ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV019

PVEcho: DESIGN OF A Vista/EPICS BRIDGE FOR THE ISIS CONTROL SYSTEM TRANSITION

K.R.L. Baker[*], I.D. Finch, G.D. Howells, A. Saoulis,
ISIS Neutron and Muon Source, Didcot, United Kingdom
M. Romanovschi, University of Manchester, United Kingdom

Abstract

The migration of the ISIS Accelerator Controls System from Vsystem to EPICS presents a significant challenge and risk to the day-to-day operations of the accelerator. An evaluation of potential options has indicated that the most effective migration method to mitigate against this risk is to make use of a ‘hybrid’ system running Vsystem and EPICS simultaneously. This allows for a phased porting of controls hardware from the existing software to EPICS. This work will outline the prototype Vsystem/EPICS bridge that will facilitate this hybrid operation, referred to as PVEcho. The bridge has been developed in Python, utilising existing communication from Vsystem to an MQTT broker developed as part of a previous project. Docker containers have been used for its development to create a test environment that allows the software to communicate with other active services currently used at ISIS.

INTRODUCTION

The ISIS Neutron and Muon Source [1] has been operated using Vista Control Systems’ commercial product Vsystem [2], colloquially referred to as Vista, since 1998. Recently, a study was undertaken to evaluate its use against that of the Experimental Physics and Industrial Control System [3] (EPICS) at ISIS. The study determined that the control system should be migrated to EPICS [4]. One of the key benefits of the migration will be to promote collaboration with other facilities at Rutherford Appleton Laboratory, such as ISIS Instrumentation [5], Central Laser Facility [6] and Diamond Light Source [7], also using EPICS.

In order to minimise the impact on business-as-usual at the facility, a phased porting of control of hardware is the desired option for the transition. This requires the development of bridging software that will map approximately 33,000 channels that exist in Vsystem to an equivalent EPICS Process Variable (PV). This will allow a progressive migration of the operator control screens in the main control room (MCR) alongside a gradual transition of hardware without interrupting operator control. The current control screens are produced using Vsystem’s Vdraw [8] tool but will be migrated to screens created using CS-Studio Phoebus [9] as part of the transition. Phoebus is the suite of applications that can be used to display screens as well as probe EPICS PVs.

The use of a bridge also elegantly permits some legacy hardware that cannot be converted to EPICS to still be run using Vsystem but using Phoebus control screens. PVEcho will serve as this bridge, responsible for replicating the behaviour of the current control system in EPICS.

PVEcho

Each component of hardware that comprises the accelerator controls system is associated with a group of unique channels through which value changes in the hardware can be communicated. During the migration, we will need to exactly replicate each of these channels that currently operate through Vsystem as an equivalent EPICS PV, including their value, alarm and display configuration. Each Vsystem channel and corresponding mirrored EPICS PV needs to be kept synchronised. This in turn will allow the control system to operate with either the Vsystem channel or the EPICS PV acting as the source of truth.

In the early stages of the transition, the majority of channels will still be operated through Vsystem while long-term EPICS equivalents are being developed. In this case, the source of truth will be the Vista channels and changes in their values and alarm limits will be broadcast from Vista to EPICS, handled by the vistatoepics (v2e) program. Once the control of hardware of specific channels has been ported to EPICS, the source of truth will transfer. Now, changes to the PV will be monitored and transferred from EPICS to Vista via MQTT [10] messages. This is handled by the epicstovista (e2v) program. The vistatoepics and epicstovista applications work in tandem to make up the PVEcho bridge, a visual representation of which can be seen in Fig. 1. The long-form names are used for descriptions of the applications, while the shorthand versions (v2e and e2v) are used for simplicity in the codebase and in diagrams. More details about the individual programs will be given in subsequent sections of this paper.

To track which channels have Vista or EPICS representing their true value, we have created a `PVMonitor` utility class to use in both programs. A list of channels to monitor are defined in a file which the `PVMonitor` regularly checks for changes (in our case once per second). When a change is noted, the monitor determines which channels have been added or removed relative to the file’s previous state, and starts or stops tracking those channels accordingly. This method allows a lot of control and flexibility over the channels to be replicated in EPICS, allowing the transition of individual channels at a time. As the same class is used in both applications, “channel” may be substituted for PVs when referring to epicstovista, where the same approach is used.

[*] k.baker@stfc.ac.uk

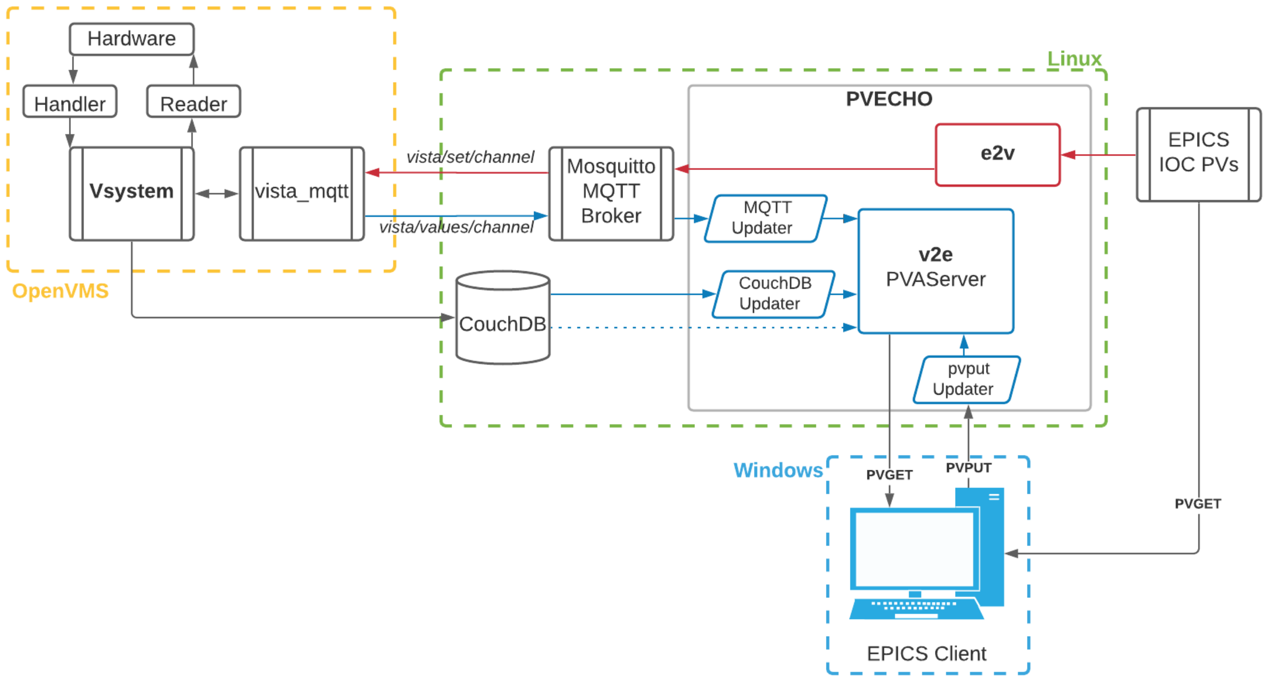

Figure 1: An overview of how PVEcho will integrate into the ISIS Accelerator Control System. VSystem will remain operating on OpenVMS, while newer systems and EPICS will operate on Linux, with the MCR running Phoebus [9] screens through Windows. Arrows in dark blue represent the flow of data from a Vsystem channel through to EPICS (v2e). They also indicate the Updater classes that parse incoming data. Dotted lines indicate an initialization step in the creation of PVs. Arrows in red follow the flow of information from EPICS to Vsystem (e2v). PVEcho interfaces to Vsystem through a Mosquitto MQTT broker [10] and the vista_mqtt service [11].

Both vistatoepics and epicstovista components of PVEcho will leverage the MQTT messaging service linked to Vista, named vista_mqtt, set up as part of a previous project within the ISIS controls group [11]. The vista_mqtt service interfaces to Vsystem through topics associated with each Vsystem channel. When a value in Vista changes, the application (originally developed in Python but being migrated to C++) generates an MQTT message with the updated value and sends it to a read-only "values" topic associated with the channel. If a message is published to the channel's "set" topic, the application triggers a Vsystem set command to change the value of the channel within the control system. Vistatoepics will make use of the former, read-only case, while epicstovista will utilise the writable topics.

By subscribing to the MQTT value topic associated with a Vsystem channel we can monitor changes to its value through the clients and callbacks available in MQTT's python library [12]. The changes are then echoed to EPICS through vistatoepics. Similarly, we track changes to the EPICS PVs through pvapy's [13] `Channel` client and we set the value of its corresponding channel in epicstovista by publishing a message to the channel's set topic in MQTT.

The programs will also exploit the CouchDB [14] database set up by the group to store metadata associated with each Vsystem channel. The database stores information such as the channel type (e.g. integer, float, boolean, string) and display formatting, as well as alarm type and limits if applicable. In the case of vistatoepics, this information is used to define the structure of the PV to be created. In both programs it is also used to define how to parse incoming values to a format appropriate for the system where it will be echoed, based on the channel's type.

VISTA TO EPICS (V2E)

The PVEcho bridge is designed to replicate the entire Vista control system, rather than individual devices that usually constitute a distributed control system. This places a focus on the ability to dynamically add, remove or edit PVs on the server created to host the replica PVs without having to restart the program, which would interrupt operation of the accelerator. Therefore, a PVAServer from the `pvapy` library was favoured over the traditional IOC for the vistatoepics component of PVEcho. On running the vistatoepics program, the configuration that dictates the services to connect to and the necessary files to be read into memory is loaded and parsed. In this case, files include a record of the CouchDB entries containing the metadata for the Vsystem channels, the mappings of reference channels to their setter channels and the list of PVs to monitor. The benefit of using this approach is that we can tailor our configuration file for production or development and testing purposes.

Once all of the relevant classes have been instantiated, we enter the main loop of the program. It continuously checks the monitor file for changes, acts accordingly for the PVs that have been added or removed from the list and then goes on to check for any changes to CouchDB. This continuously runs until the program is interrupted, as illustrated by Fig. 2.

Initialisation

On start-up and initialization of the vistatoepics component of PVEcho, a snapshot of the existing configuration for each of the channels on Vsystem is captured using the ISIS CouchDB database. The channel metadata (such as the display and alarm configuration) associated with each channel in the monitor list is then translated into a format appropriate for EPICS.

Monitoring

Once the PVs have been created on the server, changes to their value and metadata are tracked through three different avenues: MQTT topic messages from Vista, CouchDB updates and EPICS `pvput` commands. Each source produces updates in a different `json` format so a fleet of customised 'Updater' classes are used to manipulate the data from the raw input into a structure appropriate for use with the `pvapy` `PvObject.set()` method which is used to apply the changes to the PV. The PV is then updated on the server so that changes are visible to EPICS clients such as Phoebus [9] control screens.

1. **MQTT**
   As described previously, once a PV has been created, we subscribe to the original channel's corresponding MQTT topic, published from Vsystem, which tracks changes to the PV's value.

2. **CouchDB**
   CouchDB shadows the Vsystem databases for each channel so it acts as a source of truth for the channel metadata that is not accessible through MQTT. To keep PVs consistent with their channel, we track changes to Vsystem database files through alterations to CouchDB using the CouchDB Python client [15] and modify the PV's metadata to match the updated values from Vista.

3. **pvput**
   PV values and metadata may also be changed by the operators through `pvput` commands using Phoebus control screens. As we are using custom classes that inherit from the original `pvapy` `PvObject` class for different channel types, when an update comes through we need to ensure that our object is updated correctly on the server.

Alarm Management

The vistatoepics component of PVEcho is also responsible for mirroring the current alarm management in Vsystem. Implementing the alarms from the outset will allow us to start using EPICS alarm handling and monitoring software throughout the transition, as well as minimise operational changes for the crew. The mapping of range and binary match alarms is relatively straightforward as EPICS defines these alarm types natively. Vsystem also offers the option of integer match and reference alarms [16], which do not have a corresponding native type in EPICS.

An example of where an integer match alarm might exist is a pump that operates in multiple modes. These could include pump operating states such as `0=off, on=1, 2=fault, 3=standby, 4=turbo,` where only state 2 should trigger an alarm.

A reference alarm is a channel where the alarm limits are determined by the value of a different Vsystem channel according to some calculation. The channel that dictates the reference channel's alarm limits is referred to as its "setter" channel. The alarm limits are recalculated for the reference channel whenever the value of the setter channel changes.

In order to replicate Vsystem as closely as possible within EPICS, the logic behind the Vista alarms has been implemented manually within the `Updater` classes. This was primarily possible because of the freedom of control over alarms that the `pvapy` library allows as well as the availability of the alarm definitions (for example type, limits, setter channel) for each channel in CouchDB.

- **Range** With each new update to the value or the metadata, the current value is compared to the most recent alarm limits to determine its alarm state, either MAJOR, MINOR or NO ALARM.
- **Binary Match** The alarm state is set to raise a major alarm if the current value matches that of the defined alarm match value.
- **Integer Match** The alarm state for integer match alarms is determined by comparing the value of the PV with its highAlarmLimit. On initialisation, this and all the other limits are set to be equal to the alarm match value from Vsystem. To circumvent the issue where a user could change the value of the match alarm through one of the other limits (e.g. lowAlarmLimit), a custom function has been devised and implemented in the `PvPutUpdater` (seen in Fig. 1), whereby if the value of any of the alarm limits is changed on a match alarm PV, all of the alarm limits are modified to equal the updated value. We use the same approach to update the severity of the match alarm.
- **Reference** A mapping of reference alarm PVs and their setter channels are stored in memory. Whenever a PV updates, a check is performed to see whether the channel is a setter. If so, all of its reference channels are also updated to reflect the new limits and their alarm state is recalculated with the new limits.

EPICS TO VISTA (E2V)

As previously mentioned, currently the ISIS control screens are created and run using Vista's VDraw tool [8]. The intention during the transition is to operate a hybrid User Interface (UI), transferring some screens to Phoebus/EPICS while others remain in VDraw/VSystem as control of hardware is ported. If a user changes the value of the PV in

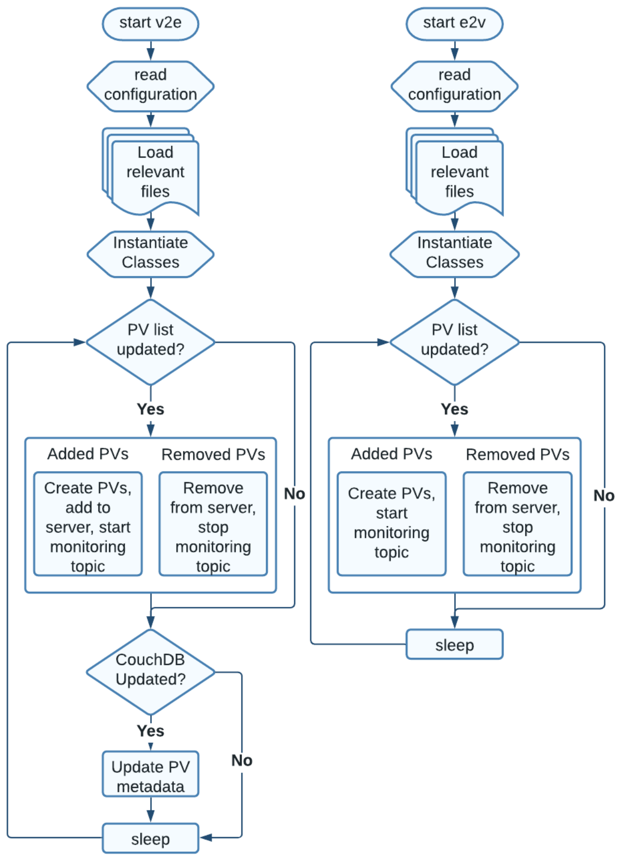

Figure 2: Flowcharts demonstrating the logic behind vistatoepics and epicstovista.

the ISIS MCR through a Phoebus screen where the hardware is still controlled by Vsystem, the change needs to be propagated from the EPICS Client to Vsystem to update the hardware. The same is true in reverse; if hardware is read via an EPICS IOC but the control screen is still operated by a VDraw screen, the PV values will need to be propagated to Vsystem. A `pvapy Channel` connection and callback is used to do this. As the update comes through via the EPICS callback, the values are converted to a JSON format that the Vsystem code on OpenVMS is expecting. The message is then published to the correct setter topic on the MQTT broker, which activates a set command in Vsystem [11]. The main loop of the program follows a similar structure to vistatoepics and can be seen in Fig. 2.

Alarmed channels are monitored in a similar way, but changes to the channel metadata are also tracked to provide updates if the alarm limits are changed through EPICS.

DEVELOPMENT ENVIRONMENT

Environment

The ISIS accelerator control system is also transitioning from OpenVMS to Linux as a base operating system. Therefore, in order to future-proof the design of the bridging software, PVEcho has been developed using Docker [17] environments with a Linux operating system as the base of the containers, to isolate and replicate PVEcho's operating environment. Two separate containers will be used to run vistatoepics and epicstovista simultaneously, while isolating the processes to give greater control over their execution.

Services

Alongside Linux and EPICS, PVEcho requires a connection to a number of other active services that ISIS utilises to monitor and maintain its control system. These include the ISIS MQTT broker [11] and the instance of CouchDB where channel metadata is stored. The use of Docker while developing PVEcho allowed connection to personally managed containers to prevent premature interaction with the live control system, minimising the risk of unintentional changes to existing channels.

Testing

Throughout the development of the software, we have tried to implement software good practice wherever possible. This includes the application of unit and integration testing utilising GitLab's CI/CD facility and the unittest [18] framework in Python. Performance has also been monitored using an in-house virtual logging system [19] to track the CPU and memory use of each container while the software is running.

FUTURE WORK

While functional, work on the current implementation of PVEcho is still ongoing. The accelerators at ISIS will primarily use the newer pvaccess protocol rather than channel access [11]. The current version of PVEcho only implements PVs using the pvaccess protocol, as highlighted by the use of a PVAServer to host the PVs. Future work on the software will incorporate channel access, ensuring that any changes to Vsystem channel values are broadcast to channel access PVs as well as their pvaccess equivalents.

Future developments will also include the addition of delayed alarms that are currently in use with VSystem. Additionally, the PVMonitor class should be modified to allow the bulk addition and removal of PVs from the monitor list as devices containing groups of PVs are migrated, to reduce the risk of user error in the transfer of channel names. Stress testing of the live system should also be completed to ensure that the software can cope with the day-to-day frequency of messages and surges in demand.

CONCLUSION

A prototype of the bridge connecting Vsystem to EPICS has been created to mimic more than 33,000 channels live on the ISIS accelerator control system. PVEcho will allow a hybrid system running both Vsystem and EPICS to be utilised as control of hardware is progressively migrated to EPICS, without incurring the risk of interrupting accelerator operations. The prototype is due to be tested in deployment before the end of the 2021 ISIS long shutdown [20].

REFERENCES

[1] J.W.G. Thomason, "The ISIS Spallation Neutron and Muon Source—The first thirty-three years", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 91, pp. 61–67, Feb 2019. doi:10.1016/j.nima.2018.11.129

[2] Vista Controls Systems Inc., https://www.vista-control.com/.

[3] EPICS Control System, https://epics-controls.org/.

[4] I.D. Finch, "Evaluating VISTA and EPICS With Regard to Future Control Systems Development at ISIS", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 291–292. doi:10.18429/JACoW-ICALEPCS2019-MOPHA042

[5] K.V.L. Baker, F.A. Akeroyd, J.R. Holt, D.P. Keymer, T. Löhnert, C. Moreton-Smith, *et al.*, "IBEX: Beamline Control at ISIS Pulsed Neutron and Muon Source", in *Proc. ICALEPCS'19*, New York, NY, USA, Oct. 2019, pp. 59–64. doi:10.18429/JACoW-ICALEPCS2019-MOCPL01

[6] Central Laser Facility, Rutherford Appleton Laboratory, United Kingdom, https://www.clf.stfc.ac.uk

[7] Diamond Light Source, Rutherford Appleton Laboratory, United Kingdom, https://www.diamond.ac.uk

[8] Vdraw, Vsystem User's Guide v4.3, Vista Controls Systems Inc., Jan 2014

[9] CS Studio Phoebus, https://controlssoftware.sns.ornl.gov/css_phoebus/.

[10] MQTT standard: MQTT Version 3.1.1, OASIS, October 2014, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html

[11] I.D. Finch, A. Saoulis and G.D. Howells, "Controls Data Archiving at the ISIS Neutron and Muon Source for in-depth analysis and ML applications", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, paper WEPV049, this conference.

[12] Eclipse Paho MQTT Python client library, https://pypi.org/project/paho-mqtt/.

[13] pvapy, https://epics.anl.gov/extensions/pvapy/production/index.html

[14] Apache CouchDB, http://couchdb.apache.org/.

[15] CouchDB's python client, https://couchdb-python.readthedocs.io/en/latest/.

[16] Vaccess Concepts, Vsystem User's Guide v4.3, Vista Controls Systems Inc., Jan 2014.

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

[18] python unittest framework, https://docs.python.org/3/library/unittest.html

[19] G.D. Howells and I.D. Finch, "Containerised Control Systems Development at Isis and Potential Use in an Epics System", in *presented at ICALEPCS'21*, Shanghai, China, Oct. 2021, paper WEPV050, this conference.

[20] ISIS Long Shutdown 2021 Update, https://www.isis.stfc.ac.uk/Pages/ShutdownUpdate.aspx.