Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 52-27 (2019) 317–322

General methodology for building of OPC UA gateways

Tomáš Ausberger [*] Milan Štětina [**]

[*] *NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Technická 8, 306 14 Pilsen, CZ (e-mail: tomasa@ntis.zcu.cz.)*
[**] *NTIS – New Technologies for the Information Society, Faculty of Applied Sciences, University of West Bohemia, Technická 8, 306 14 Pilsen, CZ (e-mail: mstetin2@ntis.zcu.cz.)*

Abstract:
This article describes an original generalized methodology for building OPC UA bridges for different industrial communications and internal command protocols. This methodology is aimed for fast development of OPC UA gateways for less popular data transferring communications which are not available on the market. A final gateway can be added to existing solutions without modification of data source control devices. This gateway needs only a minimal configuration and obtains information about data source automatically. This design has been already implemented in three different solutions with various communication (REXYGEN, PerNet, a Keysight middleware protocol). The solutions has been deployed in various projects and proved to be efficient and reliable.

Keywords: OPC UA, industrial communication, OPC UA gateway, OPC UA bridge

1. INTRODUCTION

This article describes an original methodology for building OPC UA bridges for general industrial communications. It identifies and map together crucial methods of OPC UA and mandatory methods of data transferring communication. It also defines a processes that have to be implemented in OPC UA gateway in order to provide efficient and reliable interface between original data source device and OPC UA clients.

This methodology provides general, easily implementable instructions for designing an effective OPC UA gateway. It's suitable for most data transferring protocols and therefore it is ideal for internal or less popular data transferring protocols that don't provide any usable OPC UA gateway yet. The concept itself has been repeatedly implemented with different data transferring communications and tested in long-term monitored projects. It has been proven as efficient, reliable and customizable for different behaviours of various communications.

The gateway is designed as a standalone application that can be easily configured and installed without modifying the target control device (see Figure 1). It consists of an original communication client and an OPC UA server. It communicates with a data source device which serves as an original communication server. The gateway browses the data structure, reads and writes data of the device and stores the data in the OPC UA Address Space. OPC UA clients can then access the data via standard OPC UA services. For the device, the gateway acts like any other original client; therefore an OPC UA server interface can be added to the device on the fly.

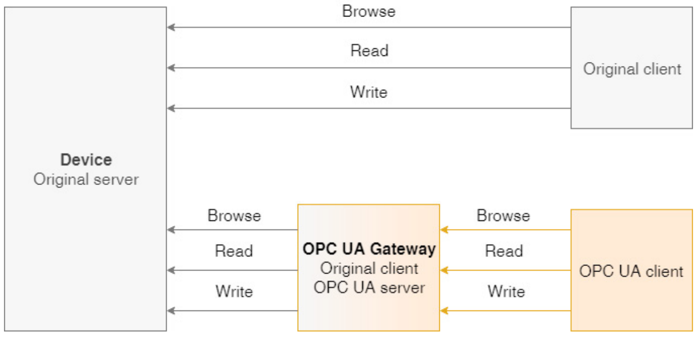

Fig. 1. The gateway provides an OPC UA interface without any modifications of target device

The used approach is derived from a common concept of gateway between two data transfer communications. This concept has been examined in the context of OPC UA and its crucial processes have been identified. A derived methodology has been designed as general and easy to implement as possible, while preserving reliability and optimizing network overhead. Therefore the gateway doesn't only resend all OPC UA clients' requests to data source device but it tries to lower the network overhead as much as possible.

The gateway is not fully dependable on its data sources but it itself contains information about connected devices and their last acquired data. Therefore it can provide last known data to OPC UA clients even when a data source is disconnected. Clients themselves decide how old data are still acceptable for their purposes.

On the market are actually available many OPC UA gateways, which are bridging mostly common industrial communications, namely OPC DA (e.g. OPC UA – OPC DA gateways from Unified Automation GmbH[1], Prosys OPC Ltd[2], Software Toolbox Inc.[3], OPC UA – MODBUS gateway from Advantech Co., Ltd.[4]). However those OPC UA gateways are always dedicated to common industrial communications, leaving behind other data transferring communications.

The main result of this article is a general methodology for designing OPC UA gateways, which can be used with any applicable data transfer communication. This methodology is easy to use and provides stable, efficient and reliable solutions. The methodology is flexible enough to fit behaviour of most data transferring communications. Optimal requirements for used data transferring communications are browse, read and write methods. However, a limited gateway can be designed even for communications that does not fully support these methods. The only requirement is the ability to implement a client for original communication. This article can be availed mainly by small and medium companies. It can help with implementing and deploying of OPC UA servers and therefore with spreading of OPC UA in industry.

2. OVERVIEW

2.1 OPC UA

OPC UA (OPC Unified Architecture) is a new industrial communication standard developed by the OPC Foundation, see OPC Foundation (2018). It's designed as a platform independent, scalable and modular service-oriented communication with strong security mechanisms. This makes it an ideal solution for the IoT (Internet of Things) and Industry 4.0, for more details see Zezulka et al. (2018). The OPC UA can be used for connecting two PLCs (Programmable Logic Controller) on the same floor or for data aggregation and transfer between devices on the control floor and other floors (operations, management and enterprise) with scalable security.

An OPC UA gateway created with presented methodology can be designed either as a simple gateway or as an aggregation server for several connected control devices. A new OPC UA PubSub communication can be also implemented in the OPC UA Gateway; however, the preferred use of Publisher interface is to implement it directly in control devices and sensors and omit the Gateway entirely.

Detailed information about customizable OPC UA functions and settings like discovery services, endpoints or security are described in the specification OPC Foundation (2018). Security and endpoint settings may affect the performance of OPC UA communication. It is recommended to consider requirements of the target environment. For a detailed analysis see Cavalieri and Chiacchio (2013) and Post et al. (2009). A gateway's performance and cost can be also affected by chosen platform, see (Cho and Jeong, 2018).

2.2 Address Space

All shared content of OPC UA server is stored in its Address Space. OPC UA clients can access this content via OPC UA services (see OPC Foundation (2018)). OPC UA Address Space consists of nodes, references and events. Address Space can be built by managing nodes, binding them together with references and setting their attributes. Even at the start of an OPC UA server, the Address Space isn't empty. The OPC Foundation pre-defines all fundamental nodes in the OPC Foundation namespace, and every OPC UA server has to implement them.

Nodes can be classified as VariableType, Variable, ObjecType, Object, ReferenceType, EventType, DataType, Method or View. Each node has its attributes like NodeId, BrowseName, DisplayName or NodeClass. Some of these NodeClasses also contain other attributes, e.g. Variables also contain following attributes: Value, ValueRank, ArrayDimensions, AccessLevel, UserAccessLevel, etc. These attributes are defined for each NodeClass and cannot be further extended.

2.3 Namespaces and NodeIds

The principle of Namespaces is to uniquely identify and group data of one logical part. Two nodes with the same NodeId (Namespace and Identifier) represent one virtual (or physical) object in the OPC UA universe.

Namespace is uniquely identified by its URI. However, NodeIds are composed of Identifier and NamespaceId. NamespaceId is only an index to the local Namespace Table. A common mistake is to compare only NamespaceId and Identifier of NodeId instead of comparing Namespace URI and Identifier.

Every OPC UA server provides at least two namespaces: the first one is the OPC Foundation namespace with build-in nodes, the second one is a namespace of the OPC UA server which is used for its internal needs. A gateway built with presented approach will typically also have one type namespace for declaring new NodeTypes and one device namespace for each connected data source device.

The type namespace is present in all gateway instances, and its nodes shall be consistent. This means that new NodeTypes are virtually declared only once in the whole OPC UA universe and all application instances are only making references to them. Any modification of nodes in the type namespace should be backward compatible.

A device namespace should be created for each connected device. A device namespace's URI can be generated automatically using the gateway's configuration and information about connected control devices. A device namespace's URI shall be composed of static identifier of the gateway (Gateway-Company), given identifier of the environment (Project) and given identifier of target control device (Device). The form of a device namespace's URI can be something like this:

[1] https://www.unified-automation.com/products/wrapper-and-proxy/uagateway.html
[2] https://www.prosysopc.com/products/opc-ua-gateway/
[3] https://www.softwaretoolbox.com/datahubopcgatewaypack.html
[4] https://advdownload.blob.core.windows.net/productfile/PIS/EKI-1242IOUMS/Product%20-%20Datasheet/EKI-1242OUMS-A_1242IOUMS-A_DS(09.26.18)20181007183020.pdf

urn:Gateway-Company:<Project>:<Device>

An Identifier can be defined statically or dynamically. Static Identifiers (usually numeric or string) are suitable for namespaces with nodes that don't change over time (e.g. type namespace) or nodes that can be determined from their Identifier (e.g. device namespace of devices with distinguishable variables). Dynamic Identifiers (usually numeric or GUID) are generated automatically by OPC UA server and don't contain any information about their virtual objects. The gateway should use static Identifiers for its device namespaces' nodes if possible, otherwise the use of the gateway shall be restricted (e.g. no redundancy).

3. GATEWAY

The OPC UA gateway is a standalone application which implements two interfaces, an OPC UA server and an original communication client. The gateway has to implement methods of original communication for managing a connection with a control device (connect, reconnect, disconnect, handle errors, etc.), browsing a data structure of the device, checking a data structure's consistence, reading and writing values of device's variables. All of these functions has to be implemented to develop a fully functional gateway, however, a limited gateway can still be built without some of these functions.

The main purpose of the gateway is to browse a data structure of control device (namely its objects and variables), replicate it to the OPC UA Address Space and let OPC UA clients read and write values of its variables. The data structure of control device is stored in the Address Space in a form of nodes and references. OPC UA clients can then access them with OPC UA services. Whenever a data structure in the control device changes, the gateway removes all its nodes and re-browses its data structure.

After establishing a connection and browsing a control device's data structure, the gateway starts synchronizing data with the target control device. Data are generally synchronized in bulk and on-demand. When the gateway receives a request from an OPC UA client for reading or writing a value of a synchronized variable, it stores the request to a synchronization stack and triggers a synchronization process. A result of the process is sent back to the OPC UA client as a status code of the response.

Monitoring of synchronized variables is handled differently. The gateway has to fetch and store values of monitored synchronized variables from the target device periodically. Classic OPC UA notifications on data change are then generated automatically. The gateway adds a variable to the synchronization stack when it starts being monitored by first OPC UA client and remove it when it stops being monitored. Thus, the Gateway's performance is more affected by the count of monitoring items and less by connected clients. A synchronizing interval represents a maximum period between two synchronization calls. The minimal sampling interval should be higher or equal to the synchronizing interval to make data fetching faster than data sampling and to limit useless notifications.

To utilize the gateway as an aggregation server, a list of connected devices and their connection details has to be defined in the configuration. The gateway creates a synchronization thread and a synchronization stack for each device. Browsed variables of one device have to be added to its dedicated namespace. When an OPC UA client requests to read, write or monitor a device's variable, the gateway have to select the right synchronization stack and add the requested variable into it.

A modification of the gateway may be necessary while implementing different original protocols. When an original protocol doesn't provide a browsing method then all devices' variables have to be specified explicitly in the configuration. If an original communication uses values with quality then these quality codes shall be mapped to OPC UA status codes. Finally, if it doesn't support a data consistence check then the gateway shall either compare browsed data structure with device's data structure, use constant data structures or re-browse a data structure after each connection error.

3.1 Data structure

A structure of nodes which represents a data structure of the connected device depends on a behaviour of used original communication and connected control devices. A hierarchy of the structure can be static or dynamic. A static hierarchy uses predefined nodes (and their types) for certain levels. A dynamic hierarchy is browsed recursively. Used objects and variables can be generic or specialized. Some attributes of variables and objects can be determined from browsed information: BrowseName, DisplayName, numeric or string part of NodeId, user permissions to read and write, value rank and array dimensions of variables. Even a value of additional properties like Min and Max can be used for preserving information about variables.

The recommended way to store devices' data structures is to create a separate namespace for each device, create a folder in this namespace and organize the folder from the well-known node Objects. This folder should be created when a data structure is browsed. All nodes of browsed data structure should be placed in this folder. It is recommended to create these nodes first and then add the whole tree of nodes in the Address Space at once. When a device's data structure changes then the gateway should remove its folder and connected tree of nodes from the Address Space. That should apply even when the device disconnects for an extended period or when it's uncertain if the device's data structure remains consistent. Folder nodes and the well-known node Objects should be versioned. NodeVersions of the Objects node and particular folder shall be changed every time a device folder is added or removed from the Address Space. OPC UA clients shall be notified about these model changes by standard OPC UA events. A name of each device can be stored in a DisplayName of its folder, thus the Address Space remains clearly organized.

The gateway can define and use its own NodeTypes. These NodeTypes should be created in the type namespace and should represent every type of variable or object used in data structures. A type of node can be used for filtration of synchronized objects and variables. When the gateway processes a request for reading, writing or monitoring a value of a node which is an instance of new VariableType, it selects the right synchronization stack and insert the processed node into it. However, data structures can even be made using build-in nodes only. In that case another method has to be used for identification and filtration of synchronized nodes (e.g. all variables in a specific namespace).

3.2 Gateway functions

A purpose of the gateway is to provide data of connected devices, handle client's requests for reading, writing and monitoring data and synchronize requested data with connected devices. However this synchronization is not always instant, the gateway tries to lower the network overhead by caching data and synchronize monitored data periodically in bulk.

At the start an OPC UA server has to be configured and has to build its Address Space. If the gateway uses its own NodeTypes, it shall insert them in the Address Space. It shall also define and initialize connections to target control devices and create a synchronization stack and a synchronization thread for each connection. After initialization, the OPC UA server and synchronization threads have to be started. The gateway then waits for a close signal to close all synchronization threads, the OPC UA server and the application itself.

A synchronization stack is composed of lists of read requests and write requests and lists of nodes requested to be read, written or monitored. These nodes represent variables of the connected device and should be instances of the gateway's new NodeTypes. All synchronization lists have to be unique (without duplicate items) but a node can be present in all three lists of nodes at once (it can be queued to read, write and monitor).

When the gateway receives a read or write request about a synchronized node, it has to insert the synchronized node into the corresponding node list of synchronization stack; the request itself has to be inserted into the corresponding request list. Read and write nodes and requests are presented in the synchronization lists until synchronized and responded. The list of monitored nodes, on the other hand, remains unchanged after synchronization. A synchronized node is inserted into the monitored node list when the first client starts its monitoring and remains there until the last client stops its monitoring.

A synchronization thread consists of an infinite loop which can be interrupted by a close signal. The thread has to manage a connection with the device (connect, reconnect, handle errors), check a consistence of its data structure, browse and re-browse its data structure if required and synchronize requested data from the synchronization stack. A semaphore may be present in this loop, pausing a thread when no synchronization is required. The server shall send a signal to this semaphore when a client requests data synchronization with the device. However, this semaphore cannot wait indefinitely, but it has to use a synchronizing interval as its timeout. The synchronizing interval represents a minimal interval in which the synchronization thread has to synchronize data for monitored nodes. This interval can be set in the gateway's configuration, or it can be determined from sampling interval of monitored items. The synchronization interval has to be lower (or at least equal) than minimal sampling interval of synchronized monitored nodes.

A thread's synchronization method synchronizes values of nodes and process stored clients' requests. Initially, the method has to safely copy all lists of synchronized nodes and requests from the synchronization stack to local copies. The list of monitored nodes has to be merged to the local list of read nodes; duplicities shall be removed. Subsequently, the method has to clear lists of the synchronized stack except the monitored nodes list and release the synchronization stack to be filled again by new requests and nodes. The method shall further use only local copies of synchronization lists. Next, the method has to write values from written nodes to the device's variables and send a result of the operation in a write response to OPC UA clients. Finally, the method has to read values of requested read nodes from the device's variables, save them to corresponding nodes and deliver a result of the operation in a read response to clients.

It is necessary to modify methods for reading and writing values of synchronized variables. Every time the gateway processes a client's request for reading or writing a value attribute of a synchronized variable, it shall check the request and insert requested nodes from valid requests to the synchronization stack. If the gateway processes a write request, it should first set a written value from the request to the target node and then insert the node to the synchronization stack; thus, only the last written value is synchronized with the device. Synchronized variables can be determined by their namespace or NodeType. All synchronized nodes may contain a reference to the desired synchronization stack to facilitate its selection.

4. IMPLEMENTATION

In the next chapters are described three OPC UA gateways that has been implemented using described methodology. These gateways are used as a proof of concept and as an implementation guide. Each of these gateways uses a data transfer communication with different behaviour. In each chapter are described a behaviour and a data structure of used communication, implemented methods, chosen OPC UA representation of data structure, deployed solutions and obtained results.

The complexity of OPC UA makes it difficult to implement from scratch. That's why some organisations and groups makes its own toolkit (a set of libraries) which implements some core functions of the OPC UA specification and provides an interface for development of OPC UA applications. Some of these toolkits are free to use (e.g. open6241[5]) but some have to be purchased from their authors (e.g. ANSI C[6] and C++[7] toolkits of Unified Automation GmbH). An efficiency and provided functions may vary from toolkit to toolkit.

[5] `https://open62541.org/`

[6] `https://www.unified-automation.com/products/client-sdk/ansi-c-ua-client-sdk.html`

[7] `https://www.unified-automation.com/products/server-sdk/c-ua-server-sdk.html`

The presented gateway implementations were developed using the OPC UA c++ toolkit by Softing Industrial Automation GmbH for Windows [8] and Linux [9]. This toolkit generates a fully functional OPC UA server on its own with minimum provided information. Developers can make changes to this server and modify it to work as requested. The toolkit allows developers to configure settings of the server (e.g. Endpoints, certificates, server information), to define and manage nodes and references in the Address Space and even to overwrite some of its internal handlers.

4.1 REX OPC UA

REXYGEN is a system developed by REX Controls s.r.o. The essential part of this solution is a RexCore which can be commanded via its own internal command protocol based on TCP (Transmission Control Protocol). This protocol allows uploading, running and stopping an executive, browsing its blocks, reading and writing values of blocks' variables and diagnosing the executive. The executive is an application code that can be compiled, uploaded to RexCore and executed. The executive consists of multiple periodically executed tasks. Each task is constituted of function blocks and subsystems that are connected together and can be displayed in a form of FBD (Function Block Diagram). Subsystems serve merely for grouping other function blocks to logical units. Function blocks, subsystems and tasks can contain multiple variables. These variables are qualified as inputs, outputs, parameters and states. The RexCore allows users to read all values of these variables, but to write only values of parameters and inputs. Permissions can be further limited by RexCore authorization.

REX OPC UA has been built with browse, read and write method. It can be connected to multiple RexCore instances at once and can work as an aggregation server. To its work the gateway needs administrator's credentials for RexCore authentication, these credentials has to be specified in target device's configuration. The gateway is therefore allowed to browse all device's blocks and read and write their variables. An access of OPC UA clients to write values can be restricted by used User Token Policy (authentication mode) of endpoints.

The system can theoretically change its executive at any moment. If that happens, the internal command protocol starts returning an ExecutiveChanged error until it's reloaded. The gateway checks error codes of command protocol responses and re-browses the data structure of connected RexCore (and reload the command protocol) when an ExecutiveChanged error is returned. If an executive is changed while a connection to target RexCore is interrupted, the command protocol doesn't return an ExecutiveChanged error. For this situation the start time of browsed executive is stored and verified after each reconnection. If the new start time doesn't match the stored one, an executive has been changed and the data structure has to be re-browsed.

The gateway creates a folder organized by the well-known node Objects for each connected RexCore. The gateway browses recursively executive's tasks, subsystems, blocks and their variables and organizes them with HasComponent references to maintain the same structure as in the diagnostic tool REXYGEN Diagnostics using instances of TaskType, SubystemType, BlockType and IVariableType subtypes (Input, Output, Parameter and State). Only values of IVariableType instances are synchronized with their corresponding target RexCore. Read and write requests are synchronized in bulk. The gateway stores an ID of each block or variable for synchronization, a name in node's DisplayName and a browsed path in its NodeId's string Identifier. It also creates Min and Max properties for each variable for the storage of limits of its value.

REX OPC UA is a part of REXYGEN solution, and it's available on the market [10]. It has been used in various projects by both REXYGEN developers and users for more than five years. Reported issues have been processed and led to updates of some core principles. An example application with 556 objects and 2,472 variables and 50 ms synchronizing period uses 15.3 MB of RAM. For further information look at REX Controls s.r.o. (2019).

4.2 OPC UA PerNet Gateway

PerNet is a communication used by SandRA (Safe and Reliable Automation) system of ZAT a.s. It is a service oriented communication based on UDP (User Datagram Protocol) and allows clients to browse structures and their variable arrays, read data of structures and write data of variables. Data of PerNet server are stored in structures. A structure consists of arrays of different date types; each array is dedicated to one predefined date type and contains constant amount of variables. The size of every array is contained in browsed structure's description. Predefined date types are for example Boolean, Integer-32, Float, TimeDate, BitSafe, RealSafe and others. BitSafe and RealSafe contain value with validity, other date types contain value only. An access for reading and writing data of structures is configured for each structure separately, this information is also contained in structure's description.

OPC UA PerNet Gateway has been built with browse, read and write functions. It can be connected to multiple PerNet servers and can be utilized as an aggregation server or to maintain redundancy. A PerNet data structure can be theoretically changed, but that requires a change of application code in connected device (PerNet server). The gateway stores a print of browsed structures' description for each server and compares it periodically with a print of re-browsed structures' description. If these prints do not match, affected old structures should be removed and new ones should be created from re-browsed structures' description.

The gateway creates a StructureFolder for each connected PerNet server with its name stored in the node's DisplayName. This StructureFolder contains a data structure of corresponding PerNet server represented by Structures. Each Structure contains a StructureArray component for each data type array used in the particular structure, each

[8] `https://data-intelligence.softing.com/products/datafeed-opc-sdks/datafeed-opc-ua-c-server-client-sdk-for-windows/`
[9] `https://data-intelligence.softing.com/products/datafeed-opc-sdks/datafeed-opc-ua-c-server-client-sdk-for-linux/`

[10] `https://www.rexygen.com/`

StructureArray has as many StructureVariables as is the size of selected data type array. A StructureArray and its StructureVariables have the same data type. Structure-Variable contains a scalar value, StructureArray contains a one-dimensional fixed-length array with values of its StructureVariables in it. A value can be read both from StructureArray or StructureVariables but only a value of StructureVariable can be written.

OPC UA PerNet Gateway has been deployed in a few projects requiring a high reliability of the gateway. It has been carefully verified before its deployment and it's continuously monitored since then. An application with 29 objects and 11,709 variables and 100 ms synchronizing period uses 8.7 MB of RAM. At this moment it runs continuously for two years with no serious issue.

4.3 Keysight OPC UA Gateway

Keysight third-party proprietary middleware is a system used for data collection from Keysight measuring units. The system can be operated by its own internal communication protocol, similar to the OPC DA communication. This protocol is built on COM (Component Object Model) technology and allows clients to browse groups and points, read and write data of points and more. Points and groups are unique in the system. A point belongs always to one group. Groups and points contained in the system cannot be easily redefined without a restart of whole system. The system can be distributed on more physical machines but each of them can be used as an entry point and has access to points of other machines.

Keysight OPC UA Gateway has been built with browse, read and write functions. However, the write access has to be allowed explicitly for each group in the gateway's configuration file. The gateway browses all groups and their points only once at the start and browses only groups which are specified in the configuration file. It creates a GroupFolder for each group and a KeyVariable node for each point. GroupFolders are aggregated by the well-known node Objects in the Address Space. KeyVariables are components of their GroupFolders.

The gateway reads data of all points of a group at once but writes data of each point separately. Values of points are specified by triplet VTQ (Value, Time, Quality). Time and quality conversion functions had been specified for data conversion between Keysight middleware and OPC UA. Data type of Keysight values is always a float.

This OPC UA gateway has been deployed in a solution with Keysight middleware and a set of 34980A Mutifuntion Switch/Measure Mainframes and Modules. An application with 8 objects and 178 variables and 100 ms synchronizing period uses 9.3 MB of RAM and runs continuously for 6 months with no acknowledged error.

5. CONCLUSION

The article presents an original generalized methodology for designing of OPC UA gateways for various data transferring communications. The presented methodology provides a quick and easy way to implement a reliable and efficient OPC UA gateway. The methodology is intended to provide (in a form of gateway) an OPC UA interface to devices that uses less popular or internal data transferring communications. It can be availed mainly by small or medium size companies by adding an OPC UA interface to their products. The Gateway can provide all basic OPC UA services; however, optimizations of the network traffic make the Gateway incapable of some advanced functions, like historical data access.

As a proof of concept, three gateway have been implemented using this methodology. Each of them bridged a communication with different behaviour and data structure. These implementations has been described in previous chapters. All three gateways have been successfully deployed in long-term monitored projects and are observed since then. No fatal problem have been acknowledged so far. Implemented gateways seems to be reliable and efficient enough to be used in industry. In conclusion, the presented methodology seems to fulfil its purpose perfectly.

ACKNOWLEDGEMENTS

Authors would like to thank ERDF project "Research and Development of Intelligent Components of Advanced Technologies for Pilsen Metropolitan Area (InteCom)" No. CZ.02.1.01/0.0/0.0/17_048/0007267, Project No. LO1506 (PUNTIS) of MEYS and Grant No. FV10427 of the Ministry of Industry and Trade.

REFERENCES

Cavalieri, S. and Chiacchio, F. (2013). Analysis of opc ua performances. *Computer Standards & Interfaces*, 36(1), 165 – 177. doi:https://doi.org/10.1016/j.csi.2013.06.004. URL `http://www.sciencedirect.com/science/article/pii/S0920548913000640`.

Cho, H. and Jeong, J. (2018). Implementation and performance analysis of power and cost-reduced opc ua gateway for industrial iot platforms. In *2018 28th International Telecommunication Networks and Applications Conference (ITNAC)*, 1–3. doi:10.1109/ATNAC.2018.8615377.

OPC Foundation (2018). *OPC Unified Architecture Specification*, 1.04 edition. URL `https://opcfoundation.org/developer-tools/specifications-unified-architecture`.

Post, O., Seppälä, J., and Koivisto, H. (2009). Certificate based security at device level of automation system. *IFAC Proceedings Volumes*, 42(21), 120 – 124. doi:https://doi.org/10.3182/20091006-3-ES-4010.00023. URL `http://www.sciencedirect.com/science/article/pii/S1474667016323527`. 4th IFAC Workshop on Discrete-Event System Design.

REX Controls s.r.o. (2019). *OPC UA server for REXYGEN - User Guide.* Pilsen, Czech Republic, 2.50.9 edition. URL `https://www.rexygen.com/doc/PDF/ENGLISH/RexOpcUa_ENG.pdf`. Accessed: 2019-05-21.

Zezulka, F., Marcon, P., Bradac, Z., Arm, J., Benesl, T., and Vesely, I. (2018). Communication systems for industry 4.0 and the iiot. *IFAC-PapersOnLine*, 51(6), 150 – 155. doi:https://doi.org/10.1016/j.ifacol.2018.07.145. URL `http://www.sciencedirect.com/science/article/pii/S2405896318308899`. 15th IFAC Conference on Programmable Devices and Embedded Systems PDeS 2018.