EXPERIMENT CONTROL AND ANALYSIS FOR HIGH-RESOLUTION TOMOGRAPHY*

N. Schwarz†, F. De Carlo, A. Glowacki, J.P. Hammonds, F. Khan, K. Yue
ANL, Argonne, IL 60439, USA

Abstract

X-ray Computed Tomography (XCT) is a powerful technique for imaging 3D structures at the micro- and nano-levels. Recent upgrades to tomography beamlines at the APS have enabled imaging at resolutions up to 20 nm at increased pixel counts and speeds. As detector resolution and speed increase, the amount of data that must be transferred and analyzed also increases. This coupled with growing experiment complexity drives the need for software to automate data acquisition and processing. We present an experiment control and data processing system for tomography beamlines that helps address this concern. The software, written in C++ using Qt, interfaces with EPICS for beamline control and provides live and offline data viewing, basic image manipulation features, and scan sequencing that coordinates EPICS-enabled apparatus. Post acquisition, the software triggers a workflow pipeline, written using ActiveMQ, that transfers data from the detector computer to an analysis computer, and launches a reconstruction process. Experiment metadata and provenance information is stored along with raw and analyzed data in a single HDF5 file.

INTRODUCTION

The Advanced Photon Source (APS) is the nation's premier source of hard (10-100 keV) x-rays. In recent years, the very high x-ray photon flux generated by the APS has been matched by a new generation of detectors that are able to collect x-ray images at faster speeds. This enables real-time, in situ, and dynamic studies, but also presents challenges related to experiment control, analysis, and data management.

In standard tomography experiments, a series of regularly spaced projections is acquired as the sample is rotated 180 degrees about its center axis. Using monochromatic beam illumination from an APS bending magnet source, an old generation Coolsnap K4 CCD camera with 2048x2048 pixels was able to acquire the needed 1500 projections in 15 to 25 minutes. A new generation sCMOS camera like the Cooke pco.edge camera with 2560x2160 pixels can acquire the same data in 15 seconds. An even faster camera like the Cooke pco.dimax can collect the same data in 300 ms using polychromatic beam illumination.

* Work supported by U.S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357.

† nschwarz@aps.anl.gov

The addition of experiment equipment, such as furnaces and pressure chambers, adds complexity to standard tomography acquisition. Tomography beamlines may also take advantage of automated sample changers to increase throughput. After data acquisition completes, but before any further analysis can be performed, the tomographic projections must be reconstructed to produce a 3D volumetric representation of the sample. Figure 1 shows the overall architecture of a tomography system at the APS.

EXPERIMENT CONTROL SOFTWARE

The experiment control software shown in Figure 2 is the main interface through which users set up and run tomography measurements. The system has a number of features related to image viewing and data collection.

Features

Live and offline data can be displayed. The application can capture single, multiple, or continually streamed images. All standard Area Detector [1] properties such as binning, region-of-interest (ROI), gain, exposure time, acquire period, data type, image mode, etc. can be configured. Detector-specific features exposed with Area Detector may also be made accessible.

Graphical annotations are available as image overlays for simple analysis or system calibration. A marker annotation allows users to mark static points in the image. A calibrated ruler annotation allows quick measurements in user-defined units.

Color maps and contrast can be adjusted by setting RGB values, and the low and high values can be clamped using a graphical editor. An auto-levels function searches for the lowest and highest non-zero value and clamps the map appropriately.

EPICS-enabled devices, such as motors, are displayed on a separate calibration window. These devices may be configured dynamically via the application's preferences window. Devices may be added, removed, or disabled during run time. The addition or removal of devices is reflected automatically in the user interface without the need to edit traditional display manager configuration files.

Configurable scanning is the most important feature of the experiment control software. The system is designed to be extensible and easy for a non-expert user to configure. Users can configure all scan parameters via the user interface.

ISBN 978-3-95450-139-7

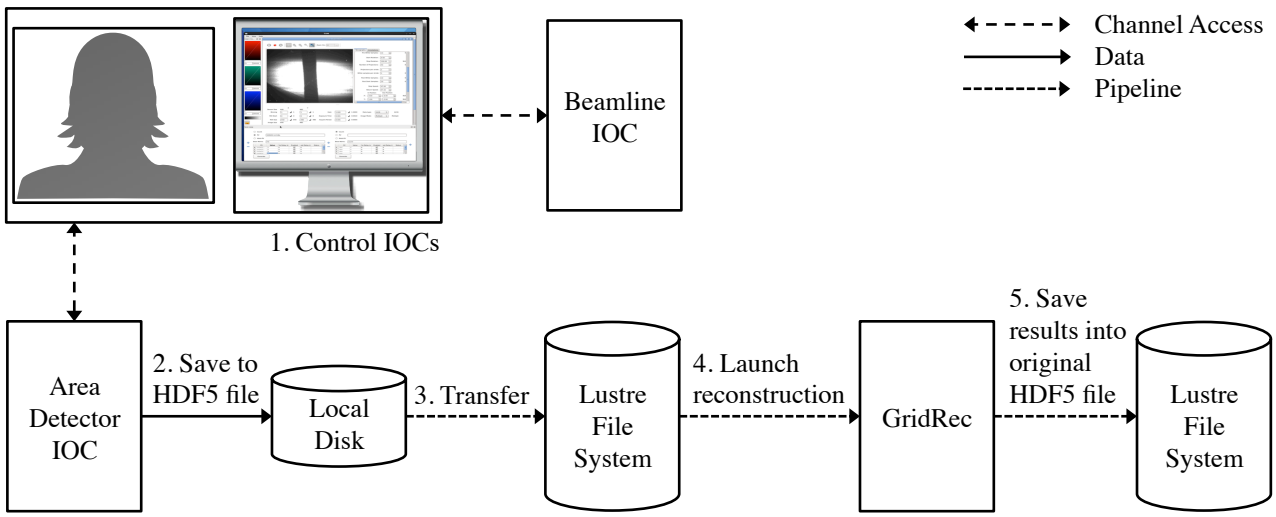

Figure 1: Components of the tomography control, acquisition, and reconstruction system in use at the APS. (1) Experiment control software coordinates EPICS-enabled motors, detectors, and other apparatus using the Channel Access protocol. (2) EPICS Area Detector software writes data in HDF5 files to locally attached storage. (3) Data is transferred using gridFTP from the detector computer to a central Lustre file system. (4) Tomographic reconstructions are created from acquired projection data stored on the central Lustre file system. (5) Reconstructed data are added to the same HDF5 file containing original raw projection data.

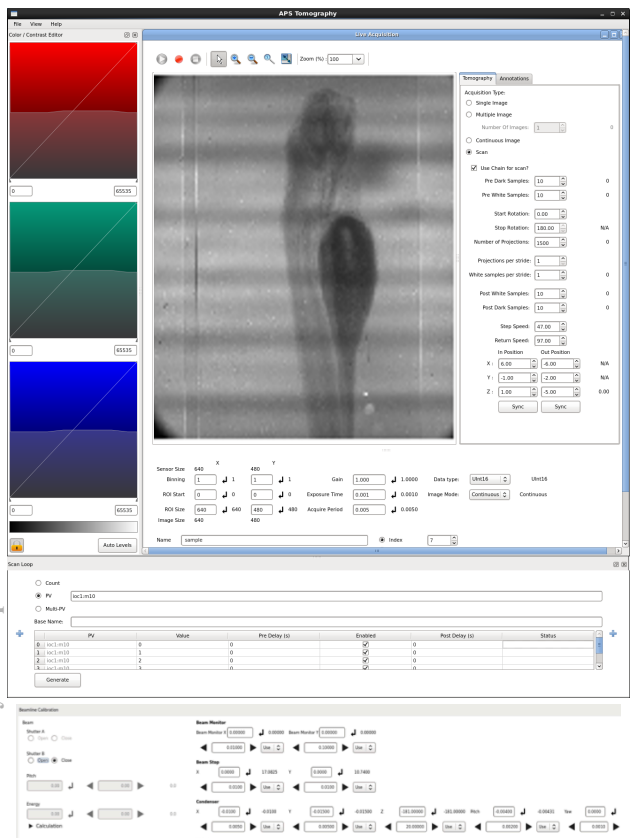

Figure 2: Experiment control software's main window (top) and calibration window (bottom). The main window shows the color editor on the left, the acquisition display in the center, tomography scan settings on the right, and nested loop controls on the bottom.

Users can configure properties for basic tomography scans using widgets on the right side of the acquisition window. Settings include the number of pre-, inner- and post-white and dark field images, the number of projections, start and stop positions, and motor speed. Tomography scans may be nested within higher-level loops. These loops can change some number of other EPICS-enabled [2] devices via their Channel Access PVs. A dockable GUI component allows the user to graphically create, edit, and monitor the progress of nested loops.

Architecture

The overall architecture of the experiment control software is shown in Figure 3. The application is written in C++ using Qt [3] as the GUI toolkit. Qt provides multi-platform GUI widgets allowing the software to run on Linux, Mac, and Windows platforms. The application is divided into three separate support libraries that provide classes for graphics tools, data types and formats, and a Channel Access scanning library.

GStar is a GUI component library utilizing the Qt framework. It consists of classes related to the user interface. There are classes for displaying images in acquisition windows, graphical annotations, color map and contrast controls, and creating histogram plots.

DStar is a data type and format library. It abstracts details of file formats by providing a single interface. DStar is built on top of the HDF5 [4] and libTIFF libraries. GStar components use DStar's interfaces and data types when using image data.

SStar is a C++ wrapper around the EPICS Channel Access API and implements classes for scanning PVs.

ISBN 978-3-95450-139-7

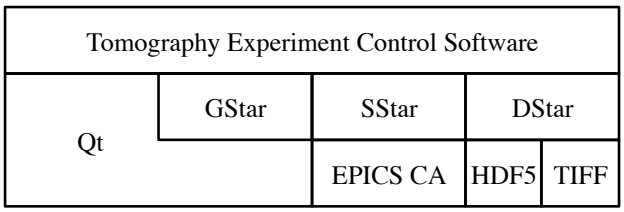

Figure 3: High-level component architecture of the tomography experiment control software.

It has classes for PV data access, and data change and connection change events. It also acts as a multiplexer between the application and PVs so that multiple software components can send and receive data to a single PV without creating multiple connections.

PV values are displayed and manipulated with a small set of PV-aware widgets. The class *PVWidget* serves as the base class from which other GUI components that connect to PVs are derived. For example, *PVLabel* displays PV values like read back data, *PVSpinBox* is used for editing numeric data, and *PVComboBox* is used to select from enumerated values. Initially, a *PVWidget* connects to a PV and queries information such as its data type. If no connection is preset the *PVWidget* will disable itself. When the connection is established the widget enables itself. Each *PVWidget* displays the PV it is connected to when hovering the mouse over it for a few seconds. More complex controls can be derived, such as the *PVMotor* class. This class is used in the calibration display. It consists of a *PVSpinBox* for editing the motor's position, *PVLabel* for reading the motor's read back value, and another *PVSpinBox* for tweak controls.

Multiple area detectors are available and more can be added by creating new Area Detector profiles. An *ADProfile* class is created for each area detector. *ADProfile* serves as a layout with properties for a particular camera, such as binning, gain, etc. EPICS-aware widgets and their relationship to an Area Detector profile are shown in Figure 4.

The scanning core is implemented using the *Command* design pattern [5]. Figure 5 shows the class diagram for this system. The *ScanCommand* is an abstract interface. Each concrete implementation performs a specific set of PV operations using the *SStar::PV* class. The *ScanSequenceCommand* is the basic type of command that performs a series of PV operations in a sequence. This sequence can be repeated in a loop using the *ScanLoopCommand*. The *Loop* interface is a template that generates any sequence of values for the loop, including a fixed step function and a fixed table of values. Multiple scan commands can be grouped together using *ScanMacroCommand*. Execution of commands is handled by the *Scan* class. Primitive operations on PVs, such as setting a value, getting a value, and waiting for the PV to be set to a specific value, are done by the *Scan* command as specified by individual instances of the command.

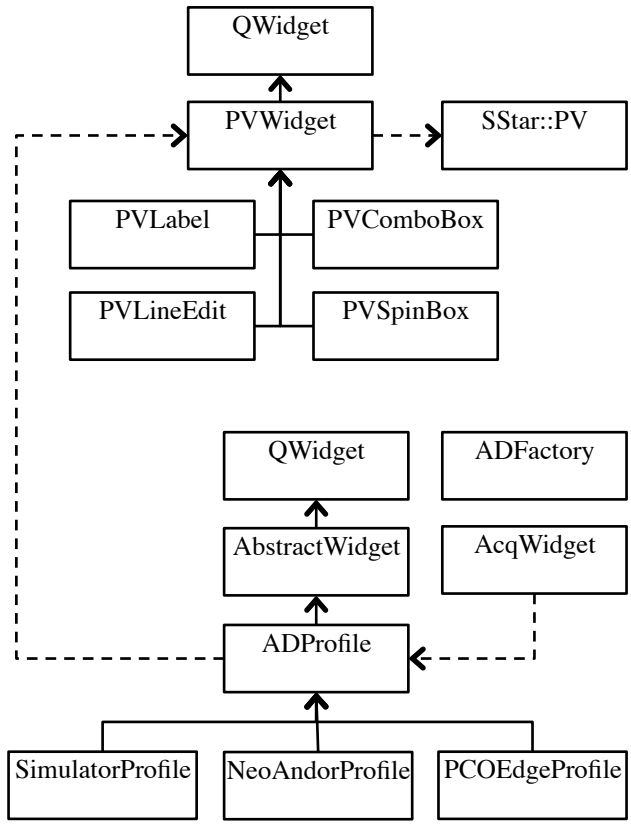

Figure 4: Class diagram of EPICS aware widgets (top) and their relationship to the Area Detector profiles (bottom).

RECONSTRUCTION

Once acquisition is complete, the experiment control software uses the workflow pipeline to transfer data to the Lustre central file system. It does this by queuing a message to begin the file transfer. After data transfer is complete, another message is queued to launch the reconstruction process. The reconstruction code is an implementation of the FFT-based *GRIDREC* algorithm that reconstructs two data sets at once: one in the real part and one in the imaginary part of the FFT [6]. The application is multi-threaded and runs on a single high-end workstation.

The reconstructed data is stored in the same HDF5 file as the raw projection data and experiment metadata. Provenance information about the experiment and processing steps are stored as well. The exact structure of the HDF5 file is well defined and extensively documented by the Scientific Data Exchange format [7].

PIPELINE

A workflow pipeline is used to connect phases of the process. It uses an industry-standard messaging system for reliable task sequencing and triggering. Generic actors handle common tasks such as file transfers. Technique-specific analysis code is implemented or called from custom actors that may be written in Java, C++ or Python.

The pipeline is a series of stages connected by message

ISBN 978-3-95450-139-7

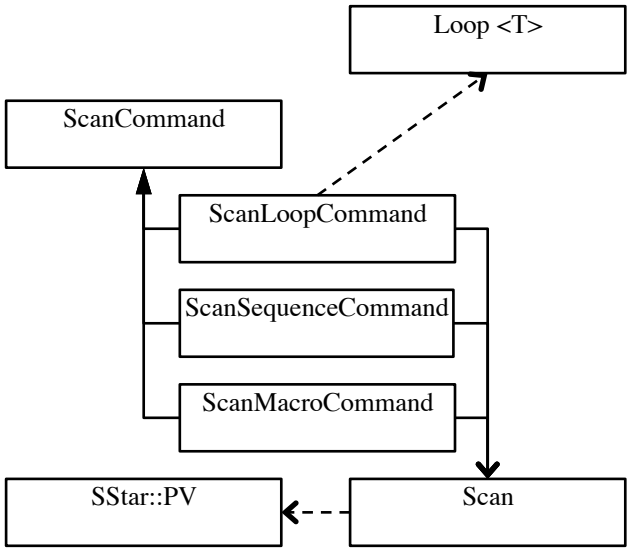

Figure 5: Class diagram of Channel Access PV scanning components in the SStar library.

queues. Stages may interface with the acquisition software, they may be data analysis software, or they may transfer files. Stages may be written in a variety of different programming languages, including C++, Java, and Python. Message queues are how stages are connected. When a stage is done with its task, it queues a message for the next stage in line. That stage will eventually get to that message and process it. The ActiveMQ [8] implementation of the JMS standard is used.

A stage in the pipeline is composed of two classes: the *Director* and the *Actor*. The *Director* is the interface to the message queue. The *Director* handles processing incoming and outgoing job messages, constructing signal messages for the next stage, and handles control signals for killing or restarting jobs. Additionally, the *Director* keeps a history of all messages being processed by the queuing system. The *Actor* is a simple class definition that requires the implementation of two methods: *execute()* and *abort()*. Both methods return a status to the pipeline. Figure 6 summarizes these components.

CONCLUSIONS

The system is in production use at the newly upgraded nano-tomography station at the APS 32-ID beamline. It is soon to be deployed at the existing micro-tomography station at the APS 2-BM beamline. The experiment control software has been well received by staff and users. A typical raw tomographic dataset of 1500 projections at 2048x2048 resolution takes about 5 minutes to reconstruct using a modern 12-core workstation with 32 GB of RAM. The combination of configurable experiment control software, a multi-threaded reconstruction application, and a modern messaging backend that ties together different components allows more efficient utilization of equipment and beam time.

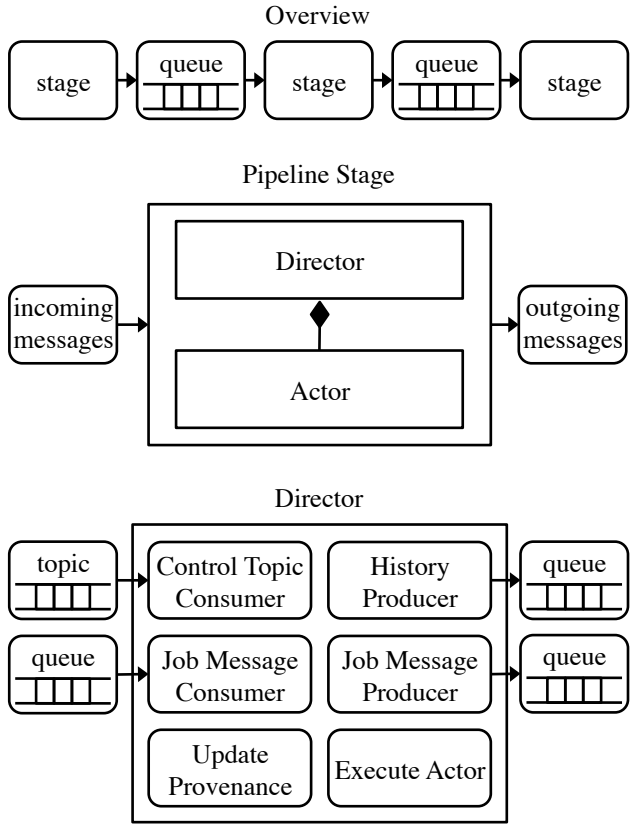

Figure 6: The overview of the pipeline architecture (top). A pipeline stage that performs an action (center). The pipeline *Director* that regulates messages and actives of the pipeline (bottom).

ACKNOWLEDGMENT

We'd like to thank Alex Deriy, Chris Jacobsen, Tim Mooney, Mark Rivers, Giampiero Sciutto, Pavel Shevchenko, Steve Wang, and Xianghui Xiao for their assistance and support.

REFERENCES

[1] Area Detector, `http://cars9.uchicago.edu/software/epics`

[2] EPICS, `http://www.aps.anl.gov/epics`

[3] Qt, `http://qt-project.org`

[4] Hierarchical Data Format version 5 (HDF5), 2000-2010. `http://www.hdfgroup.org/HDF5`

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, (Addison-Wesley Longman, 1995).

[6] M. L. Rivers, "tomoRecon: High-speed tomography reconstruction on workstations using multi-threading," Proc. of SPIE 8506, Developments in X-Ray Tomography VIII, 85060U (2012).

[7] The Scientific Data Exchange, `http://www.aps.anl.gov/DataExchange`

[8] Apache ActiveMQ, `http://activemq.apache.org`

ISBN 978-3-95450-139-7