The DECam Data Acquisition and Control System

K. Honscheid[a], J. Eiting[a], A. Elliott[a], J. Annis[c], M. Bonati[b], E. Buckley-Geer[c], F. Castander[d], L. da Costa[i], M. Haney[e], W. Hanlon[e]. I. Karliner[e], K. Kuehn[f], S. Kuhlmann[f], S. Marshall[h], M. Meyer[e], E. Neilsen[c], R. Ogando[i], A. Roodman[h], T. Schalk[g], G. Schumacher[b], M. Selen[e], S. Serrano[d], J. Thaler[e], W. Wester[c]
for the Dark Energy Survey Collaboration

[a]The Ohio State University, Columbus, OH, USA; [b]National Optical Astronomy Observatories, Tucson, AZ, USA; [c]Fermi National Accelerator Laboratory, Batavia, IL, USA; [d]IEEC, Barcelona, Spain; [e]University of Illinois, Urbana, IL, USA; [f]Argonne National Laboratory, Argonne IL USA; [g]University of California Santa Cruz, Santa Cruz, CA, USA; [h]SLAC, Menlo Park, CA, USA; [i]Observatorio National, Rio de Janeiro, Brazil

ABSTRACT

In this paper we describe the data acquisition and control system of the Dark Energy Camera (DECam), which will be the primary instrument used in the Dark Energy Survey (DES). DES is a high precision multi-bandpath wide area survey of 5000 square degrees of the southern sky. DECam currently under construction at Fermilab will be a 3 square degree mosaic camera mounted at the prime focus of the Blanco 4m telescope at the Cerro-Tololo International Observatory (CTIO). The DECam data acquisition system (SISPI) is implemented as a distributed multi-processor system with a software architecture built on the Client-Server and Publish-Subscribe design patterns. The underlying message passing protocol is based on PYRO, a powerful distributed object technology system written entirely in Python. A distributed shared variable system was added to support exchange of telemetry data and other information between different components of the system. In this paper we discuss the SISPI infrastructure software, the image pipeline, the observer interface and quality monitoring system, and the instrument control system.

Keywords: Dark Energy Survey, DECam, data acquisition, control system, camera, survey, Blanco, CTIO

1. INTRODUCTION

In an effort to find out why 95% of the universe is dark, a number of large scale surveys of the sky with new ground based instruments have been proposed. Among the most promising initiatives is the Dark Energy Survey (DES). DES is a next generation sky survey aimed directly at understanding this mystery. Over 525 nights DES will survey a 5000 sq-degrees area of the sky in five filter bands using a new 3 deg^2 camera (DECam) on the Blanco 4m telescope [1, 2]. The survey area will overlap with the South Pole Telescope [3] and the VHS VISTA survey [4]. DES is designed to measure the dark energy equation of state parameter with four complementary techniques: galaxy cluster counts, weak lensing, angular power spectrum and type Ia supernovae. The combination of these techniques will produce a factor of ~4.6 improvement over current experiments in the figure of merit defined by the Dark Energy Task Force. The Dark Energy Camera, the primary instrument for DES is

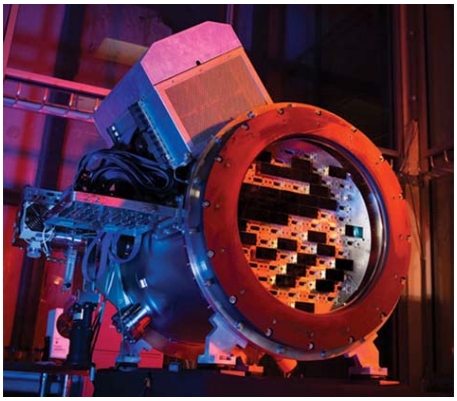

Figure 1: The DECam test vessel with a partly populated focal plane and 1 of 3 electronics crates. The optical corrector, hexapod, filter changer and shutter are not shown.

currently being assembled at Fermilab. A photo of the test vessel with a partly assembled focal plane is shown in Figure 1. In a little more than a year DECam will be mounted at the prime focus of the Blanco 4m telescope at CTIO. The instrument consists of a 3 square degrees focal plane covered by 62 2kx4k science CCDs and 12 2kx2k CCDs for focus and guiding, a five element optical corrector, a hexapod to position the camera, up to eight filters, a modern readout and control system, and the associated infrastructure for operation in a new prime focus cage. A more detailed description of the DECam instrument can be found in [1].

2. SISPI

The DES mountain top data acquisition system is called the Survey Image System Process Integration (SISPI). A schematic overview of the system is shown in Figure 2. An exposure sequence starts with the observer sending a request to the Observation Control System or OCS in the center of the block diagram. Exposure parameters such as coordinates, exposure time etc can be entered manually by the observer, come from a script or are submitted by an automated process (ObsTac) that knows the survey history and strategy as well as the current observing conditions. The latter is available only in survey mode. Upon receipt of the exposure request the OCS queries the state of the instrument and sends requests to the telescope control system to slew the telescope and to the filter changer mechanism to load the filter requested for the next exposure.

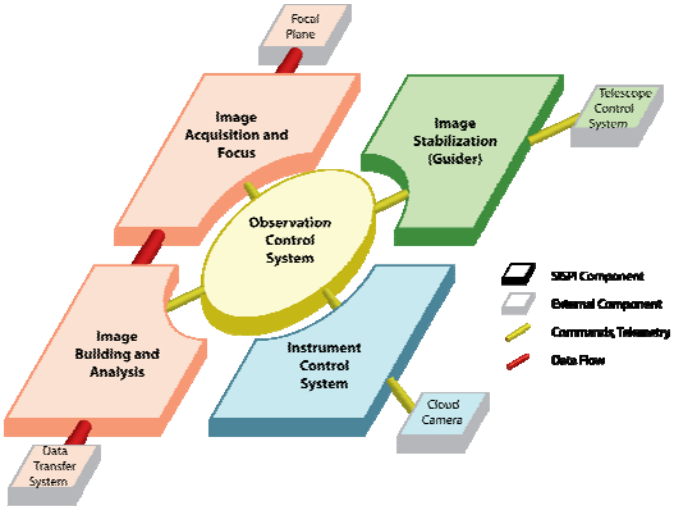

Figure 2: Block diagram of the DECam read-out and control system (SISPI).

The OCS waits for these activities to complete before prepping the CCDs and the front end electronics to receive a new image. At the end of an exposure the OCS closes the shutter, assigns an image builder process to assemble all pixel streams into the full image and triggers the electronics to readout the CCDs. Image data flows from the focal plane CCDs and the Monsoon front end electronics to the Image Acquisition and Image Builder systems before it is recorded on a storage device and handed over to the DES Data Management system. The image acquisition system is pipelined and the OCS can start processing the next exposure as soon as all pixels are digitized. The performance requirements for the DECam read-out and control system are set by the size of the focal plane, the read-out time and the typical DES exposure time of 100 seconds. With 62 science CCDs or 520 Mpixels and 16 bits per pixel the size of a DECam exposure is approximately 1 GByte. At a rate of 250 kpix/s it takes about 17 seconds to transfer the data from the focal plane to the computers of

the Image Acquisition system. During this time the telescope slews to the next position. In order to maximize survey efficiency SISPI is designed to operate within these constraints and not to add any additional dead time.

The SISPI architecture can be broken up into four functional units: the image pipeline, the instrument control system, data quality monitoring and the user interfaces including the observer console. We will discuss each of these components as well as the underlying software infrastructure in the next sections.

3. INFRASTRUCTURE SOFTWARE

The division of SISPI in an application and an infrastructure layer is shown schematically in Figure 3. SISPI is coded in Python with some elements of the instrument control system written in Labview. Supported operating systems include Linux and Max OS X. Due to its distributed architecture inter-process communication takes a central place in the design of the SISPI infrastructure software. Our solution is based on the Python Remote Objects (PYRO) software package developed by I. de Jong [5]. PYRO is an advanced and powerful distributed object technology system written entirely in Python. It allows objects to interact just like normal Python objects even when they are spread over different computers on the network. PYRO handles the network communication transparently. PYRO provides an object-oriented form of remote procedure calls similar to Java's Remote Method Invocation (RMI). A name server supports dynamic object location rendering (network) configuration files obsolete. Using the name server SISPI processes can locate and establish communication links with other processes irrespective of the underlying hardware architecture; an important feature for the any multi-processor communication system. In a test system, for example, most of the applications might share one or two computers whereas on the mountain everything is spread out over many computers to maximize performance.

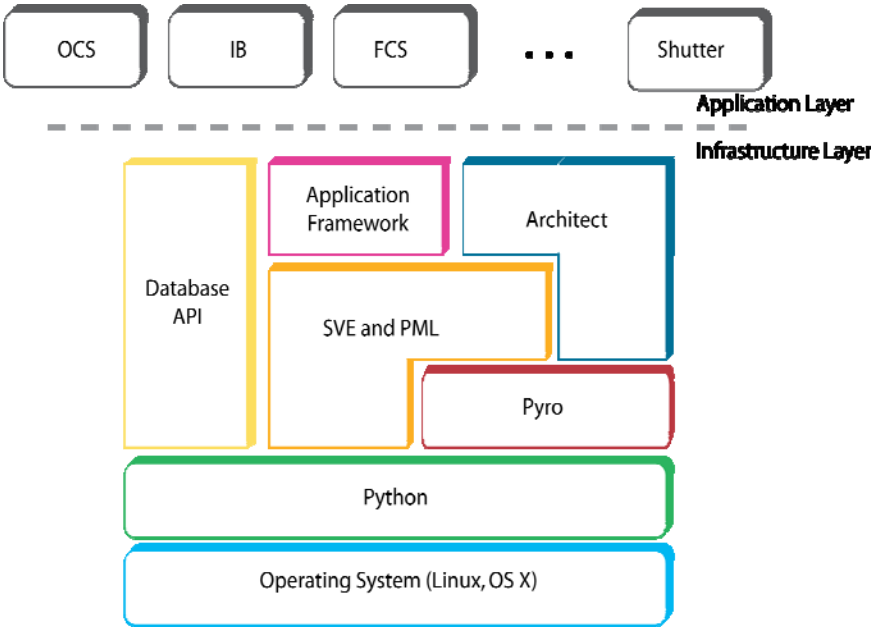

Figure 3: Application and Infrastructure layers of the DECam read-out and control system (SISPI).

We distinguish between *Command* messages and *Telemetry* data. Commands are used to request information from a remote application or to activate a remote action. The Command or Message Passing system is implemented using a Client-Server design pattern with a thin software layer (PML) on top of PYRO. PML

introduces the concepts of *Component* and *Device* and with this a uniform naming scheme for SISPI applications.

The telemetry system is based on the publish-subscribe design pattern using ideas similar to data distribution services, an emerging standard for publish-subscribe middleware for distributed systems [6]. Again built upon the core functionality provided by PYRO, SISPI introduces a concept called Shared Variables. Consisting of a client stub library and a central server (Shared Variable Engine, SVE) this system allows user applications to publish information such as a temperature reading or the readout status to a virtual data space. Other applications can subscribe to information placed in this virtual data space and will receive updates whenever a publisher submits a new value. Publishing and subscribing to a shared variable are completely decoupled. A publisher needs no knowledge of who will subscribe to this variable and vice versa. The shared variable system supports asynchronous callbacks, guaranteed delivery, multiple publishers of the same shared variable and group subscriptions. Additional options are available but a detailed description of the SVE API is beyond the scope of this paper. Production versions of the SVE and the PML messaging layer are complete and both are used for all DECam prototype tests and studies.

3.1 Configuration

Initialization and configuration of a complex distributed system such as SISPI is a multi-step process. SISPI employs a subversion-based [7] code management and distribution system to support concurrent code development. We have selected eUPS [8] for installation and version management. eUPS organizes SISPI applications into products and supports simple command line tools to select specific versions with automatic configuration of environment variables such as PATH. The eUPS concept of current versions allows the code manager to define a standard configuration for regular use while software developers can select a specific version of a product to debug and test new features.

With the software installed it is the job of the Architect, the SISPI configuration system, to load and start the SISPI processes in the correct order and with the correct arguments. A SISPI configuration is described by an initialization file like the one shown below. The collection of processes started by the Architect is called a SISPI Instance. Multiple instances can be run by the same user and on the same computers as long as there are no hardware conflicts.

```
image_basename = Image_${SISPI_INSTANCE}      ← Define variables, use shell symbols
[Nodes]
  [[cosmos7]]                                 ← Specify which computers (Nodes) run
    host = cosmos7.mps.ohio-state.edu           which process (Role)
    roles = FCS,IB1,IB2,PSL,CONSOLE,OCS
            GUISERVER,SHUTTER,SHUTTERGUI
  [[cosmos6]]
    host = cosmos6.mps.ohio-state.edu
    roles = PANA, DHSA          # bkp1
  [[cosmos12]]
    host = cosmos12.mps.ohio-state.edu
    roles = PANB,DHSB           # bkp3
[Roles]
  [[PSL]]                                     ← Roles have names. The program is
    product = PSL                               identified by its eUPS product name
    application_args=-default_delay,3000      ← Choose arguments to be passed to role
  [[PANA]]
    product = PanviewHelper
    fpaname = _decam63_vib2_bkp4              ← Configuration constants can be defined
  [[DHSA]]
    product = DHS
    im_name = %(image_basename)s              ← Internal variables are resolved
  [[OCS]]
    product = OCS                             ← Specific eUPS version of a product
    version = 1.4.0                             to use
    xterm = True                              ← Start process in a separate xterm
    xterm_args = -iconic,                     ← Set xterm parameters
```

The configuration files use standard Windows .ini file notation [9] and the Architect uses the configObj Python module [10] to parse the configuration file. The example in the text box above shows some of the options available for SISPI configuration files. The computers that participate in this instance are specified in the [[Nodes]] section in which we also list how the application processes (Roles) are distributed. In the [[Roles]] section each application is identified by its eUPS product name. The Architect can pass command line arguments specified in the .ini file when it starts an application. Variables are supported as well as configuration constants that are passed to the application as a Python dictionary. The Architect itself consists of a manager application and architect_node processes started on each node in the instance via ssh. A management interface implemented by the Architect provides full process control. Individual roles can be stopped and restarted. With a centralized shutdown command an entire SISPI instance can be stopped. The Architect is integrated in the SISPI web-based user interface system discussed later in this paper. A screen shot of the Architect Console is shown in Figure 4. Development of the Architect is complete and the SISPI configuration system is fully operational.

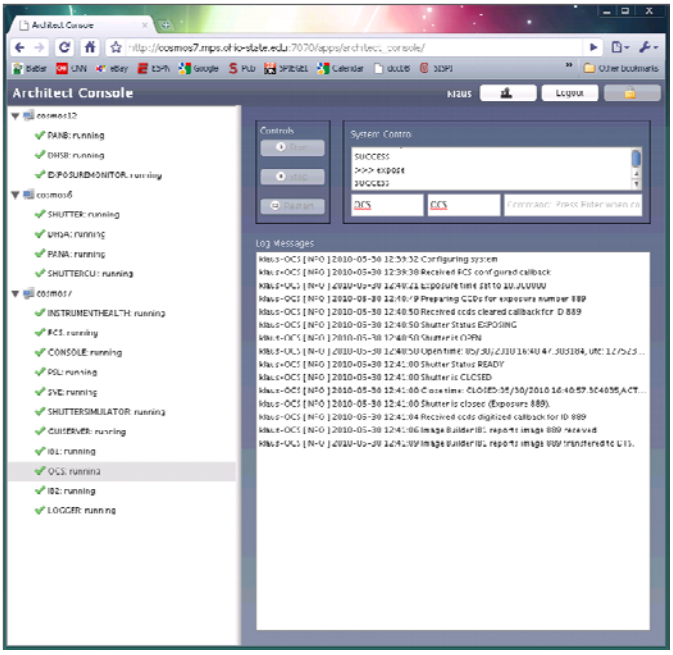

Figure 4: The Architect Console showing the active nodes and roles. The top right panel allows user input and the lower panel on the right shows log messages.

3.2 Facility Database

The SISPI facility database stores all non-image data that are relevant to the operation of DECam. There are a number of different types of data that will be stored including camera calibration and configuration information, telemetry data such as environmental information from the instrument control system, alarm and error messages, information from the cloud camera, and log messages about all events that take place during camera operations. Our implementation of the facility database is based on PostgreSQL [11]. PostgreSQL is a freeware database that is widely used in the Linux community as well as at both NOAO and FNAL. We have developed a set of routines and libraries written in C++ and in Python using the psycopg2 and sqlalchemy modules [12] that hide SQL and other database details from the application programmer. Database access is fully integrated with the SISPI Application Framework discussed in the next section. The database will be located on Cerro Tololo. A tool was developed to provide daily backups to CTIO and Fermilab. The data will also be made available to DES Data Management.

3.3 Application Framework

The SISPI Application Framework is a Python class that serves as a base class for all SISPI application. It gives the same basic structure to all applications and brings together all of the SISPI services such as SVE, PML, Alarms, Logging and configuration management (Architect). The Framework manages all resources centrally which allows it to unsubscribe all shared variables and to close all PML connections when the

application is about to exit. Proper exit handling is critical for SISPI's ability to stop and restart individual processes without the need to end the entire instance. Additional functionality provided by the Application Framework include a heartbeat that can be used to monitor the overall state of the system and a standardized management interface with process control functions and access to an application state variable. Interfaces to the alarm and constants database are also provided by the framework. With the Application Framework we were able to reduce the complexity of application development significantly.

4. THE IMAGE PIPELINE

Built on top of the infrastructure software, the Observation Control System or OCS is the central component of SISPI coordinating all aspects of camera operation and the observation sequence. Connected to the OCS is the tactical observation package (ObsTac) which determines a sequence of pointings for the telescope based on a number of inputs. A typical DES algorithm would take as inputs: 1) current date and time to compute

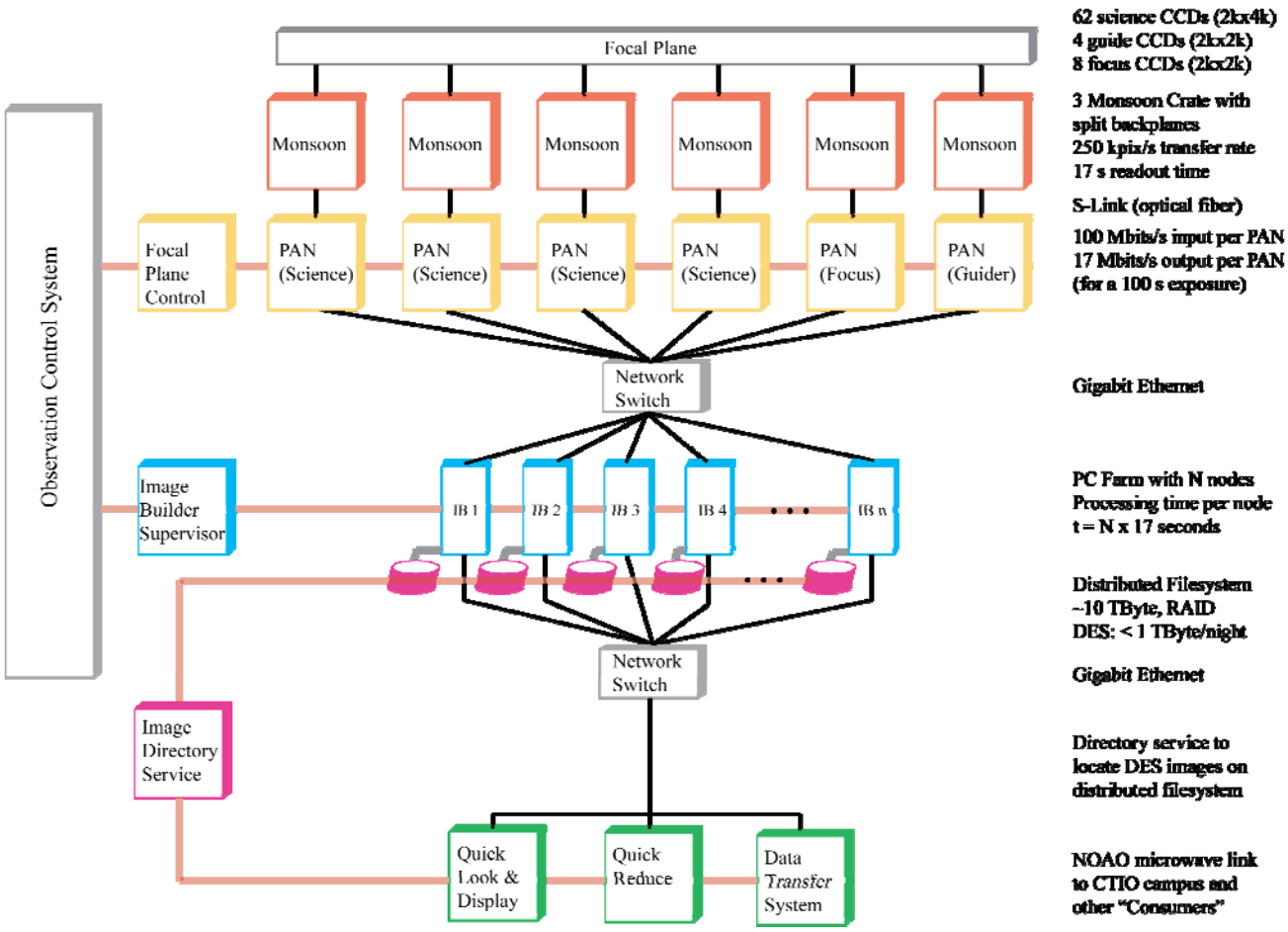

Figure 5: Data Flow architecture of the DECam read-out system

moon position and twilight status, 2) survey status, in the form of fields previously observed, and 3) the current observing conditions as reported by the cloud camera and other instruments. The algorithm will then locate candidate tiles from the survey status information, and compute air-masses for the candidates. Given this information it computes a metric for each candidate and then chooses the best tile to observe. OCS

receives this information and proceeds to take these images unless the sequence is overwritten by the observer. Besides survey mode (ObsTac) and manual mode the OCS also supports simple scripting of a given series of exposures and provides standard features such as general purpose dither patterns. Figure 5 shows the data flow architecture of the DECam read-out system. At the end of an exposure data is read from the CCDs and digitized using Monsoon based detector head electronics [13]. A detailed description of this system and of some of the newly developed front end modules for DES can be found in [14]. The front end electronics is arranged in three crates each with a split backplane to improve system throughput. These CCD readout crates will be located in the prime focus cage in thermally controlled housings. These housings will maintain the CCD electronics at a constant temperature to ensure constant gain performance. From the Monsoon crates image data are transferred to pixel acquisition nodes (PAN) via an optical data link based on the S-Link specification [15]. At a pixel rate of 250 kpix/s readout of the entire focal plane takes about 17 seconds. Our readout software (panview) is based on ArcVIEW package developed by M. Bonati at CTIO [16].
The focal plane is configured such that the data from all 8 focus and alignment CCDs are read by the same PAN. This processor will analyze the focus images in real time to compute adjustments to the hexapod support system in order to maintain the best possible image quality. This arrangement simplifies the architecture and it minimizes the movement of the data, thereby maximizing the time available for processing before the next image is taken. In order to complete the hexapod motion before the beginning of the next exposure, any adjustment commands must be issued before 9.5 seconds after the shutter closes. Since it takes about 8.5 s to readout the 2kx2k alignment CCDs the focus algorithm has to complete in less than 1 second. We are considering implementing region of interest readout using know star positions to reduce the readout time. A prototype algorithm has been developed and readout and algorithm timing measurements are underway. An additional PAN computer is allocated to read the guide CCDs and to execute the guide algorithm. To provide guide signals to the telescope, the guide CCDs will be operated independently of the image CCDs by using a separate Monsoon system for their readout. In order to achieve the required update frequency of the guider correction signals of about 1 Hz the guide CCDs will operate in window or region-of-interest (ROI) mode which has been successfully tested in our studies with the Multi-CCD Test Vessel at Fermilab. The guider algorithm itself was tested in a dedicated observing run using a DECam 2kx2k CCD mounted to the CTIO 1m telescope.

Using high speed Gigabit Ethernet links each PAN sends its data to the Image Builder (IB) system which is implemented as a processor farm. The data transfer protocol is optimized for performance and instead of standard TCP/IP sockets we are using netcat [17] to achieve transfer times of less than 10 seconds for the entire image (1 GB). Throughput is further enhanced by using a memory based filesystem (tmpfs) for temporary files. We are currently evaluating the use of compression schemes such as FITS tile compression.
Each image builder process, written in Python, receives data from every PAN and when an image is complete, it assembles the exposure data in a multi-extension FITS format using pyFits. While processing the image the IB completes the primary FITS header by inserting information such as telescope position, airmass, filter etc. published by other applications. Before the data is written to disk each image is passed through a quality insurance process that analyses the image data and determines standard quantities such as sky noise and seeing. Assuming a multi-core CPU is used for the IB processor farm allowing data I/O to proceed in parallel, the available processing time t_{Proc} is simply $t_{Proc} = t_{Exp} \times N$ where t_{Exp} is the exposure time and N is the number of IB nodes. We have estimated that 4-5 nodes will be sufficient for DES but the system can easily be expanded. The IB nodes are controlled by the Image Builder Supervisor (IBS) which is implemented as part of the OCS. The IBS maintains a queue of available IB nodes which is used by the OCS to instruct the PANs of where to send the next image data.
Images are stored on raid disks local to each image builder. An Image Directory Service provides a catalog of available images and provides users and other applications, for example Quick Reduce (discussed below), with access tools. Once again using netcat or similar tools for the data transfer we obtain significant performance gains compared to an nsf based system. The NOAO Data Transport System (DTS) is used to

transfer data from the Cerro Tololo mountain-top to the CTIO campus in La Serena via a 155 Mbits/s microwave link. From there DTS transfer the image data to the DES Data Management system at NCSA at the University of Illinois.

5. INTRUMENT CONTROL SYSTEM

Hardware monitoring and control of the DECam instrument is the responsibility of the instrument control system (ICS). Shown schematically in Figure 6, the system can be divided into two parts. Critical systems such as the liquid nitrogen cooling system and the monitor system for the front end electronics crates are always on. Fail safe systems are implemented in hardware. Control loops and monitor functions are programmed in Labview and use programmable automation controllers from National Instruments (a mixture of Compact RIO and FieldPoint systems). These applications operate stand alone. The only other SISPI component they use is the DECam database to archive alarm message and telemetry information. A local disk

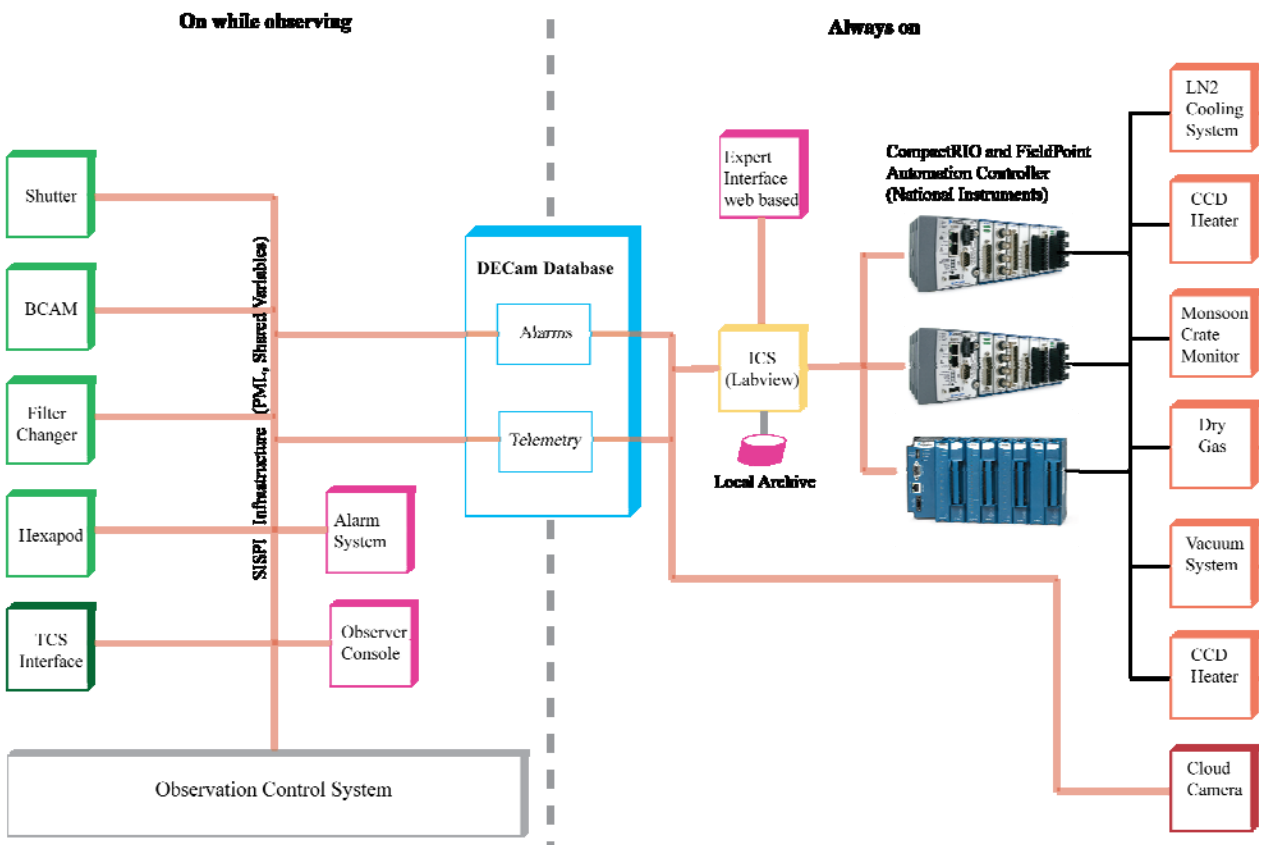

Figure 6: The DECam Instrument Control System.

is used should the database is unavailable. No other SISPI services are used. We use Labview web services to make the expert user interfaces available remotely.

We use a similar model to integrate other mountain top instrumentation. This includes in particular the radiometric all-sky infrared camera (RASICAM) currently under construction by the DES collaboration [18]. This camera allows real-time quantitative assessment of night sky and determines whether the conditions are photometric. Via the DECam database this information is made available to ObsTac as an input for the exposure selection algorithm.

The second part of the instrument control system consists of components that participate more actively in the image acquisition process such as the shutter and the filter changer mechanism. These applications are started by the Architect and require other SISPI components to be fully functional. They typically consist of a network enabled micro controller with the firmware written in C The hardware controller interacts with a Python process via a TCP/IP socket connection. The DECam shutter developed by the University of Bonn and the filter changer mechanism built by the University of Michigan [19] have been integrated so far. We complete these applications as the hardware arrives at Fermilab.

The connection between DECam/SISPI and the Blanco telescope control system [20] is accomplished by the TCS Interface process shown on the left in Figure 6. While similar in nature the two control systems use different protocols and the TCS interface supports bi-directional flow of control and status information. It includes algorithms to convert between the different communication protocols used for the telescope and DECam.

6. QUALITY ASSURANCE

Continuous monitoring of both the instrument and the image quality is required to control systematic uncertainties to achieve the science goals of the Dark Energy Survey and to allow continuous, error-free operation of DECam. Several SISPI components are designed to implement these quality assurance procedures. Image-Health, a first check of the image quality of every exposure, has already been discussed as part of the image builder process. In addition, a sample of the images will be processed on the mountain top with a complete reconstruction and analysis pipeline that uses algorithms identical or in some cases very similar to the procedures used by the full Data Management processing pipeline. This Quick Reduce pipeline is being developed by DES-Brazil and has already been successfully tested using simulated images. Quick Reduce consists of a flexible framework written in Python that can call a user defined list of (astronomy) algorithms to process the image. Results are collected in an internal database and a combination of Matplotlib [21] and the Django framework [22] are used to present them graphically in a web browser as shown in Figure 7. Quick Reduce will be available for both survey operations and community observing.

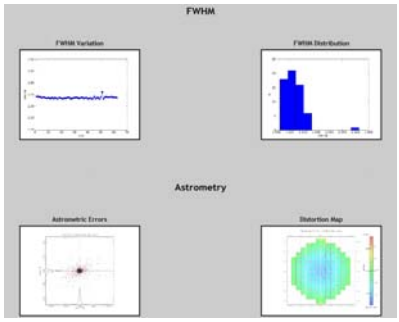

Figure 7: Sample output plots from the Quick Reduce pipeline.

Additional quality assurance tools available to the observer include a real time display that automatically shows a down-sampled image of every exposure and the observer workstation. SISPI will copy every image to the observer workstation where it can be accessed by standard tools such as ds9 or iraf for an in depth interactive analysis. The observer can choose to process an image with his private, customized algorithms. These operations are completely decoupled from the SISPI image pipeline and do not affect the overall throughput. We have developed a a prototype image server interface shown in Figure 8 that assists the observer in locating the images of interest. In a future version the observer will be able to select images based on exposure type, filter etc.

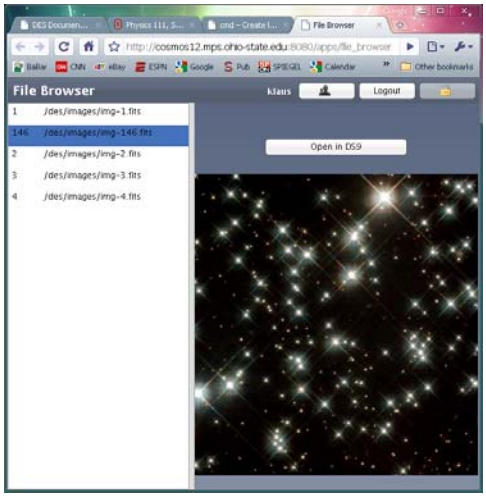

Figure 8: Prototype Image Directory Service

We also continuously check the information provided by the instrument control system. Instrument-Health is a SISPI application that monitors the state of the instrument. Deviations from normal operating conditions cause alarms that will be logged in the database and displayed on the alarm panel of the

observer console. Instrument-Health publishes select quantities as shared variables to be displayed on the observer console either as a function of time or as a function of position within the focal plane, where possible.

7. USER INTERFACE AND OBSERVER CONSOLE

The SISPI GUI architecture follows the Model-View-Controller (MVC) pattern first developed for Smalltalk but now in common use for large applications [23]. The MVC pattern is based on the realization that all applications are, essentially, interfaces that manipulate data. For SISPI the controller component is provided by a GUIserver and standard web browsers are used to render the view. Using a web browser provides many desirable features such as platform independence, remote access, a large number of 3rd party tools, and a certain level of security. There are, however, some concerns regarding usability and performance in a real time environment. Recent advances in browser technology have addressed this issue. Thanks to faster rendering and JavaScript execution as well as new standards such as HTML5 and the Web-Socket API that

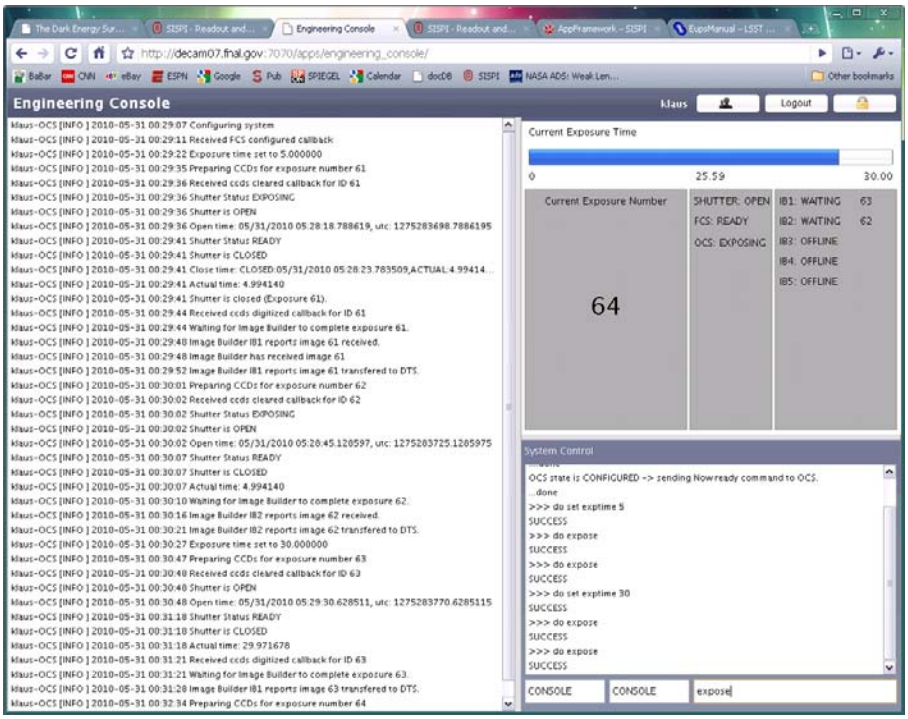

Figure 9: The engineering console showing the remaining exposure time and the status of the image pipeline. The left panel shows messages from the OCS and the lower panel on the right implements a console interface.

allow for advanced functionality such as bidirectional socket communication and 2D drawing contexts, web browsers are now on a near equal footing with desktop GUI toolkits. The SISPI GUIserver is based on the Twisted framework. The client side (web browser) code is based on HTML/CSS and JavaScript. We use SproutCore, a modern JavaScript application framework. A detailed description of the SISPI graphical user interfaces can be found elsewhere in these proceedings [24].

8. CONCLUSIONS AND OUTLOOK

The Dark Energy Survey Collaboration is building a new powerful CCD camera that will be installed on the Blanco 4m telescope at CTIO. In this paper we presented the design of the read-out and control system for this instrument (SISPI). Development of the SISPI infrastructure software is complete. Near production versions of the image pipeline, the instrument control system, the data quality monitoring software and the user interfaces are becoming available. Extensive tests will be performed this summer with the full scale telescope simulator [25] at Fermilab before SISPI will be shipped and installed at CTIO in the new Blanco control room early in 2011. System integration at CTIO will be complete by the time DECam arrives in Chile and we will be ready for the start of survey observations with DECam in October 2011.

9. ACKNOWLEDGEMENTS

Funding for the DES Projects has been provided by the U.S. Department of Energy, the U.S. National Science Foundation, the Ministry of Science and Education of Spain, the Science and Technology Facilities Council of the United Kingdom, the Higher Education Funding Council for England, the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign, the Kavli Institute of Cosmological Physics at the University of Chicago, Financiadora de Estudos e Projetos, Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro, Conselho Nacional de Desenvolvimento Científico e Tecnológico and the Ministério da Ciência e Tecnologia, the German Research Foundation-DFG sponsored cluster of excellence "Origin and Structure of the Universe" and the Collaborating Institutions in the Dark Energy Survey.

The Collaborating Institutions are Argonne National Laboratories, the University of California at Santa Cruz, the University of Cambridge, Centro de Investigaciones Energeticas, Medioambientales y Tecnologicas-Madrid, the University of Chicago, University College London, DES-Brazil, Fermilab, the University of Edinburgh, the University of Illinois at Urbana-Champaign, the Institut de Ciencies de l'Espai (IEEC/CSIC), the Institut de Fisica d'Altes Energies, the Lawrence Berkeley National Laboratory, the Ludwig-Maximilians Universität, the University of Michigan, the National Optical Astronomy Observatory, the University of Nottingham, the Ohio State University, the University of Pennsylvania, the University of Portsmouth, SLAC, Stanford University, and the University of Sussex.

10. REFERENCES

[1] Flaugher, B., et al., "The Dark Energy Camera (DECam)", these proceedings (2010).
[2] Abbott, T., et al., "Improving the Blanco Telescope's delivered image quality", these proceedings (2010)
[3] Information on the South Pole Telescope can be found at pole.uchicago.edu/.
[4] Information on the VHS VISTA Survey can be found at www.ast.cam.ac.uk/~rgm/vhs/
[5] Information on PYRO (Python Remote Objects) can be found at pyro.sourceforge.net.
[6] See for example http://en.wikipedia.org/wiki/Data_Distribution_Service
[7] Details on svn can be found at http://subversion.tigris.org/
[8] Details on eUPS can be found at http://dev.lsstcorp.org/trac/wiki/Eups
[9] Information on Windows INI format can be found at http://en.wikipedia.org/wiki/INI_file.
[10] Details on the configObj module can be found at http://www.voidspace.org.uk/python/configobj.html
[11] Details on PostgreSQL can be found at http://www.postgresql.org/
[12] Details on psycopg2 can be found at http://initd.org/psycopg/; details on sqlalchemy can be found at http://www.sqlalchemy.org/
[13] The Monsoon system was developed by NOAO (www.noao.edu/etc/monsoon).

[14] Castilla, J., et al., “The Readout Electronics for DECam”, these proceedings (2010) and T. Shaw et al, “System architecture of the dark energy survey camera readout electronics”, these proceedings (2010)
[15] The S-Link system is described in http://hsi.web.cern.ch/HIS/s%2Dlink/.
[16] Ashe, M., Bonati. M., Heathcote, S., “ArcVIEW: a LabVIEW-based astronomical instrument control systems”, Proceedings of the SPIE, Volume 4848, pp. 508-518 (2002)
[17] Details about netcat can be found at http://netcat.sourceforge.net/
[18] Schindler, R., et al., “Design of a radiometric all-sky infrared camera (RASICAM) for DES/CTIO”, these proceedings (2010)
[19] Tarle, G., et al., “Large format filter changer mechanism and shutter for the dark energy survey”, these proceedings (2010)
[20] Schumacher, G., et al., “The Blanco Telescope TCS upgrade”, these proceedings (2010)
[21] Information on matplotlib can be found at http://matplotlib.sourceforge.net/
[22] Information on Django can be found at http://www.djangoproject.com/
[23] A description of the Model-View-Controller design pattern can be found at http://ootips.org/mvc-pattern.html
[24] Eiting, J., et al., “HTML 5, Websockets, and Sproutcore: a web-based user interface for the dark energy camera (DECam)”, these proceedings (2010)
[25] Diehl, T., et al., “Testing DECam with the Telescope Simulator”, these proceedings (2010)