Feature-based Comparison of Open Source OPC-UA Implementations

Nikolas Mühlbauer[1] Erkin Kirdan[2] Marc-Oliver Pahl[3] Karl Waedt[4]

Abstract: OPC UA is an industry-standard architecture for automation, process controlling and monitoring. It is a detailed and complex machine-to-machine communication protocol which makes it challenging to implement. The complexity of the protocol leads to heterogeneity among implementations. Today, there are several open-source implementations without a compliance certificate accredited by the OPC Foundation. Certified implementations undergo various tests to fulfil interoperability. Every implementation fits different use-cases and requirements as each of them comes with its own features. In this paper, we make a feature-based comparison of the most common open-source OPC UA implementations. We investigate their support for the essential features and functionalities. Furthermore, we evaluate their interoperability. Overall, our study shows that open-source implementations have good coverage of features and functionalities, especially open62541 and UA-.NETStandard. Furthermore, our tests show that they do not have any significant interoperability issue.

Keywords: opc ua; open-source; interoperability

1 Introduction

OPC UA is a machine-to-machine communication protocol widely used in industrial automation [RC18]. It has use cases across the fields of transportation, oil and gas, energy and utilities and automation [OPb]. It is platform-independent and thus can run on various operating systems and hardware platforms. OPC UA can scale down to embedded controllers or mobile devices up to powerful servers controlling a collection of machines. Furthermore, it can also be integrated into cloud platforms [OPc].

OPC UA is used in critical infrastructures. It is developed above the already mature and widely used OPC Classic protocol. OPC UA is composed of detailed and complex series of specifications. The complexity of the protocol leads to variations between implementations. Consequently, OPC UA implementations are having a different level of compliance to the specification. The OPC foundation certificates implementations from various points such as compliance, interoperability, robustness usability and efficiency to control this heterogeneity. Implementations having the compliance certificates can communicate with each other, whereas this is not guaranteed for the non-certified implementations.

[1] Technical University of Munich, n.muehlbauer@tum.de
[2] Covalion, Framatome/Technical University of Munich, erkin.kirdan@framatome.com/erkin.kirdan@tum.de
[3] IMT Atlantique/Technical University of Munich, marc-oliver.pahl@imt-atlantique.fr/pahl@tum.de
[4] Framatome GmbH, karl.waedt@framatome.com

doi:10.18420/inf2020_34

OPC UA is deployed in use cases having different operating systems and hardware together. There is no single OPC UA implementation that fits every use-case. A typical OPC UA scenario is depicted in Fig. 1. Some machines run an OPC UA server that is developed by the machine manufacturer. The manufacturer can also deploy the machine with a third-party OPC UA implementation. Some IT companies provide cloud solutions for OPC UA that are integrated into the scenario. The main server collects data from the machines and makes them available for the office computer for organizational purposes. An implementation written in JavaScript can be preferred to develop a browser-based application for the offices. A C-implementation fits well to the mobile clients as well as embedded controllers due to its efficient resource usage. Implementations written in C# or Python can be preferred in the main server because of their languages. Principally, several OPC UA implementations should be able to run together in such a scenario. However, despite the commercial ones, open-source implementations mostly lack certificates.

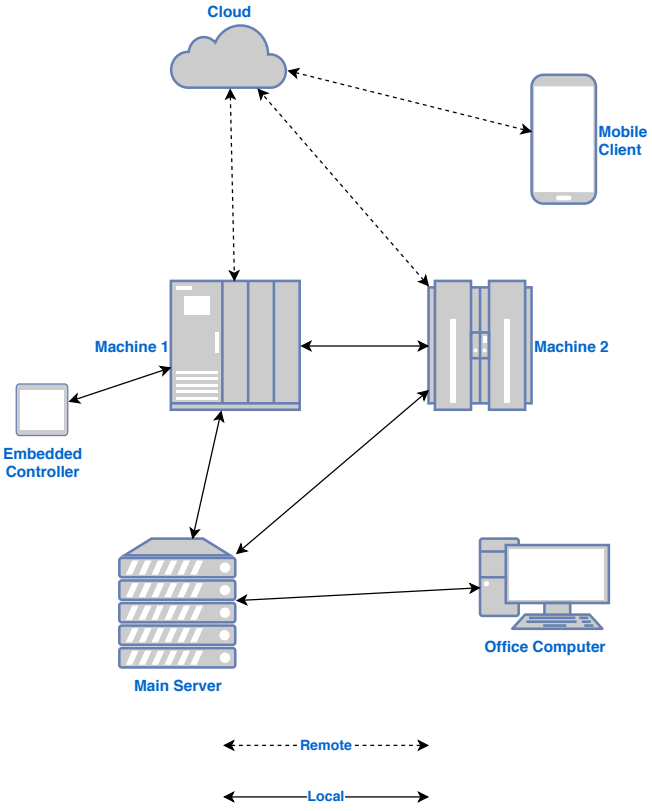

Fig. 1: OPC UA scenario

OPC UA is a client-server architecture [17a] and has been extended with the publish-subscribe pattern [18a] in 2018. However, most of the open-source implementations only support the client-server model, which relies on SecureChannels and Sessions [18b]. First, a client opens a SecureChannel which ensures integrity, confidentiality and application authenticity. Then a Session is created between the client and the server on top of a SecureChannel for user authorization and authentication. The client can access data or call methods on the server through Services [17c] such as the Browse or Read Services. In OPC UA, data and methods are implemented as nodes that are interconnected by references. Nodes and References form the hierarchical OPC UA address space [17b].

There are many open-source OPC UA implementations. For our consideration in this paper, an implementation must be

1. actively maintained,
2. full-stack server implementation and
3. completely open-source.

Among them, we selected the most popular ones based on the star counters in their GitHub pages and verified the ranking using trends in web search engines: open62541 [PP], node-opcua [Roa], UA-.NETStandard [OPd] and python-opcua [Frb]. In this group, open62541 has already the certificate and thus can be taken as reference in the interoperability tests. Furthermore, UA-.NETStandard is also certified and the reference implementation of the OPC foundation.

Since in 2015, the reference implementations of the OPC Foundation were not available for free and other open-source projects were not ready to be used in industrial automation, the authors of [Pa15] implemented open62541. The authors give a short feature comparison of open-source implementations back in 2015, including node-opcua and open62541. In [Ha17], both open-source and commercial OPC UA implementations are compared according to their features. However, some of the implementations have been discontinued, and others have been extended with new features. Also, the specification is extended with new services and security policies. The authors of [Ce19] test the open-source OPC UA implementations open62541, freeOpcUa C++ and python-opcua on embedded devices. This includes cross-wise RTT and service time measurements. A comparison in a broader sense is given in [Pr19], where the IoT protocols OPC UA, ROS, MQTT and DDS are investigated. Apart from a feature comparison of the protocols, the authors also analyse the protocol overhead and perform RTT measurements. These include tests with open62541 in client-server and in pub-sub mode. In [Mu20], the authors analysed the same open-source implementations covered in this paper from security and scalability aspects. The results show considerable differences in the scalabilities of the implementations depending on their languages. Moreover, the implementations have similar security models.

Our contributions in this paper are as follows. We compare the available features and functionalities of commonly used OPC UA implementations in Sect. 2. We evaluate their interoperabilities with various cross-wise connection tests in Sect. 3.

2 Features

OPC UA offers a set of many different functionalities which are continuously extended. Therefore, most implementations only support a subset of them. In this section, we give an overview of the supported Services, that are offered by OPC UA servers and can be used by clients. For most of the services, one needs to set up a SecureChannel that is secured by one of the standardized Security Policies, which are investigated in this section. Additionally, the different transport protocols of OPC UA are given. Finally, we provide an overview of implementations, offering clients a graphical user interface (GUI). To get this information, we inspect the source code and the readme-files in the repositories of the implementations.

2.1 Supported Services

The Specification part 4 [17c] defines the Services and aggregates them into Service Sets. Tab. 1 lists all currently standardized Services and whether they are implemented or not.

The Discovery Service Set is used by clients to find Servers and vice versa. Servers can publish their endpoints, i.e. the URIs including transport protocol and security settings. After the discovery, a client can connect to a Server by first establishing a Secure Channel and then a session using the respective services. Then, multiple services are available to the client: The NodeManagement Service Set can be used to add or delete nodes of the server's address space. For getting an overview of the address space, the Browse Service Set offers methods to show the sub-nodes of a node in the server. To view or alter the value of a variable, the View Service Set is used. When a server offers Methods, a client can call them via the Method Service Set. For clients, it is often of interest to get notified about data changes. This is achieved by creating a subscription using the Subscription Service Set, creating and binding Monitored Items to this Subscription and finally sending Publish requests to the server.

While UA-.NETStandard implements all Services, the other open-source projects miss some functionality. However, the major and commonly used services are available in all implementations. The Query Service Set is only supported by UA-.NETStandard. node-opcua does not support the management of nodes, i. e. adding and deleting nodes and references are not possible.

Tab. 1: Supported Services

Service	open62541	node-opcua	python-opcua	UA-.NETStandard
Discovery Service Set				
FindServers	✓	✓	✓	✓
FindServersOnNetwork	✓	✓		✓
GetEndpoints	✓	✓	✓	✓
RegisterServer	✓	✓	✓	✓
RegisterServer2	✓	✓	✓	✓
SecureChannel Service Set				
OpenSecureChannel	✓	✓	✓	✓
CloseSecureChannel	✓	✓	✓	✓
Session Service Set				
CreateSession	✓	✓	✓	✓
ActivateSession	✓	✓	✓	✓
CloseSession	✓	✓	✓	✓
Cancel		✓		✓
NodeManagement Service Set				
AddNodes	✓		✓	✓
AddReferences	✓		✓	✓
DeleteNodes	✓		✓	✓
DeleteReferences	✓		✓	✓
View Service Set				
Browse	✓	✓	✓	✓
BrowseNext	✓	✓		✓
TranslateBrowsePathsT.	✓	✓	✓	✓
RegisterNodes	✓	✓	✓	✓
UnregisterNodes	✓	✓	✓	✓
Query Service Set				
QueryFirst				✓
QueryNext				✓
Attribute Service Set				
Read	✓	✓	✓	✓
HistoryRead	✓		✓	✓
Write	✓	✓	✓	✓
HistoryUpdate	✓			✓
Method Service Set				
Call	✓	✓	✓	✓
MonitoredItem Service Set				
CreateMonitoredItems	✓	✓	✓	✓
ModifyMonitoredItems	✓	✓	✓	✓
SetMonitoringMode	✓	✓		✓
SetTriggering				✓
DeleteMonitoredItems	✓	✓	✓	✓
Subscription Service Set				
CreateSubscription	✓	✓	✓	✓
ModifySubscription	✓	✓	✓	✓
SetPublishingMode	✓	✓		✓
Publish	✓	✓	✓	✓
Republish	✓	✓	✓	✓
TransferSubscriptions				✓
DeleteSubscriptions	✓	✓	✓	✓

2.2 Security Policies

The security of the exchanged OPC UA messages heavily relies on the used Security Policy. They are defined in art 7 of the specification [17d] together with their security level as follows.

1. None (insecure)
2. Basic128Rsa15 (deprecated)
3. Basic256 (deprecated)
4. Basic256Sha256 (secure-high)
5. Aes128_Sha256_RsaOaep (secure-average)
6. Aes256_Sha256_RsaPss (secure-high)

None uses no cryptography at all, and thus it is insecure. *Basic128Rsa15* and *Basic256* rely on the broken SHA1 algorithm, and therefore they are deprecated. The remaining security policies are either targeting an average (128 bit) or high (256 bit) security level. The supported Policies are shown in Tab. 2.

Tab. 2: Supported Security Policies

Security Policy	open62541	node-opcua	python-opcua	UA-.NETStandard
None	✓	✓	✓	✓
Basic128Rsa15	✓	✓	✓	✓
Basic256	✓	✓	✓	✓
Basic256Sha256	✓	✓	✓	✓
Aes128_Sha256_RsaOaep	✓			✓
Aes256_Sha256_RsaPss				✓

None and the two deprecated Policies are provided by all implementations. The only secure security policy supported by all projects is *Basic256Sha256*. The support for *Aes128_Sha256_RsaOaep* and *Aes256_Sha256_RsaPss* is rather low. The open-source projects are continuously implementing new features, which can be seen in *open62541*, which added *Aes128_Sha256_RsaOaep* during our research.

2.3 Transport Protocols

OPC UA allows for different transport protocols. For client-server communication, HTTPS, WebSockets and TCP are available [17a]. For pub-sub communication, one can use UADP, a binary protocol on top of UDP, AMQP or MQTT [18a]. Tab. 3 shows the support for the specified protocols.

Tab. 3: Supported Transport Protocols. Some are experimental[e].

Transport	open62541	node-opcua	python-opcua	UA-.NETStandard
UA TCP	✓	✓	✓	✓
HTTPS				✓
Websockets	✓[e]			✓
UADP	✓[e]			
MQTT	✓[e]			
AMQP				

All implementations support the binary protocol over TCP, which is also the most efficient. UA-.NETStandard offers additional support for HTTPS and WebSockets. Open62541 has experimental support for Websockets and pub-sub with either OPC UA over UDP (UADP) or MQTT.

2.4 Client GUIs

For most of the users, the implementation of a custom client or the usage of a command-line interface (CLI) is inconvenient, and thus a GUI is favourable. Among the four investigated open-source implementations, three provide a GUI. UA-.NETStandard provides several example applications featuring a GUI in their repository [OPd]. However, these applications rely on the .Net Framework and thus only run on Microsoft Windows operating system. Additionally, a mobile client targeting Android, iOS and Windows UWP is available. The GUI of python-opcua is hosted in a separate repository [Fra] and is platform-independent. However, this project is not active since December 2019. An ncurses-based GUI, i. e. a text-based user interface (TUI), is offered by node-opcua. Again, this project is hosted separately from node-opcua and in [Rob]. Finally, open62541 does not provide any GUI.

3 Interoperability

Heterogenous OPC-UA-enabled machines are often interconnected, and users expect them to work together. While open62541 and UA-.NETStandard are officially certified by the OPC Foundation [OPd] [OPa], node-opcua and python-opcua do not have certificates. A certificate indicates that the product obeys the specification and can work with other certified implementations. In this section, we show that the open-source OPC UA solutions interoperate despite their missing certification. To do this, we first perform a simple test only consisting of a connection and retrieval of a standard value, namely the server time. In a second test, we include the usage of cryptography and further Services.

3.1 Cross-wise connection test

Our first test is performed by connecting clients to servers and requesting the server time using the Read Service with SecurityPolicy *None*. To connect to a server, a client sends a *HEL message* and further requests the following services: *OpenSecureChannel*, *OpenSession* and *ActivateSession*. To tear down the connection, the *CloseSession* and *CloseSecureChannel* services are called. All implementations are working with each other, which can be seen in Tab. 4. The results show that the connection mechanism and the package structure are compatible among the implementations.

Tab. 4: Connect and retrieve server time

server\client	**open62541**	**node-opcua**	**python-opcua**	**UA-.NETStandard**
open62541	✓	✓	✓	✓
node-opcua	✓	✓	✓	✓
python-opcua	✓	✓	✓	✓
UA-.NETStandard	✓	✓	✓	✓

3.2 Cross-wise functionality test

We perform pairwise connection tests further call various services. To limit the number of tests, we consider the following selection of services: *Browse*, *Read*, *CreateSubscription*, *CreateMonitoredItems*, *Publish* and *AddNodes*. These are some of the most useful services. Futhermore, we can get good coverage of the available Service Sets using these services. In the Services part of the specification [17c], the *Browse* service call is described as a method to retrieve the references of a node. One can start browsing the root node and recursively retrieve the whole tree of nodes. With the *Read* service, one can retrieve attributes of a node, for example, the server time in our basic connection test. To monitor changes of a value in the server, a client first creates a *Subscription*, then adds one or multiple *MonitoredItems* to the *Subscription* and finally sends *Publish* request. This service is used when supervising a production process, where a physical value such as temperature or pressure, should be monitored. For adding a node to the server, the *AddNodes* request is used. That is important when one wants to have a dynamic address space, like when a client desires to store additional information on the server. Since *Basic256Sha256* is the only secure and widely adopted Security Policy, we included it together with the security policy *None* in the test.

Since UA-.NETStandard is the reference implementation, we assume, that it obeys the standard and thus do not include it in this test. Further, open62541 can be seen as a benchmark in this test, as it is already certified.

As can be seen in Tab. 5, all implementations are working with each other. However, as node-opcua does not implement the NodeManagement Service Set, adding nodes does not

work whenever a node-opcua client or server is involved. All other implementations support all tested Services.

Tab. 5: Cross-wise functionality test. node-opcua does not support *Nodemanagement*^{n}.

server\client	open62541	node-opcua	python-opcua
open62541	✓	✓^{n}	✓
node-opcua	✓^{n}	✓^{n}	✓^{n}
python-opcua	✓	✓^{n}	✓

For the secure connection with Policy *Basic256Sha256*, the node-opcua server features the strictest certificate checking. It checks whether the client URI matches one of the certificates or not and whether the *KeyUsage* in the certificate is set correctly or not. Python-opcua and open62541 do not care about the *KeyUsage*. All three implementations come with scripts and small documentation on how to create the correct certificates.

4 Conclusion

In this paper, we investigated the four most popular open-source OPC UA implementations from the aspects of available features and interoperability. All implementations have essential functionalities and features. However, none of them implements the complete specification. The two implementations with most coverage are UA-.NETStandard and open62541. While the former implements all Service Sets of the client-server model, it does not support pub-sub communication. Whereas the latter features pub-sub and most but not all Service Sets. The newer Security Policies *Aes128_Sha256_RsaOaep* and *Aes256_Sha256_RsaPss* are only supported by UA-.NETStandard and partially by open62541. Except for open62541, all projects feature a GUI or TUI. Furthermore, we showed that the implementations work together in simple test scenarios. Our interoperability tests indicate that they obey the specification at least to a decent level. Overall, we conclude that open-source implementations can be utilized instead of or together with commercial solutions. Their features and programming languages make them suitable for different requirements. We can conclude that the implementations conform to the base standards since they support the basic features of the specification and they can interconnect.

References

[17a] OPC UA Specification Part 1: Overview and Concepts, 1.04, OPC Foundation, 2017.

[17b] OPC UA Specification Part 3: Address Space Model, 1.04, OPC Foundation, 2017.

[17c] OPC UA Specification Part 4: Services, 1.04, OPC Foundation, 2017.

[17d] OPC UA Specification Part 7: Profiles, 1.04, OPC Foundation, 2017.

[18a] OPC UA Specification Part 14: PubSub, 1.04, OPC Foundation, 2018.

[18b] OPC UA Specification Part 2: Security Model, 1.04, OPC Foundation, 2018.

[Ce19] Cenedese, A.; Frodella, M.; Tramarin, F.; Vitturi, S.: Comparative assessment of different OPC UA open–source stacks for embedded systems. In: 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, pp. 1127–1134, 2019.

[Fra] Free OPC-UA Library: opcua-client-gui, https://github.com/FreeOpcUa/opcua-client-gui Accessed 28 June. 2020.

[Frb] Free OPC-UA Library: python-opcua, https://github.com/FreeOpcUa/python-opcua Accessed 08 May. 2020.

[Ha17] Haskamp, H.; Meyer, M.; Mollmann, R.; Orth, F.; Colombo, A. W.: Benchmarking of existing OPC UA implementations for Industrie 4.0-compliant digitalization solutions. In: 2017 IEEE 15th International Conference on Industrial Informatics (INDIN). Pp. 589–594, 2017.

[Mu20] Muehlbauer, N.; Kirdan, E.; Pahl, M.-O.; Carle, G.: Open-Source OPC UA Security and Scalability. In: 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). 2020.

[OPa] OPC Foundation: Certification of open62541 Server SDK, https://opcfoundation.org/products/view/open62541-server-sdk Accessed 28 June. 2020.

[OPb] OPC Foundation: OPC UA Case Studies, https://opcfoundation.org/resources/case-studies/ Accessed 28 June. 2020.

[OPc] OPC Foundation: OPC Unified Architecture, https://opcfoundation.org/about/opc-technologies/opc-ua/ Accessed 28 June. 2020.

[OPd] OPC Foundation: UA-.NETStandard, https://github.com/OPCFoundation/UA-.NETStandard Accessed 08 May. 2020.

[Pa15] Palm, F.; Grüner, S.; Pfrommer, J.; Graube, M.; Urbas, L.: Open source as enabler for OPC UA in industrial automation. In: 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA). Pp. 1–6, 2015.

[PP] Pfrommer, J.; Profanter, S.: open62541, https://github.com/open62541/open62541 Accessed 08 May. 2020.

[Pr19] Profanter, S.; Tekat, A.; Dorofeev, K.; Rickert, M.; Knoll, A.: OPC UA versus ROS, DDS, and MQTT: performance evaluation of industry 4.0 protocols. In: Proceedings of the IEEE International Conference on Industrial Technology (ICIT). 2019.

[RC18] Resnick, C.; Clayton, D. A.: OPC Technology Well-positioned for Further Growth in Tomorrow ' s Connected World. In. 2018.

[Roa] Rossignon, E.: node-opcua, https://github.com/node-opcua/node-opcua Accessed 08 May. 2020.

[Rob] Rossignon, E.: opcua-commander, https://github.com/node-opcua/opcua-commander Accessed 29 June. 2020.