Open Source as Enabler for OPC UA in Industrial Automation

Florian Palm, Sten Grüner
Chair of Process
Control Engineering
RWTH Aaachen University
Aachen, Germany
{f.palm, s.gruener}@plt.rwth-aachen.de

Julius Pfrommer
Fraunhofer IOSB
Karlsruhe, Germany
julius.pfrommer@iosb.fraunhofer.de

Markus Graube, Leon Urbas
Chair of Process Control
Systems Engineering
Technische Universität Dresden
Dresden, Germany
{markus.graube, leon.urbas}@tu-dresden.de

***Abstract*—As a standardized communication protocol, OPC UA is the main focal point with regard to information exchange in the ongoing initiative *Industrie 4.0*. But there are also considerations to use it within the *Internet of Things*. The fact that currently no open reference implementation can be used in research for free represents a major problem in this context. The authors have the opinion that open source software can stabilize the ongoing theoretical work. Recent efforts to develop an open implementation for OPC UA were not able to meet the requirements of practical and industrial automation technology. This issue is addressed by the *open62541* project which is presented in this article including an overview of its application fields and main research issues.**

I. Introduction

In recent times, a number of concepts were presented under the terms *Internet of Things* (IoT) [1] and *Industrie 4.0* (I40). *Industrie 4.0* tries to meet the challenges which arise from closely interconnecting business processes with production systems. A central part of the definitions is the seamless linkage of all information-processing instances that are involved in the value chain [2]. IoT, on the other hand, is concerned with the connection of any type of embedded device, which inevitably leads to synergy effects. In contrast to I40, the main focus of IoT is placed on the consumer market. But there are also suggestions on how IoT could affect factory automation [3].

The resulting integration of heterogeneous communication partners calls for a specification of uniform communication standards either for the direct communication between communication partners or for the inset of a middle-ware, in order to meet the demands of automation technology that were defined in VDI/VDE 2657 [4]. In many cases, OPC UA [5] is suggested as a basis for communication in IoT, as was done in [6]. OPC UA specifies the structure of semantic information models, how information between communication partners is transferred and how these models can be expanded with user-specific models [7]. Especially the possibility of designing complex information models on target systems that can be accessed and explored by third parties represents a challenging new feature compared to the predecessor *Classic OPC* [8]. Current examples for a manufacturer-independent design of information models on an OPC UA basis are the IEC 61131-3-Model [9] the ISA95 Common Object Model [10] or the BACNet Mapping [11]. The main research questions are concerned with the transmission protocol, applied security concepts, server profiles, and the integration of new information models.

This paper is an extended version of an earlier concept paper [12] that was only published in German language. In this article we first (Section II) discuss the advantages of open source projects and point out why they are especially suitable to share concepts discussed in the context of IoT and I40. In Section IV, we point out the advantages of using an open software in the context of OPC UA and also present the currently existing open source projects. In Section V, the overview of existing solutions is used as a guideline to derive and then discuss the main requirements of an upcoming project. In the subsequent section, the current state of development and some important technical details of the open62541-project are presented. In Section VII, scenarios and applications that are of major importance for research purposes in the context of OPC UA are outlined. Furthermore, we will demonstrate that these scenarios and applications are realizable by using the stack. The article ends with a short summary and an outlook towards future research work.

II. Open Source for Concept Solidification

Open source software is widely used. Even in industrial environments, many vendors base their software on open source implementations which can be counted as further evidence for the professionality of certain open source projects.

The advantages of open source can be observed from both, the user side as well as from the developer side: Open source driven projects are less influenced by vendor policies and decisions are often the result of open conversations (forums, mailing lists, etc.) between developers. As a result, the decision-making process is always kept transparent. Furthermore, best practices can be adopted quickly.
Users can evaluate new technologies and their use-cases with low financial risks. Besides that, development is mostly driven by user needs. Furthermore, open source software can function as a feedback loop to standardization processes. A good example of how open source projects and standardization can work well together is the project “Boost” for the programming language C++ which provides libraries for a broad spectrum of use cases [13]. The purpose of the project is described

978-1-4673-7929-8/15/$31.00 © 2015 IEEE

by the authors as: “We aim to establish ‘existing practice’ and provide reference implementations so that Boost libraries are suitable for eventual standardization.” So far Boost functionality has been integrated into the C++11 standard and several libraries are proposed for standardization of the coming standard C++17 [13]. Of course, open source software also has certain drawbacks, particularly in matters concerning service and maintenance, since neither individuals nor companies have an obligation to provide these services. However, in research projects, especially for prototype development there are use cases in which the reliability of software is not critical.

III. OPC UA In A Nutshell

OPC UA is a service oriented protocol that is intended for industrial communication and is standardized in IEC 62541 [5]. It is based on a client server structure. The server is meant to hold the data which is modeled in an object-oriented manner. Therefore, OPC UA defines a meta model with two basic elements, i.e. nodes and references. References interconnect nodes and span a web which is called address space in the framework of OPC UA. The client uses standard services to explore and modify the exposed part of the address space. There are different types of protocol, namely binary protocol and SOAP-based webservices. The binary protocol is preferred when it comes to high data throughput between client server whereas the SOAP-based webservices are used to overcome firewall issues. OPC UA standard defines its own mechanisms for security, namely encryption and authentication. Thus, an extra encryption on transportation layer is not necessary.

IV. Demand for an Open Implementation of OPC UA

Data-model-based communication protocols like OPC UA have large impact on existing and new systems. Furthermore, the lack of a reference implementation of the developed concepts in the field of I40 and IoT leads to uncertainty. Even big companies are interested in open source implementations [14]. Though OPC UA is specified in the international standard IEC 62541, the communication stacks that were developed and maintained by the OPC Foundation (.NET, Java, Ansi C) are only available to members of the foundation or against payment of a fixed fee. However, an open and free implementation is necessary for a rapid dissemination of the OPC UA technology for the following reasons:

- Currently, extensions of the standard in the context of research and development are made difficult for various reasons, and alterations of the source code of commercial solutions (in case such changes are even available) cannot be passed on. This hinders innovations, since comprehensive projects cannot be build upon each other.
- A use of OPC UA beyond the original domain – for example in the context of the IoT – has so far been made difficult due to the absence of a free implementation. Other IoT-protocols such as MQTT[1] and CoAP[2] have evidently profited from open source implementations and deliberately integrate these implementations to facilitate a simple introduction and access to the technology.

[1]Message Queue Telemetry Transport: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[2]Constrained Application Protocol: https://tools.ietf.org/html/rfc7252

For these reasons, an open implementation of IEC 62541 would yield advantages for the entire OPC UA system. Since an open source project cannot fulfill the tasks of commercial operators which involve warranty and customer support, the latter will also profit from a greater dissemination of the technology.

The authors of this article currently know of eight open source implementations that use different platforms/programming languages to implement IEC 62541. A comparative overview of the different implementations with regard to aspects such as license model, functionality and activity of the developer community is provided in Table I. The requirements of automation technology (see Section V) are only partially covered by these implementations. In order to address this issue, a cooperation project between the Chair of Process Control Technology at RWTH Aachen University, the professorial chair of Process Control Systems Engineering at TU Dresden and the Fraunhofer IOSB was initiated at the beginning of 2014 and subsequently started the open62541 project.

V. Requirements and Objectives

IEC 62541 [5] specifies OPC UA as a communication standard for heterogeneous system environments regardless of the utilized hardware technology. This aspect should only be restricted by an implementation to a limited extent. Since the environment of automation technology is characterized by a large number of different hardware- and software architectures, it is very important to design the source code in a preferably platform-independent mode. This means that the stack should principally be transferable to any processor architecture. In this regard, the used programming language plays an important role since a compiler or an interpreter must be available for the respective hardware platform in the language concerned. Then again, the implementation should only stipulate preferably low requirements for the used hardware in terms of memory usage and processor load. The described requirements can be substantiated under the software quality characteristics portability and efficiency [17]. With regard to these considerations, the available implementations appear inadequate for the field of automation technology, since they either superimpose on frameworks (Node.js a Chrome JavaScript runtime) or Virtual Machines (Java, .NET), or make use of programming languages (C++) that have no complete compiler coverage in the world of embedded systems. For these reasons, C99 was selected as a programming language [18] for this project (see Table I). Hence, the requirements of open62541 for the underlying system are only a basic memory management functionality and BSD socket support.

The decision against the commercial stack, which is available as ANSI C implementation, lies in the fact that open source prototype development frequently takes place in university settings where software is often structured on a modular basis. When using commercial products, this approach will often lead to license law issues. For this project, a modified version of the non-viral LGPL [19] with linking exception is chosen as a license.

Table I: Comparison of known open-source OPC UA implementations (accessed May 2015)

Project Name	Language	License	Client/Server	Implemented Functionality	Platform	Last Contribution
node-opcua	JavaScript	MIT	both	Address Space management, Browse, Read, Subscription Services	GitHub	May 2015
FreeOpcUa	C++	LGPL	both	Address Space management, Browse, Read, Write, Subscription Services	GitHub	May 2015
OpenOpcUa [15]	C++	CeCill-C, not free of costs	both	n. a. (no open access to source code)	n. a.	n. a.
OPyCua	Python	GPL v3	both	Transport Layer implemented	SourceForge	Jun. 2012
OpcUaServer[16]	Java	n. a.	both	Address Space management, Browse Service	internal SVN	Jul. 2012
opcua4j	Java	CC BY-SA 3.0	Server only	Address Space management Browse, Read, HistoryRead, Write, CreateMonitoredItems Services	Google Code	Sep. 2013
opc-ua-stack	Java	Apache License V2	both	Adress Space management, Browse, Read, Write, Method/ Subscription Service Set	GitHub	May 2015
open62541	C99	LGPL + static linking	both	Address Space management, Browse, Read, Write Services	GitHub	May 2015

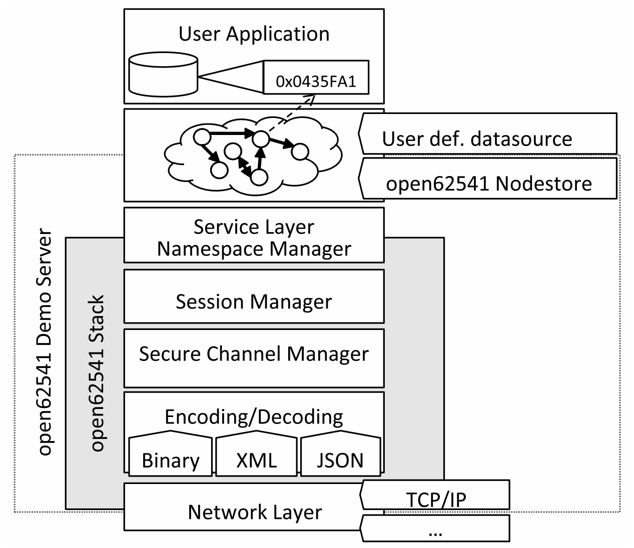

Figure 1: Structure of the open62541 stack

VI. Current State of Development

The different requirements of participants, which at times seem to be contradictory, are addressed by the strong modularization of the open62541 stack and the customized source code, generated to a large extent by XML-coded data models (currently about 85%). C structures and corresponding encoding-decoding functions are generated with the help of data type definitions. Other coding methods (XML, JSON) can easily be integrated by modifying the build-scripts.

A. Stack Structure

The stack is the core of the open62541 project. It consists of different layers similar to the ones described in [5] (see Figure 1). From the bottom-up, there is a network layer which allows the use of different protocols for data transportation such as TCP or UDP. It is the interface to the underlying hardware or operating system, respectively, and therefore needs to be adapted if the stack is run on a different architecture with no BSD Socket support.The decoding and encoding of data is performed on top of the network layer. Since OPC UA defines different data encodings (i.e. binary format and XML format), different encoders can be plugged in. Beyond this layer, there are two layers which address security (SecureChannel Manager) and authentication (Session Manager). There are two possibilities to access user-defined data, with one being the mapping (Read, Write etc.) of OPC UA standard services by means of the Namespace Manager and the other being the attachment of single values (e.g. current process values) to the data field of a variable node. On client-side an encapsulating interface is provided that gives access to the services. Besides the stack, a storage for user defined nodes and references is provided namely the open62541 *Nodestore*. User server-side applications can either use their own data storage or the provided one. Furthermore, if the provided data storage is used, a callback architecture can be used to update certain values of user defined nodes. With that, users can integrate their data, e.g. process values like the current measurement of a sensor, into an OPC UA address space.

As discussed in Section IV, the source code of open62541 shall provide a conceptional understanding of OPC UA. For this reason, the code is kept relatively brief and readable. A server for the OPC UA Nano-profile is implemented in approximately 5,500 lines of idiomatic C-Code (excepting the generated data structures).

On the one hand, the modularization focuses on the strong separation of the different layers within the stack that are described in the standard, for example, sessions and secure channels. On the other hand, it deals with the separation of stack and applications (those are server and clients). This has multiple advantages: In the downward direction, all TCP-specific functions, of which some are specific to the operating system, are shifted to the application in order that the stack remains platform-independent and that both, a synchronous and asynchronous network connection are possible. In the upward direction, the stack includes a whole range of callback-functions in order to establish, e.g. a connection to existing data sources. This type of modularization is also recognized in the licensing process. The stack is licensed under LGPL, while the exemplary server-client application is registered under a liberal CC0 license [20].

The modular structure allows the creation of custom-made server variations. As a result, the complexity of the functionality can be flexibly adapted and adjusted to the supported services and data types. This has direct consequences for the resource consumption of the server. At the present time, there is a server application that supports the binary protocol OPC UA TCP. Unencrypted connections to the server can be established (OpenSecureChannel Service Set) and sessions can also be created (CreateSession, ActivateSession). Moreover, it is possible to browse the static and internalized address space, while the node properties can be read invariably and written under certain conditions. In addition, multiple clients can establish a connection with the server simultaneously and make modifications to the address space. On the client-side, besides connection establishment, several services are implemented to browse and modify the server's address space (Browse, Read, Write, etc.). The implementation of the Subscription service set is currently in work.

VII. Application Scenarios

In this section, we will describe possible application scenarios for the research stack that have either been developed before the project was started or during the subsequent implementation. The project is solely concerned with problems in the framework of OPC UA that can be directly related to industrial automation technology.

A. Large Multi-Core Server Applications for C10K-Problem

Today's standard server hardware already features 8 processor cores including hyperthreading. This number of cores makes the simultaneous operation of 16 threads possible. However, the problem of running multiple process cores lies in the fact that they work within local memory maps of their individual caches even though they access the same central memory. These memory maps can differ from each other and lead to inconsistencies in the entire system. Hence, even purely local applications make more and more use of technologies which were developed especially for distributed systems. In open62541 the concurrent use of multiple processor cores has been factored in right from the outset. The open62541 library takes into account experiences from efficient and highly parallelized applications of the recent past, such as nginx-Webserver[3], the Redis In-Memory database[4], or the Linux-kernel.

The utilized technologies can be roughly divided into two categories: lock free data structures for synchronizing the individual processor cores, and event-based load distribution via asynchronous function requests. All mechanisms that are described in the following section can be removed through a simple flag in the build system. As a result, no disadvantages will emerge when open62541 is run on non-multithreaded systems like microcontrollers without operating systems.

a) Lock Free Data Structures: When processing OPC UA-requests in parallel threads, the differentiation between local states (which only concern this request) and global states (with a write access of potentially multiple threads) is a necessary prerequisite. In open62541, the global state occurs precisely at two points: during the management of secure channels and sessions, and at the central data storage which includes the nodes of the UA-address space. In the past locks, mutexes and semaphores were used to enforce the synchronization of the global state between threads. However, these approaches quickly reach their limitations, because the work load required for synchronization even for a small number of processor cores (managing the locks, frequent context switches in the operating system and waiting times for lock-releases) significantly exceeds the actual processing load for the request. Thus, the open62541 stack only makes use of lock free data structures for synchronization.

For example, if a node is changed in the UA-address space, it must be replaced completely (thus, nodes are "immutable" data objects). Writing into already existing nodes would mean that a parallel thread could read an incorrect state during the writing process. This is avoided by generating a completely new node and by only replacing the node pointer from a central position. In multicore-CPUs this can be realized with the help of atomic compare-and-swap (CAS) operations. In our case, the CAS operation compares the current memory space value with the old expected value and replaces it with the new value in case of agreement (without any disturbing influence of a parallel thread). Similarly, all processor cores that include the pointer in their caches will be notified to reload their memory space. For small memory spaces (e.g. a pointer or an integer value), this operation is implemented in the hardware and is therefore very efficient. We still have not addressed issue concerning when the old (replaced) node can be deleted. To make sure that no thread still holds a node pointer, a reference-counting mechanism is utilized. The RCU-mechanism (Read-Copy-Update) from the Linux-kernel ensures that no thread has a node pointer and at the same time, that no thread has incremented the reference-counter [21]. As a result, parallel threads can now process the same data without the occurrence of lock caused waiting time.

b) Event-based Network Communication: The issue of handling 10,000 concurrent connections while keeping up the same reaction time has recently gained a lot of prominence under the name C10K-Problem. In order to support such applications, most operating systems today support event-based network stacks[5] as an alternative to the POSIX-Modell via poll/select. A central advantage of these approaches is the disappearance of $\mathcal{O}(n)$-expenditure required for the iteration across all open connections. Instead, network messages (that include a pointer to the data structure for the respective connection) are kept in a waiting list from where they are asynchronously processed by available threads. Similarly, the response messages are also forwarded to a waiting list from

[3]http://nginx.org
[4]http://redis.io
[5]e.g. epoll (Linux), kqueue (BSDs), or IOCP (Windows)

where they are later deleted after they were dispatched (by the operating system). The fact that only one processor core is activated per thread means that no complex context switches can take place, which in turn ensures that messages can then be processed rapidly.. In open62541, both the use of event-based network stacks and the use of lock free data structures are optional. The open62541 library defines only one standard-interface for communication where respective implementations can be incorporated. An exemplary implementation of event-based network communication for open62541 based on the Libuv-library[6] was realized as part of the project and can be looked up in the repository.

B. Integration Strategies For Existing Object Management Systems

OPC UA is employed as a communication protocol by most users. However, this kind of utilization neglects the fact that OPC UA has a much more comprehensive meta model for information processing at its disposal than Classic OPC. This meta model does not only provide much simpler data, but it also maps the semantics of this data directly into the target system (sensor, actuator, controller) via complex object hierarchies.

On the one hand, the standardized meta model offers considerable advantages with regard to interoperability. On the other hand, this can lead to more complex data management issues on the server side. Generally, one can differentiate between two different types of application: either an application with data management and no meta model or one with inherent data management based on a proprietary meta model. In order to address the first case, the open62541 project offers the possibility to create an address space and to deposit existing data in the form of objects by using the OPC UA meta model. For this purpose, the open62541 project provides a *NodeStore* that can take over data management and enable the stack to gain access. With the standardized meta model functioning as a basis, a concrete number of standardized branch-specific information models can be set up in order to support interoperability [8]. Thus, a whole range of information models like the IEC 61131-3-Model can be created in addition to the existing information models.

In case an application has an individual meta model which differs from the OPC UA meta model, one has to decide to what extent existing data should be explorable via an OPC UA client. On the one hand, the *NodeStore* of the open 62541-project can be used to deposit a certain part of the existing data as an explorable OPC UA-conform model. This is the preferred way to couple legacy applications with OPC Classic interfaces etc. [22]. However, if the entire data retention is to be made available to an OPC UA Client, one will require a model transformation. The meta model defines a set of different nodes and references which are inherent to OPC UA. The address space is then set up on the basis of these two structured elements.

C. Design-for-Evolution-Strategies

The evolution of information models presents a major challenge for open62541. At the point of creation, information models typically describe the desired domain in reasonable detail. However, the world moves on and as a result, the demands regarding applications that are used by OPC UA also change with time. In many cases, the development will also require changes to the semantic information model in OPC UA. In the worst case, the information model and the existing applications will become incompatible.

One potential solution to this problem involves a clever modeling approach that already considers the possibility of extension. On the other hand, a server-based automatic transformation between different versions of one or multiple information models can minimize the problem to a large extent. For this purpose, one must first develop an internal revision management system for information models in OPC UA, similar to [23]. Importantly, one should also make use of model-based transformations. These transformations should support bidirectional transformations for better maintainability. Since the information model OPC UA can also be regarded as a graph, Triple-Graph-Grammars [24] represent a good choice for this particular task.

D. Resource-Oriented Service Architecture

Resource orientation is an architecture paradigm that is characterized by precisely carved services. In contrast to service-oriented architectures (SOA), a resource-oriented architecture (ROA) [25] is composed of a small number of services that are known to all participants. Thus, the necessity of including service descriptions (e.g. for web services using WSDL [26]) is eliminated. The variability of the architecture is therefore not achieved through service descriptions but by means of variable resources that are processed by the standardized services. A prominent example of ROA is the so-called RESTful (Representational State Transfer) interface. This interface is realized with URLs, which are interpreted as CRUD (Create, Read, Update, Delete) operations, and the standard methods of the HTTP-protocol.

A similar set of services that can be interpreted as CRUD operations and utilized in the sense of an ROA is defined in IEC 62541. Crucially, negotiations on at least three levels of the stack have to be conducted and completed in OPC UA before a CRUD operation can be executed, namely connection, secure channel and session. The question of how ROA can be realized with stateless services by means of OPC UA remains an open research issue. A first contribution towards the evaluation of protocol extensions with the help of open62541 was presented in [27]. In this work possibilities of realization of a RESTful architecture style based on OPC UA protocol were studied. The following extensions two were introduced: stateless calls, i.e., calls to inherently stateless OPC UA services without establishing a session or secure channel, and secondly, a non-standard UDP transport layer which replaces the standard OPC UA transport layer that is based on TCP. Both proposals were studied independently and also in combination with regard to the reduction of communication overheads for sporadic interactions. Notably, there are scenarios where the throughput of the server could be increased by an order of magnitude.

[6]https://github.com/joyent/libuv is the network and multithreading-core of Node.js application server (available as a standalone C-library)

E. OPC UA on Embedded Systems

By understanding sensors such as temperature sensors as embedded systems, one can assess that they are characterized by a small memory size (RAM and ROM) and a small amount of CPU power. With the binary protocol, OPC UA is even dedicated to those systems [8]. Compared to HTTP/XML the overhead is drastically reduced. Still, the OPC UA standard requires the specification/definition of a large amount of data types. The complex data types are composed of so called built-in data types (Integer, Boolean, String, etc.). As mentioned above, the code for handling those types is generated automatically. This includes structures as well as handling functions (create, delete, copy, encode, decode, size calculation). A significant reduction to the size of the executable can be achieved if only one single set of handling functions is created. The fact that every complex data type is just a composition of built-in data types implies that two kinds of functions are required: concrete handling functions for built-in types and generic functions that are called recursively. If a complex data type is handled by the generic function, each member is repeatedly called until the data type that is to be handled becomes a built-in type. Moreover, the provided DataType structure makes sure that the generic function can handle each standard data type, by packaging information about the current type (data type Id, count of members, memory layout, and specific other things) in a compact manner. Overall, the improvement results in an executable size of around 70 KByte for a OPC UA server application with the described functionality.

F. Benchmarks for OPC UA

An architecture with a lightweight service structure requires high-performing servers. The open62541 stack can be used for setting up benchmark-clients by means of which different performance indicators of servers can be measured. These indicators include values such as e.g. response time and server resource consumption per request/session. An open source benchmark would contribute to the transparency and reproducibility of the obtained test results.

VIII. Summary and Outlook

In summary, this article points out why an open source realization of OPC UA is important for initiatives like I40 in industrial automation and also presents an open source solution for OPC UA. At first, we discussed why OPC UA is relevant and presented reasons why an open platform is required. An analysis of currently available open source implementations revealed that they do not meet the requirements of automation technology. This consequently led to the formation of the open62541 project. In addition, we also presented scenarios where the open62541 project is currently in use and where it may serve as a basis in the future. The current state of development was then demonstrated in the penultimate section. What we currently know is that the presented scenarios as well as the stack itself will undergo further examinations. This will eventually result in the implementation of additional standard-specified functionality in order to support a wider field of applications. In particular the plan is to implement the functionality of the Nano-profile in the near future. Furthermore, the results from the presented scenarios will also be released.

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] Forschungsunion and acatech, “Recommendations for implementing the strategic initiative INDUSTRIE 4.0,” 2013.

[3] A. M. Houyou, H.-P. Huth, H. Trsek, C. Kloukinas, and D. Rotondi, “Agile manufacturing: General challenges and an iot@ work perspective,” in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–7.

[4] *VDI/VDE 2657. Middleware in industrial automation – Fundamentals*, VDI/VDE, 2013.

[5] *OPC Unified Architecture Part 1-10*, IEC Std. 62 541, Rev. 1.0, 2010.

[6] S. Haller, S. Karnouskos, and C. Schroth, *The Internet of Things in an Enterprise Context*. Springer, 2009.

[7] W. Mahnke, A. Gössling, M. Graube, and L. Urbas, “Information Modeling for Middleware in Automation,” in *Proc. IEEE ETFA 2011*. IEEE, 2011, pp. 1–7. [Online]. Available: http://dx.doi.org/10.1109/ETFA.2011.6059111

[8] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC Unified Architecture*. Springer, 2009.

[9] (2015) OPC UA Information Model for IEC 61131-3 version 1.00. [Online]. Available: http://www.plcopen.org/pages/tc4_communication/

[10] OPC Foundation, “OPC UA for ISA-95 Common Object Model – Companion Specification,” 2013.

[11] (2014) ISO 16484-5 and IEC 62541 – Mapping between BACnet and OPC-UA Presented. [Online]. Available: https://opcfoundation.org/

[12] F. Palm, S. Grüner, J. Pfrommer, M. Graube, and L. Urbas, “open62541 - der offene OPC UA-Stack,” Lemgo, Nov. 2014, [in German].

[13] (2015) Boost C++ Libraries. [Online]. Available: http://www.boost.org/

[14] Meinrad Happacher. (2015) Taugt OPC UA zum IoT Standard? [in German]. [Online]. Available: http://www.computer-automation.de/steuerungsebene/steuern-regeln/artikel/114926/1/

[15] (2015) OpenOpcUa. [Online]. Available: http://www.openopcua.org/

[16] (2015) OpcUaServer. [Online]. Available: http://most.bpi.tuwien.ac.at/opc-ua-server-progress/

[17] *Systems and Software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*, IEC Std. 25 010, Rev. 1.0, 2011.

[18] *Programming Languages – C*, ISO/IEC Std. 9899:1999, 2011.

[19] (2015) GNU Lesser General Public License, Version 3.0. [Online]. Available: http://www.gnu.org/licenses/lgpl.html

[20] (2015) Creative Commons License, Version 1.0. [Online]. Available: http://creativecommons.org/publicdomain/zero/1.0/legalcode

[21] J. Triplett, P. McKenney, and J. Walpole, “Resizable, Scalable, Concurrent Hash Tables via Relativistic Programming.” in *USENIX Annual Technical Conference*, 2011, p. 11.

[22] T. Hannelius, M. Salmenpera, and S. Kuikka, “Roadmap to adopting OPC UA,” in *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, July 2008, pp. 756–761.

[23] M. Graube, S. Hensel, and L. Urbas, “R43ples: Revisions for Triples - An Approach for Version Control in the Semantic Web,” in *Proc. of Linked Data Quality (LDQ) Workshop at Semantics Conf.*, Leipzig, 2014.

[24] E. Kindler and R. Wagner, “Triple graph grammars: Concepts, extensions, implementations, and application scenarios,” *University of Paderborn*, 2007. [Online]. Available: http://svn-st.inf.tu-dresden.de/svn/reuseware/tags/20100220-emftext-1.2.2/Tornado/de.tudresden.tornado2.metamodel.modelCorrespondence/documentation/TGG-TechnicalReoprt.pdf

[25] T. Erl, *SOA Design Patterns*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2009.

[26] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana *et al.*, “Web services description language (WSDL) 1.1,” 2001.

[27] S. Grüner, J. Pfrommer, and F. Palm, “A RESTful Extension of OPC UA,” in *Factory Communication Systems (WFCS), 11th IEEE World Conference on*, May 2015, pp. 25–27.