A RESTful Extension of OPC UA

Sten Grüner
Chair of Process Control Engineering
RWTH Aaachen University
Aachen, Germany
Email: s.gruener@plt.rwth-aachen.de

Julius Pfrommer
Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB)
Karlsruhe, Germany
Email: julius.pfrommer@iosb.fraunhofer.de

Florian Palm
Chair of Process Control Engineering
RWTH Aaachen University
Aachen, Germany
Email: f.palm@plt.rwth-aachen.de

***Abstract*—RESTful interfaces are a wide-spread architecture style for webservice implementations and are built upon the resource-oriented approach to decentralized architectures (ROA). REST postulates a set of requirements that are not covered by the OPC Unified Architecture (OPC UA) communication protocol per se. We propose a set of simple extensions to the OPC UA binary protocol that enable RESTful communication. The evaluation shows an order of magnitude improvement in the use of communication resources for sporadic service requests. Additionally, RESTful OPC UA allows applications to profit from the advantages of the resource-oriented architecture style, such as caching and loose application coupling.**

I. Introduction

Currently, the traditional manufacturing industries and the information and communications technology industry are coming together in the so-called Internet of Things (IoT) [1]. Both industries bring their favoured protocols. On the one hand, this includes protocols with a web-centric perspective, such as application-level protocols with HTTP transport, Message Queue Telemetry Transport (MQTT) [2] and Constrained Application Protocol (CoAP) [3]. On the other hand, this includes protocols stemming from industrial networks. These are either fieldbuses or ethernet-based application-level protocols, such as OLE for Process Control (OPC Classic) [4] and the more recent OPC Unified Architecture (OPC UA) [5].

The difference between the protocols from either industry lies in the nature of the communication they need to support. On the Internet, stateless communication has played a more crucial role, as communication was mostly short-lived and less predictable than in confined industrial settings. However, this technological gap at the heart of IoT can be overcome. We show that, with a simple extension of the standard, OPC UA can support the popular RESTful webservice-style that underlies many popular web-technologies. To do so, we

1) extend OPC UA to enable stateless service requests,
2) introduce a caching control mechanism by an additional expiration header in messages, and
3) study the optional transportation of OPC UA messages via the User Datagram Protocol (UDP).

The paper is structured as follows. First, some background on OPC UA and REST is given in Section II. Section III discusses motivating use cases for RESTful OPC UA. A set of simple extensions to OPC UA that enable RESTful communication is proposed in Section IV and evaluated in Section V. Finally, Section VI provides a summary of the paper.

The work of Julius Pfrommer is partially supported by the BMBF (German Ministry of Education, Science, Research and Technology) under grant number 01IS12053.

II. Background

A. OPC Unified Architecture

OPC UA is a protocol for industrial communication and has been standardized in IEC 62541. At its core, OPC UA defines a set of services (see Figure 1) to interact with a server-side object-oriented information model. The information model is a web of nodes (see Figure 2 for possible node types), each identified by a unique identifier and interconnected by directed reference-triples of the form (`nodeid, referencetype, target-nodeid`). Reference types can be standard or user-defined and either non-hierarchical (e.g., indicating the type of a variable-node) or hierarchical (e.g., defining a parent-child relationship). Besides the service-calls initiated by the client, push-notification of remote events (such as data changes) can be negotiated with the server. The client/server interaction is mapped either to a binary encoding and TCP-based transmission or to SOAP-based webservices. A unique characteristic of OPC UA is that the information model can be fully introspected and—in theory—modified at runtime via the standardized services. Common and reusable (for some domain of interest) information models are released as companion standards that enable vendor interoperability. Most of these translate the semantics of existing standards into an OPC UA information model and are developed jointly with the standardisation bodies and industry working groups of the specific domain. The OPC Foundation[1] drives the continuous improvement of OPC UA, the development of companion standards, and the adoption of the technology in the industry by hosting events and providing the infrastructure for compliance certification.

B. Representational State Transfer (REST)

The REST architecture style for resource-oriented distributed applications [6], [7] proposes the use of simple, uniform interfaces to interact with resources by the transfer of their representations. For generality, assume that the representations may refer to virtual as well as to physical objects. Currently, RESTful HTTP is a popular way to implement webservices

[1] https://opcfoundation.org

978-1-4799-8244-8/15/$31.00 ©2015 IEEE

where resources can be accessed under some URL. The REST principles are however independent of specific protocols or technologies.

Richardson and Ruby [7] postulate four properties of RESTful webservices as part of a resource-oriented architecture:

1) Addressability: Resources are assigned to an address that acts as their identifier and that contains enough information to establish a communication channel.

2) Statelessness: The interaction with resources does not depend on some state of the communication channel. The requests and responses are self-contained and do not rely on the server storing information besides that persisted in a resource. This enables the introduction of caching layers that "replay" responses to identical requests when the resource state has not changed. Furthermore, statelessness simplifies asynchronous communication where multiple requests are sent in parallel.

3) Connectedness: Connectedness describes how resources reference each other in their representations via their identifying addresses.

4) Uniform Interface: The same set of interface mechanisms is used to interact with all resources. Differences between resources are manifest in their representations, but not in the interface. For example, the available operations in HTTP (GET, POST, PUT, DELETE, and so on) are the same for every website and for every type of media that is transferred. For pure REST, there can be no out-of-band information transfer [6]. On the web, this is achieved by browsers retrieving hypertext and JavaScript code (both well-defined representation formats), that can be interpreted to access additional resources with unknown representations without updating the browser itself. In different settings, this constraint is however routinely violated by manually integrating custom APIs on top of a RESTful interface for data transport. The out-of-band interface information then has to be updated with every interface change.

C. How RESTful is OPC UA already?

1) Addressability: Addressability in OPC UA is given. Nodes can be found either via their unique ID or, since every node must have a hierarchical reference to a parent up to the root node, by walking a *BrowsePath* on the hierarchical references. Paths can be encoded in simple human-readable form, such as `/site1/plant1/line1/mill42`.

2) Stateless: In general, OPC UA cannot be considered stateless. First, the protocol defines compulsory *SecureChannels* and *Sessions* that expire without regular renewal. Second, many of the OPC UA services are inherently stateful. For example, the *Browse* service can be requested with a maximum number of returned items. The server than returns a *ContinuationPoint* token, used to request the remaining items. Services that are inherently stateless are marked with a star in Figure 1. Third, push-notification by the server (*MonitoredItems* and *Subscriptions*) obviously cannot be implemented without storing state.

3) Connectedness: It is a core concept of OPC UA to have nodes referring to each other with their unique identifier. Node identifiers may also point to a remote server and are then called *ExpandedNodeId*.

Discovery Service Set
- FindServers⋆
- GetEndpoints⋆
- RegisterServer⋆

SecureChannel Service Set
- OpenSecureChannel
- CloseSecureChannel

Session Service Set
- CreateSession
- ActivateSession
- CloseSession
- Cancel

NodeManagement Service Set
- AddNodes⋆
- AddReferences⋆
- DeleteNodes⋆
- DeleteReferences⋆

View Service Set
- Browse⋆
- BrowseNext
- TranslateBrowsePathsToNodeIds⋆
- 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

Fig. 1: Services defined in the OPC UA standard. Services that are inherently stateless are marked with a star.

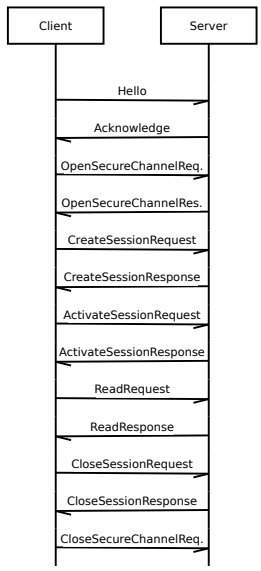
- ReferenceTypeNode
- DataTypeNode
- VariableTypeNode
- VariableNode
- MethodNode
- ObjectTypeNode
- ObjectNode
- ViewNode

Fig. 2: Node types used in OPC UA information models.

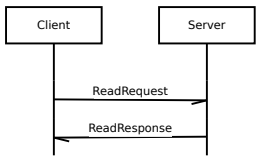
4) Uniform Interface: The services defined in OPC UA are fixed and cannot be changed by the user. In that sense, there is a uniform interface that is common to all applications. But its capabilities go well beyond the standardized access to resource representations. For example, the *Call* service can be used for remote procedure calls with arbitrary arguments and return values that are defined in the corresponding *MethodNode*. But knowing the prototype of a function is no substitute for knowing its semantics and side-effects (also on the physical world). So the possibility to fully introspect an OPC UA information model does not by itself remove the need for out-of-band interface information transfer.

III. Motivation for RESTful OPC UA

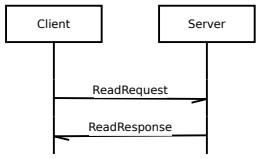
A typical message exchange for a binary OPC UA service call (a *Read* service request) is depicted in Figure 3a. The initial connection is established with a HEL-ACK handshake to exchange basic connection information (these are OPC UA messages, not to be confounded with the TCP handshake). Consequently, a pair of messages is exchanged to open a secure communication channel (that may have encryption and



(a) OPC UA over TCP (8 TCP overhead packets omitted).



(b) Stateless OPC UA over TCP (8 TCP overhead packets omitted).



(c) Stateless OPC UA over UDP (no packets omitted).

Fig. 3: Message exchange diagrams for establishing connections and a single *Read* request.

message signing disabled). The next pair of messages creates a session that holds the client's state on the server. As we can see from Figure 3a, eight messages were exchanged before the *Read* service is invoked. Finally, the session and the secure channel are closed, which also aborts the TCP connection. The resulting communication overhead and the need to store communication channel state information in the server has implications for some relevant industrial and IoT scenarios.

Low-Energy Clients: Low-energy clients, e.g., distributed IoT devices running on batteries, must reduce the communication overhead to increase their lifetime. An example for those are low-energy Bluetooth devices for which OPC UA interfaces have already been studied [8]. For maximum energy efficiency, it is even possible to have clients wake up, send a status message to the server and go back to sleep without waiting for a response.

Constrained or Non-Scalable Servers: Theses scenarios aim for situations where the server could normally not cope with the high load of many clients/number of requests. With stateless connections, the server only has to keep a socket connection without any additional per-client information. Modern high-bay storages contain tens of thousands of pallets. The periodic status transmission of the pallet status (e.g., its temperature and position) can be handled with existing server applications. But peak loads for alert notifications, for example 30,000 pallets trying to send status information when the temperature drops under a minimum level, could easily bring conventional systems to their limits. Here, the limiting quantity is not only the required network throughput, but also the ability of the server and the underlying operating system to manage concurrent stateful access under the given time constraints.

IV. Realizing RESTful OPC UA

To reduce the described communication overhead, we propose a minimally invasive enhancement of the OPC UA binary protocol that enables stateless REST-style communication.

A. SecureChannel Zero and Session Zero

First, we omit the need for elaborate handshakes by defining fallbacks with sensible configurations that are used if no alternative has been negotiated by the client. If the client does not open a *SecureChannel* and *Session*, it can still send messages with reference to the SecureChannel with identifier 0 (having the security profile `http://opcfoundation.org/UA/SecurityPolicy#None`) and the anonymous session with identifier 0. Session Zero allows clients to request stateless UA services (see Figure 1). These services are invoked with a self-contained request and can be answered with a single, self-contained response. Additional access control policies for Session Zero are left as an implementation choice. Note that, since the identifier 0 is never used by negotiated SecureChannels and Sessions, our approach is fully backward compatible with existing clients.

A message exchange for a stateless *Read* service invocation is shown in Figure 3b. We observe that the amount of exchanged messages goes down from 13 to 2 (or from 21 to 10 if we count all TCP packets) which is equivalent to an improvement of more than 50% in terms of the number of exchanged messages.

B. Expiration Tags for Cached Responses

Every service response in OPC UA already includes a time stamp – the time of message creation. This data is contained in the so-called response header and is comparable with HTTP's *Date* header field [9]. The simplest cache control strategy is to add an additional time stamp to the header to define the data lifetime. This time stamp then denotes the expiration of the data in analogy to the *Expires* HTTP header field. It can be stored e.g. in the currently unused *AdditionalHeader* field of the response header (reserved for further use by the standard [5] and ignored by current clients) and can be expanded to an array of time stamps when the response makes references to multiple nodes. But clearly, the addition of an expiration time stamp is only a first step towards a scalable caching architecture that is available for most internet-scale technologies and protocols.

C. Optional UDP Transport

The OPC UA specification clearly requires TCP transmission for its binary protocol. However, OPC UA implementations can be easily adapted to use UDP sockets for the stateless requests introduced before.

Unreliable UDP transport makes chunking (splitting long messages into several smaller packets) impracticable, since chunks may not arrive or arrive in the wrong order. The messages must therefore fit into a single packet. A typical *Read* request for a single node and attribute is about 70 bytes long. The actual packet size depends on the number of requested nodes and attributes and its data types.

TABLE I: Duration for establishing a connection and a single binary protocol invocation of the *Read* service to retrieve an integer scalar (the average of 1,000 measurements).

Server device	Standard compliant (TCP)	Stateless (TCP)	Stateless (UDP)
WAGO-750-860	26.60 ms	11.35 ms	2.91 ms
	100%	42.67%	10.94%
Raspberry Pi	7.77 ms	2.15 ms	0.86 ms
	100%	27.62%	11.06%
x86 PC	5.02 ms	1.43 ms	0.62 ms
	100%	28.49%	12.35%

UDP guarantees to support packet payloads of at least 576 bytes [10]. Support for larger payloads depends on the actual infrastructure. In case a packet is unavoidably big, it is possible to switch to the TCP communication.

One upside of using UDP is the decrease of required roundtrips between the server and the client (cf. Section IV and Figure 3c). UDP communication can be implemented with less effort in both, the client and the server. UDP avoids head-of-line blocking and makes the processing of messages more efficient. Another advantage is a more deterministic transport delay, since no reordering or retransmission is performed.

V. Evaluation

In this section we evaluate the performance improvements due to the proposed extensions of the OPC UA standard. The changes were implemented in an open-source OPC UA communication stack called open62541[2] with a patch-set that changes about 50 lines of code in total.

The RESTful extensions were benchmarked on the following three systems. The first system is a WAGO-750-860 fieldbus controller. The controller has an 44MHz ARM7TDMI CPU and 16 MB of RAM. The system was chosen as a representative of a low cost and constrained device for industrial applications. The second evaluated system is a Raspberry Pi Model B with an 700 MHz ARM1176 CPU and 512 MB of RAM. The third system is a typical consumer PC with an 2.5GHz Intel Core i5-2520M CPU and 8 GB of RAM. The clients were also implemented with the open62541 stack and ran on a dedicated desktop PC. The client PC and the three devices running OPC UA servers are interconnected via switched Fast Ethernet. Security features, such as authentication, message encryption and signing were not used for the measurements.

The measured scenario is the duration of a complete message exchange, i.e. establishing connection, sending a single *Read* request, retrieving an integer scalar value in the service response, and closing the connection. The entire sequence is depicted in Figure 3. The duration of a standard compliant message exchange (Figure 3a) is compared to the proposed protocol modifications (Figure 3b and Figure 3c, respectively). The result of the performed measurements is shown in Table I.

[2]http://www.open62541.org

We observe a time reduction of 60-70% when comparing stateful and stateless requests over TCP and a speedup of about an order of magnitude when comparing stateful TCP and stateless UDP requests. Note that the presented numbers were obtained under almost perfect connectivity conditions with a roundtrip time of under 0.5 ms. Due to the large number of exchanged packets for stateful requests, the speedup is expected to rise with increasing network latency.

VI. Summary

In this paper we presented a possibility of using the RESTful architecture style with the OPC UA protocol. In addition, a non-standard UDP transport layer for OPC UA was evaluated with regards to the reduction of communication overheads for sporadic interactions.

The proposed extensions were implemented in an open source OPC UA communication stack called open62541 and evaluated on two embedded systems including an industrial fieldbus controller. The evaluation shows that stateless UDP requests result in an up to an order of magnitude improvement in terms of service invocation time. Clearly, these improvements come at the cost of restricting use cases of OPC UA usage, e.g., decreasing the number of usable services and maximal message length. Still, we have outlined scenarios where RESTful extensions are preferable over the standard compliant protocol.

The presented extensions are fully reverse compatible with the standard and support mixed-mode operation i.e., standard conformal clients can access REST-enabled servers without any changes. RESTful extensions can also be easily included into existing OPC UA servers and clients by means of the addition of a couple of dozen lines of code.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] D. Locke, "MQ telemetry transport (MQTT) v3.1 protocol specification. IBM developerWorks Technical Library," 2010.

[3] C. Bormann, A. P. Castellani, and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, 2012.

[4] Y. Shimanuki, "OLE for process control (OPC) for new industrial automation systems," in *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, vol. 6. IEEE, 1999, pp. 1048–1050.

[5] *IEC 62541. OPC Unified Architecture Part 1-10, Release 1.0*, 2010.

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

[7] L. Richardson and S. Ruby, *RESTful web services*. O'Reilly, 2007.

[8] G. M. Shrestha, J. Imtiaz, and J. Jasperneite, "An optimized OPC UA transport profile to bringing Bluetooth Low Energy Device into IP networks," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*. IEEE, 2013, pp. 1–5.

[9] R. Fielding and J. Reschke, *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, Internet Engineering Task Force, Jun. 2014. [Online]. Available: http://tools.ietf.org/html/rfc7231

[10] R. Braden, *RFC 1122 Requirements for Internet Hosts - Communication Layers*, Internet Engineering Task Force, Oct. 1989. [Online]. Available: http://tools.ietf.org/html/rfc1122