Hietala, Jani; Ala-Laurinaho, Riku; Autiosalo, Juuso; Laaki, Heikki

GraphQL Interface for OPC UA

Published in:
Proceedings - 2020 IEEE Conference on Industrial Cyberphysical Systems, ICPS 2020

DOI:
10.1109/ICPS48405.2020.9274754

Published: 10/06/2020

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

GraphQL Interface for OPC UA

1st Jani Hietala
Department of Mechanical Engineering
Aalto University
Espoo, Finland
jani.hietala@aalto.fi

2nd Riku Ala-Laurinaho
Department of Mechanical Engineering
Aalto University
Espoo, Finland
riku.ala-laurinaho@aalto.fi

3rd Juuso Autiosalo
Department of Mechanical Engineering
Aalto University
Espoo, Finland
juuso.autiosalo@aalto.fi

4th Heikki Laaki
Department of Mechanical Engineering
Aalto University
Espoo, Finland
heikki.laaki@aalto.fi

***Abstract*—Industrial Cyber-Physical Systems consist of multiple machines working together and demand efficient and flexible communication methods to function as intended. The protocols used in industrial operations and web applications are often contradictory in regards to the latency and security characteristics. Due to these differences, the intersection of operation and information technologies is a challenging area. But the rewards in smoother information flow are also high, providing a fruitful area for development. This paper introduces a general wrapper application to enable the use of the industrial OPC UA server through an interface implemented with web technology GraphQL. The results demonstrate sufficient performance for the middleware to be used in an overhead crane control application, bringing the agility of web development to industrial environments.**

***Index Terms*—GraphQL, Interface, OPC UA**

I. Introduction

Cyber-Physical Systems, Industry 4.0, and Industrial Internet of Things (IIoT) aim at enabling applications that consist of multiple intelligent machines communicating with each other. The communication requires the use of the same standardized interfaces and protocols across different stakeholders and applications. However, the various applications often have diverse and even contradictory demands, which has led to a situation where there is a plethora of different interfaces and protocols [1]–[4]. Each of them was developed to serve a distinct purpose and there are both technological and sociological reasons why they all exist at the same time even though it might be more sensible to have just a couple of good methods of communication. This paper concentrates on the intersection of two groups: the industrial and the web development communities.

Industrial environments often include time-sensitive applications, ranging from inverter-controlled electrical motors and surgery robots [5] to high accuracy load positioning for cranes [6]. The applications typically require high reliability and security of communication protocols, as human lives may be at risk if communication fails. Traditional web applications are usually less time-sensitive and delays in communication mostly cause discomfort in users instead of risking lives. From the security point of view, compromises in web technologies may cause financial losses or information breaches, of which the latter may enable malicious actors to cause physical harm. With the introduction of Internet of Things devices, security issues in web technologies [7] are causing also physical security risks. Regardless of the security risks, the developments of web technologies enable their use also in control applications, e.g. in smart homes [8].

Traditional industrial applications have been implemented in closed environments, enabling strict control of both physical and digital security. However, isolation from the outside world comes at a cost, one of which is the opportunity cost of having slower and more complicated information flow than would be possible with the latest technologies. The operational level is always providing information to the other hierarchy levels of companies, creating a link between operations technology (OT) and information technology (IT). The first application leveraging this link is typically the monitoring of operations for performance enhancement and even direct control.

Digital twin has emerged as a conceptual entity that connects the digital and physical worlds. The added value of the digital twin concept is mostly in providing semantic structure for information networks and therefore it is inherently dependent on other technologies before it can provide any actual benefits. Digital twins need other technologies as building blocks and data from physical devices. As digital twins exist in the information domain, the connection between OT and IT is crucial for them. This paper facilitates digital twins by introducing a wrapper software implementation between an OT protocol ”OPC Unified Architecture” (OPC UA) [9] and an IT protocol ”GraphQL”[10].

OPC UA is a widely employed industrial communication protocol, which allows monitoring and control of industrial machinery. However, it is a relatively complex protocol and induces large overhead for individual requests, because several handshakes are required in connection establishment [11]. In addition, it is not suitable for the communication of constrained devices without modifications [12].

This work was supported by the Business Finland under Grant 8205/31/2017 “DigiTwin,” and Grant 3508/31/2019 “MACHINAIDE”.

Developers are seldom familiar with industrial protocols – unlike Web protocols. Web protocols are more scalable, efficient, suitable for constrained devices, and simpler to use compared to OPC UA. Thus, several papers [11], [13], [14] have proposed methods to make OPC UA RESTful. REST is an architectural style based on stateless and resource-oriented communication. RESTful architecture is at the core of the current development of web applications. However, despite its benefits such as scalability and ease of use, RESTful communication is an inefficient way of querying multiple objects. Therefore, this paper suggests a novel GraphQL wrapper for the OPC UA industrial interface, which offers a stateless, easy-to-use and efficient way to access objects in the OPC UA server. The paper builds upon the master's thesis [15] of the first author. The main contributions of the paper are as follows:

- Introducing GrahpQL wrapper for OPC UA with source code available in GitHub [16]
- Demonstrating the use of the developed wrapper by introducing a control application for an industrial overhead crane
- Accelerate application development for Industry 4.0 and Cyber-Physical Systems with GraphQL wrapper for commonly used industrial protocol

II. Literature review

A. OPC UA

OPC UA is an industrial protocol for communication standardized in IEC 62541 and currently developed by OPC Foundation [11]. It is designed to be platform-independent and can be used in IIoT, M2M, and Industry 4.0 [9]. The OPC UA client/server communication is based on services offered by OPC UA servers. These services are used to interact with the resources, which are organized with the object-oriented information model [11]. The resources are represented with nodes, which can describe real-world objects such as process variables and their relations [12]. Nodes have attributes based on their NodeClass [17], and one of the benefits of the OPC UA information model is that it can provide information also on the quality of the data [18].

B. RESTful architectural style

REST is a "set of design criteria" [19] for "distributed hypermedia systems" introduced by Fielding in his dissertation [20]. According to Fielding, these design criteria include e.g. statelessness, client-server model, in which a client requests services from a server, and a possibility for caching responses. If a software fullfills this set of criteria, it is called RESTful. Richardson and Ruby present [19] that a RESTful system can be implemented by following Resource-Oriented Architecture (ROA). In ROA, each resource has an address, which identifies the object and can be used to access it. In addition, all resources can be interacted with via similar interfaces – in Web – using HTTP and its methods (GET, POST, PUT, DELETE).

RESTful architectural style is not bound to a specific protocol [11]. Yet, REST is often used as a description for HTTP APIs – even though these APIs often do not fulfill RESTful criteria completely. REST APIs are currently the most prevailing way of implementing web communication. Statelessness of RESTful communication allows high scalability as information about the client is not stored on the server and responses can be cached.

C. Approaches for RESTful OPC UA

Several solutions to allow RESTful communication with OPC UA has been presented. Grüner et al. enabled RESTful communication by allowing communication without several handshakes and adding expiration tags for caching [11], [21]. They also propose an extension to OPC UA standard which allows communication optionally over UDP (User Datagram Protocol). The comparison of stateless OPC UA and standard OPC UA showed a significant improvement in terms of latency for stateless communication. Paronen developed a monitoring application for IIoT following the RESTful architectural style [22]. The service layer of the application is used to map HTTP requests to corresponding OPC UA services and the presentation layer offers graphical human-machine interface (HMI).

Schiekofer et al. [14] present an approach for RESTful OPC UA interface, which also addresses the problem with dynamic Namespace and ServerArray, unlike [21]. In addition, they implemented a prototype following the approach based on the Java OPC UA Stack. Cavalier et al. introduce an OPC UA Web Platform consisting of Web Service Interface module, which handles the requests, and a Middleware module, which includes OPC UA Client enabling communication with OPC UA Servers [13], [23]. In addition, the developed platform has a broker, which allows monitoring services i.e. changes to certain Variable Nodes can be subscribed by a client. To enhance the interoperability of OPC UA, Derhamy et al. propose a protocol translator, which converts requests made with common IoT protocols such as HTTP, CoAP or MQTT to OPC UA compatible [24]. CoAP is a request/response protocol, which allows RESTful communication on constrained devices [25].

D. GraphQL

GraphQL is both a query language for building queries and an execution engine for performing these queries on the server-side. The development of GraphQL originally started at Facebook in 2012 and it was used internally until 2015 when an open specification was released [10]. The specification does not define any programming language specific instructions. Consequently, various community projects have been created since then to make GraphQL available for different programming languages and platforms. GraphQL has been growing in popularity among web service providers since its release to the public. Some of this can be attributed to its front-end developer and application centered design principles. Queries are structured in a way that all desired data can be fetched with a single HTTP request to the GraphQL endpoint. Moreover, as only necessary resources are returned, network traffic is

minimized. Despite these benefits, a GraphQL interface for OPC UA has not yet been implemented.

E. Comparison of GraphQL and REST

Both GraphQL and REST are used to retrieve data via an application programming interface (API). In addition, both protocols use the HTTP protocol. However, REST APIs use HTTP Methods (GET, POST, PUT, DELETE) for implementing CRUD (Create, Read, Update, Delete) operations, whereas GraphQL uses API specific methods. GraphQL supports GET and POST requests to make queries.

GraphQL offers several benefits compared to REST APIs. With GraphQL, one query can interact with multiple resources whereas REST APIs follow the resource-oriented architecture and, thus, only one resource can be modified with a single query. GraphQL allows introspection of the schema and supported queries. In addition,I the hierarchical queries of GraphQL [10] matches to OPC UA Information Model which is a graph-like [21]. GraphQL supports subscriptions and it has possibility to deprecate outdated fields [10].

III. OPC UA - GraphQL wrapper

The purpose of the wrapper is to serve as an additional interface for the information that is available on any OPC UA server. The wrapper functions as a broker between the client and the OPC UA server. It translates GraphQL queries from clients into OPC UA service requests and passes them forward to the OPC UA server. The response is also transformed into a GraphQL response. Thus, clients only need to know how to communicate with the GraphQL wrapper in order to consume data from the OPC UA server. One of the requirements for the wrapper is that it can be added retrospectively to any system without modifying the existing OPC UA server implementation. Hence, approaches similar to those used by [11] could not be used. Moreover, due to the wrapper not being integrated into a single OPC UA server, it is capable of aggregating multiple unique servers under the same GraphQL interface. Besides basic server setup, the wrapper was required to support operations for nodes on the OPC UA server as seen in Table I.

TABLE I: OPC UA - GraphQL Wrapper node operations.

Read	**Write**	**Add**
DisplayName Description Variable • DataValue • DataType • SourceTimestamp • Statuscode NodeId Child nodes	DataValue Description	Folder node Variable node

The wrapper is built on the Starlette framework and is run by the Uvicorn Asynchronous Server Gateway Interface (ASGI) server. Starlette with Uvicorn is considered as one of the fastest Python-based web frameworks [26]. Starlette readily supports GraphQL for which schemas are built using the Graphene library. Communication with OPC UA servers rely on Python OPC UA library. The wrapper application has been containerized with Docker to allow an effortless setup of new implementations. Containerization may prove particularly useful in the future if Docker workers reach the factory floor as Alam et al. [27] suggested. If Docker workers are already on the factory floor, it is trivial to add the GraphQL wrapper to existing systems as an additional feature.

The wrapper typically retrieves data from the OPC UA server in batched requests. Clients may request any combination of node attributes which are then batched together into a single service request for the OPC UA server. The first query of a session when retrieving data via the GraphQL wrapper takes longer than its subsequent requests as the session is formed between the wrapper and the OPC UA server. Following queries, even from new clients, use the existing session to reduce the communication traffic overhead and latency.

IV. Performance analysis

A. Measuring setup for performance analysis

The performance measurements for the GraphQL wrapper were conducted using a test setup shown in Fig. 1. In the test setup, GraphQL wrapper and a test OPC UA server were run on Raspberry Pi 1 and 2, respectively. Laptop 1 was used as a test client, sending the test requests with a Python script. Measurements were made with Laptop 2 using Wireshark, a network traffic capture software. The Raspberry Pis, the client and the measuring laptop are connected via an Ethernet switch and cables. To ensure disturbance-free measurements, we configured the switch to act as a normal switch for ports 1-3 and to mirror all packets from them to port 8. Each test were performed 50 times to minimize the effects of minor disturbances.

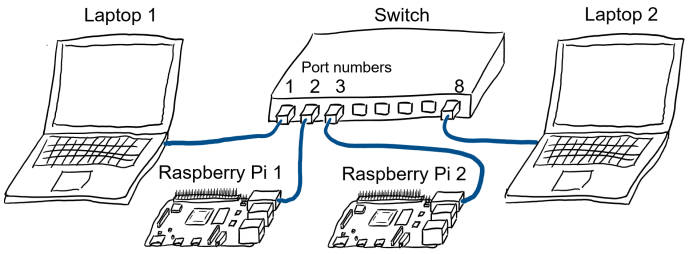
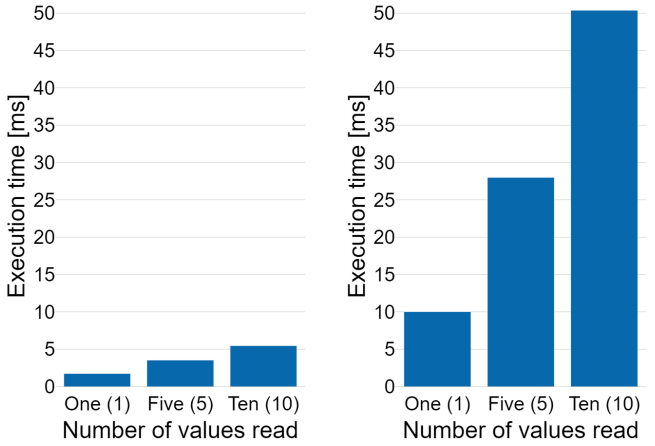


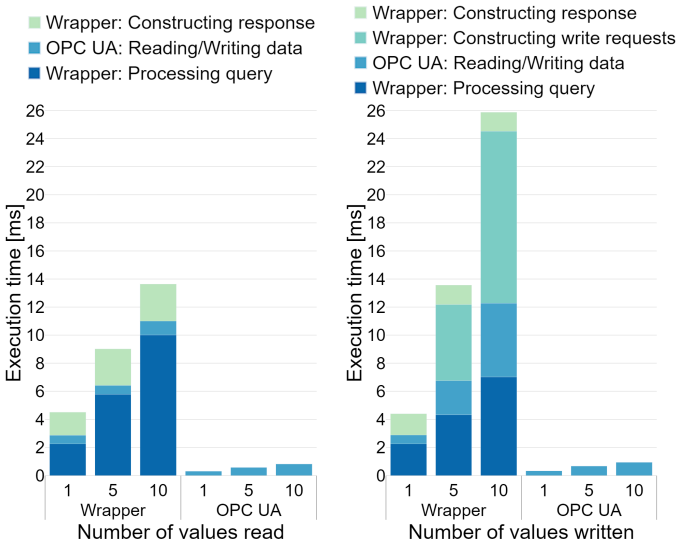
Fig. 1: Layout of the test setup.

The same value or values on the OPC UA server were requested from the wrapper and directly from OPC UA server in the tests. This way, each time a network packet passed the Ethernet switch, the measuring software Wireshark on Laptop 2 captured it with timestamps. By using these timestamps, the time to complete the request by each device could be measured. Additionally, to evaluate the performance of the wrapper alone, the time to return a value that is not fetched from the OPC UA server was measured. This eliminates the



(a) Local (b) Test setup

Fig. 2: GraphQL wrapper query execution times when resolving value internally without the OPC UA server (average of 50 queries).



(a) Read (b) Write

Fig. 3: The average query execution times for reading and writing values with the system ran locally (average of 50 queries).

effect of OPC UA server to the performance measurements. To examine the effect of the system performance on the request execution times, the tests were run both in the test system and internally on a laptop which had a considerably better performance than a Raspberry Pi. The Raspberry Pis used in the tests were 3 Model B+ and had ARM Cortex-A53 1.4 GHz quad-core processor and 1 GB RAM. The test laptop running the tests locally had a quad-core processor (Intel i5-8365U @ 1.6 GHz) with a max turbo frequency of 4.10 GHz and 16 GB RAM.

B. Measurement results

The measurement results on fetching data from GraphQL wrapper without the wrapper having to fetch data from the OPC UA server are presented in Fig. 2 and Table II. The differences in execution times between the local and test setup tests are considerable. This can be explained with the difference in performance between the Raspberry Pi and the laptop.

TABLE II: Statistical properties of GraphQL wrapper query execution times when resolving value internally without the OPC UA server.

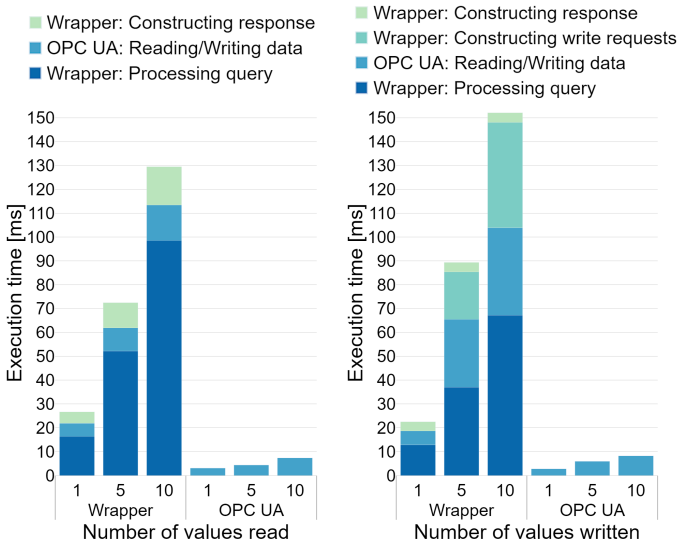
Test	Min	Max	Mean	Median	SD
Local 1	1.555	2.169	1.745	1.754	0.128
Local 5	3.148	5.425	3.487	3.408	0.347
Local 10	5.127	7.117	5.503	5.412	0.349
Test setup 1	9.786	10.280	10.051	10.048	0.101
Test setup 5	27.671	29.145	27.852	27.806	0.202
Test setup 10	50.000	50.819	50.293	50.206	0.212

The query execution times on reading and writing data to the OPC UA server both via the wrapper and directly to the OPC UA server are shown in Table III. These tests were run locally on the test laptop. The request execution times with wrapper are noticeably slower than directly requesting the OPC UA server. Processing within the wrapper takes up most of the total execution time (Fig. 3). Wrapper read requests to the OPC UA server take roughly the same amount of time than direct read requests, whereas write requests are significantly slower. The difference between reading and writing times with wrapper can be explained by the different way GraphQL handles these operations. Reading is done asynchronously which enables request batching for the OPC UA server, whereas writing is handled synchronously i.e. each value is written separately to the OPC UA server.

TABLE III: Statistical properties of query execution times for reading and writing values with the system ran locally.

Test	Min	Max	Mean	Median	SD
Read wrapper 1	3.662	6.879	4.460	4.312	0.638
Read wrapper 5	8.027	16.285	9.013	8.588	1.448
Read wrapper 10	12.745	25.386	13.657	13.168	1.817
Read OPC UA 1	0.342	0.471	0.364	0.356	0.023
Read OPC UA 5	0.539	0.695	0.567	0.558	0.029
Read OPC UA 10	0.738	10.083	0.809	0.790	0.062
Write wrapper 1	3.373	11.780	4.420	4.129	1.211
Write wrapper 5	11.958	17.599	13.542	13.065	1.268
Write wrapper 10	22.886	42.133	25.893	24.997	2.908
Write OPC UA 1	0.382	0.669	0.442	0.432	0.053
Write OPC UA 5	0.542	1.109	0.682	0.676	0.094
Write OPC UA 10	0.835	1.219	0.936	0.918	0.061

The similar measurements as in the previous section using the test setup are shown in Fig. 4 and Table IV. Compared to the locally run tests, execution times are significantly higher. This can be mostly explained by the performance difference between the test setups.



(a) Read (b) Write

Fig. 4: Query execution times for reading and writing values with the test setup (average of 50 queries).

TABLE IV: Statistical properties of query execution times for reading (=R) and writing (=W) values with the test setup.

Test	Min	Max	Mean	Median	SD
R wrapper 1	26.342	27.812	26.751	26.728	0.263
R wrapper 5	71.82	74.152	72.452	72.304	0.560
R wrapper 10	126.918	202.497	129.440	127.670	10.469
R OPC UA 1	2.445	5.596	3.211	2.568	1.184
R OPC UA 5	4.026	4.650	4.321	4.320	0.110
R OPC UA 10	6.448	14.751	7.382	6.584	2.403
W wrapper 1	22.199	23.157	22.605	22.568	0.209
W wrapper 5	88.294	90.674	89.254	89.262	0.550
W wrapper 10	142.077	173.207	152.102	142.790	12.722
W OPC UA 1	2.356	2.963	2.662	2.670	0.093
W OPC UA 5	5.030	11.487	5.911	5.134	2.052
W OPC UA 10	8.052	8.472	8.148	8.179	0.065

V. Control Application for an Overhead Crane

For demonstrating the possibilities of GraphQL wrapper and validating the goal of easy application development, a control application for an industrial overhead crane [28] was implemented. The control application allows moving the crane using the web user interface, which is designed for mobile devices. In addition to controlling the crane, the application can be used to monitor the state of the crane. The monitored variables can be freely chosen by the user and the application is able to update them continuously. Fig. 5 shows the user interface with buttons used to control the crane and monitored variables. The user interface is also able to show live stream from the camera attached above the hook.

The application is implemented with Flask, which is a lightweight web framework for Python. The application makes several requests per second to the GraphQL interface. These messages consist of control commands, reading values from the OPC UA Server of the crane, and incrementing a watchdog. The watchdog is used as an additional safety feature to prevent receiving commands from malfunctioning software.

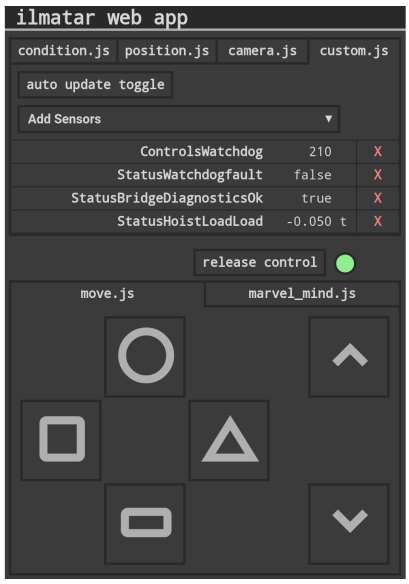


Fig. 5: An overhead crane can be controlled with a mobile device using the developed control application [15].

VI. Conclusion

This paper presented a GraphQL wrapper for OPC UA to allow more developer-friendly and quick application development for Cyber-Physical Systems and Industry 4.0, bridging the gap between web technologies and industrial applications. The wrapper can be plugged parallel to an existing OPC UA server to provide a GraphQL API to the OPC UA nodes. Hence, our wrapper implementation enhances the interoperability of industrial machines with OPC UA interface. Compared to RESTful OPC UA implementations, GraphQL offers a more compatible hierarchical query structure, allows interaction with several resources with a single query, and enables introspection of the schema.

The wrapper was implemented with the Starlette Python web-framework and the Uvicorn server. The measurements show that adding GraphQL wrapper for OPC UA significantly increases the latency compared to the direct requests to an OPC UA server. However, for example, the latency of reading five values from the OPC UA server is below 75 ms, indicating the wrapper is suitable for many web-based applications. The use of the wrapper was demonstrated with a web control application for an industrial overhead crane. The tests also show that increasing the performance of the wrapper server significantly reduces the latency.

Future research activities include identifying the sources of the high execution times and lowering the overall latency caused by the wrapper. Furthermore, the wrapper should be validated on diverse OPC UA server implementations.

References

[1] A. Talaminos-Barroso, M. A. Estudillo-Valderrama, L. M. Roa, J. Reina-Tosina, and F. Ortega-Ruiz, "A machine-to-machine protocol benchmark for eHealth applications – use case: Respiratory rehabilitation," *Computer Methods and Programs in Biomedicine*, vol. 129, pp. 1–11, 2016, ISSN: 0169-2607. DOI: 10.1016/j.cmpb.2016.03.004.

[2] L. Durkop, B. Czybik, and J. Jasperneite, "Performance evaluation of m2m protocols over cellular networks in a lab environment," presented at the 2015 18th International Conference on Intelligence in Next Generation Networks, 2015, pp. 70–75. DOI: 10.1109/ICIN.2015.7073809.

[3] R. Ala-Laurinaho, "Sensor data transmission from a physical twin to a digital twin," 2019. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201905123028.

[4] S. Rasool, R. Khan, and A. N. Mian, "GraphQL and DC-WSN-based cloud of things," *IT Professional*, vol. 21, no. 1, pp. 59–66, 2019, ISSN: 1941-045X. DOI: 10.1109/MITP.2018.2876982.

[5] H. Laaki, Y. Miche, and K. Tammi, "Prototyping a digital twin for real time remote control over mobile networks: Application of remote surgery," *IEEE Access*, vol. 7, pp. 20 325–20 336, 2019, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2897018.

[6] H. Sjöman, J. Autiosalo, J. Juhanko, P. Kuosmanen, and M. Steinert, "Using low-cost sensors to develop a high precision lifting controller device for an overhead crane—insights and hypotheses from prototyping a heavy industrial internet project," *Sensors*, vol. 18, no. 10, p. 3328, 2018. DOI: 10.3390/s18103328.

[7] M. A. Khan and K. Salah, "IoT security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018, ISSN: 0167-739X. DOI: 10.1016/j.future.2017.11.022.

[8] S. Kim, J.-Y. Hong, S. Kim, S.-H. Kim, J.-H. Kim, and J. Chun, "Restful design and implementation of smart appliances for smart home," presented at the 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops, ISSN: null, 2014, pp. 717–722. DOI: 10.1109/UIC-ATC-ScalCom.2014.64.

[9] OPC Foundation, *OPC unified architecture specification part 1: Overview and concepts release 1.04*, 2017.

[10] GraphQL Foundation. (2020). GraphQL specification versions, [Online]. Available: http://spec.graphql.org/draft (visited on 01/20/2020).

[11] S. Grüner, J. Pfrommer, and F. Palm, "A RESTful extension of OPC UA," presented at the 2015 IEEE World Conference on Factory Communication Systems (WFCS), 2015, pp. 1–4. DOI: 10.1109/WFCS.2015.7160557.

[12] J. Imtiaz and J. Jasperneite, "Scalability of OPC-UA down to the chip level enables "Internet of Things"," in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, ISSN: 2378-363X, 2013, pp. 500–505. DOI: 10.1109/INDIN.2013.6622935.

[13] S. Cavalieri, D. Di Stefano, M. G. Salafia, and M. S. Scroppo, "A web-based platform for OPC UA integration in IIoT environment," presented at the 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), ISSN: 1946-0759, 2017, pp. 1–6. DOI: 10.1109/ETFA.2017.8247713.

[14] R. Schiekofer, A. Scholz, and M. Weyrich, "REST based OPC UA for the IIoT," presented at the 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), ISSN: 1946-0740, vol. 1, 2018, pp. 274–281. DOI: 10.1109/ETFA.2018.8502516.

[15] J. Hietala, "Real-time two-way data transfer with a digital twin via web interface," 2020. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-202003222557.

[16] J. Hietala. (2019). GraphQL API for OPC UA servers, [Online]. Available: https://github.com/AaltoIIC/OPC-UA-GraphQL-Wrapper (visited on 01/10/2020).

[17] S. Cavalieri, D. Di Stefano, M. G. Salafia, and M. S. Scroppo, "Integration of OPC UA into a web-based platform to enhance interoperability," presented at the 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), ISSN: 2163-5145, 2017, pp. 1206–1211. DOI: 10.1109/ISIE.2017.8001417.

[18] K. Folkert and M. Fojcik, "Ontology-based integrated monitoring of Hadoop clusters in industrial environments with OPC UA and RESTful web services," in *Computer Networks*, P. Gaj, A. Kwiecień, and P. Stera, Eds., Cham: Springer International Publishing, 2015, pp. 162–171, ISBN: 978-3-319-19419-6.

[19] L. Richardson and S. Ruby, *RESTful web services*. Farnham: O'Reilly, 2007, 419 pp., OCLC: ocm82671871, ISBN: 978-0-596-52926-0.

[20] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Doctoral dissertation, University of California, Irvine, 2000.

[21] S. Grüner, J. Pfrommer, and F. Palm, "RESTful industrial communication with OPC UA," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, pp. 1832–1841, 2016, ISSN: 1941-0050. DOI: 10.1109/TII.2016.2530404.

[22] T. Paronen, "A web-based monitoring system for the industrial internet," p. 107, 2015. [Online]. Available: http://urn.fi/URN:NBN:fi:aalto-201505142682.

[23] S. Cavalieri, M. G. Salafia, and M. S. Scroppo, "Integrating OPC UA with web technologies to enhance interoperability," *Computer Standards & Interfaces*,

vol. 61, pp. 45–64, 2019, ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2018.04.004.

[24] H. Derhamy, J. Rönnholm, J. Delsing, J. Eliasson, and J. van Deventer, "Protocol interoperability of OPC UA in service oriented architectures," presented at the 2017 IEEE 15th International Conference on Industrial Informatics (INDIN), ISSN: 2378-363X, 2017, pp. 44–50. DOI: 10.1109/INDIN.2017.8104744.

[25] Z. Shelby, K. Hartke, and C. Bormann. (2014). The constrained application protocol (CoAP), [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt (visited on 01/20/2020).

[26] TechEmpower. (2019). Web framework performance comparison, [Online]. Available: https://www.techempower.com/benchmarks/#section=data-r18&hw=ph&test=query&l=zijzen-7 (visited on 01/20/2020).

[27] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for IoT using docker and edge computing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 118–123, 2018, ISSN: 0163-6804. DOI: 10.1109/MCOM.2018.1701233.

[28] J. Autiosalo, "Platform for industrial internet and digital twin focused education, research, and innovation: Ilmatar the overhead crane," presented at the 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), ISBN: 978-1-4673-9944-9, 2018, pp. 241–244. DOI: 10.1109/WF-IoT.2018.8355217.