GHOST and GIAPI: experience using Gemini's new instrument control system framework

Peter Young, Jon Nielsen

Peter J. Young, Jon G. Nielsen, "GHOST and GIAPI: experience using Gemini's new instrument control system framework," Proc. SPIE 9913, Software and Cyberinfrastructure for Astronomy IV, 99132Q (26 July 2016); doi: 10.1117/12.2232235

SPIE.

Event: SPIE Astronomical Telescopes + Instrumentation, 2016, Edinburgh, United Kingdom

GHOST and GIAPI: experience using Gemini's new instrument control system framework

Peter J. Young and Jon G. Nielsen
Research School of Astronomy & Astrophysics, The Australian National University, Canberra, ACT, Australia, 2611

ABSTRACT

The new Gemini High Resolution Optical Spectrograph (GHOST) will be controlled with software developed against the new Gemini software framework - the Gemini Instrument Application Programmer Interface (GIAPI). The developers describe their experience using this framework and compare it to control systems developed for earlier Gemini instruments using the original Gemini Core Instrument Control System (CICS) framework.

Keywords: GHOST, GIAPI, Gemini CICS, ANU CICADA, EPICS, software framework

1. INTRODUCTION

The software team at the Research School of Astronomy and Astrophysics, ANU (RSAA) have developed the control system software for two of the Gemini Observatory's current suite of instruments – NIFS (McGregor et al., 2003[1]) and GSAOI (McGregor et al., 2004[2]). These systems were developed using the initial Gemini instrument control system framework – CICS (Core Instrument Control System – Beard, 1996[3]) which is an EPICS based system that requires the use of a set of Gemini specific EPICS records for implementation. CICS was designed in the 1990's when the prevailing ideas for instrument control were to use dedicated industrial real-time computers interfaced to high-level workstations running Unix. This was the typical EPICS deployment environment and usually involved VxWorks IOCs (Input Output Controllers) hosting the hardware interfaces to the instrument and the companion EPICS IOC database. The Unix platform chosen was often Sun Solaris workstations. NIFS and GSAOI were delivered to Gemini in 2005 and 2006 respectively (though GSAOI was only fully commissioned many years later once the Gemini GeMS (Rigaut, 2011[4]) system became available) – and both were deployed in this manner.

Soon after this time Gemini decided that CICS needed an overhaul and they therefore developed a new instrument control system framework – the Gemini Instrument Application Programmer Interface (GIAPI – Gillies & Nuñez, 2007[8]) to address the perceived problems/age of CICS. The Gemini Planet Imager (GPI – Gemini[9]) had its software developed using this new framework – effectively allowing the framework's development to be completed and tested in the real world. Using GIAPI enables a departure from the typical EPICS deployment environment described above and instead allows an instrument team to use technology based on field bus devices orchestrated by software running on Linux based workstations. The Gemini principle systems remain as they were (i.e. EPICS based) but the means by which an instrument control system interacts with them has fundamentally changed. No longer is it necessary to code the instrument software directly using the Gemini specific EPICS records of the CICS.

The RSAA team is now developing a third instrument control system for Gemini – this time for the Gemini High-Resolution Optical Spectrograph (GHOST – Sheinis et al, 2016[7]) - and this time using the new GIAPI framework. GHOST is a collaborative development led by the Australian Astronomy Observatory (AAO) partnered with the ANU and the National Research Council of Canada (NRC). The instrument has recently passed its design reviews and the build phase has commenced with commissioning expected during 2018. During the design phase many of the necessary modules required for interacting with Gemini systems using GIAPI have been developed and tested, this experience is elaborated on here.

2. GEMINI CICS (EPICS) OVERVIEW

The software for controlling the Gemini telescopes and instruments is comprised of a network of cooperating 'principal' systems – each system designed to know as little as possible about the internal workings of other systems. They have been designed to communicate control information between each other using attribute sets that move a target system into

Software and Cyberinfrastructure for Astronomy IV, edited by Gianluca Chiozzi, Juan C. Guzman, Proc. of SPIE Vol. 9913, 99132Q · © 2016 SPIE
CCC code: 0277-786X/16/$18 · doi: 10.1117/12.2232235

a desired configuration. The Observatory Control System (OCS) provides overall direction to the control system. All command and status information that flows between these systems is handled using a simple unified strategy based on the EPICS (Experimental Physics and Industrial Control System[i]) framework.

For the Core Instrument Control System (CICS) there are three (or four) components of a 'conforming' instrument implementation. The Instrument Sequencer (IS), the Components Controller (CC), the Detector Controller (DC), and, optionally, an On Instrument Wave Front Sensor (OIWFS). These are usually hosted on two (or three) IOCs with the IS and CC sharing one. Each has instances of an EPICS command database (CAD/CAR) and an EPICS status and alarms database (SAD). This arrangement is shown in Figure 1.

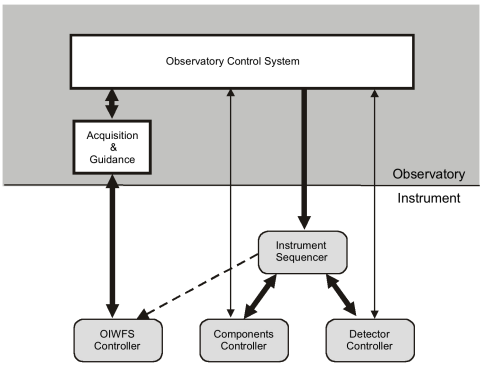

Figure 1: Gemini ICS Architecture

So the emphasis with CICS is the development of an instrument control system using the EPICS framework – which may or may not be something an instrument builder is familiar with.

3. GEMINI GIAPI OVERVIEW

Though existing Gemini instrument systems are working well, many lessons were learned from the build effort, and Gemini have been concerned about capturing the lessons learned and applying those lessons to future instrument development. Computing equipment and operating systems have drastically improved since the development of CICS - providing options that were impossible a few years ago.

The motivation for the changes being introduced for Gemini instrument development, though, are primarily pragmatic rather than technical:

- Simplify the Gemini environment (remove the complexity of having to develop within an EPICS environment)
- Allow builders more freedom (take advantage of existing systems that an instrument builder is familiar with and are mature)
- Simplify the instrument integration effort (via the provision of a well maintained API for interacting with Gemini principle systems)
- Clarify software responsibilities (ensure software module boundaries are clear with well-defined interfaces)
- Reduce software costs in effort and equipment (reuse of existing software deployed on lower cost hardware)
- Increase quality and rigor (adopt agile development strategies with Gemini staff involvement from the outset)
- Provide timely and appropriate levels of support (ensure Gemini staff are engaged during the design and development stages)

GIAPI is the new instrument software framework developed at Gemini in response to these considerations and is fully described in GIAPI Design and Use (Gillies & Nuñez, 2007[8]).

[i] http://www.aps.anl.gov/epics/

In the GIAPI context, an instrument consists of three logical software components (as in CICS): the IS, the CC, and the DC. The GIAPI guidelines suggest how the high-level components should be distributed to attain the goals of simplicity and cost-effectiveness. The idea of a Top-Level Computer (TLC) running the Linux operating system is introduced with GIAPI and should be the system for the implementation of the following features and components:

- The primary system running builder code that integrates with the Gemini software interface to commands and status.
- Provides the IS capability.
- Provides all or a major part of the instrument CC functionality.
- May or may not host the DC functionality – depending on performance requirements.

These components are co-hosted on the TLC in order to reduce the number of computers, which reduces software complexity for common operations, reduces command execution times, increases overall performance, and removes reliability issues that are increased by inter-computer communications.

The GIAPI layer is implemented in the form of a Gemini Master Process (GMP) that on the one side interacts with the Gemini principle systems (as with CICS) and on the instrument developer side provides a simple API for the developer to use in the form of common language (C++ and Java) bindings. This API allows for command handling and status information delivery using simple high-level language concepts. This arrangement is shown in Figure 2.

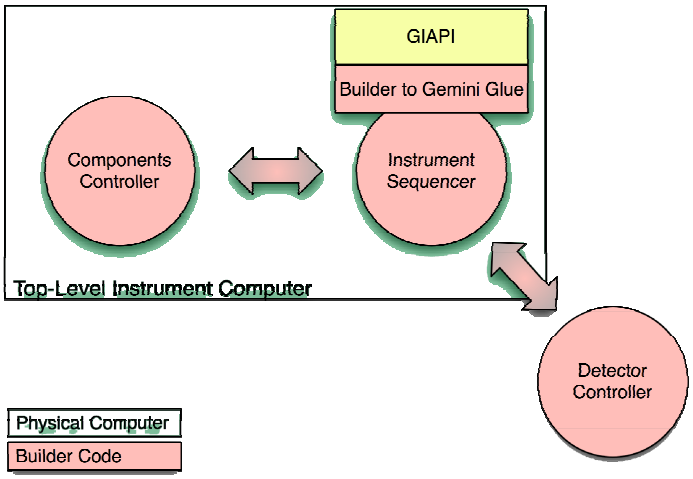

Figure 2: Gemini GIAPI Architecture

4. ANU CICADA OVERVIEW

The Research School of Astronomy and Astrophysics (RSAA), Australian National University operates (and has operated) a set of relatively small telescopes for many years. These telescopes are equipped with a variety of instruments that have been and are operated under a single control and data acquisition software environment. This software system, named CICADA (Configurable Instrument Control and Data Acquisition - Young et al, 2008[5]), has also been ported (in part) to operate two Gemini instruments constructed at RSAA. CICADA has been augmented with a higher layer of software to enable the remote and automatic operation of some instruments. This new layer, named TAROS (Telescope Automation and Remote Observing System – Wilson et al, 2005[6]), has been commissioned and is currently being used at RSAA's 2.3m telescope and the new SkyMapper survey telescope.

CICADA started development in 1993 and though only vaguely recognizable from its earliest version, the latest version (version 5) is still in operation today and in total has been used on nine telescopes with seventeen unique instruments over its history.

5. NIFS, GSAOI, CICS AND CICADA

NIFS and GSAOI employed the Gemini CICS framework in their software design. They both used two VxWorks IOCs (CC and DC) hosting the underlying EPICS databases. The operator interface ran on a Sun Solaris workstation and used the EPICS Display Manager (DM) GUI toolkit.

The main emphasis in building the software for NIFS and GSAOI was the construction of the Gemini styled EPICS databases. This was done using the Capfast database creation tool. EPICS State Notation Language (SNL) was used as a glue layer between the Gemini principle systems and the underlying instrument control software – which for the DC, was built using elements of the ANU CICADA data-acquisition system. A further glue layer was developed to allow CICADA heritage code to read and write directly into the EPICS database records – using a thin-layer approach.

6. GHOST, GIAPI AND CICADA

6.1 GHOST Software Design

In the development of an architecture that will meet the requirements of GHOST, the following have been kept in mind. Note that the whole thrust of the GIAPI from Gemini's point of view is to enhance the ability of the instrument builder to produce higher quality instrument software more efficiently – they recognize that a key element of this strategy is to enable re-use of existing software. The following is consistent with this Gemini emphasis:

- Support expected Gemini architectural components: IS, CC, DC deployed on a top-level computer (TLC). CAD/CAR/SIR model now isolated in GIAPI.
- Parameter publish/subscribe pattern: Use existing ANU parameter database class previously used in our thin-layer EPICS design (NIFS, GSAOI) with extensions for a GIAPI parameter database class, parameter subscriptions and alarms.
- Reuse ANU CICADA classes and basic master-slave architecture (as already adopted for NIFS and GSAOI).
- Requirement for Gemini Parameter Definition Format be realized using automatic generation from an RPCL (RPCL – Remote Procedure Command Language[ii]) like definition language (RPCL affords reuse from CICADA – extended for automatic parsing of command requests and parameters).
- Acceptance Test and Engineering GUI (ATEUI) – GUI technologies that will offer maximum platform flexibility, Java8 has been chosen. This needs to be loosely coupled to the GHOST software engine – using a parameter subscription model (i.e. client only), EPICS Channel Access has been chosen (to access the underlying GIAPI EPICS database).
- Support GHOST mechanisms using a distributed model with actual hardware interfaces and real-time performance met by field bus type devices, CAN bus modules have been chosen.
- Need to interface to the Allied Vision cameras – do so via construction of a new CICADA controller class. This class will use the vendor supplied C++ source code API that uses an underlying Linux binary library to access the GigE Vision interface for these cameras.
- Embedded scripting will be used for support of executing complex sequences– i.e. embedded Python (embedded Tcl already in CICADA and it will be extended to support Python). Examples of these types of sequences include the unequal exposure mode for Blue and Red camera arms and an exposure meter that stops an exposure as soon as sufficient flux has been reached. This will replace the use of SNL heavily evident in CICS built instruments.

6.2 Model of an Instrument

For NIFS and GSAOI, the software design strategy for the Detector Controller (DC) part of the instrument was to re-use as much of the software already running at ANU supporting a range of instruments. ANU instruments are operated using the CICADA package. CICADA is an object-oriented software framework that takes a generic model of a telescope instrument and builds an actual configuration based on a composition of elements that match the requirements of the actual instrument. The class diagram in Figure 3 best illustrates the model used. A configuration is expressed using a set

ii http://tools.ietf.org/html/rfc5531

of configuration files – one file corresponding to each class in the class diagram. Elements in the configuration are indexed via unique instrument part names.

We have reused this fundamental design strategy for GHOST – allowing us to take advantage of an existing mature code base.

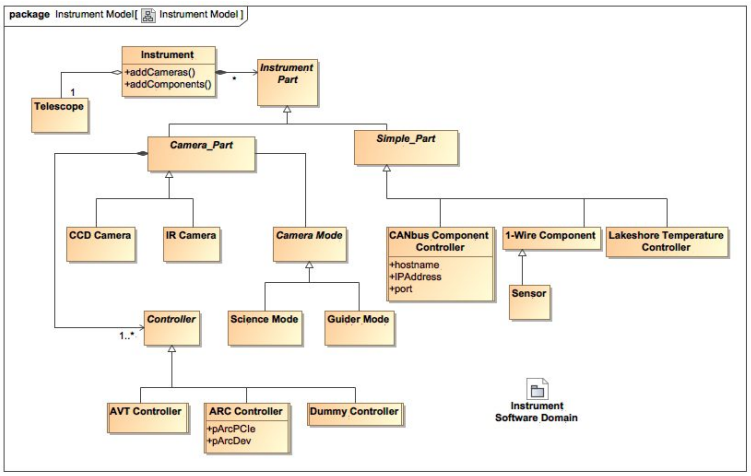

Figure 3:Partial CICADA class diagram representing the instrument model

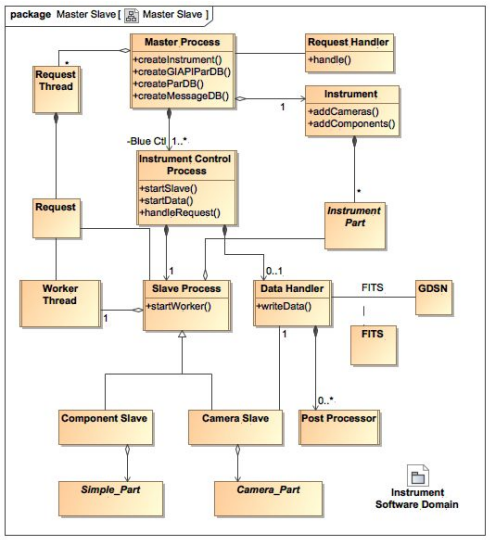

Figure 4: The Master-Slave architectural design pattern for the GHOST instrument

Note that the diagram shows a sample of the concrete Controller classes that are already implemented. We have implemented the Controller class for an AVT Controller, using the already procured Allied Vision cameras for GHOST. We have also implemented the ARC Controller class that uses the ARC Camera C++ API and tested with ARC Gen 3 controllers using the PCIe interface.

The diagram also shows elements of the current CICADA model not relevant to GHOST – but are simply included as a demonstration of the extensibility of the model.

6.3 Organization of Work, the Master-Slave Pattern

Another aspect of the CICADA architecture used in NIFS and GSAOI is the master-slave architectural design pattern. This pattern, when combined with the instrument model described above, maps neatly onto the logical arrangements of parts in a typical Gemini instrument, i.e. the Instrument Sequencer, the Components Controllers and the Detector Controllers.

The Master process's role is to coordinate activity among a set of Slave processes – one for each of the component parts of the instrument. Each Slave independently manages the activity required for its component and will either belong to the logical CC (aka an instrument mechanism or component) or the logical DC (this arrangement allows us to support the ability to control components of the instrument concurrently).

This organization of the work allows for very a flexible distribution of processes on a network of computers – if so required. All commands to the instrument are funneled through the Master process which has a complete understanding of the state of each of its child (Slave) processes.

For Camera instrument parts, each Slave process also has a companion Data Handler process whose sole responsibility is to manage data transfer to storage and provide any post processing of the data that may be required.

The Slave process is started, interrupted and stopped by an Instrument Control process.

The Master, Slave and Data Handler classes (see Figure 4) are active classes and their dynamic activities are best specified via the use of a finite state machine. These state machines are most easily designed and understood using the Unified Modelling Language[iii] (UML).

With the EPICS based solutions of older Gemini instruments, the State Notation Language[iv] (SNL) was a very nice tool for encoding much of what was needed for these processes (though companion C++ code was also used).

Because SNL will not be used for GHOST, encoding the model behavior has been done by using more components of the CICADA code base than were used in NIFS and GSAOI – complemented with the use of embedded python scripting for implementing specific instrument required sequencing behavior.

6.4 Working with the GIAPI Environment

The Gemini document describing the use of the GIAPI[8] suggests a preferred layout of software elements illustrated by a block diagram (see 5.1 of GIAPI Design and Use and expanded descriptions in chapter 7).

This arrangement requires that the GHOST specific code interact with the Gemini Master process (the realization of GIAPI) using a glue layer (or library). This library provides facilities for the instrument code to publish and subscribe to uniquely named status and command items defined using the Gemini Parameter Definition Format.

For NIFS and GSAOI a similar requirement existed and was solved by creating an EpicsParameterDB class that provided a wrapper interface to the actual EPICS interface. A higher-level ParameterDB class was in turn used by the instrument code to transparently access EPICS parameters without knowledge of any of the details of the underlying EPICS interface.

This arrangement is shown in the class diagram of Figure 5.

The same application code can be transported to a different environment – say the ANU CICADA environment, by realizing a different underlying External ParameterDB class (in CICADA's case the XdrParameterDB class) without actually modifying any higher layer code.

We have designed a specific External ParameterDB class for the GIAPI environment (the GiapiParameterDB) which re-uses CICADA code just as for NIFS and GSAOI. This GiapiParameterDB actually just extends the existing CICADA XdrParameterDB with the extra functionality required to communicate with the Gemini Master Process using the GIAPI API. Prototyping of this class has been completed and it is working smoothly.

Entries in the ParameterDB database are automatically generated from definitions described using the RPCL, which also define the complex data structures used within the entire instrument code base.

We have adopted companion JSON configuration files as a way for mapping these RPCL definitions to the GIAPI naming scheme and a combination of these input files are used to generate the required code to add each parameter to the runtime GiapiParameterDB. Included in the JSON configuration file is necessary meta-data for implementing alarms and FITS header keywords.

6.5 Layout of GHOST Software in the GIAPI Environment

Combining the architectural elements described above into a complete GHOST software architecture we have the structure shown in Figure 6. Deployment of this architecture to a computing platform logically has a natural division of processing made at the Slave process level if required allowing for a distributed deployment. This is achieved using the remote procedure call (RPC) technology.

iii http://en.wikipedia.org/wiki/Unified_Modeling_Language

iv http://www.aps.anl.gov/epics/EpicsDocumentation/AppDevManuals/Sequencer/snl_1.9_man.html

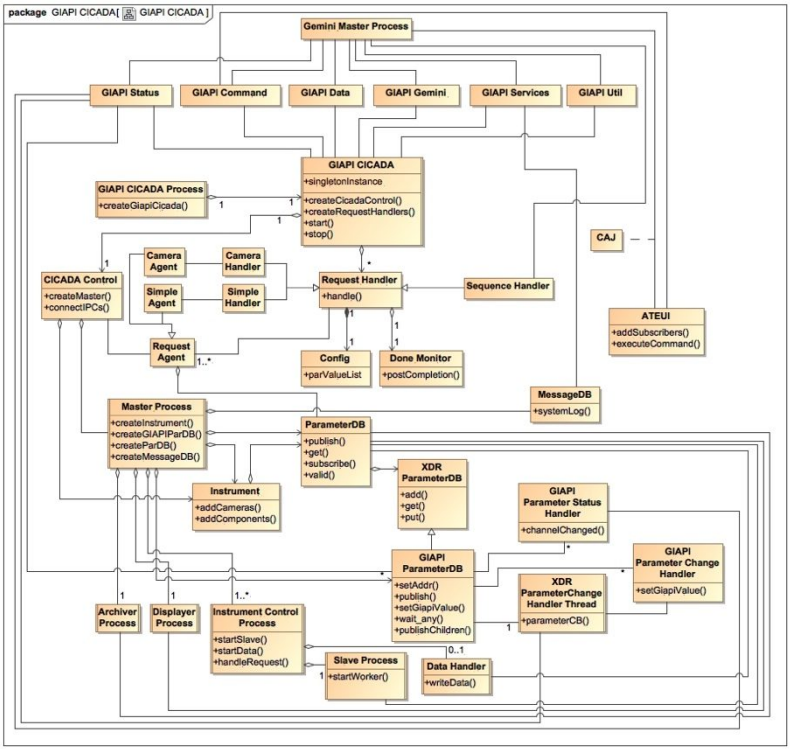

Figure 5: The ParameterDB class diagram showing the new GiapiParameterDB class

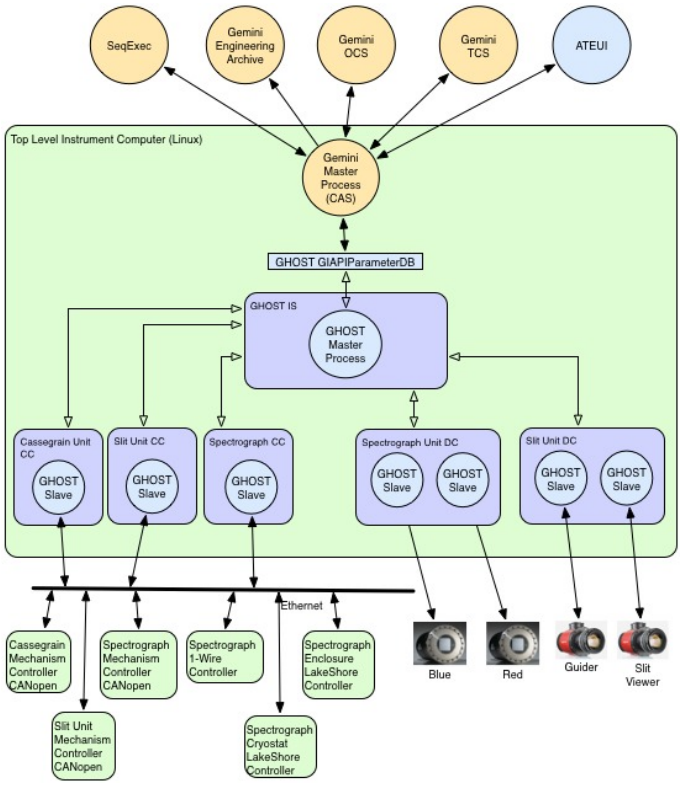

Figure 6: Block Diagram of GHOST Software Architecture

Note that the logical division of processes between IS, CC and DC is designed to be just that – i.e. a logical division, noting that attributes and parameters in each of the logical domains will be named appropriately according to the expected Gemini naming scheme.

The objects represented with blue shading are specific GHOST software objects. Those in yellow are software objects provided by Gemini.

The ATEUI will access the instrument for status parameter updates via the GiapiParameterDB that will be accessible across the network using the EPICS Channel Access (CA) protocol. This is possible because the Gemini Master Process (GMP) hosts a CA server that publishes all the GHOST parameters. The GMP is configured using a set of configuration files that describe the parameters to be made available for the CA server. These configuration files are automatically generated from the GHOST RPCL and JSON configuration files.

We have implemented an interface to Gemini sequence commands using listener threads supporting the requirement to register handlers for each sequence. Commands from the ATEUI are submitted via the same route through the GMP as normal Gemini OCS commands will travel.

Only parameters that have changed values will be published through the GiapiParameterDB class.

Other performance requirements that this parameter-handling module must meet include those describing maximum update latencies and rates as well as minimum update rates. We plan to configure the GIAPI interface module to behave according to these requirements via use of configuration tables.

Each process in the IS, CC and DC logical domains will have access to the GiapiParameterDB.

6.6 Automatic Generation of Configuration Files and Parameter Code

Missing from CICADA prior to the GHOST development was a way to automatically configure the software to directly build the ParameterDB from the RPCL data structure definition files and to automatically parse a text based CICADA command into a CICADA Request structure.

For GHOST we have devised a way of doing this using the Boost Spirit parser library that will take an RPCL file as input and generate a base JSON file. This file is then elaborated manually by inserting relevant meta-data for describing other attributes of each parameter that cannot be contained within the RPCL. This meta-data includes:

- GIAPI parameter name
- FITS keyword and related detail
- Validation data (either range or set values)
- Alarm settings
- ATEUI component name
- Documentation – including tooltip hints for the ATEUI

Not all these attributes are needed for each parameter – in fact some will not need any elaboration (being internal CICADA parameters only) – just meta-data required for each parameter will be added. Changes to the underlying RPCL would, of course, require merging a newly generated base JSON file with a previously elaborated version. We believe this is a small compromise and unlikely to be required very often.

Using the JSON file to generate an RPCL file would probably be a better design – but for historical reasons (CICADA was started in 1993) – it is more cost-effective to adopt our current strategy.

Figure 7 shows this process in a flow like diagram. Note that some steps are “compile” time preparation steps while others happen once the software has been started – i.e. during “run” time.

The input to the configuration process consists of two types of file: RPCL and JSON. The generated files include C++ headers and source, GIAPI XML configuration files, Java properties files for the GUI, and text files for the ICDs (that are converted into tables for Word documents).

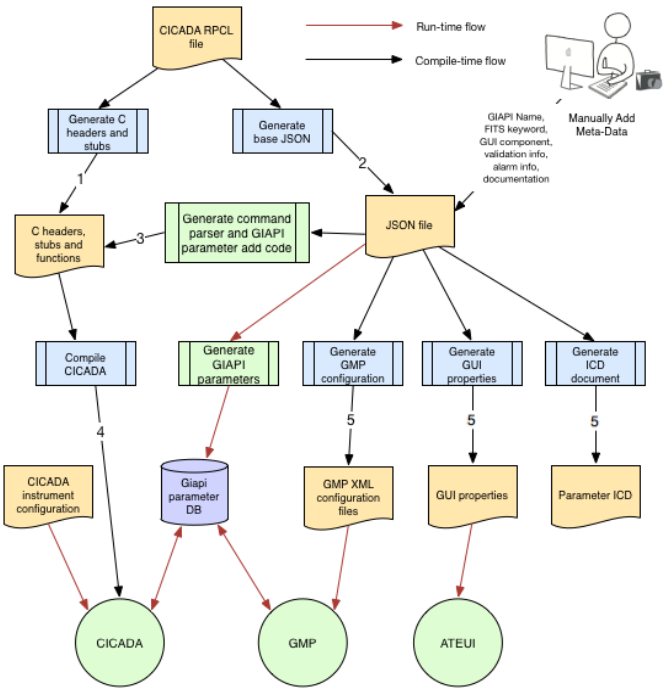

Figure 7: CICADA configuration process flow

The steps to generate each set of files are numbered in the diagram and described below:

1. Use rpcgen to generate C headers and stubs. Part of CICADA's primary Makefile.
2. Use xdr_parse (ANU code using Boost) to generate the base JSON file. This uses a specific Makefile target and will only be done on demand. If this is an incremental change – merge with the previous version of the JSON file using common merging tools.
3. Use xdr_parse (with specific arguments) to generate code for adding parameters, parsing commands and handling enum types. Also used to populate parameter verification and alarm data items. Done routinely as part of the make process with specific Makefile targets.
4. Compile CICADA – routine Makefile target.
5. Generate run-time usable configuration files for the GMP, the ATEUI and for documentation. Done using on-demand Makefile targets.

6.7 GUI Development

Java has been chosen for the GHOST ATEU after an analysis of the technologies proposed at the GHOST CoDR and after determining Gemini's preferred approach. We have attempted to preserve the advantages offered by EPICS display manager tools (i.e. the ability to attach widgets in the GUI to actual EPICS process variables using the Channel Access protocol) – but at the same time offer a modern GUI look and feel with platform portability. Java has ticked all these boxes and in particular the JavaFX[v] widget technology has proven to be good to work with.

Alternative technologies proposed have been rejected for the following reasons:

- Web browser based technologies (e.g. HTML5, IceFaces, Ajax): these technologies offer much promise including innovative visualizations and platform portability but have been found to be too unstable to be suitable for Gemini. Nothing in this area seems to stand still for long, with the risk of making a technology choice that could soon become obsolete. Gemini (A. Nunez priv. comm. June 2014) indicated that they would not accept this choice.

v http://www.oracle.com/technetwork/java/javase/overview/javafx-overview-2158620.html

- Qt: this would have been a suitable choice but neither Gemini nor ANU preferred it – principally because of existing experience with Java at both organizations. A brief comparison of Qt and Java was made with the conclusion that either platform would be suitable for GHOST.

To connect to the underlying status and control points running in the GHOST control code, the pure Java implementation of the EPICS Channel Access protocol, Channel Access for Java (CAJ) has been chosen. This allows the Java ATEUI to connect to the EPICS channel access server running in the Gemini Master Process – in the same way as dedicated EPICS display manager applications have done.

The development of the ATEUI also follows the Model-View-Controller (MVC) software design pattern[vi]. Existing Java frameworks that implement this pattern are available and one such open-source framework that offers many language bindings, including Java, is the PureMVC[vii] framework, which we have chosen for the GHOST ATEUI.

Using Java allows us to offer the ATEUI for a number of platforms (including Linux and OSX), which was another stated goal at the GHOST CoDR. However, we do not anticipate that a complex authentication mechanism will be needed as use of the GUI will be restricted to Gemini (and GHOST partner subnets during development).

It is important to note that even though Java has been chosen to develop the ATEUI, it will not affect the underlying GHOST operating engine. Using this approach, Java could be replaced with another GUI technology later on – without affecting the GHOST control code.

6.8 Real-Time Display Support for GHOST

Missing from GIAPI is support for a real-time image display. FITS files are written by the GHOST software directly to the Gemini Data Storage Network (network attached storage) and the usual image display tools (such as DS9) could then load and display the files for analysis. However, this approach would not satisfy the requirements of GHOST which needs to provide immediate visual feedback during data acquisition – both for the spectrograph cameras and the guiding and slit-view cameras.

Being somewhat loyal to EPICS as a software framework, the GHOST developers have chosen the EPICS V4 technology stack as a mechanism for transporting real-time image data from the GHOST data acquisition system to the engineering GUI tool. EPICS V4 offers two features that a GHOST real-time display required – structured data transport (in our case image chunk data) and a proven publish/subscribe data transport pattern. GHOST will provide support for multiple simultaneous engineering GUI instances running, possibly, on different platforms (e.g. Mac laptops or Linux work-stations).

This choice has proven successful so far, though some issues have had to be overcome and the learning curve for EPICS V4 proved to be steeper than anticipated (perhaps due to the open-source nature of the software where documentation sometimes lags code development).

Figure 8 shows the Spectrograph Detector Controller screen with embedded real-time display having just completed an exposure (simulated data). Note that we are using a customized version of the JSky[viii] reusable Java components as the basis for the image display.

vi http://en.wikipedia.org/wiki/Model–view–controller).
vii http://puremvc.github.io
viii http://jsky.sourceforge.net

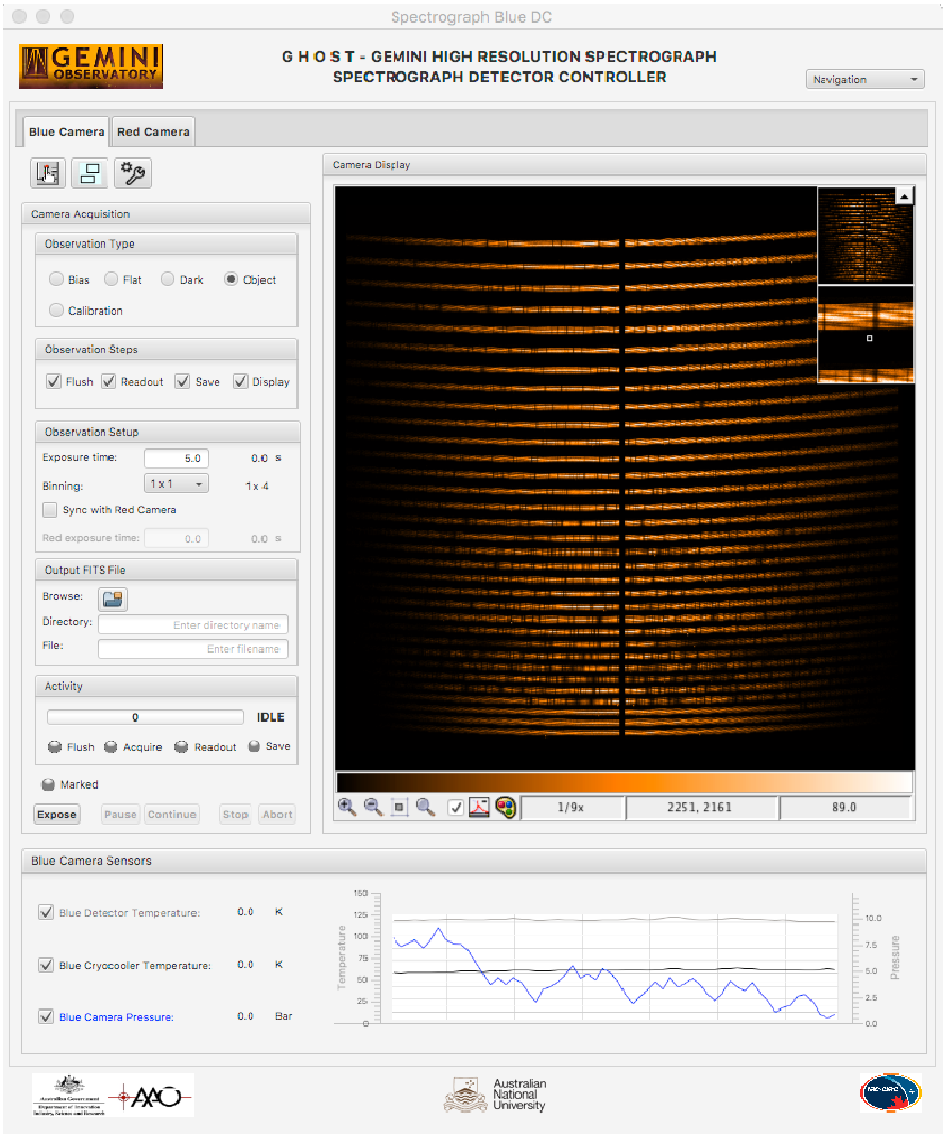

Figure 8: GHOST ATEUI Spectrograph Detector Controller screen

Figure 9 shows the GHOST ATEUI slit unit detector controller screen (mock up data). The image plane displays of the fibre bundles are actually post-processed reconstructions of the slit-view images. Both of these use the JSky display component as for the spectrograph screen.

Figure 9: GHOST ATEUI Slit Unit Detector Controller screen

6.9 Developer Experience

In contrast to ANU's earlier Gemini instruments, the instrument developer using GIAPI no longer builds a Gemini EPICS database directly. Instead the GMP is configured using XML configuration files which in turn are converted (by the GMP) into a running "soft" EPICS IOC that is accessible to the heritage Gemini principle systems. For GHOST, and as described above, these configuration files are generated directly from the CICADA RPCL data structure definitions using an automated parsing code that builds an intermediate JSON file which is then processed to form the required XML.

In principle either scheme is quite similar – i.e. a configuration file needs constructing to represent the data items that describe the interface between the instrument and Gemini. However, in practice the experience is completely different. The use of the Capfast drawing tool for constructing an EPICS database has a steep learning curve – particularly when a custom set of Gemini EPICS records are required to be used.

On the other hand, building the XML files that configure the GMP has not been easy either. There are several of these files and it has been hard to understand how to build some of them. In particular, the content of the EPICS Apply configuration file was poorly documented and it required access to the GMP source code before a good understanding was gained of what had to be done. This understanding was then used to construct the parsing code for the generation of the XML configuration file from the JSON input file.

7. SUMMARY

Having developed software using both Gemini frameworks, what have we learned? Have the goals that Gemini have set itself in developing GIAPI been met? Listing the pros and cons of using GIAPI (with reference to CICS) is a way of answering these questions

7.1 GIAPI Pros

- No need to learn how to use EPICS. A developer can continue to use a known software framework and confine any interaction with Gemini to a small layer of code.
- No (real) need to understand the Gemini approach to using EPICS – i.e. using the Gemini developed EPICS records. However, understanding the Gemini specific EPICS records has helped us understand how to configure the GMP. No requirement to build an EPICS database directly – instead use of standard C-like data structures is possible.
- Reuse of existing data acquisition code is encouraged.
- Modern object oriented language support with the GIAPI API – C++ and Java.
- Simplicity of interface (API).

7.2 GIAPI Cons

- Black box (restrictive) nature of the GMP. Ultimately we sought access to the GMP source code to better understand how the GMP configuration files worked. This perhaps implies inadequate documentation for configuring the GMP. It also could imply that GIAPI has limitations which will need addressing.
- Restricted data type support (i.e. no structured data). This is essentially a restriction caused by the underlying EPICS data structures used by GIAPI.
- Missing real-time display support. Gemini discourage use of the heritage Gemini Data Handling System (DHS) for new instrument developments. This system had support for a real-time display sub-system based on the ESO Skycat tool. Though the DHS is probably dated technology, perhaps a replacement real-time display system needs to be provided as part of GIAPI. Otherwise each developer will end up developing something different.

7.3 Conclusions

Looking at Gemini's goals prior to undertaking the development of GIAPI we can conclude that:

- Simplification: we think that this has been achieved – but perhaps at the cost of introducing more obscurity and it places a greater responsibility on the Gemini software maintainers to provide a strong, reliable working system. Previously this responsibility largely laid with the EPICS community.
- Freedom: yes – developers have a lot of freedom to use (re-use) instrument control software that they understand well and that is mature. We can imagine though, that the complexity of interfacing with GIAPI may vary greatly depending on the pre-existing framework being used. In our case this went very well as we had a software design (in CICADA) that matched quite closely the architectural aspects of GIAPI.
- Simplify instrument integration: we cannot comment on this – commissioning of GHOST is scheduled for 2018.
- Clarify software responsibilities: here we don't think there have been many changes – at least for us. When developing software using the older CICS framework – it was always reasonably clear to us what had to be done by us, particularly since we, effectively, confined our use of EPICS to a small layer of software (for the DC at least). The caveat being that some of the original CICS documentation was somewhat hard to follow or inadequate.
- Reduce costs: Even though we are re-using an existing framework for GHOST, the instrument's requirements differ significantly enough from our earlier instruments that overall software costs are not too different. Our previous

instrument developments had much re-use as well. One significant cost reduction would be the learning curve associated with EPICS and its associated tools – but not relevant for us this time given our previous CICS experience. The question for us in our first use of GIAPI could be that we had the cost of learning GIAPI whereas if we had used CICS again there would not have been such a learning curve cost. Very difficult to assess.

- Quality and rigor: We cannot see how this can be controlled – i.e. introducing freedom to developers may in fact allow for a reduction in quality. Though using a long proven framework which has undertaken a thorough review process helps.
- Timely support: Gemini have attempted to be more closely involved during our code development – including spending time at our site. It is still a bit early to make a final conclusion on this.

REFERENCES

[1] McGregor, P., Hart, J., Conroy, P.G., Pfitzner, M.L., Bloxham, G., Jones, D.J., Downing, M.D., Dawson, M., Young, P. and Jarnyk, M., van Harmelen, J. "Gemini near-infrared integral field spectrograph (NIFS)", Proc. SPIE 4841, (2003).

[2] McGregor, P., Hart, J., Stevanovic, D., Bloxham, G., Jones, D., van Harmelen, J., Griesbach, J., Dawson, M., Young, P. and Jarnyk, M. "Gemini South Adaptive Optics Imager (GSAOI)", Proc. SPIE 5492, (2004).

[3] Beard, S.M. "The Gemini Core Instrument Control System", ASP Conf. Ser. 101: Astronomical Data Analysis Software and Systems V, 101 (1996).

[4] Rigaut, F. and Neichel B. "Gemini South MCAO on-sky results", 2nd AO4ELT Conference, Victoria, Canada, (25 September 2011).

[5] Young, P.J., Nielsen, J., Roberts, W.H. and Wilson, G.M. "Achieving Design Reuse: A Case Study", Proc. SPIE Volume 7019, pp. 70192M-70192M-12 (2008).

[6] Wilson, G.M., Czezowski, A., Hovey, G.R., Jarnyk, M.A., Nielsen, J.N., Roberts, W.H., Sebo, K.M., Smith, D.F., Vaccarella, A.L. and Young, P.J., "Telescope Automation and Remote Observing System (TAROS)", ASP Conf. Ser. 347: Astronomical Data Analysis and Systems XIV, 347, 563 (2005).

[7] Sheinis, A., Anthony, A., Baker, G., Burley, G., Churilov, V., Edgar, M., Ireland, M.J., Kondrat, Y., McDermid, R.M., Pazder, J., Robertson, G., Young, P.J. and Zhelem, R., "The Gemini High Resolution Optical Spectrograph (GHOST)", SPIE Astronomical Telescopes + Instrumentation, Paper 9908-44, (2016).

[8] Gillies, K., Nuñez, A. "GIAPI Design and Use", https://www.gemini.edu/sciops/instruments/specifications/Gemini_GIAPIDesignAndUse.pdf (10 September 2007).

[9] Gemini Observatory, "GPI, The Gemini Planet Imager", http://www.gemini.edu/sciops/instruments/gpi (26 May 2015).