Observatory software for the Maunakea Spectroscopic Explorer

Tom Vermeulen*[a], Sidik Isani[a], Kanoa Withington[a], Kevin Ho[a], Kei Szeto[a], Rick Murowinski[a]
[a]Canada-France-Hawaii Telescope, 65-1238 Mamalahoa Highway, Kamuela, HI, USA 96743

ABSTRACT

The Canada-France-Hawaii Telescope is currently in the conceptual design phase to redevelop its facility into the new Maunakea Spectroscopic Explorer (MSE). MSE is designed to be the largest non-ELT optical/NIR astronomical telescope, and will be a fully dedicated facility for multi-object spectroscopy over a broad range of spectral resolutions. This paper outlines the software and control architecture envisioned for the new facility. The architecture will be designed around much of the existing software infrastructure currently used at CFHT as well as the latest proven open-source software. CFHT plans to minimize risk and development time by leveraging existing technology.

Keywords: MSE, CFHT, Maunakea, spectroscopy, software

1. INTRODUCTION

The Maunakea Spectroscopic Explorer project will transform the CFHT 3.6m optical telescope into a 10m class dedicated multi-object spectroscopic facility, with an ability to simultaneously measure thousands of objects with a resolving power range spanning 2,000 to 20,000. The project is currently in design phase, with full science operations nominally in 2025.

As part of this process, a critical eye is focused on identifying what the software infrastructure will look like for this newly reimagined facility. MSE will require large-scale changes to the physical infrastructure of the CFHT observatory. Along with these physical changes, new software must be developed to support the new telescope, dome enclosure, and multi-object spectrograph. As part of these changes, the entire software infrastructure for MSE must be addressed. The proposed approach for developing MSE software will be to reap maximum benefits from the stable production software systems of CFHT and the open source and telescope community.

In order to provide a bit of perspective on software for MSE, a diagram illustrating the lifespan of software produced for CFHT is shown in figure 1. This figure is included to illustrate that the software for MSE is also likely to evolve over its lifespan. As a result, it will be important to design and develop software with flexibility and maintainability in mind.

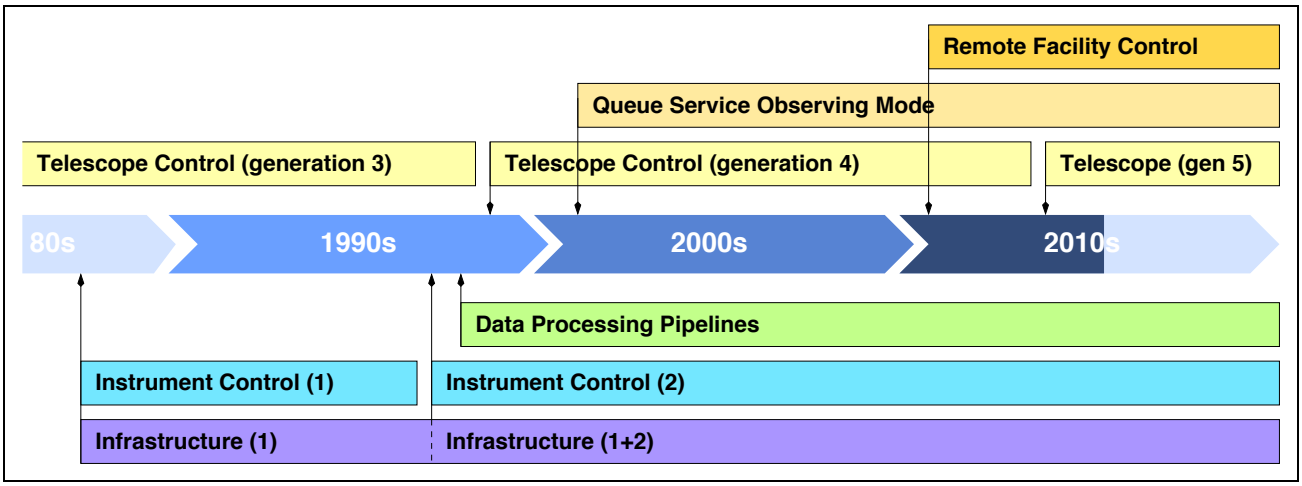

Figure 1. CFHT software subsystems lifespan

2. SOFTWARE DESIGN AND IMPLEMENTATION PHILOSOPHY

Many of the software challenges for MSE are similar to those faced by most observatories. In order to put the software for MSE in the proper context, it is useful to take a step back and understand the overall software design and implementation philosophy that will be used with MSE.

2.1 Development Methodology

For the most part the software designed and developed for MSE will follow the traditional waterfall development approach (requirements, conceptual design, preliminary design, final design, implementation, integration, and testing phases). Areas with high user involvement, such as graphical user interfaces (GUIs), will follow a more iterative design cycle such as the agile development methodology. To the extent that is necessary, prototyping may be performed, however, this will not be used as a way to bypass the design process. The waterfall and agile methodologies were successfully used on large-scale software projects at CFHT such as Queued Service Observing (QSO), remote observing, and the Status Server. In each case, these projects were integrated with virtually no downtime and a very low maintenance overhead.

2.2 Open Source

Where possible, open source software will be used with MSE. CFHT has traditionally taken advantage of open source software. As a result, CFHT also places a GPL license on internally developed software so that the observatory can give back to the community. It is the hope that any software that is developed for MSE, either by MSE staff or by partners, should be freely shared in a non-commercial way. The anticipated lifetime of MSE will span several decades and it is important that source code be available to adapt to possible changes to the hardware or operating systems over time. In addition, should issues arise the MSE staff will have the ability to fully address them instead of critical components of the software behaving like a "black box".

2.3 Reuse

CFHT has an underlying common software infrastructure that can be reused for MSE. By reusing existing software for MSE it will be possible to reduce risk and start with a base of robust software. It could be possible to integrate proposed changes to the common software for MSE within CFHT prior to commissioning. There is no substitute for testing in an operational environment. Though this comes with its risks to the efficiency of the existing observatory, upgrades have already been a constant process through the life of CFHT and carry the benefit of bringing modern tools, functionality, and efficiency to older instruments. This paper will highlight the primary areas for software reuse from CFHT.

2.4 Let the Requirements Dictate Design

Rather than identifying a specific programming language that all software solutions for MSE must conform to, the software will be designed and implemented based on what the requirements dictate would be the best solution. Those languages and packages that are ideally suited for user interfaces and data reduction may not be the best solutions for implementing low-level device drivers and GUIs. The software infrastructure at CFHT is currently composed of a mix of compiled and interpreted languages based on matching the best solution to each particular need.

2.5 Simple Communications Protocol

The software infrastructure for MSE will be composed of a distributed network of computers and embedded controllers running software. To the extent possible, the software for MSE will be designed in such a way that the communication between subsystems is based on a simple TCP/IP socket protocol. This allows for a looser coupling between software components and provides greater implementation flexibility for each individual component. In many ways this is patterned after the success of extensible protocols, such as HTTP, which have contributed to the success of the open Web. The same principles can help prevent observatory software from becoming constrained within tightly controlled language-specific or architecture-specific and difficult to debug or trace communication frameworks. By simplifying the coupling dependencies between subsystems it should be possible to provide the flexibility to adapt to future changes beyond the 2025 commissioning date currently envisioned for MSE. Figure 2 contains a simple example where the current telescope coordinates are requested from the Status Server at CFHT.

```
% nc statserv.cfht.hawaii.edu 909
> get /t/status/tcoords
< . /t/status/tcoords "F 21:27:31 19:59:00 appa zenith"
```

Figure 2. Simple TCP/IP socket based protocol example

2.6 Separate User Interface and Logic Layers

Where possible, control logic will be separated into a layer that can be triggered in multiple ways. The most obvious approach is to limit embedding control logic within GUIs outside of event handling and basic field validation. This approach allows for control operations to be triggered via GUIs, scripts, sequencers, or business rule driven autonomous operations. This also allows for future replacement of GUI layers or logic layers with limited disruption.

2.7 Maintainability

This is an area which often doesn't receive the necessary thought early in the design process. Large software development projects often have one group design and develop software before it is handed over to another group for support. In this scenario a motivation of the software developer can be to learn the latest software technology by applying it on a project. While it may be an interesting exercise to apply the latest bleeding edge frameworks and technologies to an observatory, it may be more useful to utilize software technologies that have been in use for at least several years and appear to be widely accepted with a long-term support projection. It will be helpful to standardize on a handful of languages and technologies that will be used for MSE. This allows for the development and reuse of existing libraries. At CFHT the majority of software is written using C, Java, or Bash scripts with Python emerging as a language of choice among astronomers involved with data reduction. At this point, these noted languages are both popular and mature. The challenge with MSE is finding the correct balance between popularity, maturity, and support for the languages and platforms that are chosen as part of its future.

3. OPERATIONAL OBSERVING ARCHITECTURE

Figure 3 illustrates the operational observing architecture for MSE. A Queued Service Observing (QSO) mode matching the current operating model for CFHT consists of three primary phases. QSO will be the exclusive operating mode of MSE. The first phase contains the software required for proposal submission, program definition, and observational scheduling. The second phase contains the software required for the execution of observations and resulting real-time analysis of acquired data. The final phase contains the software required for the archiving, in-house reduction, and distribution of science data to the principal investigators (PIs).

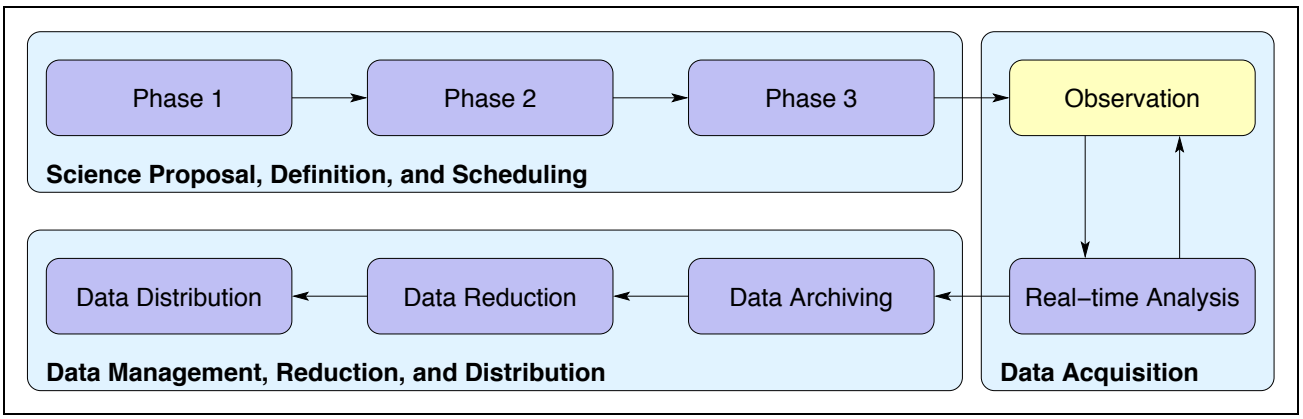

Figure 3. Operational observing architecture

The "Observation" block within the diagram is highlighted, since it will be a focal point of the software within this paper. The software within this block is responsible for configuring the observatory, telescope, and instrument and

acquiring science data. At this point, the phase 1 definition (science proposal, definition, and scheduling) as well as the phase 3 definition (data management, reduction, and distribution) aren't complete to the extent that the software to support these areas can be fully designed. As a result, the remainder of this paper will focus primarily on the software to support science observations and overall data management.

4. CONTROL SYSTEM INFRASTRUCTURE

Figure 4 illustrates the general categories of software necessary to acquire on-sky science data and operate an observatory. These categories are broken into hardware control, interfaces, and general infrastructure. Each of the categories contains some components that are highlighted within a dark box. These components contain software that could be reused from what currently exists at CFHT.

At the hardware control level, MSE will have an entirely new telescope, mirror, instrument, and certain facility components such as the dome enclosure. In general these new components will require newly designed software. Some parts of the facility will remain intact from what CFHT already has. Examples include parts of the power infrastructure, glycol cooling, dry air system, lighting, computer room humidifier, water system, security, and IT infrastructure.

The second category contains various software interfaces. These interfaces include human interfaces or GUIs, QSO interfaces for the queue system, autonomous interfaces, and engineering interfaces. All four interface categories currently exist at CFHT and it is likely that some of the existing GUI, QSO, and autonomous interfaces will be adapted or modified in some way to be used for MSE.

The final category covers the software infrastructure. This infrastructure is composed of fairly generic elements that are used at all major observatories. CFHT currently has a well-defined and robust infrastructure that can be extended for use with MSE.

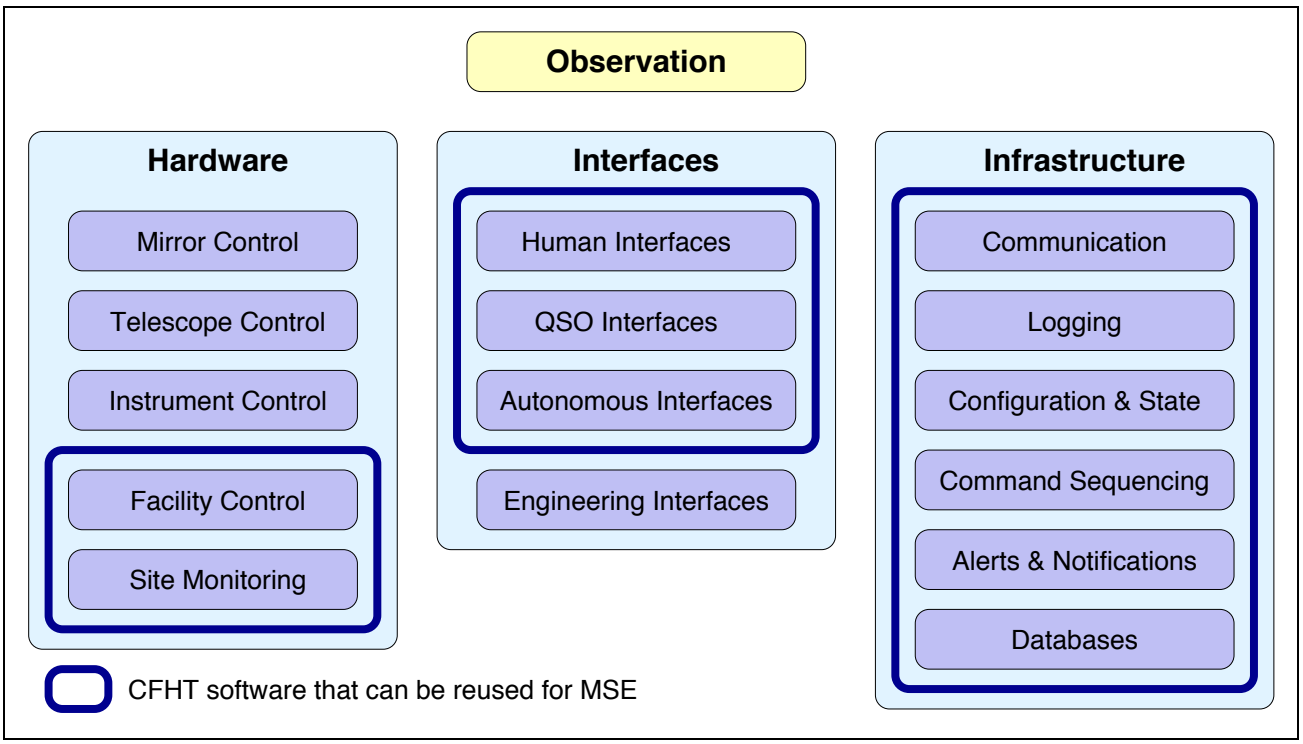

Figure 4. Control system infrastructure

5. SOFTWARE CONTROL ARCHITECTURE

Figure 5 expands upon the control software within the block labeled "Observation" in figure 3. This software is responsible for configuring the telescope, observatory, and instrument in order to acquire science data. MSE will be operated remotely from Waimea without staff present at the observatory. CFHT had a successful transition to remote observing at the beginning of 2011 and the plan is for MSE is to follow the same approach.

There will be two primary control interfaces for MSE. The first is the observing interface. This interface consists of commands generated by the queue system during the normal observing process as well as commands generated from GUIs or an engineering-based command line interface. These commands are sent via a TCP/IP socket to the observatory control sequencer. The second interface is an autonomous rules engine. This interface is necessary in order to support remote observing. This software constantly monitors a set of conditions that could trigger the initiation of autonomous actions. An example from CFHT, is the closing of the dome shutter and mirror covers when precipitation is detected. The autonomous rules engine is designed to initiate activities to protect the observatory, but it can also be used to initiate background actions as part of normal operations. For example, at CFHT the autonomous rules engine is used to modify the dome vent configuration based on the current telescope pointing, wind direction, and wind speed in order to optimize dome flushing. In the long-term future, perhaps MSE will be operated autonomously. Since the input to the observatory control sequencer is a TCP/IP socket connection with commands for each initiated activity, it should be possible to modify the observing interfaces and autonomous rules engine components to support autonomous observing.

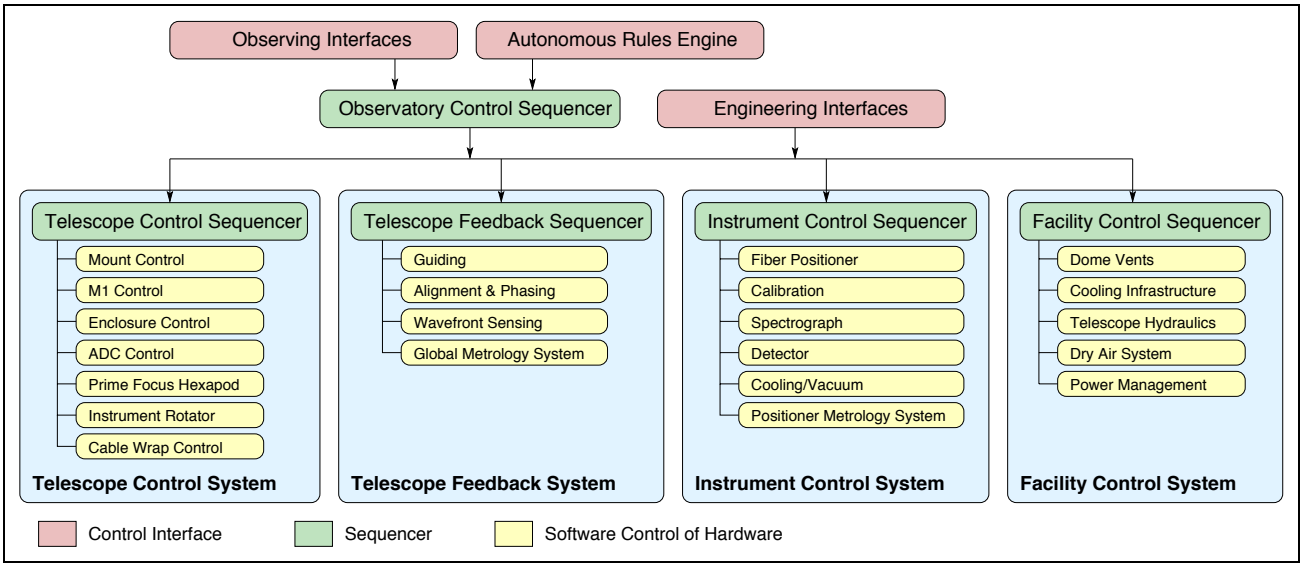

Figure 5. Software control architecture

The observatory control sequencer receives commands for all initiated telescope, instrument, and facility actions via a TCP/IP socket. A single high-level command may result in multiple lower-level actions. It is the responsibility of the observatory control sequencer to handle the high-level commands and initiate the lower-level actions by interacting with each of the low-level sequencers. For example, a new on-sky target request would result in configuration commands being sent to the telescope control sequencer, telescope feedback sequencer, and instrument control sequencer.

Below the observatory control sequencer are three primary areas of control software. Each of these areas of control contain a sequencer that is focused on configuration and control of components within a logical subsystem. The role of each sequencer is to synchronize action in a robust and efficient manner.

6. COMMON SYSTEM CONTROL ARCHITECTURE

Within figure 5 it may be apparent that some software reuse can be realized between subsystems. This leads to what is known as the common system control architecture. An illustration of this architecture can be found in figure 6. This architecture essentially expands on the control system architecture from figure 4.

Starting at the center of the diagram, the core software is composed of a hardware and software stack of components that start from hardware at the bottom layer to high-level sequencing at the top layer. Between the sequencer and hardware layers is a layer labeled "agents". A software agent is a component that handles communications and control for a hardware subsystem. Between the software agent and hardware itself there could be a device driver. An example of a software agent, which is used for all the instrument control systems at CFHT, is a detector control agent. In this example, a Linux device driver is used to handle PCI fiber cards. The detector control agent interacts with the detector controller on the other end of the fiber by sending requests and transferring pixel data, which is ultimately stored on the file system.

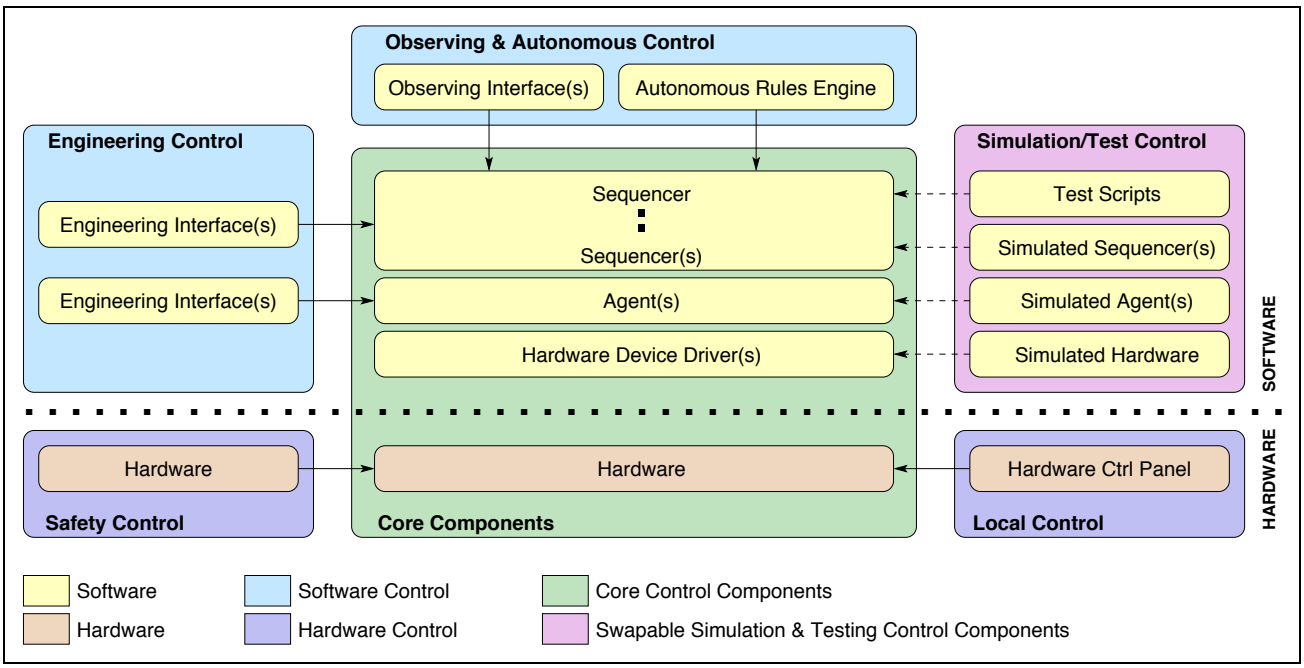

Figure 6. Common system control architecture

At the bottom of the diagram below the dotted line is a hardware only layer. Any low level safety interlocks and local hardware control are performed at this layer. The hardware layer is classified by the use of PLC devices (such as Allen Bradley PLCs) or custom hardware. Although actions, alerts, and warnings may be initiated by high-level software, philosophically the ultimate responsibility for both personnel and equipment safety is kept at the hardware level.

On the left side of the figure are boxes indicating engineering interfaces. For engineering and testing purposes it is possible to interact with both the sequencers and software agents through engineering interfaces. At the most basic level this can be done via a command line interface. Commands can be sent to both the sequencer and agents via a TCP/IP telnet style socket connection.

On the right side of the figure are boxes indicating ways that various parts of the system can be tested or simulated. Each of the layers that make up the core components can be substituted with simulation sequencers, agents, or hardware. In addition, at the top layer the observing interfaces and autonomous rules engine can be replaced with test scripts as necessary.

The common system control architecture is designed to be flexible and adaptable. The input at both the sequencer and agent levels is a simple command/response protocol over a TCP/IP telnet style interface. As a result, the middleware communication layer doesn't dictate the required programming language for each layer. At CFHT, the sequencer and agent layers are currently implemented in C. However, the flexibility exists to use a different language if the requirements dictate a better solution or if the future yields a better alternative. Almost all languages nowadays support TCP/IP communications. In addition, by supporting a simple TCP/IP interface it becomes simple to diagnose problems at the network level, if necessary.

7. DATA MANAGEMENT

Up until this point, this paper has focused on the software involved with hardware control. This is obviously a big part of the software infrastructure required for MSE. The other major part of software is everything involved with the capture, storage, and manipulation of data. Figure 7 contains an overview of the data management infrastructure for MSE.

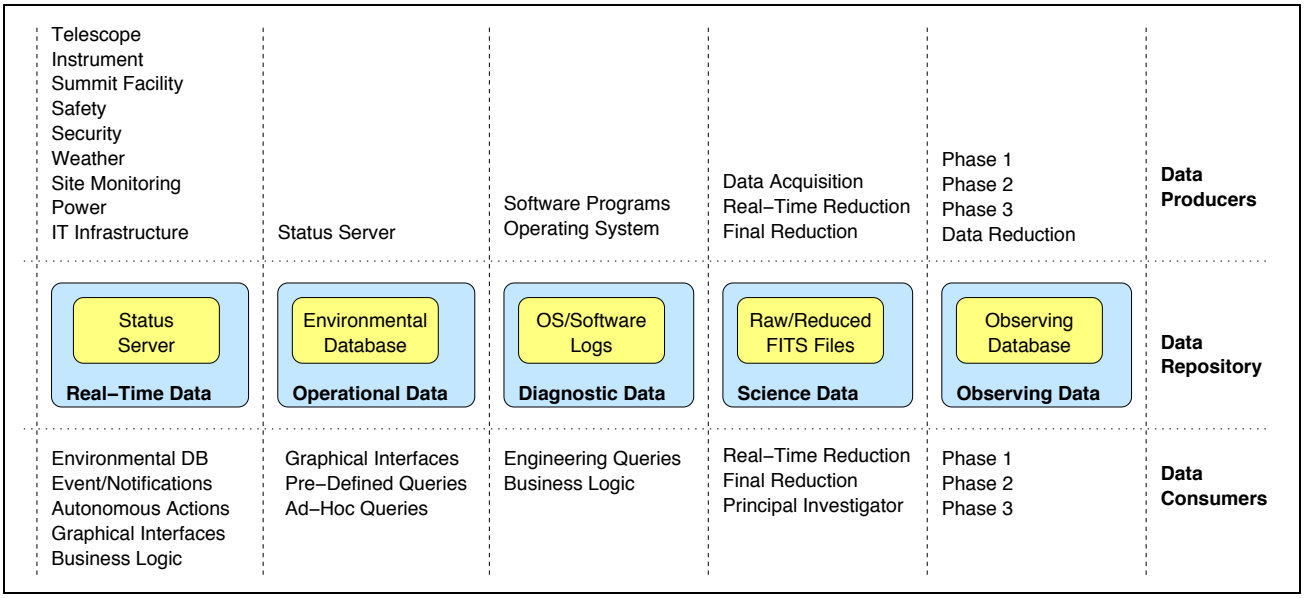

Figure 7. Data management breakdown

The figure breaks down the data management infrastructure from the basis of data producers, data repositories, and data consumers. It is possible for some systems to be both producers and consumers of data. The repositories that are listed in the figure are repositories that currently exist at CFHT. It is quite possible that these repositories can be extended for use with MSE.

7.1 Real-Time Data

The Status Server is an in-memory, real-time repository containing the bulk of all state and status information across the observatory. The Status Server provides capabilities that are similar to EPICS Channel Access. Clients can publish, retrieve, and subscribe to data stored within the repository. The ability for clients to subscribe to data stored in the repository and be notified on changes allow for the creation of useful client software such as notification, data logging, autonomous action initiation, and GUIs. The Status Server was developed at CFHT many years ago and has become a core component within the data management infrastructure. Information within the Status Server is stored as hierarchical name-value pairs and can be traversed in a similar fashion to a file system.

7.2 Operational Data

Since the Status Server does not keep historical information, data values that are useful to archive are stored within an environmental database. The primary producer of information for this database is a Status Server client. This client receives notifications when subscribed items in the Status Server change and updates the environmental database as necessary. CFHT has been utilizing a customized database for this purpose in the past and is currently evaluating the feasibility of using a no-SQL database such as MongoDB for this purpose in the future. If this evaluation proves successful, this will be the approach used for MSE. Clients of the operational data repository include GUIs, plots, pre-defined queries and customizable ad-hoc queries.

7.3 Diagnostic Data

As part of software maintenance and post failure analysis, diagnostic data is critical. The question with diagnostic data isn't really whether it should be logged. Instead, the question is often how much to log, how to log it, and where to store this information. There are both advantages and disadvantages to having diagnostic log data stored in a single repository. On one hand it is convenient to cross-correlate information. However, this can come at the expense of performance. In addition, a consolidated repository can create additional network dependencies in case of failures. With MSE the proposed approach is to use localized logging with possible downstream aggregation as necessary.

7.4 Science Data

Without a doubt, the most valuable data product at an observatory is the science data. More work must happen on the science requirements and instrument definition side to identify what the exact science data products will be for MSE. These data products will include reduced science data, since a data reduction pipeline is part of the software requirements for MSE. Access methods to the science data archive will be defined for the science community. At this point in time, the location of the MSE science data archive hasn't been finalized.

7.5 Observing Data

In order to support Queued Service Observing (QSO) a database will be used to store data associated with PI proposal and program definition as well as execution, analysis, reduction and distribution. This database must contain the information associated with each of the stages defined in figure 3. At CFHT, a standard relational database was developed to support the phase 1, phase 2, and phase 3 components involved with QSO.

NEXT STEPS

The next major step in the MSE project is a conceptual design review at the end of 2016. As part of this process, the requirements with regards to the software are being finalized such that a proper conceptual design can be written. In addition, CFHT is investigating what role existing available internal as well as external software could play within the design for MSE. While it would be an interesting and fun exercise to develop all new software for MSE from scratch, realistically this likely won't yield a final solution that is more robust and higher quality at a lower cost than evaluating existing solutions that could become candidates for modification and reuse.

REFERENCES

[1] Simons, D. A., Crampton, D., and Cote, P., et al, "Current status and future plans for the Maunakea Spectroscopic Explorer (MSE)", SPIE Proc. 9145 (2014)

[2] Szeto, K., Vogiatzis, K., Hangan, et al, "Conceptual design study to determine the optimal vent configuration for the Maunakea Spectroscopic Explorer (MSE)", SPIE Proc. 9145 (2014)

[3] Szeto, K., et al, "Mauna Kea Spectrographic Explorer design development from feasibility concept to baseline design", SPIE Proc. 9906 (2016)

[4] Bauman, S., Green, G., Szeto, K. "MSE observatory upgrade: a revised and optimized astronomical facility", SPIE Proc. 9906 (2016)

[5] Zhang, K., Zhu, Y., Hu, Z. “Mauna Kea Spectrographic Explorer (MSE): conceptual design of multi-object high resolution spectrograph”, SPIE Proc. 9908 (2016)
[6] Minot, S., et al, “Systems budgets architecture and development for the Maunakea Spectroscopic Explorer”, SPIE Proc. 9911 (2016)
[7] Saunders, W., Gillingham, R., “Optical designs for the Maunakea Spectroscopic Explorer Telescope”, SPIE Proc. 9906 (2016)
[8] McConnachie, A., “Science-based requirement and operations development for the Maunakea Spectroscopic Explorer”, SPIE Proc. 9906 (2016)
[9] Murowinski, A., McConnachie, A., Simons, D., Szeto, K., “Mauna Kea Spectroscopic Explorer: the status and progress of a major site redevelopment project”, SPIE Proc. 9906 (2016)
[10] Spano, P. “MSE spectrograph optical design: a novel pupil slicing technique”, SPIE Proc. 9147 (2016)
[11] McConnachie, A., et al, “The Maunakea Spectroscopic Explorer: the science driver design rationale”, SPIE Proc. 9145 (2014)
[12] Flagey, N., et al, “The Maunakea Spectroscopic Explorer: throughput optimization”, SPIE Proc. 9908 (2016)
[13] Flagey, N., et al, “Spectral calibration for the Maunakea Spectroscopic Explorer: challenges and solutions”, SPIE Proc. 9910 (2016)