ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

IMPLEMENTING ODIN AS A CONTROL AND DATA ACQUISITION FRAMEWORK FOR EIGER DETECTORS

G. Yendell, M. Taylor, U. Pedersen, Diamond Light Source, Oxfordshire, UK
T. Nicholls, A. Neaves, STFC/RAL, Chilton, Didcot, Oxon, UK
A. Greer, OSL, Cambridgeshire, UK

Abstract

The increasing data throughput of modern detectors is a growing challenge for back-end data acquisition systems. OdinData provides a scalable framework for data acquisition used by multiple beamlines at Diamond Light Source (DLS). While it can be implemented standalone, OdinControl is used to provide a convenient interface to OdinData. Eiger detectors at DLS were initially integrated into the Odin framework specifically for the data acquisition capability, but the addition of detector control provides a more coherent and easily deployable system. OdinControl provides a generic HTTP API as a single point of control for various devices and applications. Adapters can abstract the low-level control of a detector into a consistent API, making it easier for high-level applications to support different types of detector. This paper sets out the design and development of Odin as a control system agnostic interface to integrate Eiger detectors into EPICS beamline control systems at DLS, as well as the current status of operation.

INTRODUCTION

In an effort to stay on the forefront of scientific research in their respective fields, many beamlines at DLS have chosen to integrate Eiger detectors. Eiger detectors are very extensible, exemplified by the different use cases of the various deployments. Some beamlines require the maximum frame rate possible, while others run at much lower rates of less than 10 Hz and simply benefit from the quality of the data produced. With these varied use cases, a dynamic control and data acquisition framework is required to cope with large data output in some cases, while keeping more basic deployments as simple as possible. The system should only be as complex as is necessary to achieve the given requirements of the beamline.

Effective integration of Eiger detectors into control systems at DLS is enabling beamlines to provide world-leading research opportunities in various science cases and will also provide the scope for experiments that were not feasible before [1].

Eiger Detectors at Diamond Light Source

Eiger detectors have a photon count rate of 5×10^8 s^{-1}mm^{-2}, a pixel size of 75 µm and a read out time of 3 µs, which, amongst many other cutting-edge features, has led to quick uptake on beamlines at DLS. The Eiger comes in a variety of sizes suited to different applications, of which DLS currently makes use of the Eiger2 XE 16M, Eiger1 X and Eiger2 X 4M and Eiger1 X 500K models [2]. Eiger detectors have become commonplace on beamlines at DLS in the last 18 months and more are scheduled to be commissioned in the near future. This presents many challenges, not least the file system load of running multiple instances at the full data rate simultaneously and in some cases with minimal down time between acquisitions. It also presents a challenge in continually commissioning control and data acquisition frameworks for detectors as they arrive at various beamlines, as well as keeping them up to date.

Having to support various detector models might complicate the software required to drive them, due to different demands. Presenting the data rate of different models of Eiger is not trivial, as it is obfuscated by variable bit depths, sustained/burst thresholds and compression ratio depending on signal strength. However, broadly speaking, the maximum data rate for all models is approximately the same and in any case primarily limited by the 40 Gb network link. The choice of model is effectively a decision balancing time-resolution against the collecting area. This makes the software design decisions somewhat simpler; because different models have roughly equivalent performance requirements, the scale of the system to be deployed is mainly defined by the use case of the beamline.

X-ray Imaging and Coherence beamline I13 employs an Eiger1 X 500K to make efficient use of the high coherent flux available at the sample. The major use case is for ultrafast diffraction imaging, primarily ptychography [3]. This is a method of collecting a series of diffraction patterns from a sample, in a grid with overlapping illumination regions, and processing the datasets to create an image. Ptychography allows higher spatial resolution images than using conventional lens imaging. The combination of high flux and the inherent noise resilience of the technique enables scans with exposures of the order of 100 µs, which makes scans at 10 kHz feasible. This pushes the boundaries of trajectory scanning and data acquisition technologies.

Macromolecular Crystallography (MX) beamlines, such as Microfocus MX beamline I04, make the greatest use of Eiger detectors at DLS. The large collection area of the 16M and 4M models is ideal for collecting crystal diffraction data, while the small pixel size enables better sampling of diffraction spot profiles and improved signal-to-noise from smaller diffraction spots. The high frame rates are a particular asset for MX beamlines that invest in automation and high throughput of samples, due to reduced acquisitions times. For these beamlines, Eiger is the natural upgrade from the Pilatus detectors that have been depended on for many years.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

Odin Framework

Odin [4] is a scalable and modular control and data acquisition framework for detectors. It is a combination of a high-throughput C++ data acquisition stack OdinData (OD) with a Python control server OdinControl (OC), providing a balance of performance and convenience. It is primarily designed for the modular, parallel detector systems, such as Excalibur [5] [6] and Percival [7]. This led naturally to the Odin framework being modular in its own design to enable it to be scalable, for performance and to support parallel detectors, while providing a single point of control for integration.

INITIAL EIGER INTEGRATION

An internal HDF5 file writer is included on the provided Eiger Detector Control Unit (DCU), allowing files to be copied off via HTTP, but this sacrifices data throughput for convenience, so it is not used at DLS. Instead, the ZeroMQ (ZMQ) [8] data stream provided by the Dectris SIMPLON API [9] is used. This operation mode compresses the data frames from the detector and wraps the data blob with some header information, exposing a data stream that then needs to be gathered and processed by a listening application. The data throughput on this stream is only limited by the 40 Gbit ethernet link, assuming detector performance and compression can exhaust it. This is a substantial improvement on what is possible with the internal file writer.

When the first Eiger was integrated at DLS, ADEiger [10] was used to provide the control API alongside an OdinData acquisition pipeline. OdinData had already been developed, for use with other high throughput detectors Excalibur and Percival, and the plugin-based design makes it simple to integrate new detectors into the framework, so it was decided to use it with Eiger. This worked well and has been successful on multiple beamlines. It is also, however, a bulky and complicated system to deploy, use and maintain. The main complication is the requirement of two independent areaDetector [11] drivers, ADEiger for detector control and ADOdin for the control of the data acquisition stack. This meant that many parameters have to be synchronised between the two systems. It is too easy for users to accidentally configure the two separate applications in inconsistent ways, which could lead to confusing errors and difficult debugging sessions. To simplify things, the new system described here removes ADEiger and replaces it with OdinControl, integrating Eiger fully into the Odin framework.

The network architecture of Eiger systems used at DLS is shown in Fig. 1. The Odin processes are all run on the Eiger Processing Unit (EPU) in a compute room, while the ADOdin IOC is run on a beamline server.

EIGER ODIN ARCHITECTURE

Functionality in an OC server is primarily implemented by loading a set of Adapters, providing a coherent interface to many individual devices and applications. The server exposes a set of HTTP URIs corresponding to each Adapter's

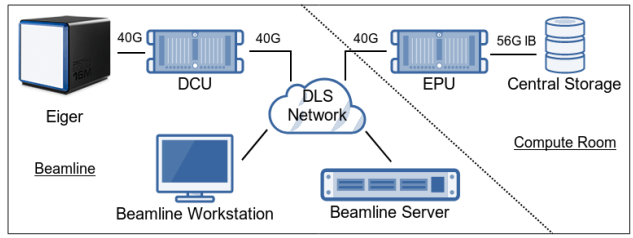

Figure 1: Network architecture of an Eiger system at DLS. DCU: Detector Control Unit provided by Dectris. EPU: DLS Eiger Processing Unit to run Odin processes.

parameters and methods, resulting in a hierarchical parameter tree representing the overall system. The OC API represents multiple instances of an object by presenting parameters as arrays. This allows high level control to request the value of a parameter for each instance in a single request, as well as to send a single value to all instances. It is also possible to index into a parameter array to interact with individual instances where the values do need to be different. OC parameters are intended to be self-describing through the provision of meta data, such as type, limits, units and description. This makes the server, and therefore the applications underlying the Adapters, introspectable by higher level control applications, aiding integration with a wider control system. Conveniently, the meta data in an OC parameter maps intuitively to the SIMPLON API [9] provided by the Eiger, which makes the Eiger Adapter API look very similar to talking directly to the detector.

The architecture of OD is also parallel in its design, but the logic is reversed; there are multiple data acquisition nodes operating in parallel to serve the same data stream, or set of data streams. The core logic of a given node is determined by its perspective of being one of many, which makes the software inherently scalable. The primary intent behind OD is high throughput. To achieve this it is stripped back and simple. Its function is limited both by design and philosophy to follow this principle. Only the bare minimum processing should be carried out in serial with the file writing, so that there is a minimal path to disk. Only processing that is required to make the images useful is allowed. Use cases that generate extra data from the raw images or calculate additional statistics should implement the logic in a processing pipeline on a separate machine or cluster via SWMR [12] live monitoring. Any processing that is required during an acquisition, such as a live view of the data, should be done in a parallel plugin chain. An exception to this might be data reduction processes, which would trade some processing load for reduced file writing load, as well as smaller files. As the data acquisition pipeline is usually I/O bound, this could serve to increase the overall throughput.

The application of this philosophy to the HDF5 file writer is key to the high throughput of the system. The configuration is restricted to the essentials: datatype, compression and chunking. Image datasets are 3D and scalar datasets are 1D; no further dimensionality is supported. Additional datasets

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

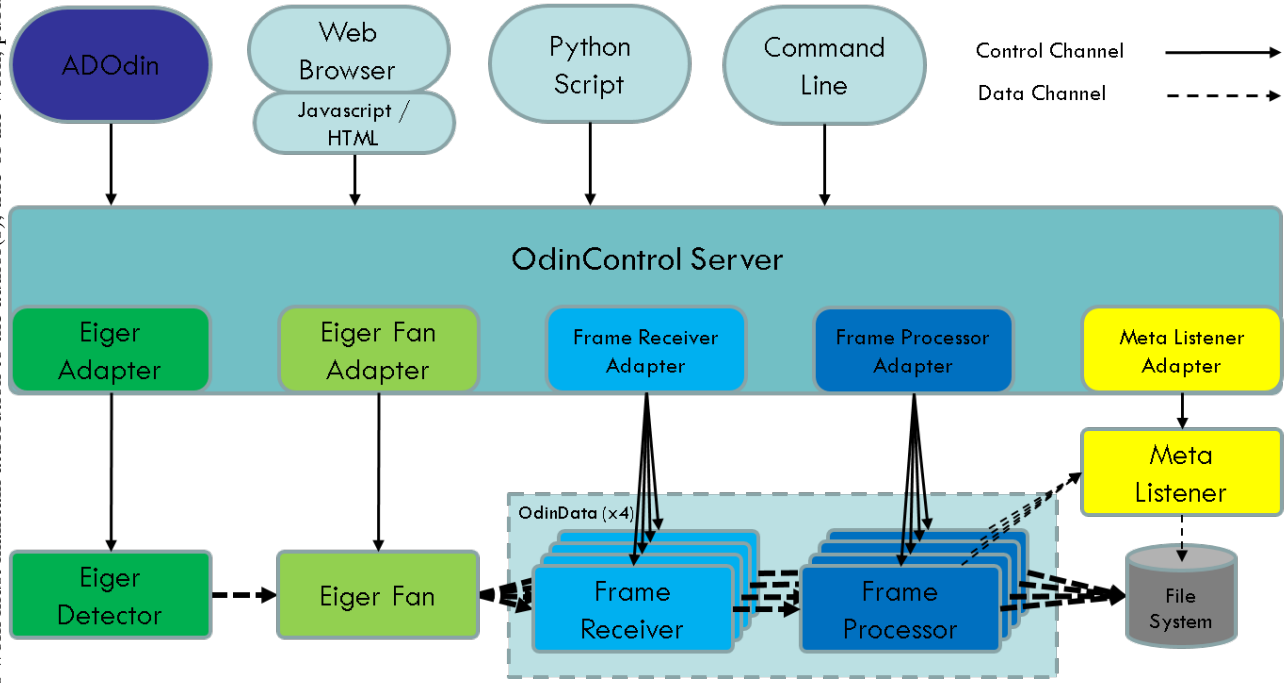

Figure 2: Architecture of a single OdinData node of an Eiger Odin system.

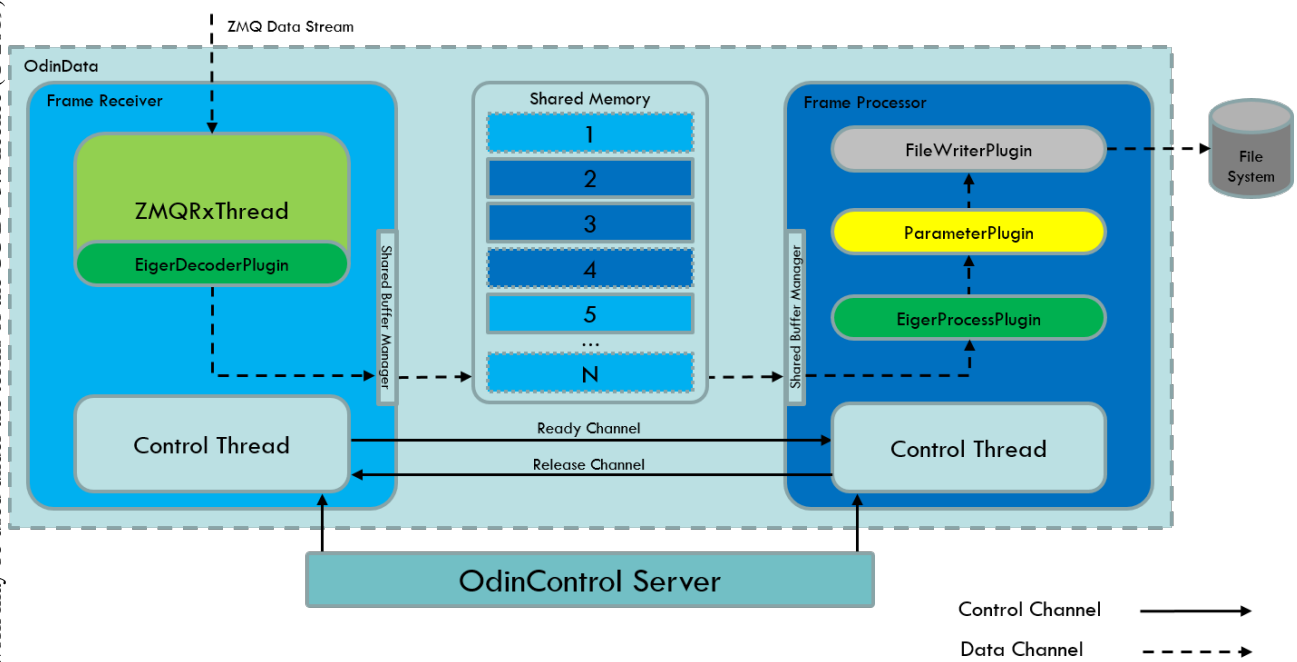

Figure 3: OdinControl Server architecture for a four-node Eiger Odin system.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

are only included in the file if they are required during an acquisition, such as unique identifier (UID) and image sum datasets used for SWMR live monitoring and processing. Anything else is published on a ZMQ socket and written to a separate file if required. This reduces the demand on the main file writer by not having to handle datasets that are not time critical or otherwise not required until after the acquisition is complete.

OC supports control of an OD stack of any size with only minor configuration changes and superficial, but clear, differences in the user facing API. The primary desire of designing OD to be scalable is to allow performance increases if a bottleneck is reached, without overhauling the entire system. However, a secondary benefit is that it also allows systems to be only as complex as is necessary for a beamlines' requirements. It also has the effect that, for the most part, the system looks the same whether it has one OD node or eight.

Data Flow

The problem to overcome in integrating Eiger is that the data stream is not parallel, which OD depends upon. There is a single ZMQ stream from the detector. A single file writer is not sufficient to keep up with the data rate, so we use the EigerFan application to parallelise the Eiger data stream onto a configured number of consumers, to share the workload between multiple OD nodes. Messages are gathered from the detector and forwarded on to the correct consumer in a round-robin fashion. The number of consecutive frames to send to a given consumer, the block size, is configurable.

OD consists of two separate applications: the FrameReceiver (FR), for gathering data from the detector into shared memory as fast as possible, and the FrameProcessor (FP), for labour intensive processing and writing to an HDF5 file. The EigerDecoderPlugin in the FR gathers a set of global header messages for an acquisition and then a set of messages per image. Parts of the global header are passed on through shared memory, such as the flat field and pixel mask datasets. The data blob is extracted from the image message parts and placed in shared memory with a simplified header containing the datatype, compression and a frame identifier. For each shared memory buffer, a Ready message with a buffer index is passed to the FP, which then wraps the data in shared memory and passes it through its plugin chain. When the EigerProcessPlugin receives the pointer, the pre-compressed image blob is passed on to be written to disk by the FileWriterPlugin, while the details of the global header messages are published on the meta channel. For hardware-triggered scanning and mapping applications [13], the compressed size is set as a parameter to be written to the data file, in the absence of a sum dataset, and a UID dataset is created by the Parameter plugin, based on frame number. This provides a visual display of the overall intensity of the image for each point of a scan, as it progresses. When the pointer is released by the final FP plugin, a Release message is sent back to the the FR to allow it to re-use the buffer.

The FileWriterPlugin is detector agnostic, so must be configured to correctly handle Eiger data. It will tag the dataset with the compression mode and write the data using Direct Chunk Write [12], bypassing the HDF5 processing pipline. The Eiger use case would be greatly limited without this, as it allows writing the pre-compressed data from the ZMQ data stream directly to file. Like the EigerProcessPlugin, it will also publish some meta data, such as the time and order the frames were written in. This is very useful for visualising and debugging performance of the file writing. Any data published on the meta channel can be captured by the MetaListener, a separate Python-based ZMQ stream listener and HDF5 file writer. The architecture of a single OD node for an Eiger system is shown in Fig. 2 and a schematic of the OdinServer and overall system for a four-node Eiger Odin system is shown in Fig. 3. Importantly, throughout this process, there are no copies made of the data. The data blob from the ZMQ data stream is stored in shared memory by the FR and then the pointer given to the FP is passed directly to the HDF5 write call. This is vital to the throughput of the data acquisition pipeline.

At the end of an acquisition there will be one data file for each node, though this is configurable, plus a meta file. If the dataset needs to read by a simple application, a new HDF5 file can be created with a Virtual Dataset (VDS) [12] to interleave the frames from each raw file, alongside links to each of the datasets in the meta file. The layout of the interleaved VDS is shown in Fig. 4. Another VDS can be created to superimpose the dimensionality of a scan onto the flat datasets. An application could also be given a definition of the scan and the file layout in order to intelligently read frames from the raw datasets in the correct order using any slicing that is required.

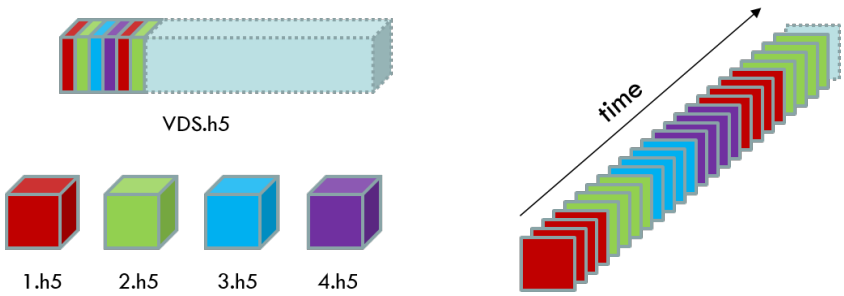

Figure 4: Virtual Dataset layout for Eiger data [14].

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

Figure 5: ADOdin EDM screen for Eiger detector control.

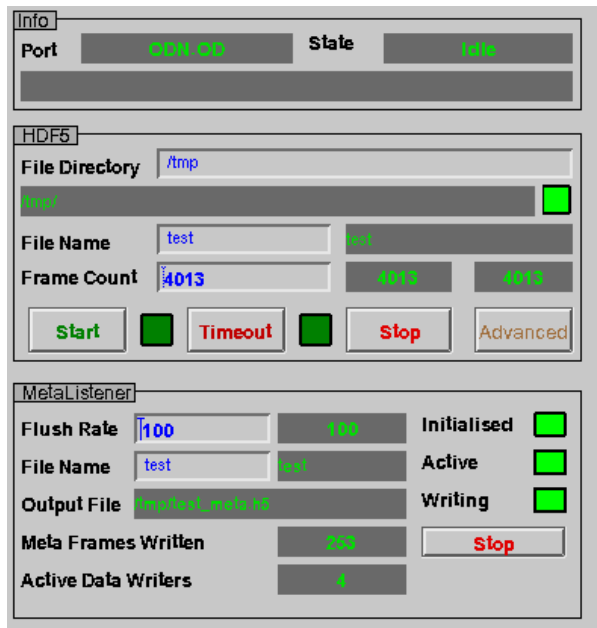

Figure 6: ADOdin EDM screen for acquisition control.

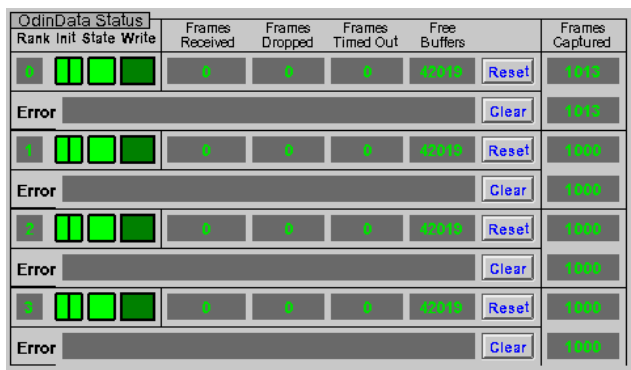

Figure 7: ADOdin EDM screen for OdinData status.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

DEPLOYMENT

This modular framework brings some complication in deployment. The four-node Eiger Odin system includes 12 separate processes in total, all of which need to be configured correctly to fit into the overall system. A Python library has been written to generate all of the start up scripts and configuration files for the required applications. This makes it much easier to deploy and update an Odin system without knowing the detail of how everything works. All that is required to configure a full system is to define the number of OdinData processes, the detector type and IP address, and the servers that will host the processes. The applications will be correctly configured to intercommunicate and operate as a single entity.

All that is required at this point is to integrate the API provided by the OC server into a wider control system. This can be as simple as a web page or a command line script exposing the HTTP endpoints. At DLS we use ADOdin to integrate Odin into our EPICS control system. This provides a familiar areaDetector interface to users and controls engineers and makes integration easier and more efficient. It also builds upon a robust and convenient driver architecture to keeps the code base small and maintainable. The majority of ADOdin is implemented by mapping EPICS process variables to HTTP endpoints, while letting the OC server handle the complex logic and coordination of the various applications. Integration with areaDetector also allows the driver to present a reduced live view of the data as NDArrays. This can then be passed through any areaDetector plugin and onto a viewing application, such as imagej via pvAccess [15]. Fig. 5 shows the main ADOdin EDM screen for controlling Eiger and the EigerFan.

ADOdin hides the underlying complexity of the OdinData stack, presenting a simple API, as shown in Fig. 6 and Fig. 7. The only required configuration for an acquisition is a directory, file name and number of frames, regardless of how many OdinData nodes there are. Some of these parameters will trigger underlying logic when set to uniquely configure each node. For example setting the frame count will send the same value to all nodes and the individual process will calculate the number of frames it expects, based on its rank.

PERFORMANCE

Performance profiling up to this point has only been carried to ensure viability for specific use cases at DLS. The highest data output is with the Eiger 2 XE 16M at 560Hz. Tests were carried out with this system on a beamline while attempting to maximise the signal on the detector, which produced frames of approximately 2 MB after compression. The four-node OD deployment was able to cope with the throughput in this case. All Eiger systems that expect full data rates use the same configuration. More basic tests show the throughput of a single node in isolation is approximately 15 Gb/s and that there is a diminishing return in adding more processes to one server, most likely due to the I/O to the file system. At a certain point it is much more effective to split the OD processes across multiple machines. This will be essential for future faster detector systems.

STATUS AND FUTURE

OD is now running in production with Eigers on five beamlines at DLS. After a successful test deployment of OC Eiger support on a beamline, we are confident in rolling it out to all existing and future Eiger systems to provide a simpler and more coherent interface to Eiger detectors. This system is something that can be deployed outside of DLS as a solution to integrate Eiger detectors into any control system.

A Docker [16] container has been produced to, firstly, install all dependencies required for Odin to run on any compatible system and then secondly to deploy specific Eiger Odin systems. The deployment consists of a set of Docker containers to run each specific application instance. This should also serve as an instruction set to build and configure a more tailored deployment from source. An Eiger simulator for the control API and data stream has been produced for testing during development. This will be deployed with the Docker container to demonstrate the system without the requirement of a real detector. A web GUI is provided to present some of the most fundamental control and status parameters of the system, which will enable manual data collection to be carried out. We are investigating the usage of Docker containers to deploy Odin on beamlines at DLS with orchestration such as Kubernetes [17] to further simplify deployment for beamline controls engineers.

There is growing interest in implementing a Kafka [18] cluster to integrate with OD. Data could be consumed from the Kafka cluster by processing pipelines, instead of effectively transferring the data via the file system, which would reduce file system load and improve latency of processing pipelines. This would enable better real-time feedback to the user. An OD Kafka plugin has been developed and tested on Eiger systems and further work is required to integrate this into existing processing applications.

CONCLUSION

Odin has become an integral part of beamline control systems at DLS. OdinData has been well proven as a robust and fast data acquisition framework for use with Eiger detectors, but the use of a separate ADEiger driver for control was bulky and error prone. To simplify the architecture, an OdinControl adapter has been written for Eiger producing a full software framework for controlling and acquiring data, which could be integrated into any control system using the HTTP API. This will be make it much easier to deploy and maintain Odin systems on beamlines and also improve the future development of the system. The Odin framework is open source and available on GitHub [4] [19].

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-THBPP05

REFERENCES

[1] M. Guizar-Sicairos *et al.*, "High-throughput ptychography using eiger: scanning x-ray nano-imaging of extended regions," *Optics Express*, vol. 22, pp. 14859–14870, 2014.

[2] EIGER X 4M, https://www.dectris.com/products/eiger/eiger-x-for-synchrotron/.

[3] D. Batey, S. Cipiccia, F. Van Assche, S. Vanheule, J. Vanmechelen, M. Boone, and C. Rau, "Spectroscopic imaging with single acquisition ptychography and a hyperspectral detector," *Scientific Reports*, vol. 9, p. 12278, 2019.

[4] odin-detector, https://github.com/odin-detector/.

[5] J. Marchal *et al.*, "Excalibur: a small-pixel photon counting area detector for coherent x-ray diffraction - front-end design, fabrication and characterisation," in *Journal Of Physics: Conference Series*, vol. 425, pp. 530–533, 2013.

[6] G. Yendell, U. Pedersen, N. Tartoni, S. Williams, N. Nicholls, and A. Greer, "Odin - a control and data acquisition framework for excalibur 1m and 3m detectors," in *Proc. ICALEPCS'17*, pp. 966–969, 2017.

[7] C. B. Wunderer *et al.*, "The percival 2-megapixel monolithic active pixel imager," in *JINST*, vol. 14, 2019.

[8] ZeroMQ, https://zeromq.org/.

[9] EIGER SIMPLON API, https://www.dectris.com/support/manuals-docs/overview/.

[10] Martins B. ADEiger, https://github.com/areaDetector/ADEiger/.

[11] M. L. Rivers, "areadetector: Epics software for 2-d detectors," in *Proc. ICALEPCS'17*, pp. 1245–1251, 2017.

[12] N. Rees *et al.*, "Developing hdf5 for the synchrotron community," in *Proc. ICALEPCS'15*, pp. 845–848, 2015.

[13] T. Cobb, M. Basham, G. Knap, C. Mita, M. Taylor, G. Yendell, and A. Greer, "Malcolm: A middlelayer framework for generic continuous scanning," in *Proc. ICALEPCS'17*, pp. 780–784, 2017.

[14] HDF Group, https://support.hdfgroup.org/HDF5/docNewFeatures/VDS/HDF5-VDS-requirements-use-cases-2014-12-10.pdf/.

[15] L. R. Dalesio *et al.*, "Epics 7 provides major enhancements to the epics toolkit," in *Proc. ICALEPCS'17*, pp. 22–26, 2017.

[16] Docker, https://docs.docker.com/get-started/.

[17] Kubernetes, https://kubernetes.io/docs/home/.

[18] Kafka, https://kafka.apache.org/documentation/.

[19] eiger-detector, https://github.com/dls-controls/eiger-detector/.