ACSYS CAMERA IMPLEMENTATION UTILIZING AN ERLANG FRAMEWORK TO C++ INTERFACE

C. Briegel, J. Diamond, FNAL[†], Batavia, IL 60510, U.S.A.

Abstract

Multiple cameras are integrated into the Accelerator Control System (ACSys) utilizing an Erlang framework. Message passing is implemented to provide access into C++ methods. The framework runs in a multi-core processor running Scientific Linux. The system provides full access to any 3 cameras out of approximately 20 cameras collecting 5 Hz frames. JPEG images in memory or as files are provided for visual information. PNG files are provided in memory or as files for analysis. Histograms over the X & Y coordinates are filtered and analyzed using Root. This implementation is described and the framework is evaluated.

INTRODUCTION

The Fermilab control system's [1] [2] camera implementation consists of a 1U X86 multi-core computer connecting a dedicated 1G Ethernet to several Prosilica cameras. A framework written in Erlang runs on a Scientific Linux (32-bit or 64-bit) operating system to provide access and control to the control system. The user has the choice of writing interfaces for data acquisition in either Erlang or C++ .

The utilization of cameras for instrumentation has evolved with research and development of electron accelerators. The application is targeted for the test facility proposed as ASTA [3] (Advanced Superconducting Test Accelerator). The cameras range is between 1.3 megapixel to 5 megapixel with up to 12-bit resolution.

REQUIREMENTS

The following is a summary of the important requirements. The requirements document [4] specifies up to 3 cameras can capture and write to disk loss-less PNG images at 1Hz without gaps. The cameras can be triggered with any clock event plus delay. These images must be accessible via the control system as well as available to the user for off-line analysis. The images must have the capability for background subtraction. Histograms of the X and Y axis will be evaluated with a Gaussian fit providing the mean, standard deviation, peak intensity, sigma, and "goodness of fit." Essentially, all the available control and read back functions available via the camera interface should map to parameters accessible by the control system.

[†]Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

ERLANG INTERFACE

The Erlang-based front end framework [5] provides a very reliable interface for C++ via a standard Erlang-C message passing protocol. The framework provides a set of generic functionality to monitor and control the framework itself.

The user adds functionality via a configuration file as in Figure 1. This file loads the added functionality (5 methods in this case) to the framework. The integer 35 in Figure 1 is the object id used by the request to route the query to the correct method. The data is typed with a maximum return size which is checked by the framework. The method is specified with associated descriptive text.

```
{daq,
  [{apps,[]},
 {device_list, [
  {35, cexternal, {"prosilica",cdev, ["nothing"]
   [{'UInt32',256,readData,"Prosilica Rd"}
    {'Char',32768,readDataArray,"Prosilica Ascii"},
    {'Float',256,readsetData,"Prosilica Rd/Wt"},
    {'UInt32',4096,readDataArray32,"Prosilica UI32"},
    {'Float',4096,readDataArrayFP,"Prosilica FP"}]}}
   ]}
 ]
```

Figure 1: Configuration file.

The user's code registers these callbacks as in Figure 2. The methods are passed request information to index the data to be returned or set. The framework automatically detects illegal requests before it can be received by the user's methods. The framework enables the user to easily add devices for all attributes of the control system.

```
registerMethods(0,*this,&prosilica::readData,
                &prosilica::setData);
registerMethods(1,*this,&prosilica::readDataArray,
                &prosilica::setDataArray,8192*32);
registerMethods(2,*this,&prosilica::readsetfpData,
                &prosilica::setfpData);
registerMethods(3,*this,&prosilica::readDataArray32,
                &prosilica::setDataArray32,8192*32);
registerMethods(4,*this,&prosilica::readDataArrayFP,
                &prosilica::setDataArrayFP,(4096+11)*32)
```

Figure 2: User callbacks.

To track status and errors, the framework provides a revolving set of log files for user diagnostics. Also, the Erlang framework provides alarm announcements to the owner's email every 12 hours reporting potential problems or abnormal behaviour.

CAMERA IMPLEMENTATION

The camera front end utilizes the Erlang framework as well as several C++ libraries to complete the solution. All of the camera specific code is written in C++. Each active camera (up to 3) instance has a prosilica class instantiated. This creates a unique object id corresponding to the instantiated class.

The following describes how the libraries were utilized to make up the implementation.

PvLib

Allied[6], the manufacture of the Prosilica camera, provides a library called the PvAPI driver. This library provides a proprietary interface to Prosilica cameras with varying attributes depending on the camera model.

Since other manufacturers were evaluated, an attempt was made to use a camera “agnostic” library via the GenICam Standard [7]. This proved to be difficult since the manufacturers did not embrace the interface at the user API. A class was created for the Prosilica cameras utilizing the PvLIb enabling a camera specific class for each unique camera manufacturer. Currently, the software only implements a single vendor solution.

Recently an additional library, “VIMBA” [8], has been implemented for the Prosilica camera following the GenTL Specification that compiles to the GenICam Standard. Although we have not yet investigated this interface, the library potentially offers a generic interface for multiple manufacturers.

Most of the controls for the camera are directly available in the hardware. The hardware ROI (region of interest) and binning posed the most difficult to manage for the user. First, when binning is implemented in hardware, the pixels are all added to provide a brighter pixel and the image gets brighter. To keep the image the same, software binning conditionally averages or drops pixels. The software binning does not affect the actually image collected for archiving and future analysis. It only affects the memory image primarily available for display. Also, the corresponding software ROI affects only the memory image. The image sent to disk contains the entire hardware image. These software controls enable the user to manipulate the image for display without affecting the quality and content of the image for analysis.

wxLib

As part of the early development, it was desirable to see the image. The wxWidgets [9] library was used to provide a X11 display on a specified IP address. This provided a temporary display for focusing and understanding the image until applications could be developed.

While the wxWidgets library successfully displayed 1 Hz full resolution images and provided a raw file image for diagnostics, the feature suffered from not being able to restart the environment redirecting the display to a new IP address. Thus, the implementaion is conditionally turned off at this time.

PngLib

PNG [10] files are used to provide a loss-less compressed image either in memory or written to disk. Ten writer tasks are available to accept a queue of images (up to 100). After the specified number of images is collected, the writer tasks asynchronously processes an image from the queue until all are done. The files are written to a user specified directory on the local drive. This directory becomes the user’s image run with unique filenames based on the time and date.

A secure mechanism is provided to transfer the entire directory of files to any user’s local disk. A parameter is set to create a tar-ball of the directory saved into an NFS mounted directory. Any browser can then upload the tar-ball into the local machine for off-line analysis.

One-shot PNG files can be written to memory and acquired over the network by the control system. For diagnostics and analysis, these images may also be written to disk.

Since the compression of PNG files is compute-intensive, no files are written until all requested images are captured to prevent data collection errors returned by the PvLib. The error correlates to high CPU utilization and the fundamental cause is not understood at this time. The upgrade from 2 cores to 8 cores may eliminate this problem.

JpegLib

JPEG [11] files are used to provide comfort displays for focusing and image location. JPEG files can be significantly compressed, but image data is lost depending on the quality (1-99) requested. The JPEG image conditionally can be sent to disk consisting of the last 10 images for diagnostics.

Generally, the JPEG images are kept in memory for control system access. A maximum size can be specified to optimize the control system’s access to a single request for data (typically 32000 bytes). When the max size is specified, a heuristic is invoked to change the image size by modifying the quality and software binning of the image. The software bin can either be the average of a square pixel or a selected single point within the square. Further, to minimize the size or increase the quality, the user can specify the software ROI.

Root

ROOT [12] is the physic analysis package maintained by CERN and supported at Fermilab. There is a high confidence in the results of ROOT and ROOT can adjust its boundaries to provide a focused region for the fit. Also, ROOT can easily change the type of fit with a slightly modified algorithm if desired.

CAMERA APPLICATIONS

There are effectively three applications. The primary user application is written in Java and can run either natively or on a console server. This application makes use of the JPEG image for feedback to the user and can manipulate all aspects of the camera. It can save and display one-shot PNG files. It displays histogram plots and provides local analysis of the histogram Gaussian fit contrasting the front end's fit analysis.

The ACSys parameter page can manipulate all parameters and provides access to generic parameters for the Erlang framework. This provides immediate access to all parameters in a standard application.

A set of synoptic displays [13] [14] provides an easy way to check the selected camera's information. It also provides histogram plots, Gaussian fit plots and the Gaussian analysis.

RESULTS

The camera system has been used to see the first beam achieved this summer [15]. The camera system has been improved to accommodate operational needs and is becoming an integral instrument for measuring the beam.

The utilization of software versus hardware ROI and binning is an effective mechanism to reduce the data size without sacrificing the quality of the image files.

The Erlang framework has proven to be very mature and robust. Erlang is a functional language that minimizes many run-time failures. Most of the failures have been attributable to the C++ coding and the utilization of the C++ libraries. As a user of the framework, these are the errors I expect, tolerate and fix. Thus, the framework has been a pleasant experience to integrate my C++ application into the control system.

ACKNOWLEDGEMENTS

The Erlang framework was implemented by Rich Neswold, Dennis Nicklaus, and Jerry Firebaugh from the Femilab's controls group.

REFERENCES

[1] J. Patrick, "ACNET Control System Overview," Fermilab Beams-doc-1762-v1, http://beamdocs.fnal.gov/AD-public/DocDB/ShowDocument?docid=176

[2] K. Cahill, L. Carmichael, D. Finstrom, B. Hendricks, S. Lackey, R. Neswold, J. Patrick, A. Petrov, C. Schumann, J. Smedinghoff, "Fermilab Control System," Fermilab Beams-doc-3260-v3, http://beamdocs.fnal.gov/AD-public/DocDB/ShowDocument?docid=326

[3] http://asta.fnal.gov

[4] M.Church, "Software Specification for Prosilica Cameras at ASTA (in progress)," Fermilab (July 27,2012). [3] http://synoptic.fnal.gov

[5] D. Nicklaus, "An Erlang-based Front End Framework for Accelerator Controls," ICALEPCS (2011); Grenoble, France.

[6] http://www.alliedvisiontec.com

[7] http://emva.org

[8] http://www.alliedvisiontec.com/us/products/software/vimba-sdk.html

[9] http://www.wxwidgets.org

[10] http://www.libpng.org

[11] http://www.jpeg.org

[12] http://www-root.fnal.gov

[13] http://synoptic.fnal.gov

[14] A. Petrov, Synoptic User Guide, http://beamdocs.fnal.gov/AD/DocDB/0035/003567/002/synoptic-1_2.pdf

[15] http://www.fnal.gov/pub/today/archive/archive_2013/today13-07-02.html