13th Int. Workshop Emerging Technol. Sci. Facil. Controls PCaPAC2022, Dolní Brežany, Czech Republic JACoW Publishing
ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP04

EPICS TANGO BRIDGE

T. Madej, P. Goryl, M. Nabywaniec, Ł. Żytniak, S2Innovation, Kraków, Poland

Abstract

EPICS (Experimental Physics and Industrial Control System) [1] and Tango Controls [2] are the two most popular control systems for scientific facilities. They both have advantages and disadvantages. Sometimes there is existing software driver supporting integration only with EPICS or Tango. EPICS Tango Bridge is the perfect solution for accessing Tango devices using EPICS Channel Access protocol. Using our bridge, the cost of integration is significantly lower than providing dedicated integration of specific hardware in EPICS.

The bridge provides high reliability and robustness. Test cases created during the development process verify their limitations like the response time of reading one attribute in the bridge with a different number of Process Variables (PVs) or parallel access for multiple EPICS clients and many others.

OVERVIEW

Architecture

EPICS Tango Bridge provides access to Tango servers via the EPICS Access Channel. The architecture of the tool is shown in Fig. 1. We used the PyTango [3] library as the interface of Tango devices servers.

The project is based on the PCASpy library, which provides biding to EPICS Channel Access. The main components of EPICS Tango Bridge are pcaspy's SimpleServer and TangoDriver which are inherited from pcaspy's Driver class.

SimpleServer creates PVs based on the PVDB file and runs IOC (Input Output Controller). The server encapsulates transactions performed by the channel access server.

Driver Class is the base class that connects channel access requests to a real-world data source. The base class implementation simply stores a user-written value and retrieves it on request. TangoDriver, extending Driver class allows to read and write attributes of Tango Device Server.

TangoDriver during the initialization creates multiple Manager objects, which provide access to attributes and commands of specific Tango Device Server using the PyTango DeviceProxy class. There are multiple implementations of the Manager interface according to the specific usage (e.g., polled attributes, commands).

Each Manager object creates its DriverAdapter thanks to whom, write to PV (Process Variables) causes an update of value in Tango Device Server's attribute.

Usage

To run the bridge, we need to pass the corresponding PVDB file, providing a valid mapping between EPICS PV and Tango attributes and commands. That file contains a simple Python dictionary where the keys are the base names of the PVs, and the values are their configurations.

TangoDriver creates multiple managers providing access to Tango Device attributes. Each manager implements the read and writes method. The write method is realized using the write thread. In that thread, the new value is assigned to the Tango Device Server attribute and then to the corresponding PV using the DriverAdapter object. If the Tango attribute is a two-dimensional list, it will be flattened during reading from the PV, i.e. [[1,2], [3,4]] will be read as [1,2,3,4]. During writing to the PV, the Bridge IOC will attempt to restore the correct structure before passing the data to Tango.

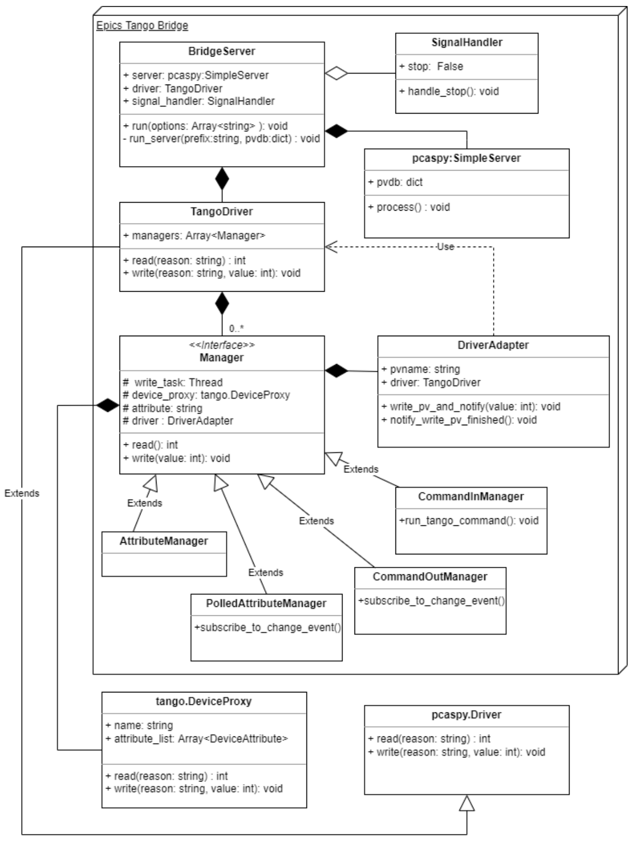
Epics Tango Bridge
BridgeServer
+ server: pcaspy:SimpleServer
+ driver: TangoDriver
+ signal_handler: SignalHandler
+ run(options: Array<string>): void
- run_server(prefix:string, pvdb:dict) : void
SignalHandler
+ stop: False
+ handle_stop(): void
pcaspy:SimpleServer
+ pvdb: dict
+ process() : void
TangoDriver
+ managers: Array<Manager>
+ read(reason: string) : int
+ write(reason: string, value: int): void
Use
0..*
<<Interface>>
Manager
# write_task: Thread
# device_proxy: tango.DeviceProxy
# attribute: string
# driver : DriverAdapter
+ read(): int
+ write(value: int): void
DriverAdapter
+ pvname: string
+ driver: TangoDriver
+ write_pv_and_notify(value: int): void
+ notify_write_pv_finished(): void
Extends
CommandInManager
+run_tango_command(): void
Extends
AttributeManager
Extends
Extends
PolledAttributeManager
+subscribe_to_change_event()
CommandOutManager
+subscribe_to_change_event()
Extends
tango.DeviceProxy
+ name: string
+ attribute_list: Array<DeviceAttribute>
+ read(reason: string) : int
+ write(reason: string, value: int): void
pcaspy.Driver
+ read(reason: string) : int
+ write(reason: string, value: int): void

Figure 1: EPICS Tango Bridge Architecture.

STRESS TEST

To recognize EPICS Tango Bridge's capabilities and limitations, we prepared a set of stress tests.

In this section, we will describe the test scenarios as well as the test setup.

Test Scenarios

To test the performance of EPICS Tango Bridge, we used the PyTest [4] library. We prepared the following test scenarios:

1. Key Performance Indicators (KPIs) test – checking the maximum size of PVDB databases due to the type of attribute (double, spectrum double, string). Tests

check the correctness of EPICS Tango Bridge operation in two ways using EPICS and Tango Controls commands.

2. Read/write time test – checking the time of read and write using PyTango and PyEpics [5] access. As parameters, we pass number of PVs which are run in Bridge IOC and the type of attribute and scan option. During the test, firstly proper PVDB file and example PyTango Device Server are generated. Then Tango Server and Bridge run the test we compute the average time of the read and write.
3. Read time during simultaneous access by multiple clients to one Bridge IOC. In that scenario, one client performs multiple writing actions to PV using the PyEpics caput command. Multiple clients read from the corresponding Tango Device Server attribute. PVDB file contains one Double Scalar attribute. As a parameter for the test, we pass the number of reading clients (default 20), and the number of write and read actions (default 200). As a result, we compute the average time of read action. During the test we validate the correctness of read attributes as the writing client writes a sequence of some numbers – every client has to read the same sequence.

The tests were conducted on a local virtual machine running Ubuntu 18.04 (64bit, x86, HVM) operating system. Following software was installed in the virtual machine: PyTango 9.3.3, PyEpics 3.5.1, PCASpy 0.7.3. Test 3 was conducted using Docker containers.

Test Results

Table 1 shows us how the reading time varies with different sizes of PVDB files performed for the double and a twenty-character string attribute. Each attribute reading was repeated one hundred times and the meantime and standard deviation were calculated for them. We can observe that the reading time is similar for both clients.

Table 1: Read Time for EPICS and Tango Tested in Python Client

Size PVDB	Type Attribute	Average TANGO read time [ms]	Average EPICS read time [ms]
1	double	0.50 ± 0.13	0.58 ± 0.82
10	double	0.50 ± 0.12	0.65 ± 0.85
100	double	0.52 ± 0.14	0.54 ± 0.75
1000	double	0.50 ± 0.14	0.40 ± 0.42
1	String	0.48 ± 0.12	0.52 ± 0.67
10	String	0.51 ± 0.11	0.66 ± 1.01
100	String	0.51 ± 0.13	0.68 ± 0.96
1000	String	0.52 ± 0.14	0.61 ± 0.76

Table 2 shows us an analogous situation as above, in this case, we check the attribute write time also using the Tango Controls and EPICS client python commands. The time of writing the attribute using the EPICS client is longer, despite all this time being satisfactory.

Table 2: Read Time for EPICS and Tango Tested in Python Client

Size PVDB	Type Attribute	Average TANGO write time [ms]	Average EPICS write time [ms]
1	double	0.23 ± 0.14	102 ± 3
10	double	0.24 ± 0.15	103 ± 6
100	double	0.23 ± 0.15	112 ± 11
1000	double	0.15 ± 0.11	112 ± 11
1	String	0.21 ± 0.14	103 ± 2
10	String	0.23 ± 0.13	105 ± 6
100	String	0.06 ± 0.04	112 ± 12
1000	String	0.14 ± 0.09	111 ± 18

During test 3, we ran twenty clients using docker and all of them were doing operations of reading, one client wrote randomly generated values, and we checked the average read time for one of the reading clients and the result was about 0.52 ± 0.24 ms.

CONCLUSION

EPICS TANGO BRIDGE is a tool for accessing Tango Classes via EPICS Channel Access, which has scalar attribute support (normal and polled), command execution, and support for spectrum (one-dimensional array) and images (two-dimensional array). Advantage of the bridge is its ease of use, it is enough to prepare a PVDB file before use, which is a dictionary linking the attribute name with a specific PV structure.

Efficiency and performance are important aspects of control systems. The bridge has no significant impact on the speed of reading, with a bigger number of PVs in the PVDB file or with more clients working on the same server, the time is quite the same and is on the order of a fraction of a millisecond. Only write time using caput in relation to Tango commands is the time is relatively longer.

REFERENCES

[1] EPICS, `https://epics-controls.org`

[2] Tango Controls, `https://www.tango-controls.org`

[3] PyTango, `https://pytango.readthedocs.io/en/stable/`

[4] PyTest, `https://readthedocs.org/projects/pytest/`

[5] PyEpics, `https://github.com/pyepics/pyepics`