Concurrent OPC UA information model access, enabling real-time OPC UA PubSub

1st Patrick Denzler
Institute of Computer Engineering
TU Wien
Vienna, Austria
patrick.denzler@tuwien.ac.at

2rd Mohammad Ashjaei
Innovation Design and Engineering School
Mälardalen University
Västerås, Sweden
mohammad.ashjaei@mdh.se

3nd Thomas Frühwirth
Research Department
Austrian Center for Digital Production
Vienna, Austria
thomas.fruehwirth@acdp.at

4th Victor Nicholas Ebirim
Innovation Design and Engineering School
Mälardalen University
Västerås, Sweden
vem20001@student.mdu.se

5th Wolfgang Kastner
Institute of Computer Engineering
TU Wien
Vienna, Austria
wolfgang.kastner@tuwien.ac.at

***Abstract*—Ongoing changes in industrial automation aim towards a flat and highly interconnected architecture that includes end-to-end real-time enabled machine-to-machine communications. Several technologies, such as OPC Unified Architecture (OPC UA) publish-subscribe and time-sensitive networking (TSN), facilitate that change. While OPC UA PubSub and TSN can already provide real-time capabilities, the parallel operation with standard OPC UA client-server remains an open challenge. This article presents preliminary results for solving concurrent information model access between OPC UA PubSub and client-server. The results include the overall RT-TSN-OPC UA concept, an analysis of common concurrent data access mechanisms for their suitability, and identifying critical code segments in the open62541 OPC UA stack. The paper concludes by outlining further research focusing on implementing and evaluating a *wait- and obstruction-free* mechanism into open62541.**

***Index Terms*—OPC Unified Architecture, TSN, real-time PubSub**

I. Introduction

Current developments in industry aim to transform industrial automation systems from a traditional hierarchical to a flat and highly interconnected architecture [1]. Such an architecture is IP-based to enable seamless communication and simplified integration of all types of devices from the shop floor up to the cloud. Essential for such an architecture is the removal of the historically grown gap between operational technology (OT) and information technology (IT), which hinders unrestricted data and information exchange [2].

Closing the OT/IT gap requires bridging two quite different environments. The field and control layer (OT), representing sensors, actuators, and programmable logic controllers (PLCs) connected by industrial communication systems, perform control loops and sometimes timely communication and processing to meet fixed deadlines [3]. On the other side, the IT layers stand for commercial off-the-shelf components, such as servers and desktop PCs running software applications, that communicate via Internet Protocol (IP) and focus on high average performance without strict timing requirements.

There are several technologies that provide functionalities necessary closing the OT/IT gap [2], for example time-sensitive networking (TSN), fog computing, and OPC Unified Architecture (OPC UA). The OPC UA protocol provides standardised, platform-independent, and secure machine-to-machine communication. TSN-enabled networks aim at providing robust deterministic data transport. The combination of both technologies, especially the OPC UA extension for publish-subscribe (PubSub), shows potential to realise an end-to-end real-time machine-to-machine communication [4].

While TSN received much attention regarding its timing behaviour, OPC UA PubSub was only recently investigated for possibilities of fulfilling strict timing guarantees [5], [6]. However, an open issue is that the timing guarantees are only valid if OPC UA PubSub is isolated from the standard OPC UA client-server communication pattern. The reason is that both communication paradigms access a shared information model, which raises issues regarding concurrent data access that must be addressed. A client-server instance might block a PubSub data access or change a value during a reading operation with unforeseeable consequences.

Within this context, this paper presents ongoing efforts to implement mechanisms that allow concurrent information model access in OPC UA and enable a distributed real-time end-to-end data transfer environment. The paper presents preliminary findings that include the presentation of the overall real-time TSN OPC UA concept and its requirements. Moreover, potential mechanisms for concurrent data access are evaluated for suitability to be implemented in the open-source OPC UA stack open62541 [7] information model, based on an analysis of the stack. The findings provide the base for future adjustment of the stack and its evaluation.

This work has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the *Austrian Competence Center for Digital Production* (CDP) under contract number 881843.

This paper contains five sections. The following Section II introduces OPC UA, TSN, and related work. Furthermore, it presents the underlying overall real-time TSN OPC UA concept and the involved challenges. Section III contains the findings regarding possible concurrent data access mechanisms to realise real-time OPC UA PubSub. Section IV presents the intended approach and preliminary results. Section V concludes the article and points out the future activities.

II. Background and Related Work

The aim of this section is to give a short and conceive description of TSN and OPC UA, as a foundation for Section II-D. The related work focuses on OPC UA PubSub in the context of real-time applications.

A. OPC Unified Architecture

The history of OPC UA starts with its predecessor the open platform communications (OPC) protocol and has evolved towards one of the possible contestant to homogenise communication in the industrial domain. Simplified, the architecture of OPC UA consists of two pillars [8]. In the first pillar, a meta model defines all rules for modelling information. The second pillar on the other hand, defines Transport Mechanisms for encoding data and data exchange between devices. Historically, OPC UA's main communication pattern is client-server, however, a recent extension also supports the publish-subscribe (OPC UA PubSub) communication pattern. The advantage of client-server is the possibility to invoke complex services like browsing the information model or calling methods. OPC UA PubSub focuses on minimal communication overhead to enable lightweight data exchange for example run-time data. It can be realised in a broker-based (e.g., MQTT) or a broker-less architecture (e.g., Ethernet multicast or TSN).

There is a wide range of OPC UA software stacks available, which are either driven by vendors or the open-source community. A widely used and well-maintained stack is the open-source stack open62541 [7]. The stack follows the official guidelines and supports a wide range of functionalities, including OPC UA PubSub. The programming language of the stack is C, what makes it a good reference for other implementations.

B. Time-Sensitive Networking (TSN)

TSN evolved from the Audio Video Bridging (AVB) for industrial real-time communication [9]. For TSN, there is no single standard, rather a collection of IEEE 802.1 standards. The authors in Bruckner et al. [4] provide an overview of all TSN standards. relevant for this paper, while just assumed to be present, are the IEEE 802.1Qbv (time-aware shaper) and IEEE 802.1AS [10] clock synchronisation.

A major distinction to classical networks in the industrial domain, is that TSN aims to provide dual-use in industrial as well as consumer applications, with a higher throughput and vendor neutrality [4]. Other functionalities of TSN are for example adding and removing connections with assured Quality of Service (QoS) properties over multiple hops in a bridged network [11].

C. Related Work

There is only a limited amount of research done concerning OPC UA PubSub in combination with distributed real-time end-to-end data transfer environments. For example, in Eymüller et al. [12], the authors describe a concept of real-time capable and long-running tasks in OPC UA. Specifically, OPC UA programs and TSN are combined to achieve a distributed and synchronised message exchange between applications. Pfrommer et al. [9] explore a similar path and analyse TSN for its usability to transport OPC UA PubSub messages in practice. In their solution, a hardware interrupt triggers an adjusted open62541 publisher to achieve real-time properties and concurrent data access between OPC UA PubSub and client-server. The publisher code is not worst-case execution time (WCET)-analysed. Panda et al. [13] used OPC UA PubSub to realise a field-level best-effort IP-based communication between OPC UA servers. The authors propose using an externally stored shared information model.

While not explicitly focusing on the real time-behaviour of the OPC UA PubSub stack, other researchers focused on TSN-enabled field devices with OPC UA running on commercial off-the-shelf (COTS) hardware and software [14] or exploring the possibilities of configuring TSN and OPC UA as an application layer protocol [15]. Similarly, Kobzan et al. [16] look into the combination of TSN, OPC UA, and software-defined networking for dynamic reconfiguration. Their approach highlights the importance of dynamic changes in future industrial networks while preserving real-time capabilities. Most of the research activities use the open62541 for their implementations OPC UA PubSub stack [7].

D. Overall RT-TSN-OPC-UA Concept

As mentioned before the wider aim of this research is to realise a distributed real-time end-to-end data transfer environment. While the issue of deterministic data transport between nodes is solved by the use of TSN, the end systems need to fulfil certain criteria. Each node is required to support TSN, additionally the OPC UA stack needs to process data packages deterministic, i.e. the stack needs to fulfil timing guarantees. Both issues have been previously investigated in Denzler et al. [5] for pure OPC UA PubSub systems including WCET analysis of the OPC UA PubSub stack. A remaining issue is to guarantee the timing behaviour of OPC UA PubSub in systems where the information model is also accessed by OPC UA client-sever.

OPC UA organises data and the corresponding semantic information in an OPC UA information model. The information model of a specific OPC UA sever is the so-called *address space*. Information in the address space can be accessed and modified via the client-server or publish-subscribe communication scheme as illustrated in Figure 1. During an address space access, the information can be read or adjusted without any restrictions, which leads to four possible concurrent data access cases:

- The publisher *reads* and the server *writes* a value.
- The publisher *reads* and the server *reads* a value.
- The subscriber *writes* and the server *writes* a value.
- The subscriber *writes* and the server *reads* a value.

All four cases could create an undesired outcome, for example, value changes during a read operation where the unit of a value changes (server *writes*) while the publisher packs the data frame (publisher *reads*). Other possible unwanted effects are, depending on the specific implementation, race conditions, deadlocks or life locks. For the OPC UA PubSub to remain time predictable, it is necessary to implement a concurrent data access mechanism that honours the time boundaries as well as the correctness of the sent value. The next sections introduce related work and possible data access mechanisms.

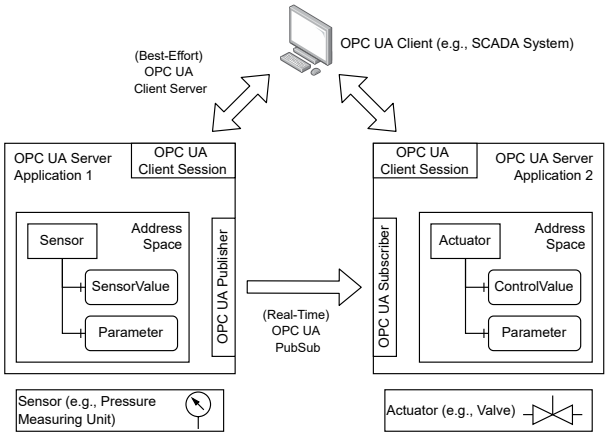

Fig. 1. OPC UA client-server and PubSub accessing a shared address space

III. Concurrent Data Access Mechanisms

The OPC UA address space is a shared resource that provides access to different processes. Therefore, the address space is a critical section or code block in OPC UA where only one task may execute at a time. This section presents the results of an investigation about suitable mechanisms to realise concurrent data access in OPC UA client-server and PubSub.

One possible solution to avoid simultaneous access is to implement only ***Single-threaded applications*** as they do not require shared data access mechanisms. If the application runs on a single-core processor (e.g., small microprocessors), another option is to allow a process to ***disable interrupts*** before entering the critical section and then re-enable interrupts after leaving it. The processor will not be able to switch between processes, if interrupts are disabled. Other solutions are required for more complex scenarios, which can be broadly categorised into blocking and non-blocking mechanisms.

A. Blocking Mechanisms

The primary requirement for a blocking mechanism is that one thread of execution never enters a critical section while a concurrent thread of execution is already accessing a critical section. In other words, while a thread accesses a shared resource, no other access is allowed during a specific time.

For mutually exclusive access, a process has to *lock* the address space before accessing it and *unlock* it when the access has been completed. Over the last decades, different implementations for *lock* and *unlock* mechanisms have been proposed; some are briefly described in the following.

A well-known blocking mechanism (semaphores) was invented by the Dutch computer scientist Dijkstra [17]. The semaphore ensures that a resource can only be exclusively accessed and must be released. There are several variations on the semaphore concept that are widely used.

Adding to the concept of a semaphore is the idea of a Mutex. A Mutex introduces ownership, meaning when a thread/task locks (acquires) a Mutex, it is the only one that can unlock (release) it. No other thread can unlock a Mutex. There are several other possibilities to implement *lock* and *unlock* mechanisms, but they must be omitted for space reasons.

B. Non-Blocking Mechanisms

If a resource must be accessed without blocking, so-called non-blocking mechanisms are used. Such mechanisms do not employ locks and eliminate the need for a critical section [18].

One mechanism proposed by Herlihy, Luchangco, and Moir in 2003 is called ***obstruction-free***. The idea is that any process can finish in a bounded number of steps if all other processes stop. Other implementations that allow concurrently access shared resources are called ***lock-free***. Such algorithms ensure that if a scheduler suspends a task in the middle of its activity, dependent tasks can still finish their activities without waiting. An extension of ***lock-free*** is ***wait-free*** algorithms that add the condition that every task using the shared resource may perform its operation in a finite number of steps, independent of the other tasks.

C. Properties of Concurrent Data Access Mechanisms

While each of the previously mentioned mechanisms aims to enable access to a shared resource, there are situations where they fail. One such situation is a **deadlock**. In a deadlock, threads/tasks obstruct each other by preventing each other from accessing the resource while making no progress in the meantime and eventually ceasing to function. Similarly, in a **livelock**, the tasks' status are continually changing while being dependent on each other and therefore can never finish their tasks. The significant difference to a deadlock is that the states are changing, but still, no progress is achieved. Another possibility is that a task cannot acquire access to a shared resource needed to progress. Such a situation is commonly known as **starvation**.

All those situations lead to no progress or ceasing function; however, shared resource access can also cause a limitation of efficiency. The desired characteristic of a concurrent data access mechanism is a high **throughput**, i.e., a high number of operations per unit of time. Therefore, an efficient mechanism has a high throughput the more successful operations can be completed in a given amount of time. High efficiency also includes a **fair** share of the resource between the tasks. These properties build a decision ground for choosing a suitable concurrent data access mechanism.

TABLE I
CONCURRENT DATA ACCESS MECHANISMS VS. PROPERTIES

Mechanism	Deadlock	Livelock	Starvation	Throughput / Fairness
Semaphore	yes	yes	yes	low / low
Mutex	yes	yes	yes	low / low
Obstruction-free	yes	yes	yes	low / low
Lock-free	no	yes	yes	high / low
Wait-Free	no	no	no	low / low

IV. APPROACH AND PRELIMINARY RESULTS

In the ongoing activities to realise the concurrent information model access in OPC UA between PubSub and client-server, the open62541 OPC UA stack [7] was analysed. Additionally, the previous mechanisms were investigated for their suitability.

A. Analysis of Concurrent Data Access Mechanisms

As indicated beforehand, a simple solution for concurrently accessing the information model is changing the stack to a single-threaded application. However, the open62541 code for the client-sever is not WCET analysable [6]. It would take a considerable effort to change the code to become deterministic enough to fulfil the necessary timing bounds. Moreover, the solution to disable the interrupts before entering the critical section of the code is not feasible for a real-time system.

Concerning the presented mechanism in Section III, a first suitability analysis resulted in Table I. The analysis considered real-time capabilities only for OPC UA PubSub. At first glance, the blocking mechanisms ***Semaphores, Mutex*** would not make a good choice, as both carry the risk of deadlocks.

The non-blocking mechanisms show better compatibility for the intended implementation. Primarily, the ***Lock and Wait-Free*** mechanisms could provide a feasible solution if the throughput is high enough. However, considering the specific setup, it might be possible to stop the client-server application and realise a ***obstruction-free*** solution. Such a solution would need some more profound analysis of the consequences.

This limitation also applies to the other mechanisms, seemingly unsuitable from the above analysis, which could be practical in a system where certain limitations are acceptable.

B. OPC UA Code Analysis

In addition, the analysis of the open62541 stack identified the critical code passages where a potential mechanism needs to be implemented. The analysis was performed using standard code analysis tools such as call graphs. Most relevant code segments connect directly to the OPC UA Node Store. A detailed code analysis will be provided in future work.

V. CONCLUSION AND ONGOING WORK

This paper presents an ongoing work to implement a concurrent data access mechanism into the OPC UA stack to allow parallel operation of OPC UA client-server and a real-time capable OPC UA PubSub. Identifying critical code segments and analysing potential concurrent data access mechanisms provided the ground for the next steps. The future steps will involve the implementation of ***wait- and obstruction-free*** mechanisms and evaluating their suitability. Combined with ongoing activities related to buffer management, the results will contribute to realising an end to end real-time enabled machine-to-machine communication environment.

REFERENCES

[1] T. J. Williams, "The Purdue enterprise reference architecture," *Computers in Industry*, vol. 24, no. 2-3, pp. 141–158, 1994.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: Association for Computing Machinery, 2012, pp. 13–16.

[3] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The Future of Industrial Communication: Automation Networks in the Era of the Internet of Things and Industry 4.0," *IEEE Industrial Electronics Magazine*, vol. 11, no. 1, pp. 17–27, 3 2017.

[4] D. Bruckner, M. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An Introduction to OPC UA TSN for Industrial Communication Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019.

[5] P. Denzler, T. Frühwirth, A. Kirchberger, M. Schoeberl, and W. Kastner, "Static Timing Analysis of OPC UA PubSub," in *2021 17th IEEE International Conference on Factory Communication Systems (WFCS)*, 2021, pp. 167–174.

[6] ——, "Experiences from Adjusting Industrial Software for Worst-Case Execution Time Analysis," in *2021 IEEE 24th International Symposium on Real-Time Distributed Computing, ISORC*, 2021.

[7] F. Palm et al., "open62541," *Available at https://github.com/open62541*.

[8] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer Science & Business Media, 2009.

[9] J. Pfrommer, A. Ebner, S. Ravikumar, and B. Karunakaran, "Open Source OPC UA PubSub Over TSN for Realtime Industrial Communication," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 1087–1090.

[10] S. Schriegel and J. Jasperneite, "Investigation of Industrial Environmental Influences on Clock Sources and their Effect on the Synchronization Accuracy of IEEE 1588," in *2007 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, 2007, pp. 50–55.

[11] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8.

[12] C. Eymüller, J. Hanke, A. Hoffmann, M. Kugelmann, and W. Reif, "Real-time capable OPC-UA Programs over TSN for distributed industrial control," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 278–285.

[13] S. K. Panda, M. Majumder, L. Wisniewski, and J. Jasperneite, "Real-time Industrial Communication by using OPC UA Field Level Communication," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1143–1146.

[14] A. Gogolev, R. Braun, and P. Bauer, "TSN Traffic Shaping for OPC UA Field Devices," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 951–956.

[15] F. Prinz, M. Schoeffler, A. Eckhardt, A. Lechler, and A. Verl, "Configuration of Application Layer Protocols within Real-time I4.0 Components," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 971–976.

[16] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, "Configuration Solution for TSN-based Industrial Networks utilizing SDN and OPC UA," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1629–1636.

[17] E. W. Dijkstra, "The structure of the "THE"-multiprogramming system," *Communications of the ACM*, vol. 11, no. 5, pp. 341–346, 1968.

[18] L. Lamport, "Concurrent reading and writing," *Communications of the ACM*, vol. 20, no. 11, pp. 806–811, 1977.