THIRTY METER TELESCOPE OBSERVATORY SOFTWARE ARCHITECTURE

K. Gillies#, C. Boyer, TMT Observatory Corporation, Pasadena, CA 91105, USA

Abstract

The Thirty Meter Telescope (TMT) will be a ground-based, 30-m optical-IR telescope with a highly segmented primary mirror located on the summit of Mauna Kea in Hawaii. The TMT Observatory Software (OSW) system will deliver the software applications and infrastructure necessary to integrate all TMT software into a single system and implement a minimal end-to-end science operations system. At the telescope, OSW is focused on the task of integrating and efficiently controlling and coordinating the telescope, adaptive optics, science instruments, and their subsystems during observation execution. From the software architecture viewpoint, the software system is viewed as a set of software components distributed across many machines that are integrated using a shared software base and a set of services that provide communications and other needed functionality. This paper describes the current state of the TMT Observatory Software focusing on its unique requirements, architecture, and the use of middleware technologies and solutions that enable the OSW design.

TMT INTRODUCTION

The Thirty Meter Telescope (TMT) is an advanced, wide field (20 arcmin), altitude-azimuth telescope with a primary mirror consisting of 492, 1.44 meter segments. At first light, a facility multi-conjugate adaptive optics (MCAO) system will be available using a laser guide star (LGS) system. The facility's twin Nasmyth platforms will concurrently support multiple instruments, which are all available during the night for observations. Two science instruments will be delivered for use with the MCAO-LGS system: IRIS, a near-infrared instrument with parallel imaging and integral-field-spectroscopy support; and IRMS, an imaging, multi-slit near-infrared instrument. A seeing-limited, wide-field, multi-object optical imaging spectrograph (MOBIE) will also be available at first light. The operations model includes PI-directed observing from remote facilities and on-site service observing by staff. The telescope and facilities are currently in the advanced design phase. The telescope is planned for the summit of Mauna Kea on the island of Hawai'i in the United States.

TMT SOFTWARE ARCHITECTURE

Significant aspects of software systems for large observatories have converged around a few common ideas and solutions due to similarities in the facilities, their requirements for operation and use, and the issues related to development and maintenance [1]. The TMT software architecture takes advantage of these prior solutions when possible. It's then possible to focus attention on the problems unique to TMT and reuse common solutions for the parts of the software system that are known or of little risk. We can also improve upon the solutions used in the previous generation of systems when experience has shown that aspects of the known solutions have issues.

The complexity of some aspects of the TMT software control system scale with the telescope aperture size, and the software must also scale to handle this complexity. Segmented mirror control for TMT requires more moving parts behind the mirror and with it more sophisticated control software than existing segmented mirror systems.

It's also true that not every aspect of TMT software complexity scales with the size of the aperture. Many things that work for 8m class telescopes work just as well with TMT. An example is proposal submission and planning software.

A complete description of the architecture for a system the size of the TMT software system requires more than a few pages. Table 1 shows some of the broad range of challenges related to the software that executes at the telescope site. This paper will show one way the architecture addresses the challenges of Table 1.

Table 1: Architecture Challenges

Challenge	Description
Acquisition	There are demanding requirements on the system performance and coordination for target acquisition and observation setup.
Composition challenges	Some components must be used in a variety of situations or composed in different ways to support different applications. For instance, the MOBIE wavefront sensors must act as part of MOBIE but also work as part of the phasing system.
Wavefront measurement	Wavefront measurement functionality and hardware is potentially spread throughout telescope systems and instruments making it a challenge to cleanly decompose the system and software. This was a problem with 8m class telescopes as well.
Distributed development	Multiple, distributed, international partners provides a new level of software construction and management complexity.
Operations maintenance	The long lifetime of TMT requires an architecture that can be enhanced and modified without impacting the operations system.

#kgillies@tmt.org

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

Architecture Overview

TMT has adopted the idea of a technical architecture and functional architecture from other recent large telescope projects (e.g. ALMA [2], ATST[4]). The *technical architecture* is the software infrastructure that provides the foundation for the functional architecture. System features, such as logging and command support, are part of the technical architecture. The *functional architecture* consists of the decisions and software components that enable the activities of the observatory from the point of view of the users. For instance, how does the system collect header information for a science data frame?

A software system should be viewed in many ways. At the highest level in the functional architecture, TMT software is modeled as the 5 large *principal systems* shown in Figure 1. Each principal system is focused on specific, easily-identifiable functionality, and each is itself a collection of other software components or systems. Viewing the system at this level allows one to think more easily about flow of control and where software functionality exists within the system.

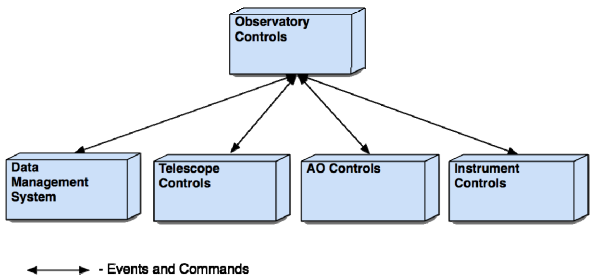

Figure 1: One view of the TMT software architecture splits the system into five large principal systems.

This view shows communication is hierarchical and flows down from Observatory Controls to the other principal systems. This command communication is low-bandwidth by design; any high-speed communication occurs within a single principal system.

Figure 2 drills down one more level to show sub-systems that represent major hardware and software components within the principal systems. At the top of the figure are the observing user interfaces and the software components within Observatory Controls that sequence and synchronize the activities of the other systems to generate the user's desired science data.

The integration of these software components requires software infrastructure that is outside the scope of the individual components themselves. The horizontal bar dividing Figure 2 in half represents this software infrastructure. The idea of shared software infrastructure based on a set of services and associated software focused on integration is a successful strategy in large observatory software architecture [1, 2, 4]. TMT calls this software Common Software (CSW). CSW is the implementation of the technical architecture.

Common Software is a collection of software and services. A *service* is a set of related software functionality together with behavior and the policies that control its usage. TMT CSW uses external packages (i.e., software not developed by TMT–COTS or open source middleware, etc.) to implement the CSW services. Abstractions and wrappers are present between the CSW services and the external packages to enable some isolation from specific product choices. For a component programmer integration of a component with TMT requires the use of a service-oriented API and library code that must be linked with the component.

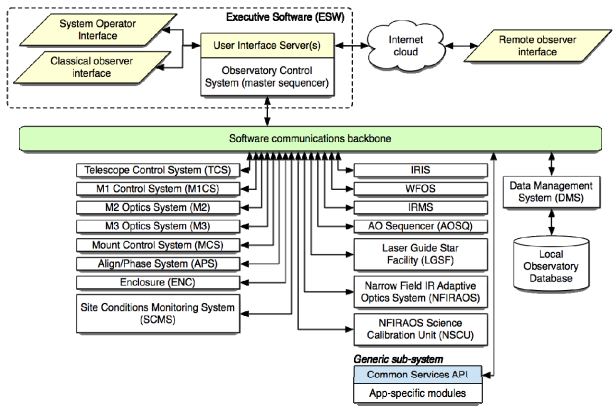

Figure 2: The major systems of the Telescope Controls, AO Controls and Instruments connected by the software infrastructure provided by the technical architecture.

Table 2 is a list of services provided by CSW that are needed to enable the functionality of the Functional Architecture.

Table 2: List of Planned CSW Services

Service	Task Description
User single sign on	Centrally manage user authentication and access control
Commands	Support for subscribing to, receiving, sending, and completing commands in the form of configurations
Location/Connection	Locate and connect to components within the distributed system
Events/Telemetry	Enable event-based functionality based on publish, subscribe paradigm
Alarm/Health	Support monitoring and publishing component alarm and health signals
Configuration	Manage initial values and system and component configurations history
Logging	Capture and store system logging information
Time	Standards-based, precision time access

One important example is the event services. These services allow one component to send a piece of information (i.e., an event) to one or more other

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

components. The Event Service provides a high-performance, publish-subscribe message system. Components can signal actions (events), publish status information (telemetry), or publish control information (event streams). One big advantage of this type of service is that the publishers and subscribers are decoupled; each requires no knowledge of the existence of the other.

OBSERVING MODE ORIENTED ARCHITECTURE

Operations experience has shown there are drawbacks to limiting the modeling and construction of the software structure to the principal system level, and this has driven architecture changes for TMT. For instance, principal systems are large systems that must handle all observing modes and operations scenarios. This results in broad, complex software interfaces that are difficult to verify and modify during operations. It continues to be valuable to view the system at the principal system level, but to address this and the challenges of Table 1 a more flexible approach is required. Flexibility is a key design concept that can result in a software system that can respond to the changing needs of science operations over the project lifetime.

The functional structuring approach planned for TMT that addresses these issues is called Observing Mode Oriented Architecture (OMOA) [3]. An observing mode is a well-defined instrument observing task and an associated set of owned resources, procedures, and capabilities that implement the mode. An example observing mode for TMT is: IRIS multi-filter integral field spectroscopy using the NFIRAOS adaptive optics unit with AO laser guide star correction. An instrument will generally have several associated observing modes for acquisition, science objects, and calibrations.

A goal of this architecture is to eliminate software waste and run as little software as is necessary to execute an observation using a specific observing mode. To accomplish this, the software within principal systems must consist of smaller components that are then assembled into a dynamic system configuration that is specific to the observing mode.

To explain this approach, the software components you might find in a principal system are shown as layers in Figure 3 with specific responsibilities described in the following sections.

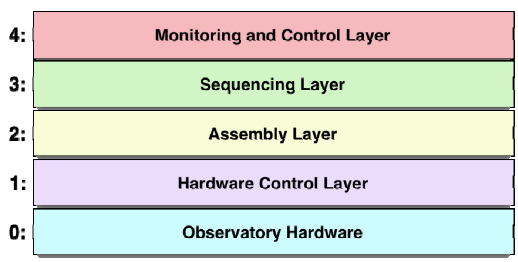

Figure 3: OMOA software structure layers.

Hardware Control Layer

In 2011 the trend is towards motor controllers and other hardware controllers and sensors that are network-resident devices capable of controlling multiple channels or to Programmable Application Controllers communicating via high-level commands over a standard TCP/IP-based network. It is generally no longer necessary to develop single use, low-level device drivers; a software investment that requires skilled programmers and significant effort that generates long-term technical debt and hinders change during operations.

The lowest layer in the OMOA software system, called the Hardware Control Layer, consists of all the controllable hardware that is available for use by higher levels of software. A sea of similar software components called Hardware Control Daemons (HCD) at layer 1 controls the TMT low-level hardware of the telescope, adaptive optics, and instruments.

Each HCD is associated with one or more networked motion controllers or other low-level hardware controllers (shown as layer 0 in Figure 3). The HCDs act as adapters and provide a uniform software interface and feature set focused on device control to the layers above. The HCDs are *always executing,* and each can be accessed at any time by the software layers above.

This layer is one place where external systems can be integrated. As an adapter, a HCD can isolate a proprietary connection to an external system making it look like a conforming system device. The HCD also provides a suitable location for device simulation allowing device end-to-end system testing without hardware presence.

Assembly Layer

The Assembly Layer exists just above the Hardware Layer at layer 2 in Figure 3. Software at this layer consists of components called *Assemblies* with two roles in the OMOA. The first role is to allow the grouping of HCDs into higher-level entities. This is required when individual hardware devices must be considered as a unit or requiring processing. The second role of components in the Assembly Layer is to provide more sophisticated hardware control functionality that integrates devices across different HCDs to produce higher-level devices or add uniformly useful capabilities.

Assemblies can be transient or long-lived. An example of a long-lived Assembly is one that provides telescope pointing, tracking, and offsetting. An Assembly can also be created dynamically to provide combinations of HCDs that need to be coupled for a specific observing mode during an observation. An example might be the coordination of wavefront sensor detector readout processing and the control of the probes for the wavefront sensors.

Sequencing Layer

The Sequencing Layer is layer 3 in Figure 3. Components at this level are called sequencers because they control and synchronize the actions of the HCDs and Assemblies. The sequencer components are dynamically

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

created and composed in order to execute a specific observing mode. This approach is possibly the most innovative part of this software architecture, because it is this layer that gives the software its compositional flexibility and other qualities. Individual sequencers can provide higher-level control of a set of distributed hardware. The ability to cut across the development hardware boundaries and compose and control hardware as needed is what allows this approach to eliminate many of the challenges mentioned in Table 2.

The components in this layer share software interfaces that allow them to be plugged together to form the sequencing engine for a specific observing mode. There can be one or many sequencing components in an observing mode sequencer. The goal is that the sequencing components for a mode be assembled into a single process. This minimizes complexity and performance issues related to distributed processes and communication of commands across process and machine boundaries. A large amount of software related to command input/output in systems is eliminated, and fewer tiers result in higher performance and simplicity.

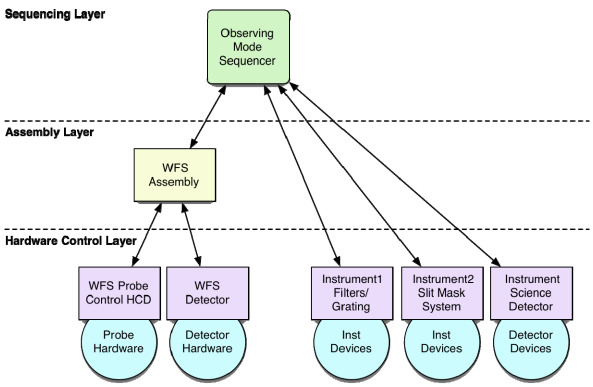

Figure 4: A simple example showing the lowest 4 layers of an OMOA observing mode configuration.

The sequencer for an observing mode is constructed during observation execution with a *Configuration Factory* that takes as input an observing mode and an observation description from another software tool such as a high-level planning GUI. The Configuration Factory has been programmed with instructions on how to construct the matching sequencing process for the observing mode.

Figure 4 shows a simplified, partial example of an instrument like TMT's MOBIE in a seeing-limited observing mode. At the lowest level are the HCDs for the wavefront sensor probes, instrument hardware, and detectors. An assembly exists to integrate and provide higher level functions for the wavefront sensor hardware and detector. All other sequencing for the observing mode is in the Observing Mode Sequencer that is created dynamically for the observation. The HCD components could be separate processes running on distributed hardware while the sequencing components exist in a single process within the Observatory Controls.

This example demonstrates the flexibility inherent in this approach. It allows the grouping of software and hardware in an optimal arrangement for a specific observing mode with only the functionality needed to support a specific mode. During operations this allows new observing modes to be rolled out more easily with minimal influence on current functionality.

Monitoring/Control Layer

The Monitoring/Control Layer (layer 4 in Figure 3) is the layer of software that contains the user interface programs that are used to observe with the telescope. At TMT there will be graphical user interfaces for use by observers during the night. These applications use the CSW services to control and monitor the system.

SUMMARY

The TMT software architecture is similar to the systems constructed for 8m telescopes, but it has been enhanced to take advantage of changes in software and hardware technology as well as 10 years of experience with the principal system architecture approach.

The TMT technical architecture is based on a set of shared software services, each of which is itself based on open-source or commercial software.

The functional architecture uses the structuring approach of the Observing Mode Oriented Architecture. The goal of OMOA during observation execution is to allow, for any given observing mode, the minimal amount of software needed to execute the mode. By taking this approach, many of the problems outlined in Table 1 are minimized or eliminated.

The innovation in OMOA is the implementation of the principal systems as more focused fine-grained components at the architectural level. This single change provides the opportunity to reduce the amount of software needed at the telescope, thereby reducing the complexity of the runtime system. This, coupled with the use of dynamic system configurations focused on executing individual observing modes, addresses the need to sequence systems with more flexibility and higher performance than currently possible.

REFERENCES

[1] K. Gillies, J. Dunn, D. Silva, "Defining common software for the Thirty Meter Telescope," Proceedings of SPIE Vol. 6274, 62740E (2006).

[2] J. Schwarz, G.Chiozzi, P. Grosbol, H. Sommer, D. Muders, ALMA Software Architecture, http://www.alma.nrao.edu/development/computing/docs/joint/draft/ALMASoftwareArchitecture.pdf.

[3] K. Gillies, S. Walker, "An observation execution system for next-generation large telescopes," Proceedings of SPIE Vol. 7740, 7740Q (2010).

[4] S. Wampler, B. Goodrich, "ATST Software Concepts Definition," Document SPEC-0013, Rev B, http://atst.nso.edu/files/docs/SPEC-0013.pdf

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)