DE-FOA-DE-SC0007551 SBIR Phase 2 Grants
A Client/Server Architecture for Supporting Science Data Using EPICS Version 4

Submitted as a phase II grant follow on to:

DE-FOA-0000577 SBIR Phase 1 Grants
Topic 12 (a): Ancillary Technologies for Accelerator Facilities
(Accelerator Modeling and Control)

Technical Abstract

The Phase 1 grant that serves as a precursor to this proposal, prototyped complex storage techniques for high speed structured data that is being produced in accelerator diagnostics and beam line experiments. It demonstrates the technologies that can be used to archive and retrieve complex data structures and provide the performance required by our new accelerators, instrumentations, and detectors. Phase 2 is proposed to develop a high-performance platform for data acquisition and analysis to provide physicists and operators a better understanding of the beam dynamics. This proposal includes developing a platform for reading 109 MHz data at 10 KHz rates through a multicore front end processor, archiving the data to an archive repository that is then indexed for fast retrieval. The data is then retrieved from this data archive, integrated with the scalar data, to provide data sets to client applications for analysis, use in feedback, and to aid in identifying problem with the instrumentation, plant, beam steering, or model.

This development is built on EPICS version 4[1], which is being successfully deployed to implement physics applications. Through prior SBIR grants, EPICS version 4 has a solid communication protocol for middle layer services (PVAccess), structured data representation and methods for efficient transportation and access (PVData), an operational hierarchical record environment (JAVA IOC), and prototypes for standard structured data (Normative Types). This work was further developed through project funding to successfully deploy the first service based physics application environment with demonstrated services that provide arbitrary object views, save sets, model, lattice, and unit conversion. Thin client physics applications have been developed in Python that implement quad centering, orbit display, bump control, and slow orbit feedback. This service based architecture has provided a very modular and robust environment that enables commissioning teams to rapidly develop and deploy small scripts that build on powerful services. These services are all built on relational database data stores and scalar data. The work proposed herein, builds on these previous successes to provide data acquisition of high speed data for online analysis clients.

All of EPICS V4 Core Code developed as part of this grant or developed in conjunction with this grant is open source and available on Github along with EPICS base at https://github.com/epics-base?page=1. All reference to Normative Types), NTTypes, PVAccess, PVData, Java IOC, PVServices or PVClients that are implemented in C++ Java are available there.[11]

Key words

Control System, EPICS, PVAccess, PVData, Normative Types, Analytics

This page contains proprietary information

Table of content

1 Phase II Technical Objective

The goal of this program is to extend the EPICS V4 architecture to demonstrate high speed data acquisition and data analysis. This requires demonstration of the ability to integrate new FPGA based devices that are taking data at the MHz rate with large GB on board memories, into a distributed architecture that can efficiently read the devices and serve the data into processing and storage components. As important, is the ability to access this data, in conjunction with other facility data, to create comprehensive, flexible, and modular analysis environment. This work provides the basis for supporting machine and science analysis.

2 Phase II Deliverables

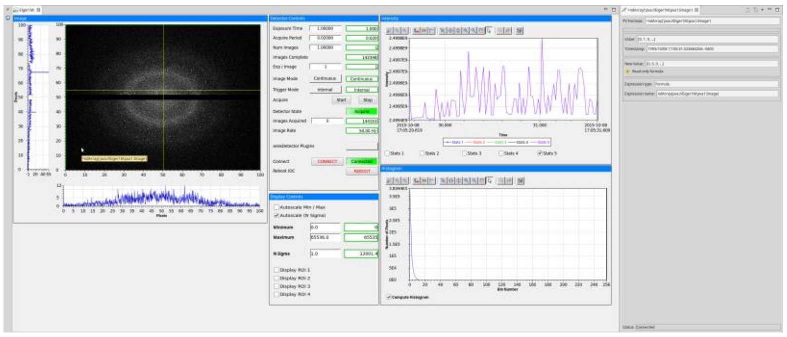
The were two primary deliverables for this phase II grant: 1) improve the V4 base to provide high speed, robust and reliable transport of structured data 2) Demonstrate this through the archiving and retrieval of an archiver for the normative types of EPICS V4, particularly multi-dimensional arrays. All of the code in this area is available at: https://github.com/epics-base and https://slacmshankar.github.io/epicsarchiver_docs/.

2.1 Task 1 Integrate high speed instruments into the EPICS V4 IOC.

2.1.1 Services to Produce Matrices

Several developments occurred to provide high speed instrument integration in EPICS V4. As part of the SBIR work, a PVAccess Service that demonstrated the throughput of the protocol was written. We were able to transmit large data sets at 90% of the Ethernet line speed. This is a significant improvement over the previous Channel Access implementation.

As part of this work, a normative type (NTNDArray) was completed that supports NDimensional Arrays: vectors, 2D data, time series of 2D data, tiled images, and any N Dimensional Arrays.

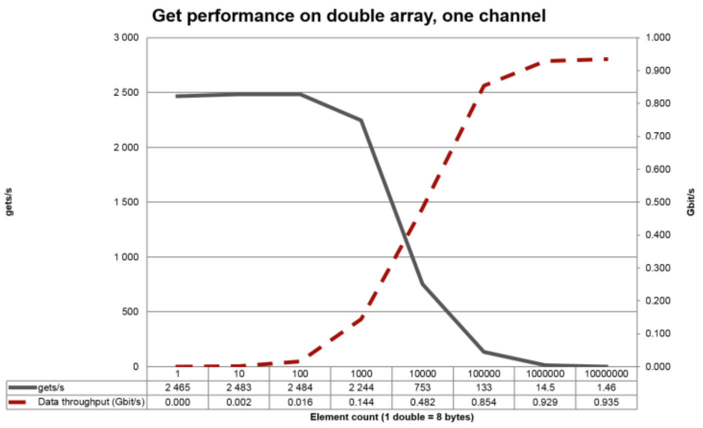


This NTNDArray represents a narrow interface that is easily handled by general purpose clients. This was demonstrated by a python client that can receive NTNDArrays and print them out in test. This demonstration was used to implement an NTNDArray in Control System Studio (CS Studio), a Graphical User Interface that is developed as an operator workflow application with a synoptic editor and high level applications in Java.

2.1.2 Data Passing Mechanisms

Demonstrated the integration of high speed data sources, such as modern Beam Position Monitors electronics and area detectors..

2.1.2.1 A test service was developed that pumps out matrices through a PVAccess server that was used to develop and demonstrate the capability of PVAccess.
PVAccess was extended to transmit large arrays. The performance greatly surpassed EPICS existing protocol, Channel Access, which only transmits arrays as a vector with no metadata to describe more than the number of datum and the size of each datum.



	1	10	100	1000	10000	100000	1000000	10000000
gets/s	2 465	2 483	2 484	2 244	753	133	14.5	1.46
Data throughput (Gbit/s)	0.000	0.002	0.016	0.144	0.482	0.854	0.929	0.935

2.1.2.2 The service above, was developed in C++ along with the implementation and helper classes to support this mechanism and the NTNDArray. It was run under Linux on the I/O Controller. Once this data passing mechanism, data type, and helper class was developed in V4 and made available to the EPICS community, developers at the National Synchrotron Light Source made an important improvement in areaDetector to take advantage of this new capability. The first was an extension to the areaDetactor, an XRay beam line development for processing detector data. Using the test service as a guide, they added a plugin that directly serves NTNDArray over PVAccess from the areaDetector processing pipeline. In addition, a CS Studio modification was made to the PVAccess Client plugin for the PVManager to monitor the NTNDArrays. The client application to view N-dimensional arrays was already available in CS-Studio. With these modifications, the beam lines at NSLS II could configure detector visualization directly in the operator screens through simple configuration and monitor their data in real time with demonstrated performance of up to 60 frames per second (fps).

2.1.2.3 The areaDetector work above immediately made all of the area detector drivers available through PVAccess. In addition to these drivers, the array passing in the EPICS Process Database was modified so that arrays would be passed by reference on the EPICS data links. This made the passing of NDimensional Data through the EPICS Process Database as fast as possible by managing and passing references on read. This eliminated copies that are very expensive on large arrays. This made all of the existing drivers for vector acquisition available through the EPICS process database to the PVAccess server (PVASrv). Further development is needed to make the NTNDArray data type available through the EPICS database.

2.2 Task 2. Demonstrate the archiving of this large data sets into a EPICS V4 Archiver

Collect V4 data into a V4 archiver. (1) Demonstrate the storage of metadata for the V4 data types. (2) Demonstrate the storage of the large data sets. (3) Demonstrate the ability to create the index files to access the data.

2.2.1 Demonstrate the Storage of Metadata

The storage of metadata for the V4 data types has been demonstrated in two ways: the areaDetector data acquisition code is able to write NTNDArrays directly to files and the Archive Appliance is able to store all NTYTypes including NTND Arrays server from PVAccess servers. The NTNDArray includes the description of the number of axis for the array and the units, data type, and dimensions of each axis. In addition, the NTNDArray supports optional metadata such as scalar values for independent variables. These independent variables can include independent variables such as position or energy, that are required to identify the significance of the NTNDArray. This has also been demonstrated for neutron experiments where detectors are recording counts and time of flight data in sparse arrays.

2.2.2 Demonstrate the Storage of Large Data Sets

In the case of both the Archive Appliance and files directly from the areaDetector data processing pipeline, the large data set is part of the NTNDArray. The metadata and the independent variables are part of the data payload. The multidimensional array is part of the NTNDArray.

2.2.3 Demonstrate the Ability to Create Index Files

The above mentioned large data sets are used for viewing and require analysis for either understanding the operation of an accelerator or to analyze the results of the xray data from some sample. In both cases, all of the data stored as part of these NTNDArrays have metadata that is used to search for data. For experimental data, beam line, user, and sample may either be metadata or part of a file descriptor. In every case, the independent variables are part of the NTNDArray. To search on data sets from a given set of independent variables, one would need to open each file, find the independent variable and compare it to the parameters being requested. This is very time consuming. On an set of HDF5 experimental data chosen from a previous experiment and scaled to current rates and data size, a search was measured to take 30 seconds. An index service was prototyped that used a relational database with 120 properties and one million samples. This test reduced the search time to 3.4 seconds. A second prototype was made with MongoDB,using the same 120 properties and one million samples and the search time was reduced to 40 msecs. A service was developed, MetaDataStore was developed as a PVAccess Service that responded to queries of independent variables and deployed as part of a suite of data acquisition and analysis tools at NSLS II.

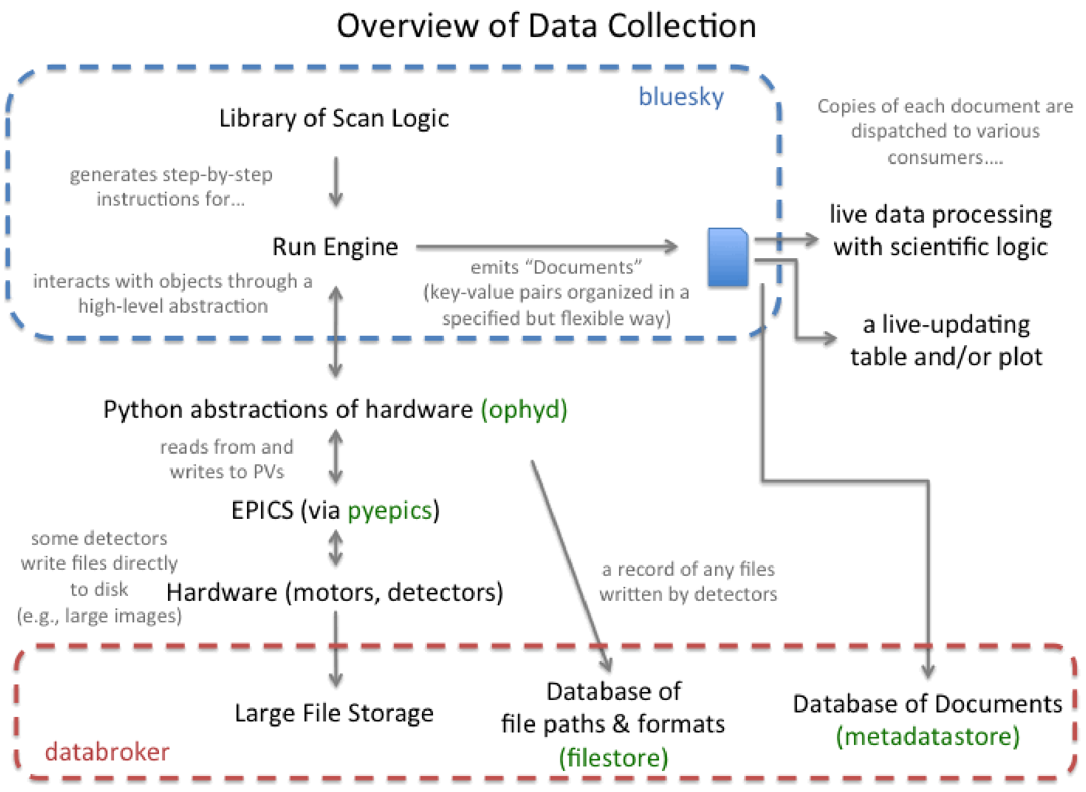

Overview of Data Collection
bluesky
Library of Scan Logic
generates step-by-step instructions for...
Run Engine
interacts with objects through a high-level abstraction
emits "Documents" (key-value pairs organized in a specified but flexible way)
Copies of each document are dispatched to various consumers....
live data processing with scientific logic
a live-updating table and/or plot
Python abstractions of hardware (ophyd)
reads from and writes to PVs
EPICS (via pyepics)
some detectors write files directly to disk (e.g., large images)
Hardware (motors, detectors)
a record of any files written by detectors
Large File Storage
Database of file paths & formats (filestore)
Database of Documents (metadatastore)
databroker

Figure 1 metadatastore and filestore for data acquisition

The PVAcccess Service can also be used to populate the MetaDataStore. Data acquisition programs can directly populate the MDS while taking data. Applications have also been written to populate the MetaDataStore with existing data stores. This can greatly improve the search and access time of data that was collected prior to the completion of the MetaDataStore

2.3 Task 3. Demonstrate the integration of the Archive Service into the SciDB Analytics Engine[2]

Integrate the V4 archiver into the SciDB analytics engine to demonstrate the ability to use existing analytics. (1)Demonstrate the ability to query the data catalogue service. (2) Demonstrate the ability to access archive data from the analytics engineers supported by the SciDB community.
2. EPICS V4 Data Access Service

The project addresses the above requirements and issues by providing a generic EPICS V4 middle layer framework for accessing composite data sources of accelerator and beamline facilities. For example, Figure 1 shows the EPICS V4 data access service in the context of the NSLS II data management and processing system. The bottom data layer represents a distributed collection of heterogeneous data sources including multiple components, such as the meta-data store and repositories of data files with time series of control data collected from accelerator devices and beamlines, detector images and associated metadata, and post-processing results from analysis and reconstruction applications. The heterogeneous structure of the data layer was driven by multiple requirements, such as configurability, modularity, incremental development, support of new extensions and multiple data formats tailored to detectors. The common client interface, Data Broker, is implemented in Python and reused by different experimental control and interactive analysis applications.

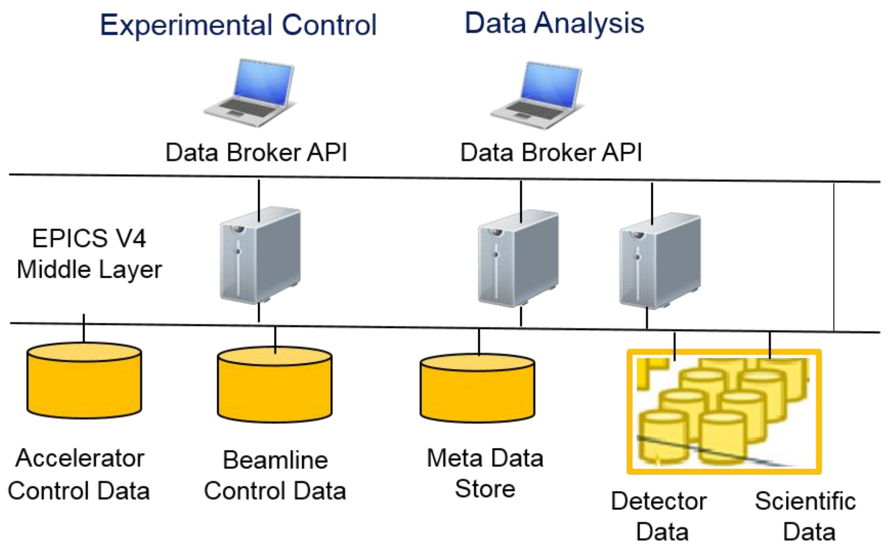


Figure 1: EPICS V4 three-tier data management and processing system

The EPICS V4 middle layer services extend the scope and scale of this infrastructure by providing an efficient interface between distributed data sources and client applications. The corresponding interface originated from the Phase I prototype and consistently developed within the Phase II project by adhering to the Spark conceptual model. In contrast with other scientific-oriented data management and processing systems (e.g., SciDB), Spark has explicitly defined an in-situ processing approach that can be connected with different data sources. As a result, the

[2] Ode is available at: https://github.com/epics-extensions/ea-cpp

Spark approach advocated the Phase I conceptual solution of the SciDB-oriented driver framework and facilitated its further development towards a much larger ecosystem. The existing version of the Spark approach, however, introduced several constraints. First, it was implemented in the Java language that was mismatched with the C/C++ I/O libraries and the Python data analysis applications of the conventional scientific-oriented environment. Second, it did not support access to heterogeneous data sources of large-scale experimental facilities. Therefore, the Phase II project addressed these issues by implementing an original compositor connector based on the EPICS V4 middleware. Figure 2 and Figure 3 show the class and sequence diagrams of the corresponding client interface.

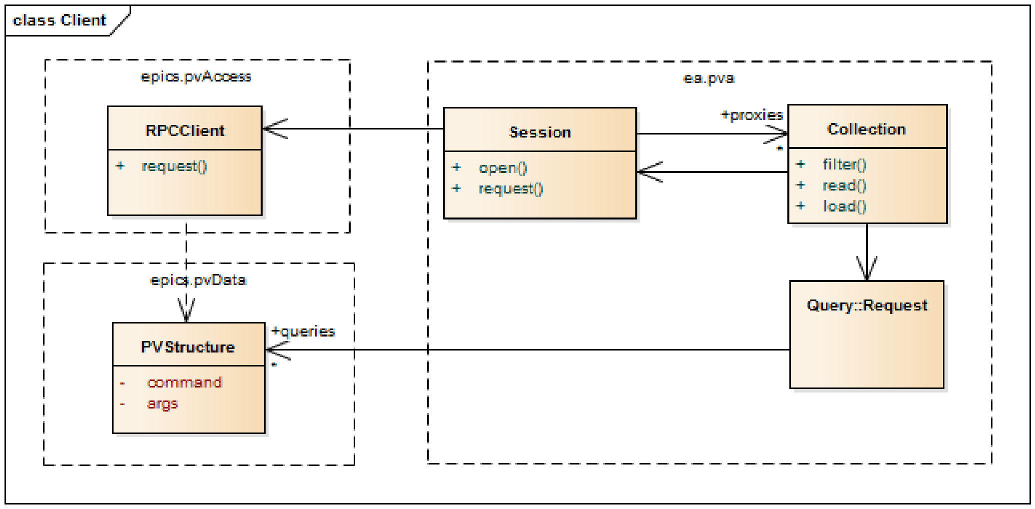


Figure 2: Class diagram of the client composite interface

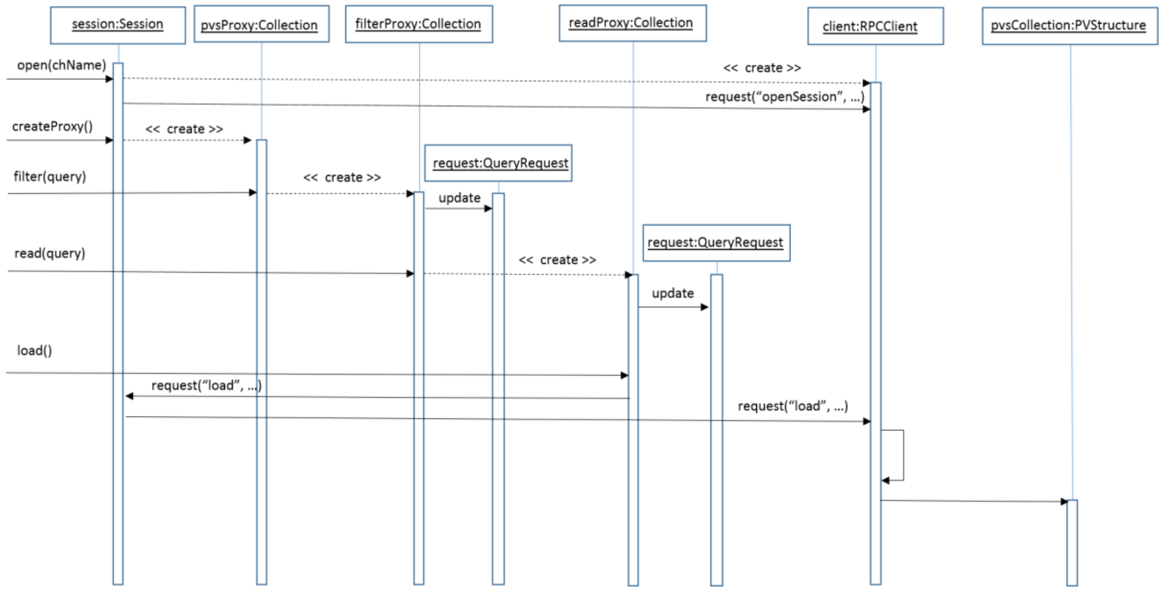


Figure 3: Sequence diagram of the client composite connector interface

The interface of the compositor connector followed the Spark approach and defined the data access as a combination of the Query requests accumulated via a sequence of the Collection transformations, *filter* and *read*. The structure of these requests is flexible and initialized via the Python dictionary that is subsequently converted into the PVStructure payload container of the EPICS V4 communication protocol. The *filter* request accommodates queries of the data catalog, for example, the aggregation pipeline of the MongoDB database. And the *read* method augments this request for reading datasets distributed in multiple data files of the hybrid data store. As a result, these requests are combined and shipped together with the *load* action to the EPICS V4 data access server. The server implementation is illustrated by the class and sequence diagrams shown in Figure 4 and Figure 5.

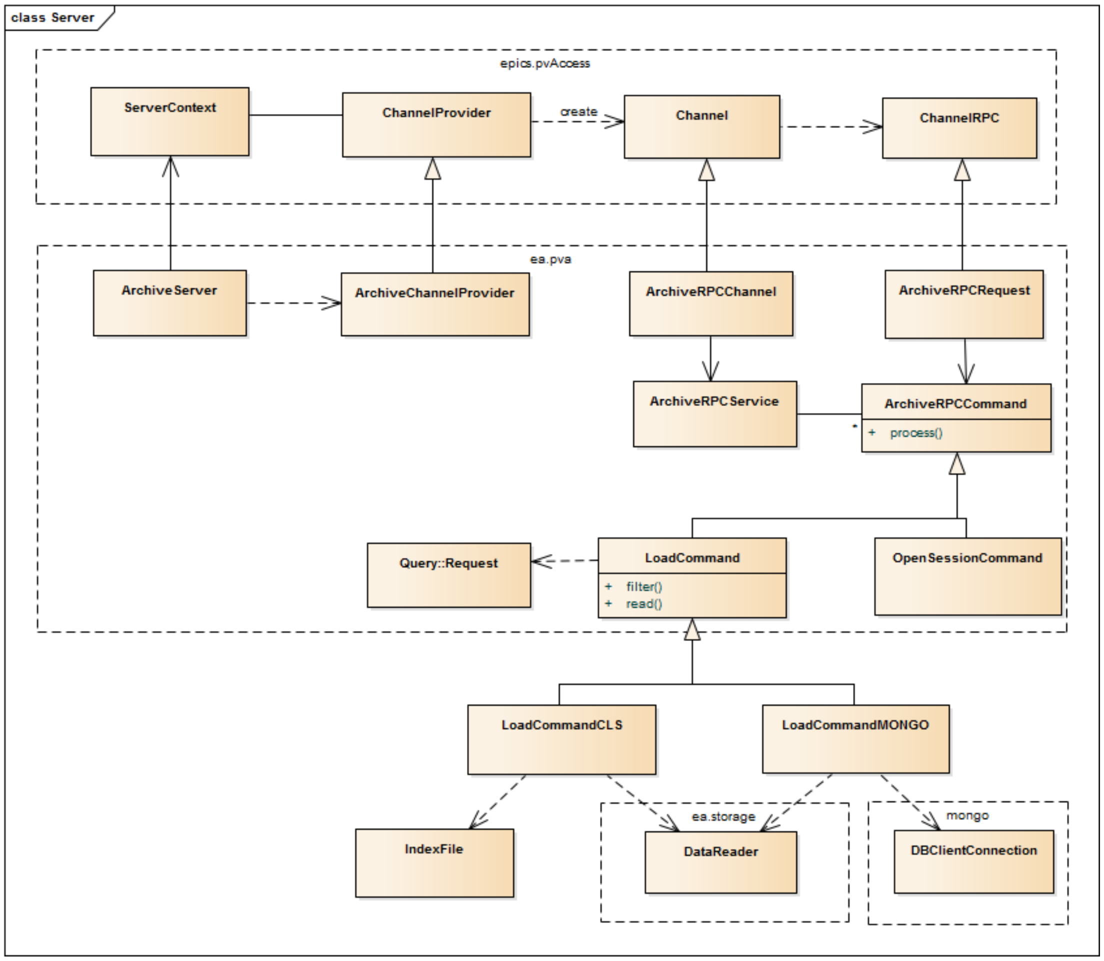


Figure 4: Class diagram of the compositor connector

The data access server is based on the composite connector framework that can be specialized for different types of hybrid data sources. Specifically, Figure 5 illustrates a sequence of steps for accessing the new EPICS Archiver data layer consisting of a MongoDB data catalog and a repository of data files. This sequence starts with accepting a composite request and breaking it down into the filter and read queries. Then, the filter request is converted into the BSON container and shipped to the MongoDB engine for fetching locations of requested datasets distributed in a repository of data files. Next, the read request is updated with the MongoDB results and processed by the corresponding file reader. The same approach was implemented for accessing the original archiver data using proprietary indexing files. As a result, the EPICS V4 data access framework provides a seamless transition to new versions of data catalogs and file formats. Moreover, the same framework was applied for accessing the MongoDB metadata catalog of the experimental data. Finally, it brought an efficient EPICS V4 middleware for connecting distributed data sources and the Python data analysis and processing environment.

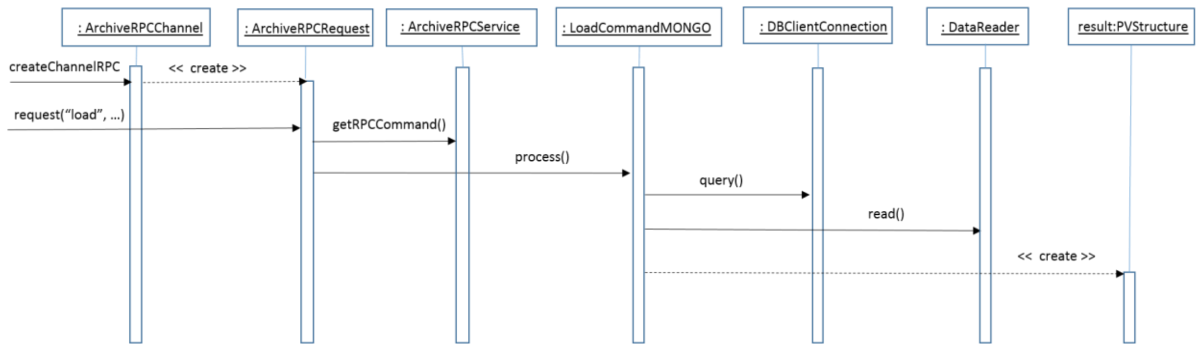
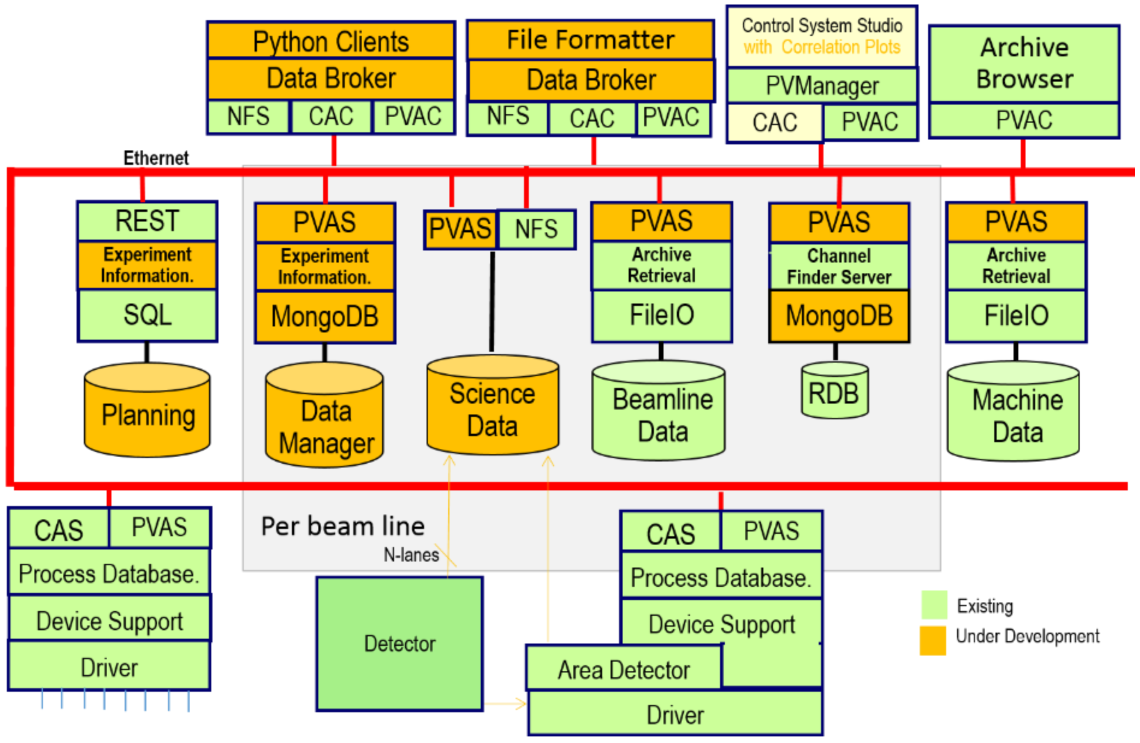


Figure 5: Sequence diagram of the compositor connector

In comparison with existing data management systems, the EPICS V4 compositor connector resolved several important issues. First, it provided a configurable framework for accessing hybrid data stores of experimental facilities. Second, the composite interface facilitated the integration of several data models. For example, the current application combined the MongoDB document-oriented and SciDB array-oriented models. Third, the composite connector approach extends the scale of conventional databases with a repository of data files. This topic encompasses multiple aspects including communication protocols. For example, EPICS V4 extends the MongoDB interface with the PVData payload container allowing to accommodate the MongoDB semi-structured data and large-scale datasets of experimental facilities. Finally, the composite data access framework extends the scope of the Spark connector mechanism and establishes a consistent route for its integration with the Spark-based parallel processing environment.

2.4 Task 4. Create a Data Catalogue Server for All Data Archives.

Build on the memory resident data catalogue demonstrated in phase 1, to provide the data catalogue as an EPICS V4 service. (1) Dynamically populate the Data Catalogue Service with data for the V4 PV Archive. (2) Dynamically populate the Catalogue Service with data for the V3 Channel Archiver. A FileStore Service that uses PVAccess, has been developed and deployed at NSLS II that gives analysis codes a single call to search the MetaDataStore, find the file and return the data as NTNDArrays. These services are available on Github at http://nsls-ii.github.io/databroker/searching.html. The scripting environment that uses this architecture to provide python client access to scientific data is found on the bluesky github repository https://nsls-ii.github.io/bluesky/documents.html. This environment and the scripting is released and in use at the NSLS II beamlines.



3 Presentations on this work

Presentations were held at EPICS meetings throughout the development and deployment of this software.

Spring 2013 meeting at Diamond Light Source, Abingdon, UK
http://www.aps.anl.gov/epics/meetings/2013-05/EPICS2013/Programme.html

"Core Development"	Bob Dalesio
"EPICS V4 Overview and Status"	Greg White
"Interoperability and migration from V3 to V4"	Andrew Johnson
pvaSrv, the IOC side bridge from PVaccess to IOC	Ralph Lange
Core Status	Marty Kraimer, Matej Sekoranja
Performance and micro benchmarking EPICS V4	Matej Sekoranja
areaDetector Image Processing Pipeline	David Hickin
Archive Service	Timo Korhonen
NSLS II V4 Services	Guobao Shen
EPICS V4 Integration into CSStudio	Matej Sekoranja
Archiving with Large Buffers / Data Type Catelog	Nikolay Malitsky
EPICS for Neutron Scattering Beamlines	Steve Hartman

Fall 2013 meeting prior to the ICALEPCS in San Francisco, USA.
http://www.aps.anl.gov/epics/meetings/2013-10/

V4 Status and Workshop Report	Bob Dalesio
PVAccess Client APIs	Marty Kraimer
Archive Engine for Large Data Sets	Nikolay Malitsky
New Archive Appliance	Murali Shankar
Archive Service	Timo Korhonen
V3 Database Throughput (Large Data)	Michael Davidsaver

Fall 2014 EPICS Meeting Saclay, France
http://irfu.cea.fr/Meetings/epics/program.php

EPICS V4 for SNS Neutron Data Collection	Kay Kasemir
PVData / PVAccess/ Normative Types	Marty Kraimer / Matej Sekoranja
EPICS V4 for Diamond Detector Data	David Hickin
AreaDetector, What's New	Mark Rivers
Bealine Control and Data Acquisition at NSLS II	Bob Dalesio
Tools and Services at NSLS II	Kunal Shroff
Writing Accelerator Physics Apps with EPICS V4	Greg White
Dirt/data integration in CSStudio	Gabriele Carcasi
NSLS II Physics Apps in an Open Architecture	Guobao Shen

4 Summary

The code that was developed to demonstrate the use of EPICS V4 protocol and data types has been completed, placed into the community as open source, and is in use at several facilities that include: NSLS II, SNS, and APS. The work has been demonstrated and is in active use. We provide training and services to support others in the community that want to adopt this approach.