ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV026

INTEGRATING OPC UA DEVICES IN EPICS

R. Lange†, ITER Organization, 13067 St. Paul lez Durance, France
R. A. Elliot, K. Vestin, European Spallation Source, 221 00 Lund, Sweden
B. Kuner, C. Winkler, Helmholtz-Zentrum Berlin, 14109 Berlin, Germany
D. Zimoch, Paul-Scherrer-Institut, 5232 Villigen PSI, Switzerland

Abstract

OPC Unified Architecture (OPC UA) is an open platform independent communication architecture for industrial automation developed by the OPC Foundation. Its key characteristics include a rich service-oriented architecture, enhanced security functionality and an integral information model, allowing to map complex data into an OPC UA namespace.

With its increasing popularity in the industrial world, OPC UA is an excellent strategic choice for integrating a wealth of different COTS devices and controllers into an existing control system infrastructure. The security functions extend its application to larger networks and across firewalls, while the support of user-defined data structures and fully symbolic addressing ensure flexibility, separation of concerns and robustness in the user interfaces.

In an international collaboration, a generic OPC UA support for the EPICS control system toolkit has been developed. It is used in operation at several facilities, integrating a variety of commercial controllers and systems. We describe design and implementation approach, discuss use cases and software quality aspects, report performance and present a roadmap of the next development steps.

INTRODUCTION

Open Platform Communications Unified Architecture (OPC UA) is an open standard architecture intended to improve and expand interoperability in the Industrial Automation industry. It is a machine-to-machine communication protocol for industrial automation developed by the OPC Foundation [1] and released in 2008.

While its predecessor, OLE for Process Control (OPC; re-branded as OPC Classic) was developed by Microsoft and used a transport layer based on proprietary Microsoft software (e.g., OLE, DCOM), OPC UA focuses on platform independence and uses well-known open standards like TCP and TLS.

Over the last years, OPC UA has become increasingly popular in the world of Industrial Automation. Across all vendors, framework and device types, it is a key word for interoperability and integration, boosted by the OPC Foundation's certification program that ensures high levels of compliance.

OPC UA ARCHITECTURE

Features

OPC UA integrates all functionality of the OPC Classic specifications into a single extensible framework, providing (see [1]):

- Functional equivalence: all OPC Classic specifications are mapped to Unified Architecture.
- Platform independence: hardware and operating system portability covers everything from embedded systems to cloud-based infrastructure.
- Security: encryption, signing, authentication and auditing allow messages to be transmitted securely with verifiable integrity.
- Extensibility: new features can be added without affecting existing applications.
- Information modelling: a framework helps defining complex information.

Core Concepts

The client-server communication model of OPC UA is layered: on top of a *transport*, which can be secured, the client opens a *session* to a server. Within that session, every piece of information is an *item* that can be written or read. Items are uniquely identified by a *nodeID*, consisting of a *namespace* number and an *identifier*, which can be numerical or a name string. To overcome the disadvantage of having to look up nodes by their name with every access, the client can *register* nodes with the server, to allow optimizing their name resolution.

Each item has a typed value, which can be of a basic data type, an array thereof, a union, or a structure consisting of multiple named elements, which are themselves of basic type, arrays, unions or structures.

Opening *subscriptions* in the session allows monitoring items for changes in their values. The subscription has a configurable *publishing period*, but any monitored item in the subscription can use a different *sampling period* for updating the value from the underlying device. For values that are sampled faster than their subscription is published, data loss can be avoided by defining a server-side queue.

OPC UA *methods* are basically remote procedure calls. They allow the client to send parameters (which can be of structured data types) and can return results to the client. All handshake and synchronization is part of the protocol specification and done within the client and server libraries.

† ralph.lange@iter.org

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV026

INTEGRATION OF CONTROLLERS

In the existing approaches to integrate industrial controllers in control systems, a number of issues can be identified that can benefit from the application of OPC UA.

Industry Standard

Integration of industrial controllers in control systems is a field of mostly proprietary protocols, which are usually vendor and sometimes device series specific. Setting up the communication with a new controller from a different vendor means switching to (or developing) a new driver, dealing with the issues and subtleties of a new protocol, gathering knowledge with a very narrow scope. For facilities with a long life time and in-kind contributions, the accumulation of driver software eventually induces a heavy maintenance burden.

Using an industry standard greatly reduces the amount of software related to the integration. Knowledge of the protocol is wide spread. Independent certification lowers the risk of particular devices behaving unexpectedly.

Symbolic Addressing

In some of the controller communication protocols, low level addressing is done through byte offsets in data blocks inside the controller, e.g. for the Siemens S7 communication [2, 3] and the TCP data block send/receive [4]. Such offset addressing is brittle and susceptible to misalignments between the client and the controller. Several configuration approaches include the generation of both the data blocks on the controller and the configuration of the client as well as strict version tagging and checking to reduce the risk of mismatch [5, 6].

Symbolic addressing makes the client configuration much more obvious and handles modification of the controller software in a robust fashion. Controller and client development workflows can be cleanly separated and have a clear interface.

Structured Data

Modern industrial controller programming uses object-like blocks that are repeated when systems are scaled up. Such programming techniques inherently produce hierarchical data structures on the controller. Mapping these objects into a flat structure for the controller communication (e.g., for the send/receive driver [4]) needs additional programming on the controller that adds complexity, needs development resources and uses run time resources on the controller.

Using OPC UA, the controller structures are directly mapped into the OPC UA namespace. No additional programming is needed on the controller. The client configuration directly uses the server-side hierarchy.

Remote Procedure Calls

Existing communication protocols are usually data-centric – they cover reading and writing data from/to the controller memory. Many control systems integrations have a concept of commands, which needs to be implemented on top of the data-centric interface. Such implementations require additional effort and programming on both the client and the controller for the synchronization (handshake).

Industrial controllers map the OPC UA *methods* feature (see below) to the execution of a function block, including the transmission of parameters and return values. This allows for a very simple yet rich and powerful implementation of commands.

Security

Specific applications, e.g. in the context of medical accelerators and for communications with safety and interlock systems, require or benefit from using the security features of OPC UA.

Secure OPC UA communication also allows running connections over insecure or open networks, e.g. wireless or remote connections.

Embedded OPC UA Servers

Some popular standard PLCs, like the Siemens S7 1200 and 1500 series, can run OPC UA servers embedded inside the controller. This allows for a slim and simple integration architecture, avoiding additional layers of hard- and software between controller and integration.

EPICS DEVICE SUPPORT

From the beginning of the project, the goal was to provide a solution completely based on free and open software.

However, based on an evaluation of existing OPC UA clients, the C++ Based OPC UA Client SDK by Unified Automation [7] was selected as the first client library to be supported. Other candidates were disregarded because of incomplete implementation of the OPC UA specification, low robustness in failure scenarios or poor consistency of their API with EPICS concepts.

Consequently, the OPC UA Device Support module is also written in C++, using an architecture that supports adding other low level client libraries. A second implementation is currently being added to the project, based on the client library of the open62541 [8] project.

Layered Structure

The interface to the EPICS Database supports all applicable record and data types from EPICS Base. In addition to the configuration of sessions and subscriptions, a set of tracing and debugging functions is available to be used from the EPICS iocShell.

Below that, a generic layer of interfaces (C++ virtual base classes) follows the core definitions of OPC UA: session, subscription, item and data element (implementing one value inside a structure) are interfaces to their respective implementations by the client library. A set of helper classes (implemented as templates) covers functionalities that all client libraries need to provide.

The lower layer provides implementation and adaptation of the generic APIs to the specifically used client library. In the case of the Unified Automation client, this part con-

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV026

tains a pretty straightforward mapping to the SDK's interfaces. The open62541 based implementation adds a fair amount of complexity in this layer, wrapping around the simpler C APIs of the open62541 client.

Recently, a second implementation of an EPICS Device Support for the open62541 client has been announced [9], suggesting that this client in its current state is a viable alternative.

Robust Support for Structured Data

Structured data needs to be well supported, as the Siemens S7 embedded OPC UA server (one of the main use cases) performs much better when using structured data compared to addressing every element as a separate OPC UA item.

Interfacing structured data to the EPICS Database pushes for a separation between the item with its configuration parameters and the elements (values) of the structure that are each connected to a different EPICS record. This is reflected in the implementation: A single item record (an additional EPICS record type) carrying the item related configuration connects to the item object below. Many regular EPICS records carrying the single value related information and their address relative to the structure root connect each to one data element object below. The addressing information from the data element records is used to create a tree of data element objects that represents the expected structure. As the OPC UA server may change its internal structure between sessions, this IOC side tree must be mapped against the structure that is actually found on the server every time a session is established or re-established.

Writeable data structures can be handled in two modes: When many elements of a structure are changed in one moment (e.g., when loading back snapshot data), OPC UA sending can be triggered explicitly by writing to a field of the item record. When only single elements of a structure are written at a time (e.g., while operating), OPC UA sending can be triggered automatically whenever a data element is written to.

SOFTWARE QA

Static Code Analysis

Static code analysis is being performed on the module as part of the Continuous Integration setups, making sure that the code is consistently showing high quality ratings. The public service Codacy is used for static analysis of the code in the public upstream repository [10] in addition to ITER's local installation of SonarQube [11].

Unit Testing

Unit tests have been introduced for the helper classes in the generic layer. Coverage, however, is still very limited. Unit tests for the lower level code would be highly desirable, but will need a noticeable investment. A possible approach would be using a mock of the client library APIs with GoogleTest [12] to allow separate testing of the integration layer.

End-to-End Testing

An end-to-end test suite was developed to provide confidence that the device support module functions as expected, in terms of correctness and performance. It also serves to test for any regression that may be introduced, as the tests can be triggered to automatically run for all new commits to the source repository.

The test setup comprises of a software OPC UA server, implemented using open62541 [8], two IOCs, and a Python-based test suite using pytest [13]. In order to communicate with both an EPICS IOC and the OPC UA software server directly, both the PyEpics [14] and opcua [15] Python modules are used. Running of the software server and IOC is handled during test execution by way of test setup and teardown methods.

The OPC UA test server hosts a set of items, covering all supported OPC UA data types. *Connection tests* validate the ability of the EPICS module to connect, reconnect and disconnect from the OPC UA server under a variety of conditions. *Variable tests* verify that that EPICS module correctly translates variable values to/from the types used on the OPC UA server. *Performance tests* are used to characterize the performance of the EPICS module in terms of execution time and memory consumption. *Negative tests* ensure that the EPICS module is able to handle error cases and incorrect inputs gracefully.

The test suite is currently in active use at the ESS facility to provide regression testing under the localized EPICS build system, ensuring that any changes to EPICS environment or OPC UA module have no detrimental impact on the relevant devices in the control system.

A second set of automated tests based on the same framework has been defined for the ABB Power SCADA integration at the ESS to allow efficient regression testing on the final application level as well as on the device support level.

Both test suites are configured to be executed as part of the ESS build pipeline for any new merges against the device support or application modules.

A COLLABORATIVE EFFORT

In 2016, B. Kuner at the Helmholtz-Zentrum Berlin evaluated available public and commercial OPC UA client libraries and settled on the C++ Based OPC UA Client SDK by Unified Automation [7]. He created a prototype of the EPICS Device Support [16], robust enough to be used in production at BESSY II and ITER for several years.

After performance measurements using the prototype at ITER were showing solid results, R. Lange started the full implementation [17] in 2018, based on the design outlined above. Other institutes were doing evaluations of the module while it was being developed.

In 2020, R. A. Elliot and K. Vestin began working on the end-to-end test suite described above, which will be merged to become part of the main project in the next version.

In 2021, C. Winkler was helping to finalize and test the Windows build of the Device Support. D. Zimoch and

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV026

C. Winkler started the integration of the open62541 client library, which is still ongoing and is likely be added to the main project soon.

CURRENT STATUS AND ROADMAP

The EPICS OPC UA Device Support module is stable, mature and robust. It is used by several institutes against different OPC UA servers in productional setups. Table 1 shows a list of currently known applications of the module and their status.

Table 1: Existing Users and Applications (incomplete)

Facility	OPC UA Server	Status
ASIPP	LabVIEW	production
	PLC Siemens S7-1500	production
Australian Synchrotron	PLC Siemens S7-1500F	near production
BESSY II @HZB	PLC Siemens S7-1500	production
	Phoenix Contact	production
	Softing uaGate	production
CHIMERA @CCFE	PLC Siemens S7-1500	development
	LabVIEW	development
ESS	PLC Siemens S7-1500F	production
	ABB Power SCADA	near production
	Siemens DESIGO	development
Fermilab	Kepware KEPServerEX	testing
	PLC Siemens S7-400	development
IPR	PLC Siemens S7-1500	testing
ITER	PLC Siemens S7-1500	production
	Siemens WinCC OA	production
	PCVue	production
KATRIN @KIT [18]	LabVIEW	prototyping
PSI	PLC Siemens S7-1500	development
Varian ProBeam[1]	PLC Siemens S7	production
	PLC Beckhoff	production

A requirement specification exists and is kept up-to-date [19]. The remaining feature from this document that needs to be implemented is the support for OPC UA methods – this effort has been started.

Comprehensive user level documentation still needs to be added. The existing "cheat sheet" documentation is sparse.

The unit tests and the end-to-end test suite need to be extended and completed to achieve a better level of test coverage.

[1] On Windows and VxWorks platforms.

REFERENCES

[1] OPC Foundation, https: //opcfoundation.org

[2] What properties, advantages and special features does the S7 protocol offer?, https://support.industry.siemens.com/cs/document/26483647/what-properties-advantages-and-special-features-does-the-s7-protocol-offer-

[3] S. Marsching, "A New EPICS Device Support for S7 PLCs", in *Proc. 14th Int. Conf on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2013)*, San Francisco, CA, USA, Oct. 2013, paper THPPC027, pp. 1147-1149.

[4] EPICS S7PLC Driver, http://epics.web.psi.ch/software/s7plc

[5] S. Pande *et al.*, "CODAC Standardisation of PLC Communication", in *Proc. 14th Int. Conf on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2013)*, San Francisco, CA, USA, 2013, paper THPPC004, pp. 1097-1099.

[6] G. Ulm *et al.*, "PLC Factory: Automating Routine Tasks in Large-Scale PLC Software Development", in *Proc. 16th Int. Conf on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2017)*, Barcelona, Spain, Oct. 2017, paper TUPHA046, pp. 495-500. doi:10.18429/JACoW-ICALEPCS2017-TUPHA046

[7] Unified Automation, C++ Based OPC UA Client SDK, https://www.unified-automation.com/products/client-sdk/c-ua-client-sdk.html

[8] open62541 project, https://open62541.org

[9] S. Marsching, "Integrating OPC UA Devices into EPICS Using the Open62541 Library", presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2021), Shanghai, China, Oct. 2021, paper TUBR05, this conference.

[10] Codacy page of the OPC UA module, https://app.codacy.com/gh/epics-modules/opcua/dashboard

[11] SonarQube, https://www.sonarqube.org

[12] GoogleTest/GoogleMock, https://google.github.io/googletest

[13] pytest, https://docs.pytest.org

[14] EPICS Channel Access for Python, https://pypi.org/project/pyepics

[15] python-opcua project, https://github.com/FreeOpcUa/python-opcua

[16] opcUaUnifiedAutomation project (prototype), https://github.com/bkuner/opcUaUnifiedAutomation

[17] EPICS OPC UA Device Support project, https://github.com/epics-modules/opcua

[18] J. Mostafa *et al.*, "Interfacing EPICS and LabVIEW Using OPC UA for Slow Control Systems", presented at the 18th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS2021), Shanghai, China, Oct. 2021, paper TUPV011, this conference.

[19] EPICS OPC UA Device Support Software Requirements Specification, revision 1.1, https://docs.google.com/document/d/1_NaSPZNGuRRt8m92Nd2NvJ6LrNAN9UkvTSSIpogSGL4