Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 55-19 (2022) 109–114

XWare: a Middleware for Smart Retrofitting in Maintenance

David Sanchez-Londono * **Giacomo Barbieri** * **Kelly Garces** **

* *Department of Mechanical Engineering, Universidad de los Andes, Bogota (Colombia) (email: {d.sanchezl, g.barbieri} @uniandes.edu.co)*
** *Department of Systems and Computing Engineering, Universidad de los Andes, Bogota (Colombia) (email: kj.garces971@uniandes.edu.co)*

Abstract:
The management of industrial maintenance has rapidly evolved in the past few decades. Thanks to the advancements of Cyber-Physical Systems (CPSs) brought by Industry 4.0, the present and future health state of machines can be estimated and, as a result, smart maintenance based on real-time sensor data has emerged. One way to implement CPSs in an enterprise is through smart retrofitting: the integration of new technologies into legacy devices to enable CPS capabilities. Among these capabilities, communication involves the integration, translation and transmission of data from legacy devices to external applications. In this work, *XWare* (eXperimental middleWare) is proposed as a low-cost middleware for smart maintenance applications that eases the implementation of the communication capability into legacy devices. This implementation is enabled through the integration of various open-source technologies: MQTT, OM2M, and a custom-made library of Python functions. The integration of these components allows sensors placed on different locations to transmit maintenance-related data to a local server through the use of gateways. By methodically designing, implementing and verifying XWare, a scalable, easy to deploy, and open-source middleware for smart retrofitting in maintenance was developed.

Keywords:
Smart Retrofitting, Maintenance, Smart Maintenance, Industry 4.0, Cyber-Physical Systems.

1. INTRODUCTION

Since its formalization in the 2010s, the vision of *Industry 4.0* has revolutionized manufacturing by enabling transparency: the ability to discover uncertainties and measure real manufacturing capability (Lee et al., 2013). The digitalization technologies brought by Industry 4.0 have allowed users to access real-time data and perform data processing on the internet, enabling the optimization of manufacturing systems (Monostori et al., 2016).

A process that benefits from Industry 4.0 is *smart maintenance*: the management of maintenance activities supported by "pervasive digital technologies" (Bokrantz et al., 2020). Here, Industry 4.0 technologies enable machines that are self-aware of their own status and can self-predict their future health (Lee et al., 2015). These capabilities are enabled by the adoption of CPSs: systems which include machinery that understands its own and other's condition through data collection (Thoben et al., 2017; Barbieri and Fantuzzi, 2016). The implementation of CPSs is generally achieved by the replacement of existing legacy devices with more modern equipment, but smart retrofitting can be adopted as a more sustainable option. Smart retrofitting consists of the upgrade of existing legacy devices with functionalities able to fulfill the capabilities of CPSs.

Guerreiro et al. (2018) were among the first authors to use the term smart retrofitting. *Smart retrofitting* involves the integration of new technologies into legacy devices (Jaspert et al., 2021), with the goal of granting Industry 4.0 capabilities to devices that were not designed for it (Guerreiro et al., 2018). As a result, machines that undergo smart retrofitting are converted into CPSs (Lins and Oliveira, 2020), enabling transparency. Nowadays, such upgrades are motivated by the drive to ensure competitiveness, increase equipment efficiency, and achieve sustainability (Jaspert et al., 2021). Another goal in smart retrofitting is the conversion of legacy devices into CPSs through low time and capital investment (Guerreiro et al., 2018).

Before the execution of smart retrofitting activities, it is important to identify which capabilities must be implemented in legacy devices. One potential answer is provided by Lee et al. (2015) in their 5C architecture. They propose a "step-by-step guideline for developing and deploying a CPS architecture for manufacturing applications". In particular, the first step is the collection of reliable data from industrial devices. Any attempt to upgrade a legacy device must then consider the implementation of the connection functionality. However, this step in smart maintenance is not trivial and warrants research.

Maintenance-related data are usually fragmented across systems because of their heterogeneous semantics and formats (Christou et al., 2020). Although various Machine-to-Machine (M2M) communication standards have been developed (such as MQTT [1], MTConnect [2], and OPC-

[1] www.mqtt.org
[2] www.mtconnect.org

10.1016/j.ifacol.2022.09.192

UA [3]), heterogeneity and non-interoperability between devices remain challenging (Younan et al., 2020). *XWare* (eXperimental middleWare) is then proposed: a middleware to enable the integration, translation, and transmission of data for smart retrofitting in the maintenance context. By designing, implementing, and testing XWare, a low-cost solution is provided for the collection of data from heterogeneous sources into a single location, which is compatible with the types of data that are most commonly used in smart maintenance applications.

This work is presented as follows: Section 2 shows the design process of XWare. Section 3 illustrates its implementation, and Section 4 describes its verification. Finally, Section 5 reports conclusions and potential future works.

2. XWARE DESIGN

To design a solution to a smart maintenance problem, Li et al. (2020) propose three steps: identify requirements, design a framework, and define an architecture.

2.1 Requirements

The main requirements identified for XWare are:

- *Metamodel*: XWare must integrate multiple-sourced data (Han et al., 2012) and context information such as operative conditions and utilized sensors (Romero et al., 2020), and translate it into a unified format (or metamodel) that is compatible with the data that are commonly used in smart maintenance.
- *Two-way communication*: XWare must use protocols and technologies that allow for two-way communications between the retrofitted devices and the final location where data are transmitted. This would enable support for Digital Twins (Kritzinger et al., 2018).
- *Cost*: XWare must utilize freely available software, and non-costly hardware. This would allow a wider range of enterprises to adopt the middleware.

2.2 Framework

The next step is to create a framework. The functions identified for XWare include: collecting samples via a Gateway (device that converts data from one format to another), transmitting and receiving samples to a Local Server (location where data from multiple devices are stored), adding context to samples, and transmitting them to an Application layer (the final location where data are transmitted). The gateway is required only for legacy devices: *M2M devices* are capable of communications without needing an external gateway. In these cases, a direct connection exists between the device and the Local Server. The relationships between these components are shown in Figure 1.

2.3 Architecture

An architecture can be built by selecting specific technological solutions for each function. With XWare, the aforementioned functions can be instantiated into two solutions: a *data metamodel* and a *communication protocol*.

[3] www.opcfoundation.org/about/opc-technologies/opc-ua

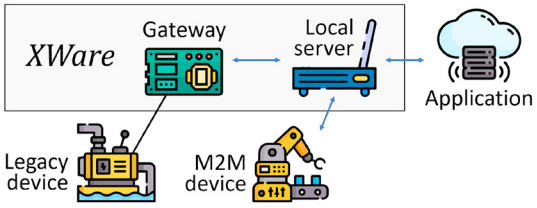

Fig. 1. Representation of the XWare framework.

Data Metamodel A standard metamodel is established for heterogeneous devices to send data to XWare. It is expected that the end user will build a translator to convert their acquired data into the XWare metamodel. Whereas, the XWare metamodel must have these characteristics:

- *Standard data types*: Represent commonly-used data in smart maintenance, such as events and time series [4] (Jardine et al., 2006).
- *Context-aware*: Include user-defined contextual information; e.g. operative condition, etc.
- *Royalty-free*: Have a royalty-free license to fulfill the cost requirement.

It was then decided that data would be represented in CSV and JSON formats and conform to the metamodel. A translator must represent event and time series data in a CSV format, and the context information of samples in a JSON format. Unlike other metamodels (such as OSA-CBM), this metamodel transmits the information without the conversion to verbose XML files (Ng et al., 2006).

The metamodel is divided in three levels: samples, measurements, and samplings as proposed in Romero et al. (2020). A *sample* includes time, tags, and a single data value. A *measurement* is a collection of samples that is stored as a CSV file. Each line in the file represents a sample – see Figure 2. Samples from multiple sensors can be included in one CSV file by using two tag fields: the first identifies the Gateway that samples come from, and the second defines the sensor to which a sample belongs. Each file also has a Sampling ID: an identifier that links the measurement with a data acquisition session, or sampling.

The *sampling* contains the context of the data acquisition session that applies to all its measurements; e.g. operative conditions and utilized sensors. This context is defined in the Application layer (Ardila et al., 2020). However, further context information must be input from the user; e.g. the sampling frequency of the acquisition (see Section 3). Since the sampling is defined in the Application layer, it would not be possible to link this information to the Sampling ID. Therefore, each measurement CSV is associated with a measurement JSON that contains the context information input by the user.

Communication Protocol Different open-source and commercial M2M technologies are available to implement the XWare functions (Naik, 2017). Section 2.1 presented the requirements for XWare. However, further requirements specific to any IIoT application should also be elicited

[4] A third commonly-used data representation is multidimensional data, such as images, but this is not considered in the current scope of XWare.

```
ID,5fc9aa0a2cfeac1222628c54,,
2021-08-26T10:45:43.866,motor1,x_accel,-3.99666
2021-08-26T10:45:43.866,motor1,y_accel,16.56835
2021-08-26T10:45:43.867,motor1,x_accel,27.26987
2021-08-26T10:45:43.867,motor1,y_accel,4.02952
2021-08-26T10:45:43.868,motor1,x_accel,12.34570
2021-08-26T10:45:43.868,motor1,y_accel,-12.98979
```

Fig. 2. Example of a measurement CSV that conforms to the XWare metamodel. Each line represents a sample. The first tag "motor1" shows that the samples come from the same Gateway. The second tag indicates if a sample comes from sensor "x_accel" or "y_accel". The first line indicates the Sampling ID of the CSV file.

(Naik, 2017). Therefore, the full list of requirements for the communication protocol is next presented:

- *Two-way communication*: It must be able to send messages from and to devices and gateways.
- *Royalty-free*: It must have a royalty-free license.
- *Quality of Service reliability*: Quality of Service (QoS) is a measurement of the certainty of messages being received. The ability to select the level of QoS should be available in the selected communication protocol.
- *Transport protocol*: between TCP and UDP, it is known that the TCP protocol is preferred when a guarantee of message reception is desired (Schenato et al., 2007).
- *Security*: Protocols should offer some level of security and privacy, as this is one of the key challenges that IIoT systems must address (Sisinni et al., 2018).
- *Encoding*: XML has been defined as an undesirable representation format for data in XWare, so binary or text encoding is preferred.

Four M2M communication protocols are compared in Table 1: CoAP [5] , OPC-UA, MQTT, and the protocol used in MTConnect networks. Out of all candidates, OPC-UA and MQTT are viable alternatives for the elicited requirements and MQTT is selected due to its better performance under similar conditions (Profanter et al., 2019; Silveira Rocha et al., 2018).

However, multiple M2M devices generally use different communication protocols (Naik, 2017). Therefore, a solution that enables XWare to be compatible with more than one protocol is preferred. Such a solution can be found in OM2M: an extensible M2M platform which allows devices with different M2M protocols to communicate (Alaya et al., 2014). Since OM2M is compatible with MQTT, OM2M is selected as a platform and MQTT is chosen as the default transmission protocol.

3. IMPLEMENTATION

This section illustrates the implementation of XWare, using the software technologies selected in Section 2.2. As hardware platform, Raspberry Pi [6] devices were used for both the Gateways and the Local Server. Raspberry Pi devices were selected for their low-cost and high software flexibility, fulfilling the requirements illustrated in Section 2.1. All code was programmed in Python 3.

[5] www.tools.ietf.org/html/rfc7252

[6] www.raspberrypi.org

3.1 XWare components

XWare is currently implemented in an environment that only uses legacy devices, and not M2M devices; see Figure 1. Sensors are connected to Gateways, which are then interfaced to the Local Server. The Gateway must fulfill the specifications shown in Section 2. The Gateway is also in charge of connecting each legacy device to the OM2M server. Figure 3 shows a diagram of the relationships between the different components, including the MQTT client and the OM2M server inside the Local Server actor.

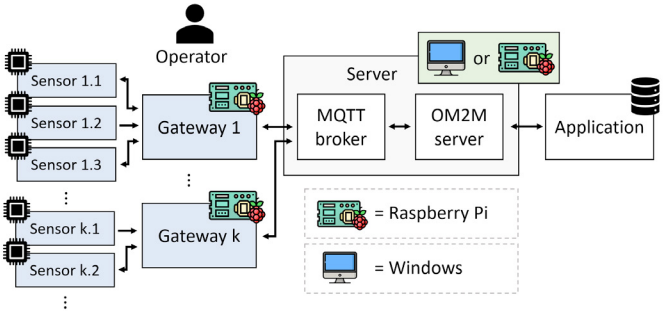

Fig. 3. Components involved in the XWare environment.

3.2 Function instantiation

Having established the components and relationships, the functions in Section 2.2 can be instantiated. Figure 4 shows how MQTT, OM2M, and custom-made XWare components (the metamodel manager and a Python library) are used to implement these functions. The steps that occur to transmit a measurement CSV are presented:

- *Initialization*: An operator first defines key parameters for each Gateway and executes the Gateway Python code. Figure 5 illustrates these parameters. The measurement time t indicates the duration of each measurement. The period T establishes the time between the beginning of each measurement [7] . Finally, the sampling frequency F (not shown in the figure) is the number of samples recorded per second.
- *Identification*: Once the Gateway code starts, the Gateway identifies itself within the OM2M Server.
- *Clock synchronization*: Knowing the starting time and sampling frequency allow the Local Server to assign timestamps to each sample. This removes discrepancies that may arise from having non-synchronized clocks between the Gateways.
- *Data collection*: The Gateway then begins collecting data following the predefined time parameters. Once the samples of a given measurement have been collected in the Gateway, they are compiled in a CSV file and sent along with the associated JSON file to the Local Server through MQTT and OM2M.

4. VERIFICATION

This section illustrates how the operation of XWare was verified. This includes: a) ensuring sample data integrity;

[7] Note that a periodic (as opposed to continuous) acquisition system was designed, since XWare is thought to be used for condition-based maintenance applications in which the state of the device changes in a timeframe of hours or days, and not within minutes or seconds.

Table 1. Comparison of various M2M communication protocols.

	MTConnect (AMT, N.D.)	CoAP (Naik, 2017)	OPC-UA [1]	MQTT (Naik, 2017)
Two-way communication	No (Sobel et al., 2019)	Pub/Sub or Request/Response	Pub/Sub or Request/Response [2]	Pub/Sub
Royalty-free license	Yes: Open Source	Yes: multiple implementations are Open Source	Yes: Open Source	Yes: multiple implementations are Open Source
QoS Reliability	*(Data not found)*	Yes	Yes [2]	Yes
Transport protocol	TCP	UDP	TCP	TCP
Security	Must be implemented externally	DTLS and IPSec	Encryption and authentication	Yes: TLS/SSL
Encoding	XML only	Binary	XML or Binary [3]	Binary

[1] (Redelinghuys et al., 2019; Schleipen et al., 2015) [2] (Profanter et al., 2019) [3] (Unified Automation GmbH, N.D.)

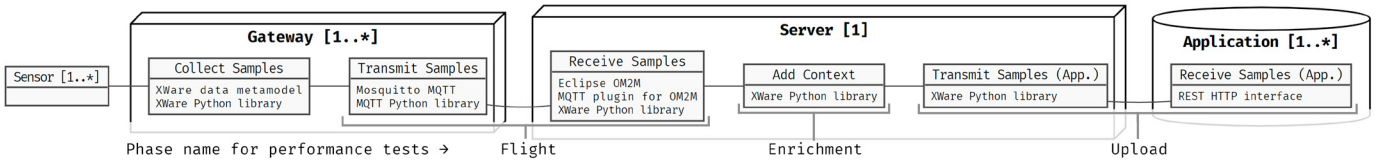

Fig. 4. Functions that the XWare components perform, along with the technological solutions that each function uses.

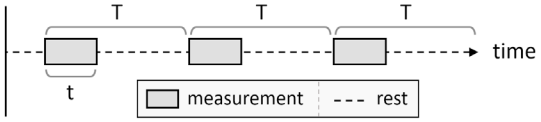

Fig. 5. Multiple measurement cycles show the t and T parameters in the sample collection process.

b) measuring the performance of the implemented architecture; c) verifying the multiple gateway functionality. Then, the fulfillment of the requirements identified in Section 2.1 is demonstrated.

To verify the correct operation of XWare, all the necessary software components were installed and tested on two separate Raspberry devices. All tests were performed with a *simulated data acquisition*, with data taken from the educational shaft unbalance dataset found in Barbieri et al. (2020). To simulate the acquisition, the Gateway read a value from this dataset every $1/F$ seconds (where F is the acquisition frequency). Furthermore, the XRepo 2.0 PHM educational information system was used as the Application layer for the testing (Romero et al., 2020). XRepo 2.0 is a big data information system that allows professors to share PHM sensor data with students and their peers. XRepo 2.0 was developed alongside the XWare metamodel, and thus the two are compatible without the need of a translator.

4.1 Data integrity verification

The goal of this verification is to ensure that the integrity of sensor data is not lost when they are sent to the Application layer. Retaining data integrity would mean that any analysis performed would give the same results whether data were acquired directly from the sensor or from the Application layer. This verification process was performed for two different samplings: one with no shaft unbalance, and the other with high shaft unbalance. Data corresponding to a single measurement file were analyzed from each dataset, taken both directly from the sensor and from XRepo. Figure 6 shows the visual comparison of both signals in the frequency domain. The equivalence of signals indicates that data integrity is maintained.

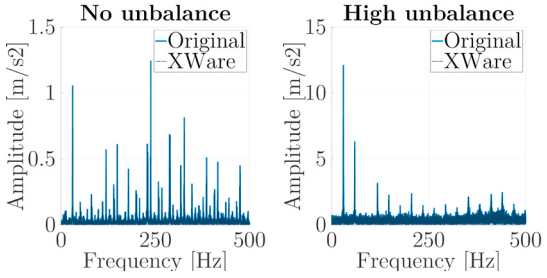

Fig. 6. Two signals (left and right) in the frequency domain, before and after passing through XWare.

4.2 Performance test

The performance of XWare is now measured. All performance tests were run using a Raspberry Pi Model 3B (Quad core ARM CPU @1.2GHz, 1GB RAM) as the Gateway and a Raspberry Pi Model 4 (Quad core ARM CPU @1.5GHz, 2GB RAM) as the Local Server.

The objective of the test was to find whether a phase of the XWare architecture required more processing time than the others, suggesting optimizations to reduce *bottlenecking*. To do this, 110 measurement cycles occurred and the time that each specific phase required was recorded. This resulted in the times found in Table 2. The mapping of the measured phases and the functions of XWare is shown in Figure 4. The uncertainty of each time was calculated using the equation: error $= 3\sigma/\sqrt{N}$, where N is the number of measurement cycles. As seen in the table, the phase that took the longest time was the "upload", which involved the process of transmitting data to XRepo. However, reducing this time resides within the scope of XRepo and not XWare. Bottleneck testing then allowed the authors to verify that the developed XWare code (along with the use of MQTT and OM2M) was sufficiently efficient without any glaring time sinks in the process.

Table 2. Time taken for 20 kHz, 1s sampling cycles to finish each stage of the XWare-XRepo data flow. All values are presented in seconds.

Sampling	Flight	Enrichment	Upload	**Total**
1±0	0.654 ±0.014	0.810 ±0.019	3.826 ±0.280	**6.290 ±0.280**

4.3 Multiple gateway verification

Next, the viability of using multiple Gateways is verified. The tests were performed by running multiple instances of the XWare Gateway code in a single Raspberry. Three Gateways were executed in one Raspberry Pi for a duration of 50 minutes. Each Gateway was assigned different time parameters, which are shown in Table 3. The same table also shows for each Gateway: how many measurements were collected and sent to the Local Server within these 50 minutes, along with the mean time that each measurement required to be sent, processed, and uploaded to the Application. If the total time for a given measurement had been higher than its respective period T, the processing of measurements would overlap and the test would fail. The total time was always lower than the period T, indicating that the architecture is able to receive measurements from multiple gateways and enrich them with context simultaneously, uploading said samples to the Application layer.

4.4 Fulfillment of requirements

Finally, the requirements in Section 2.1 are evaluated:

- *Metamodel*: a custom XWare metamodel was developed. This metamodel can represent commonly-used smart maintenance data which have been translated from multiple, potentially heterogeneous devices.
- *Two-way communication*: The XWare architecture leverages the capabilities of MQTT and OM2M, which are M2M technologies that can both transmit messages to and from any connected device.
- *Cost*: All the software technologies utilized in XWare are free to use, and required low-cost Raspberry hardware.

Table 3. Parameters and total processing time for the multiple gateway test.

	gateway0	**gateway1**	**gateway2**
Freq. F	1000 Hz	10 kHz	20 kHz
Time t	2 s	1 s	0.5 s
Period T	20 s	30 s	50 s
Population N	151	101	61
Total time	4.209 ±0.038 s	6.278 ±0.267 s	5.149 ±0.317 s

The verification results reflect that XWare was developed up to a sufficiently functional state that can successfully perform data communication.

5. CONCLUSIONS AND FUTURE WORK

XWare aims to design, implement and test a middleware architecture that fulfills the *integration*, *translation*, and *transmission* steps of data for smart retrofitting in maintenance. As part of this process, an XWare data metamodel was created, and both MQTT and OM2M were used. After the XWare architecture was generated, its functionality was verified through integrity, performance, and multiple-gateway tests. It was found that XWare does not present any performance bottlenecks with the tested low-cost hardware and that it completes a user-friendly data flow from low level sensors to the Application layer, ensuring the quality of the acquired data.

The novelty of this work does not lie on the defined CPS framework, but on its implementation. Many CPS frameworks exist in literature, and the one presented here does not intend to differentiate itself in any major way. Instead, the work focuses on building an architecture by selecting specific technological solutions for each function in the framework. XWare as an architecture then brings multiple benefits to the smart maintenance ecosystem. By establishing clear software and hardware conditions, XWare removes the ambiguity of creating a communication solution from scratch. Secondly, being able to connect multiple Gateways to a single Local Server with multiple sensors available on each Gateway (while maintaining time coherence among all the collected data) makes XWare scalable to various industrial settings. Third, the open source nature of the selected software solutions removes the need of dealing with any licensing fees, and the total cost of implementing XWare becomes substantially low.

Despite its completeness, the proposed architecture has areas where it can be improved. The XWare Local Server software can only process data from one Gateway at a time; taking advantage of code parallelization might solve this issue. Further verification (such as the use of multiple physical Gateway devices or performing data acquisition on real sensors) could help improve this middleware. Finally, compatibility with multidimensional data (e.g. images) would increase the value of XWare.

XWARE REPOSITORY

XWare is available under an open source license. The source code (along with a wiki page detailing installation and usage steps) can be found in:

`www.github.com/d-sanchezl/xware`

This repository also contains a more thorough description of the Implementation and Verification steps of XWare.

DISCLAIMER: PREVIOUS WORK ON XWARE

XWare started development as part of a graduate thesis headed by the authors at the Universidad de los Andes. The associated document can be found in the Seneca Institutional Repository: `www.repositorio.uniandes.edu.co/handle/1992/51016`. Said document was uploaded by the university to its institutional repository as part of its internal procedures. It should not be considered as previously published work, but as a draft for the present work that is not intended for consideration in the peer-reviewed academic community.

REFERENCES

Alaya, M.B., Banouar, Y., Monteil, T., Chassot, C., and Drira, K. (2014). OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32, 1079–1086.

AMT (N.D.). MTConnect Standard: Essentials Suite. URL `https://www.pathlms.com/amt/courses/10159`.

Ardila, A., Martinez, F., Garces, K., Barbieri, G., Sanchez-Londono, D., Caielli, A., Cattaneo, L., and Fumagalli, L. (2020). XRepo - Towards an information system for prognostics and health management analysis. *Procedia Manufacturing*, 42, 146–153.

Barbieri, G. and Fantuzzi, C. (2016). Design of cyber-physical systems: Definition and metamodel for reusable resources. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1–9. IEEE.

Barbieri, G., Sanchez-Londono, D., Cattaneo, L., Fumagalli, L., and Romero, D. (2020). A Case Study for Problem-based Learning Education in Fault Diagnosis Assessment. *IFAC Proceedings Volumes (IFAC PapersOnLine)*.

Bokrantz, J., Skoogh, A., Berlin, C., Wuest, T., and Stahre, J. (2020). Smart maintenance: an empirically grounded conceptualization. *International Journal of Production Economics*, 223, 107534.

Christou, I.T., Kefalakis, N., Zalonis, A., Soldatos, J., and Bröchler, R. (2020). End-to-end industrial iot platform for actionable predictive maintenance. *IFAC-PapersOnLine*, 53(3), 173–178.

Guerreiro, B.V., Lins, R.G., Sun, J., and Schmitt, R. (2018). Definition of smart retrofitting: First steps for a company to deploy aspects of industry 4.0. In *Advances in Manufacturing*, 161–170. Springer.

Han, J., Kamber, M., and Pei, J. (2012). *Data Preprocessing*.

Jardine, A.K., Lin, D., and Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7), 1483–1510.

Jaspert, D., Ebel, M., Eckhardt, A., and Poeppelbuss, J. (2021). Smart retrofitting in manufacturing: A systematic review. *Journal of Cleaner Production*, 127555.

Kritzinger, W., Karner, M., Traar, G., Henjes, J., and Sihn, W. (2018). Digital Twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11), 1016–1022.

Lee, J., Bagheri, B., and Kao, H.A. (2015). A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing Letters*, 3, 18–23.

Lee, J., Lapira, E., Yang, S., and Kao, A. (2013). Predictive manufacturing system - Trends of next-generation production systems. In *IFAC Proceedings Volumes*, volume 46, 150–156.

Li, R., Verhagen, W.J., and Curran, R. (2020). A systematic methodology for Prognostic and Health Management system architecture definition. *Reliability Engineering and System Safety*, 193, 106598.

Lins, T. and Oliveira, R.A.R. (2020). Cyber-physical production systems retrofitting in context of industry 4.0. *Computers & industrial engineering*, 139, 106193.

Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W., and Ueda, K. (2016). Cyber-physical systems in manufacturing. *CIRP Annals - Manufacturing Technology*, 65, 621–641.

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*.

Ng, W., Lam, W.Y., and Cheng, J. (2006). Comparative analysis of XML compression technologies. *World Wide Web*, 9(1), 5–33.

Profanter, S., Tekat, A., Dorofeev, K., Rickert, M., and Knoll, A. (2019). OPC UA versus ROS, DDS, and MQTT: Performance evaluation of industry 4.0 protocols. *Proceedings of the IEEE International Conference on Industrial Technology*, 2019-Febru(December 2018), 955–962.

Redelinghuys, A., Basson, A.H., and Kruger, K. (2019). A Six-Layer Digital Twin Architecture for a Manufacturing Cell. *Service Orientation in Holonic and Multi-Agent Manufacturing*, 1, 273–284.

Romero, N., Medrano, R., Garces, K., Barbieri, G., and Sanchez-Londono, D. (2020). XRepo 2.0: a Big Data Information System for Education in Prognostics and Health Management. *International Journal of Prognostics and Health Management*, –(Issue Pending), –.

Schenato, L., Sinopoli, B., Franceschetti, M., Poolla, K., and Sastry, S.S. (2007). Foundations of control and estimation over lossy networks. *Proceedings of the IEEE*, 95(1), 163–187.

Schleipen, M., Lüder, A., Sauer, O., Flatt, H., and Jasperneite, J. (2015). Requirements and concept for Plug-and-Work: Adaptivity in the context of Industry 4.0. *At-Automatisierungstechnik*, 63(10), 801–820.

Silveira Rocha, M., Serpa Sestito, G., Luis Dias, A., Celso Turcato, A., and Brandao, D. (2018). Performance Comparison between OPC UA and MQTT for Data Exchange. *2018 Workshop on Metrology for Industry 4.0 and IoT, MetroInd 4.0 and IoT 2018 - Proceedings*, 175–179.

Sisinni, E., Saifullah, A., Han, S., Jennehag, U., and Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, 14(11), 4724–4734.

Sobel, W., Armstrong, R., Turner, J., and Waddell, R. (2019). OPC UA for MTConnect - OPC UA Companion Specification Part 1 : Device Model. URL `https://www.mtconnect.org/opc-ua-companion-specification`.

Thoben, K.d., Wiesner, S., and Wuest, T. (2017). "Industrie 4.0" and Smart Manufacturing – A Review of Research Issues and Application Examples. *International Journal of Automation Technology*, 11(1).

Unified Automation GmbH (N.D.). High Performance OPC UA Server SDK. URL `https://web.archive.org/web/20200504125531/https://documentation.unified-automation.com/uasdkhp/1.2.0/html/md_opcua_binary_fileformat.html`.

Younan, M., Houssein, E.H., Elhoseny, M., and Ali, A.A. (2020). Challenges and recommended technologies for the industrial internet of things: A comprehensive review. *Measurement*, 151, 107198.