ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP075

OPC UA EPICS BRIDGE

W. D. Duckitt. Stellenbosch University, Stellenbosch, South Africa
J. K. Abraham. iThemba LABS, Cape Town, South Africa

Abstract

OPC UA is a service-orientated communication architecture that supports platform-independent, data exchange between embedded microcontrollers, PLCs or PCs and cloud-based infrastructure. This makes OPC UA ideal for developing manufacturer independent communication to vendor specific PLCs, for example. With this in mind, we present an OPC UA to EPICS bridge that has been containerized with Docker to provide a microservice for communicating between EPICS and OPC UA variables.

INTRODUCTION

Open Platform Communications United Architecture (OPC UA) [1] supports platform-independent, data exchange between embedded microcontrollers, programmable logic controllers (PLCs) or personal computers (PCs).

Of particular interest to us, is the use of OPC UA for systems integration between the Experimental Physics and Industrial Control System (EPICS) [2] and infrastructure PLCs for heating, ventilation, and air conditioning (HVAC), water supply and electrical reticulation systems.

Our experience has shown that infrastructure suppliers typically have very little knowledge of EPICS, as EPICS is mainly used for the control of large scientific experiments. These suppliers, also typically implement their control systems in proprietary technology from PLC manufacturers.

In our opinion, it is left up to the EPICS developer to specify which variables within the PLCs are of importance and should be integrated with EPICS.

With all the major PLC suppliers [3–5] it is possible to install an OPC UA server within the PLC and export and serve these PLC variables over OPC UA. Once available via OPC UA, the systems can then be integrated into EPICS.

Within the EPICS community, OPC UA integration is possible using the EPICS device support module for OPC UA [6, 7]. The device support module, in the initial release, required the integration of a commercial C++ software development kit. This limitation prompted us to investigate open-source alternatives.

With our investigation, we found that it was possible to develop open-source OPC UA clients using Python [8, 9].

With this in mind, we have drawn from our experience in developing containerized EPICS systems in the React-Automation-Studio project [10], and present a system in the next sections which is containerized with Docker [11], deployable as microservices and implemented in Python using the Python SoftIOC project [12] and OPCUA-asyncio [9] modules.

SYSTEM REQUIREMENTS

The goal was to develop a containerized software bridge capable of synchronizing and relaying process variable information between the OPC UA and EPICS domains.

It should be designed purely with open-source libraries and should be containerized with Docker [11] and version controlled as a mono-repository using Git [13].

Furthermore, the configuration mode of entry should be as docker compose files and EPICS record files.

Finally, the system should be sufficiently documented with use cases, examples and an implementation guide.

Each of the goals has been achieved, and the system overview is given below.

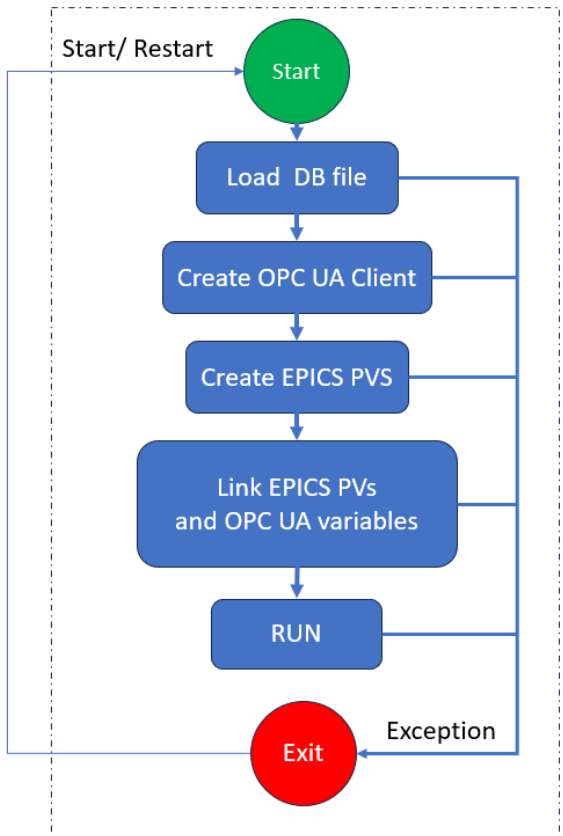

Figure 1: A state-machine diagram of the OPC UA EPICS bridge implementation.

SYSTEM OVERVIEW

A high-level state machine diagram of implementation of the OPC UA EPICS bridge microservice is shown in Fig. 1. The microservice is written in Python [8] and

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP075

containerized with Docker [11]. In particular, we make of the dbtoolspy project [14], Python SoftIOC project [12] and OPCUA-asyncio [9] modules to implement the system. Descriptions of the states of the service are given below:

Start

When the service is started, environment variables are passed to the service. These environment variables determine the uniform resource locator (URL) of the OPC UA server, the subscription rate, the EPICS record file to use, and whether to connect securly to the server.

Load DB File

Thereafter, we use dbtoolspy [14] to load EPICS records that describe the relationship between the OPC UA and EPICS variables. The dbtoolspy module conveniently loads the record information and exposes this as a dictionary in Python.

Create OPC UA Client

We make use of the Python Opcua-asyncio open source project [9] to establish a client connection to a OPC UA server. We use the information loaded in the proceeding step to configure the subscriptions to each of the OPC UA variables. The EPICS DTYP, INP and OUT field is used to make the correct subscription type to each of the variables.

As mentioned previously, the URL of the server and the subscription rate are passed to the service as environment variables.

Create EPICS PVs

We use the Python SoftIOC project [12] to establish each of the EPICS process variables (PVs) for the EPICS input output controller (IOC).

We use the information loaded in the Load DB file step to configure each of the EPICS PVs.

Link EPICS PVs and OPC UA Variables

Finally, we link between the OPC UA Client variables and the EPICS IOC PV's data change callbacks based on the information loaded in the Load DB file step. The callbacks perform type checking, and type cast the new values between the data types used internally in the OPC UA and EPICS.

Run

If no exceptions occur during the loading of previous steps, then the bridge is kept alive indefinitely. If an exception occurs, for example, if the PLC is rebooted, a connectivity exception will be thrown and the bridge service will exit.

Exit

When the microservice is stopped or an exception occurs, the system will exit. All the listening EPICS clients will therefore get a channel access (CA) exception notification.

If the microservice is configured to restart, then connectivity will resume on the EPICS CA when the new instance of the bridge establishes connection.

CONFIGURATION AND DEPLOYMENT

The system is designed to be orchestrated with Docker Compose [15], although any other orchestration system such as Kubernetes [16] can be used by porting the configuration file. An example configuration file is shown in Fig. 2.

The configuration file has numerous standard docker parameters, importantly, the restart mode is set to: always. This will make sure the microservice repeatedly tries to establish connection to an OPC UA server if an exception occurs. The network mode is also set to: host.

This enables the EPICS IOC to be exposed on the host computers subnet. The name of the client, URL of the server, the subscription rate, and a parameter that defines whether to connect securely to the OPC UA server are set in the configuration file as environment variables.

Figure 3 shows how the system may be deployed with multiple EPICS clients communicating to two PLCs though two OPC UA EPICS bridges. In this case, the configuration file would be altered to contained two microservices each connecting independently to a PLC. Furthermore, different EPICS record files, describing each of the PLCs are loaded for each microservice.

An example of a EPICS db file depicting a binary in (BI) and binary out (BO) variable, which are compatible with OPC UA Boolean type, is shown in Fig. 4.

From an EPICS developer point of view, the interface should be very familiar. The INP and OUT fields contain a string that describes OPC UA namespace (ns) and the connection to the OPC UA variables. The other fields are standard EPICS record fields and values, except for the new EPICS DTYP field values we have introduced.

We use them to perform type conversion between the OPC UA data types and the EPICS PV equivalent. The full compatibility of the OPC UA types and EPICS records are given in Table 1.

In the examples in Fig. 4, the EPICS PVs, OPCUA-Python:Bi and OPCUA-Python:Bo, connects to the OPC UA variables named GVL.BiBool and GVL.BoBool respectively.

SECURITY

The Opcua-asyncio project provides various levels of security and authentication, such as: client certificate based authentication and transport layer security. The different modes are enabled via environment variables and example docker compose configuaration files for connecting to secure and unsecure servers are provided. The Git repository also provides scripts for generating server and client certificates.

COMPATIBILITY

All the standard PLC data types shown in Table 1, have been verified to work with the OPC UA EPICS bridge. For example, Table 1 indicates that a standard PLC BOOL datatype is served as a boolean in the OPC UA domain and can be accessed as a EPICS binary in (BI) or binary out (BO) variable using the OPC UA bridge. Similarly, a PLC

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP075

```
version: '3.2'
services:
  opcuapepicsbridge:
    build:
      context: ./
      dockerfile: docker/opcuaEpicsBridge/Dockerfile
    restart: always
    network_mode: host
    tty: true
    stdin_open: true
    environment:
      - name=OpcuaTest1
      - url=opc.tcp://0.0.0.0:4840
      - subscriptionRate=100
      - secure=False
    command: "/bin/sh -c 'sleep 5; python bridge.py;'"
    volumes:
      - ./certificates:/certificates/
      - ./db/test.db:/bridge/bridge.db
```

Figure 2: An example Docker Compose configuration file.

Table 1: Compatibility Between OPC UA Data Types, PLC Data Types and EPICS Records and the OPC UA EPICS Bridge DTYP

OPC UA Data Type	PLC Data Type	New OPC UA Bridge EPICS DTYP	Compatable EPICS Records					
			AI	AO	BI	BO	STRINGIN	STRINGOUT
Boolean	BOOL	OPCUA_Boolean			Y	Y		
SByte	SINT	OPCUA_SByte	Y	Y				
Byte	USINT	OPCUA_Byte	Y	Y				
Int16	INT	OPCUA_Int16	Y	Y				
Int32	DINT	OPCUA_Int32	Y	Y				
String	STRING	OPCUA_String					Y	Y
Float	REAL	OPCUA_Float	Y	Y				
Double	LREAL	OPCUA_Double	Y	Y				
UInt16	UINT	OPCUA_UInt16	Y	Y				
UInt32	UDINT	OPCUA_UInt32	Y	Y				
Int64	LINT	OPCUA_UInt64	Y	Y				
UInt64	ULINT	OPCUA_UInt64	Y	Y				
DateTime	DT	OPCUA_DateTime					Y	

REAL datatype is served as a 32-bit float in the OPC UA domain is compatible with the EPICS analog in (AI) and analog out (AO) process variables.

The Git repository [17] for this project provides a platform independent Python based OPC UA server, that serve variables for all the standard OPC UA data types for testing purposes. The repository also contains the corresponding docker compose file that is used for testing purposes. A React-Automation-Studio (RAS) project of which a screenshot is shown in Fig.5 is also available. This RAS project provides for quick and easy testing of test system and will be used for future extensions. The system has also been verified on a Beckhoff C6015 Industrial PC running Twincat using the TF6100 OPC UA server [18].

CONCLUSION

An OPC UA to EPICS bridge that has been containerized with Docker to provide a microservice for communicating between EPICS and OPC UA variables been designed. The system supports all the standard OPC UA data types.

We urge the EPICS community to test and evaluate the system and to provide feedback via the React-Automation-Studio discussion group [19].

REFERENCES

[1] OPC UA, https://opcfoundation.org/about/opc-technologies/opc-ua/

[2] EPICS, https://epics.anl.gov/

[3] SIMATIC controller - Industrial Automation Systems, https://www.siemens.com/global/en/products/automation/systems/industrial/plc.html

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUPDP075

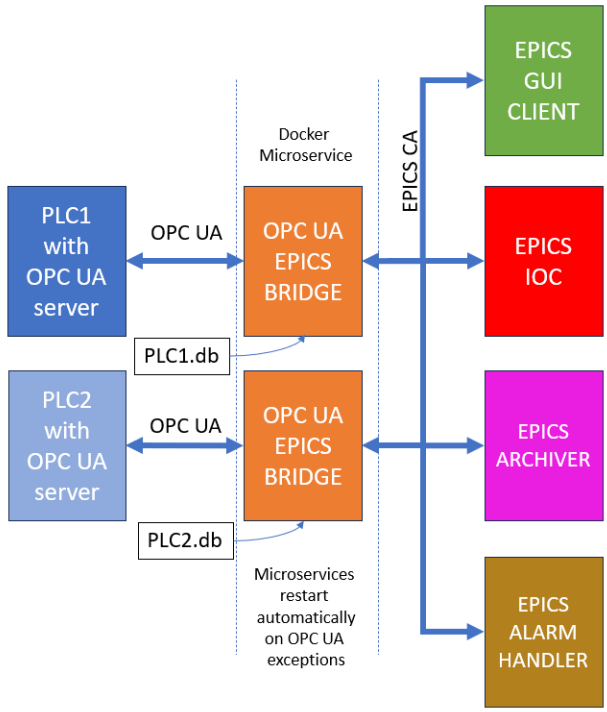

Figure 3: A diagram illustrating example deployment of two OPC UA EPICS bridges that communicate to two PLCs and multiple EPICS clients.

```
record(bi, "OPCUA-Python:Bi")
{
        field(DTYP,"OPCUA_Boolean")
        field(INP, "ns=4;s=GVL.BiBool")
        field(DESC, "BI")
        field(ZNAM, "Off")
        field(ONAM, "On")
}

record(bo, "OPCUA-Python:Bo")
{
        field(DTYP,"OPCUA_Boolean")
        field(OUT, "ns=4;s=GVL.BoBool")
        field(DESC, "BO")
        field(ZNAM, "Off")
        field(ONAM, "On")
}
```

Figure 4: Example EPICS records.

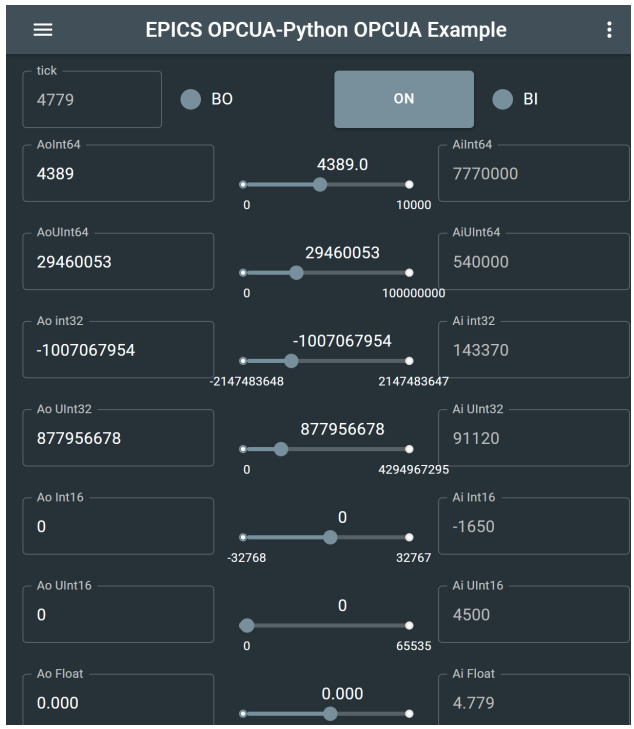

Figure 5: The React-Automation-Studio testing user interface.

[4] Automation | Open, PC-based control technology | Beckhoff, https://www.beckhoff.com/en-za/products/automation/

[5] Programmable Logic Controllers (PLC) | OMRON, https://industrial.omron.co.za/en/products/programmable-logic-controllers

[6] R. Lange, R. A. Elliot, B. Kuner, K. Vestin, C. Winkler, and D. Zimoch, "Integrating OPC UA Devices in EPICS", in *Proc. ICALEPCS'21*, Shanghai, China, 2022, pp. 184–187. doi:10.18429/JACoW-ICALEPCS2021-MOPV026

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

[8] Python, https://www.python.org/

[9] Opcua-Asyncio, https://github.com/FreeOpcUa/opcua-asyncio

[10] W. Duckitt and J. Abraham, "React Automation Studio: A New Face to Control Large Scientific Equipment", in *Proc. Cyclotrons'19*, Cape Town, South Africa, 2020, pp. 285–288. doi:10.18429/JACoW-Cyclotrons2019-THA03

[11] Docker, https://www.docker.com/

[12] pythonSoftIOC, https://dls-controls.github.io/pythonSoftIOC/master/index.html

[13] Git, https://git-scm.com/

[14] dbtoolspy: Python Module to Read EPICS Database, https://github.com/paulscherrerinstitute/dbtoolspy

[15] Docker Compose, https://docs.docker.com/compose/

[16] Kubernetes, https://kubernetes.io/

[17] React-Automation-Studio/OPCUA-EPICS-BRIDGE, https://github.com/React-Automation-Studio/OPCUA-EPICS-BRIDGE

[18] TwinCAT 3 OPC UA TF6100, https://www.beckhoff.com/en-za/products/automation/twincat/tfxxxx-twincat-3-functions/tf6xxx-connectivity/tf6100.html

[19] React-Automation-Studio · Discussions · GitHub, https://github.com/orgs/React-Automation-Studio/discussions