Introduction of Group-Subscriptions for RESTful OPC UA clients in IIoT environments

Rainer Schiekofer[1,2] and Michael Weyrich[2]
[1]Siemens AG [2]University of Stuttgart

***Abstract*—In the automation domain, OPC UA can be considered one of the most important communication protocols for IIoT applications. According to a study of McKinsey the potential economic impact of the IoT within factories may reach up to 3,7 Trillion Dollar by 2025. To unlock this potential it is very important to bridge the interoperability gap. Since V1.04 OPC UA also supports session-less clients and therefore allows the development of RESTful clients which can be used to address the cross-domain interoperability issue. However, currently, RESTful clients cannot use Subscriptions, because the corresponding Service Sets are not supported for session-less clients. In this paper, we will present how Subscriptions can be introduced for session-less clients. Furthermore, we outline how Group-Subscriptions can be introduced for session-less as well as session-based clients. Finally, our evaluation shows that Group-Subscriptions significantly outperform standard Subscriptions if the Subscription can be shared by several clients.**

***Index Terms*—OPC UA, Group-Subscriptions, Subscriptions, REST, Information Model, SessionlessInvoke, IIoT, Industry 4.0**

I. Introduction

Based on some market studies the estimated impact of the Industrial Internet of Things (IIoT) on the global GDP will be 14,3 Trillion Dollar by 2030, according to Accenture [1]. A further study of McKinsey [2] showed that the potential economic impact of the IoT within factories may reach from 1,2 up to 3,7 Trillion Dollar by 2025. However, McKinsey also stated out that up to 40 percent on average (and up to 60 percent in some settings) of the potential economic value depends on the interoperability of IoT systems. Based on that, it can be inferred that interoperability is a key to utilize the economic benefits of IIoT scenarios. This is exactly what is addressed by OPC Unified Architecture (OPC UA) [3]. OPC UA is one of the most important IIoT standards in the automation domain and promised to enable interoperability on the transport layer as well as interoperability on the semantic layer. To finally reach this goal the OPC Foundation worked very hard in the last couple of years. Some very promising activities to solve the interoperability on the semantic layer problem are the introduction of domain-specific semantics into OPC UA [4] and the support of dictionaries like eCl@ss [5]. Another very interesting activity to dramatically increase the amount of OPC UA devices is the so-called Field Level Communication (FLC) group, which aims to unify the field buses [6].

Nevertheless, to also lift the economic benefits of cross-domain use cases, OPC UA must also be able to reach out to other domains. A technology which is already present in nearly every domain is the REST architecture [7], which could also be used as a bridge between OPC UA and other domains. However, till version 1.03 of OPC UA, it was not possible to combine REST with OPC UA, because of the missing statelessness. Finally, version 1.04 of OPC UA introduced statelessness in OPC UA through a new *Service* which is called *SessionlessInvoke*. After that, it was possible to develop a RESTful interface for OPC UA [8]. Nevertheless, not each *Service* of OPC UA can be addressed through *SessionlessInvoke*. To be more concrete, this *Service* can only be used in combination with some Services of the *View Service Set*, the *Attribute Service Set*, the *Method Service Set*, the *NodeManagement Service Set*, and the *Query Service Set* [3]. Eventually, some important *Service Sets* are still not supported, as for example, the *Subscription Service Set* and the *MonitoredItem Service Set*. These *Service Sets* are necessary to get efficiently notified about data changes and therefore also would be very useful for RESTful applications.

The authors of [9] already proofed that for some OPC UA *Services* stateless interaction can improve performance. However, *Subscriptions* where out-of-scope in this work. Another interesting concept for introducing *Subscriptions* to RESTful clients can be found in HyperUA [10]. Nevertheless, in this case, the *Subscriptions* are exposed through a proprietary REST interface, which makes it impossible for standard OPC UA session-based as well as session-less clients to access the *Subscriptions*. Finally, since Version 1.04 OPC UA also supports the publish-subscribe interface [3], which also can be considered as a form of Group-Subscriptions. Nevertheless, a lot of stacks still do not support this new feature and there are also some differences between OPC UA publish-subscribe and standard OPC UA client-server *Subscriptions*, like the triggering mechanism.

In this paper, we will introduce a concept of how the *Subscription Service Set* and the *MonitoredItem Service Set* can be accessed by session-less clients and therefore this work can be considered as extension of [8]. Furthermore, we will extend the standard *Subscription* mechanism of OPC UA to also support Group-Subscriptions. Finally, our evaluation shows that Group-Subscriptions significantly outperform standard *Subscriptions* if the *Subscription* can be shared by several clients in parallel.

II. Background

In the following, we will introduce the necessary background to understand the further parts of this work, starting

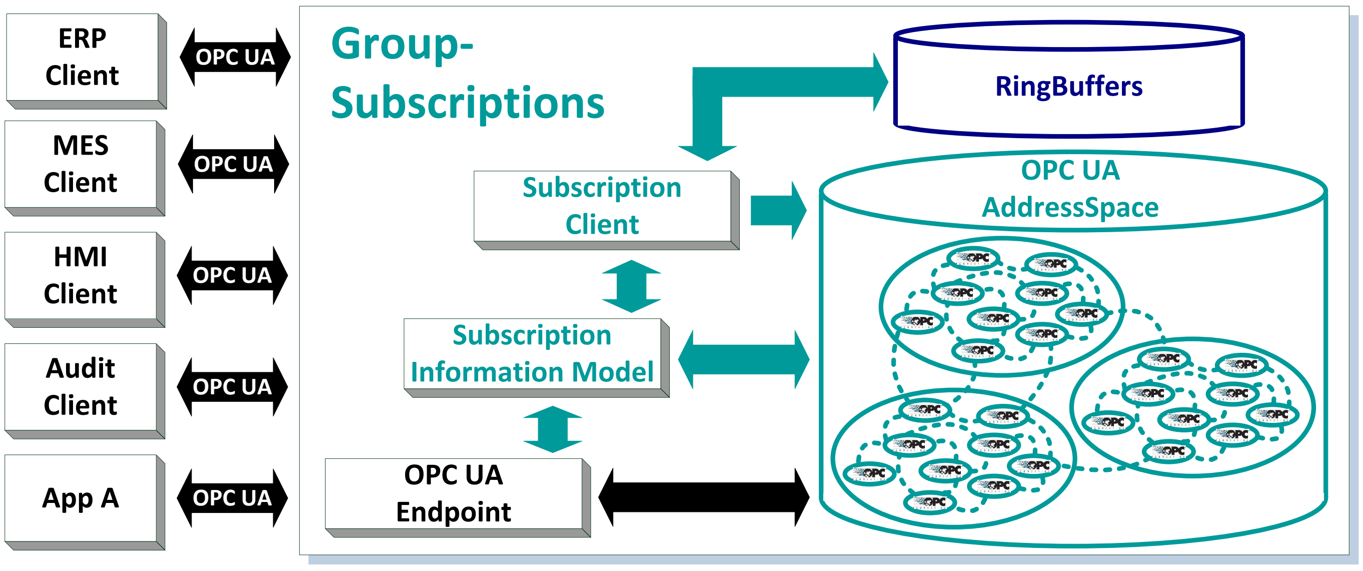


Fig. 1. Group-Subscriptions architecture.

with the basics of OPC UA (Section II-A), followed by a short explanation of OPC UA *Subscriptions* (Section II-B).

A. OPC UA

OPC UA [3] is one of the most important industrial communication standards for IIoT scenarios. OPC UA not only aims to solve the interoperability on the transport layer issue but also, the interoperability of the semantic layer shall be addressed by OPC UA.

Interoperability on the transport layer can be achieved through the standardization of transport layer protocols like OPC TCP and HTTP(S). In addition, it is necessary to also standardize different serialization formats like OPC JSON, OPC Binary, or OPC XML. In V1.04 OPC UA also introduces a cloud-ready interface based on the publish-subscribe pattern and transport protocols like MQTT and AMQP [11]. However, this is typically not enough to reach interoperability on the transport layer. The final step involves the standardization of interaction patterns with the service. In OPC UA these patterns are called *Services* and can be used to access the graph-based data model of OPC UA. OPC UA defines several *Services* to introspect (e.g., the *Read Service*) and manipulate (e.g., the *Write Service*) the graph-based information model.

Interoperability on the semantic layer can be achieved through standardizing the semantics of the graph-based data model of OPC UA. This is typically done with so-called *Companion Specifications*. In previous years most of the *Companion Specifications* were mappings from other already existing standards to OPC UA like AutomationML, PLCOpen, ISA-95, etc. [12]–[14]. Eventually, all these standards are rather abstract and therefore define mainly generic semantics. However, in the last few years also *Companion Specifications* started to emerge with detailed descriptions of the underlying devices (e.g., machine vision, robotics, powertrain, CNC machines, etc. [15]). These standards are mainly driven by the VMDA [16]. The VDMA can be considered the largest industry association in Europe and represents more than 3200 companies within the manufacturing domain.

B. OPC UA Subscriptions

Probably the most important *Service Set* in OPC UA is the *Subscription Service Set* and the *MonitoredItem Service Set*. In combination with several *Services* of these *Service Sets* it is possible to subscribe to, for example, data change notifications. This can be done by first using the *CreateSubscription Service* to create a *Subscription*. After that, the *CreateMonitoredItem Service* can be used to select what kind of *Nodes* should be monitored. Finally, the *Publish Service* is used to acknowledge received sequence numbers and to inform the server that new responses can be accepted by the client. These *Subscriptions* are highly optimized and therefore are much more efficient, besides other benefits, then periodically fetching the latest values with the *Read Service*. However, in OPC UA each client has to create its own *Subscription*, even if all items and configurations are equal across different clients. Eventually, this leads to a higher resource consumption than necessary.

III. Approach

Below, we will give an architectural overview of Group-Subscriptions. Subsequently, we will explain parts of the corresponding OPC UA information model in greater detail.

Figure 1 gives an overview of the architecture. On the left side, several clients are depicted, which want to be informed about updates in the OPC UA server. The right part of Figure 1 depicts our application. We identified two goals for our architecture: (1) Introduction of Group-Subscriptions in such a way that most parts of the existing OPC UA SDKs do not have to be altered. (2) Group-Subscriptions should be usable with session-less clients as well as with session-based clients. Eventually, we decided to model Group-Subscriptions within the OPC UA information model. The OPC UA information model itself consists of methods, which can be used to recreate the *Services* of the *MonitoringItem Service Set* and

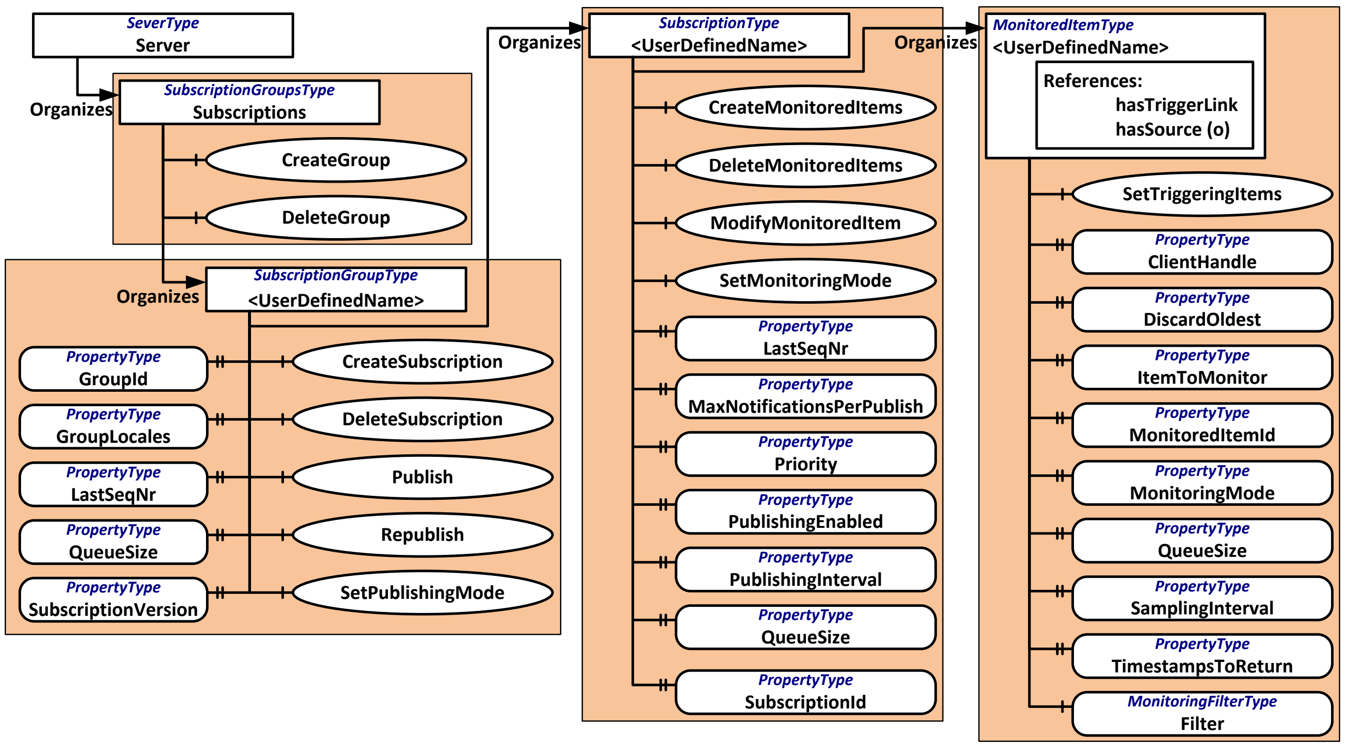


Fig. 2. Group-Subscriptions information model.

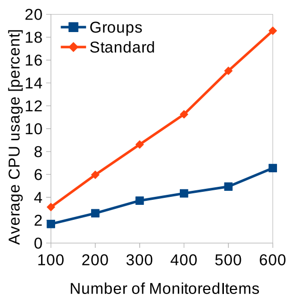
the *Subscription Service Set*, and further *Nodes*, which contain more information about the Group-Subscription configuration. Based on this information model an internal client can be controlled, which creates session-based *Subscriptions* through standard SDK calls. Nevertheless, normally *Subscriptions* and some of the corresponding *Services* are designed for a single client only. For example, the *Publish-Service* allows a client to acknowledge received sequence numbers. If a sequence number is acknowledged the server is allowed to delete the message from the internal buffers. Of course, if more than one client is using the same *Subscription* such behavior can lead to data-loss for some of the clients. Because of that, we introduced ring buffers. The size of the ring buffers can be chosen during the creation of the Group-Subscription.

Figure 2 shows parts of the developed information model. A lot of the elements should be familiar to OPC UA experts. However, some of them are altered or newly introduced. For example, our Publish-Method contains two input parameters: **publishSeqNr** and **keepAliveTime**. The first parameter is used to request a certain sequence number. This sequence number is automatically generated by the subscription client (see Figure 1) and incremented for each *Publish* response with new values. The *keepAliveTime* is used for long polling and defines the maximum time the call should be blocked. The second parameter is only important if the client requests a sequence number which is larger than the latest available sequence number. In this case, the method call will be blocked for the given time amount. If during this time the requested sequence number becomes available the call will immediately return with the new results, otherwise, the client will be informed that no results are available at the moment. A client can use this architecture to also request more than one future sequence number, which is equivalent to calling the standard *Publish-Service* several times. Of course, not all sequence numbers can be requested because of limited memory resources. The number of available sequence numbers can be configured through the queue size. However, clients which are joining late typically have no information about the actual sequence number. Because of that, we also introduced a *Property* with information about the latest sequence number (**LastSeqNr** in Figure 2). The initial values can be fetched with the standard *Read-Service* of OPC UA. The **SubscriptionVersion** *Property* of Figure 2 indicates if the Subscription-Group is changed. The actual value of the **SubscriptionVersion** *Property* is also returned in the Publish and Republish methods and can be used by the client to check if the Group-Subscription was altered.

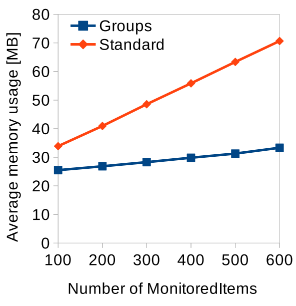
Finally, we implemented the concept above with the C++ SDK from Unified Automation (version 1.5.6) [17] to show the validity and the benefits of this architecture.

IV. Evaluation

In this Section, we compare the memory consumption and CPU usage of Group-Subscriptions to the respective characteristics of standard OPC UA *Subscriptions*. In all cases, our prototype runs on a machine with 4 logical cores, 2.6 GHz, and 8 GB RAM. Additionally, the sampling interval for each *MonitoredItem* is set to 500 ms, the publishing interval is set to 1000 ms and all queue sizes are set to twenty. Each experiment runs for two minutes and data points represent the average of three runs. If Group-Subscriptions are used, each client makes use of the same *Subscription*, which can be accessed through

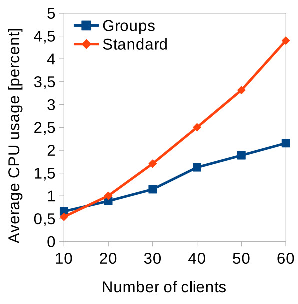


(a) CPU usage

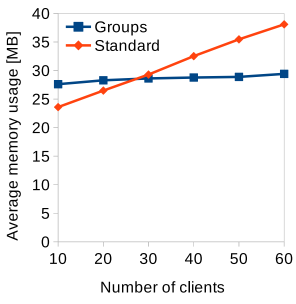


(b) Memory usage

Fig. 3. Constant number of 100 clients



(a) CPU usage



(b) Memory usage

Fig. 4. Constant number of 300 MonitoredItems

the information model of the OPC UA server. In the case of standard OPC UA *Subscriptions*, each client creates its own *Subscription* and the corresponding *MonitoredItems*.

A. Number of MonitoredItems

In our first experiment, we evaluate the average CPU load and memory consumption for different numbers of *MonitoredItems* per *Subscription*. In each experiment 100 clients connect to the OPC UA server and establish the corresponding *Subscription*. Figure 3 represents the results for the CPU load and memory consumption, respectively. As expected, the benefits of Group-Subscriptions continue to grow with the number of *MonitoredItems*. This is true for the CPU load as well as for memory usage.

B. Number of clients

In our second experiment, we evaluate the average CPU load and memory consumption for different numbers of *Clients* per *Subscription* with a constant number of 300 *MonitoredItems*. Also, for these experiments, 100 clients connect to the OPC UA server in each test run to minimize the measurement distortion through a different number of client *Sessions*. However, in this experiment, only 10 to 60 clients actively used *Subscriptions*, while the other clients were just on idle during the experiments. Figure 4 represents the results for the CPU load and memory consumption, respectively. For the average CPU usage as well as for the average memory consumption we can see that the benefits of Group-Subscriptions increase with the number of clients. However, as also depicted in Figure 4b for very few clients the standard *Subscription* mechanism offers less memory consumption than the Group-Subscription prototype. This is mainly due to the fact, that in our experiments for the standard *Subscription* mechanism the clients never were late and acknowledged instantly all sequence numbers leading to at most one publish item in the buffers. In contrast, in the Group-Subscription case, the buffer cannot be emptied, because the number of clients is unknown and therefore, constantly twenty publish requests are kept in the queue. This also leads to the fact that for Group-Subscriptions the memory-consumption is nearly constant regardless of the number of clients (see Figure 4b). Nevertheless, the CPU usage still increases with the number of clients. The explanation for this effect is based on the *Call Service Set*, which generates a higher CPU load if more calls are made and therefore is responsible for the slight increase in CPU usage.

V. Summary and Outlook

In this paper, we presented a concept of how Group-Subscriptions can be introduced into OPC UA through the information model. This architecture allows session-less clients as well as session-based clients to share the same *Subscription*, which leads to a significant reduction of the resource usage (CPU load and memory consumption) if the *Subscription* can be shared by several clients in parallel.

Nevertheless, further research is necessary to also compare this architecture with the new publish-subscribe feature of OPC UA in terms of resource consumption, latency and lines of code as well as functionality (e.g., triggering model).

References

[1] "Industrial iot and the (data) sharing economy," http://iiot-world.com/connected-industry/industrial-iot-and-the-data-sharing-economy/, 2019.

[2] "Unlocking the potential of the internet of things," https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-the-value-of-digitizing-the-physical-world, 2019.

[3] "Iec 62541: Opc unified architecture," Standard, 2015.

[4] "Vdma - opc ua working groups," https://opcua.vdma.org/en/, 2018.

[5] "Opc ua - roadmap," https://opcfoundation.org/about/opc-technologies/opc-ua/opcua-roadmap/, 2018.

[6] "Opc ua - field level communiation," https://opcfoundation.org/wp-content/uploads/2018/11/OPCF-FLC-v2.pdf, 2018.

[7] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, 2000.

[8] R. Schiekofer, A. Scholz, and M. Weyrich, "Rest based opc ua for the iiot," in *IEEE Emerging Technologies and Factory Automation*, 2018.

[9] S. Gruener, J. Pfrommer, and F. Palm, "Restful industrial communication with opc ua," *IEEE Transactions on Industrial Informatics*, vol. 12, no. 5, 2016.

[10] "Projexsys: Hyperua," http://projexsys.com/hyperua/, 2018.

[11] "Opc ua - pubsub," https://opcfoundation.org/news/press-releases/opc-foundation-announces-opc-ua-pubsub-release-important-extension-opc-ua-communication-platform/, 2018.

[12] "Automationml opc ua information model - companion specification release 1.00," Standard, 2016.

[13] "Plcopen opc ua information model - iec 61131-3 - companion specification release 1.00," Standard, 2010.

[14] "Isa-95 common object model - companion specification release 1.00," Standard, 2013.

[15] "Opc ua companion specifications," https://opcfoundation.org/markets-collaboration/, 2018.

[16] "Vdma - members," http://www.vdma.org/en/mitglieder, 2018.

[17] "Unified automation - c++ based opc ua client and server sdk (bundle)," https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html, 2019.