Draft version February 26, 2020
Typeset using LaTeX **modern** style in AASTeX62

LSST Observatory System Software Architecture

Ribeiro Tiago[1]
[1]*LSST Project Office, 950 N. Cherry Ave., Tucson, AZ 85719, USA*

(Dated: February 26, 2020)

ABSTRACT

This paper contains a description of the Software Architecture for the Vera Rubin Observatory.

1. INTRODUCTION

This paper describes the Vera Rubin Observatory Software Architecture. The Software Architecture contains high-level decisions that dictates how software for different components interact, how it is designed to control hardware and how users interact with the system.

...

2. OBSERVATORY SYSTEM SOFTWARE ARCHITECTURE

The LSST control system is based on a reactive data-driven actor-based architecture that uses a multi cast Data Distribution Service (DDS) messaging protocol middleware. component of the system (not all components are displayed here).

The LSST System Architecture is comprised mainly of;

- The Service Abstraction Layer (SAL[1]) communication middleware. Based on the DDS protocol, it provides interfaces for all the project adopted programming languages (LabView, C++, Java and Python).
- Engineering and Facility Database (EFD).
- SAL-aware reactive components, a.k.a Commandable SAL Components (CSCs).
- LSST Operators Visualization Environment (LOVE).
- User interface servers.

The SAL middleware is the backbone of the LSST system architecture. It is a high level layer on top of Data Distribution Service (DDS), a standard message passing

[1] https://docushare.lsstcorp.org/docushare/dsweb/Get/Document-21527/

system. LSST uses the PrismTech OpenSplice DDS library, community edition. It implements three distinct types of messages; Commands, Events and Telemetry, with distinct purposes. Commands are sent to a specific component, which must acknowledge its receipt and perform some action. In general, the receiving component will be the only entity listening for the commands it accepts[2]. Events and Telemetry are messages broadcast by components to the middleware and are available to any entity on the system to receive. The distinction between Events and Telemetry is that Events are output when conditions change, whereas Telemetry is output at semi-regular intervals. As such, it is much more important that Events be transmitted reliably than Telemetry. We cannot afford to lose any events, but we can lose occasional Telemetry. Thus Events are sent using a higher Quality of Service (QoS).

The EFD is responsible for capturing all SAL messages broadcasts to the middleware (including Commands, Events and Telemetry) and storing that information into a database.

CSCs are the main actors of the LSST system architecture. They are responsible for managing the incoming traffic of data and take appropriate actions, controlling hardware (e.g. M1M3, M2, Mount Controller, etc in Fig. **??**), software (e.g. Optics Controller Reconstructor, DMCS Interface, etc in Fig. **??**) or even other CSCs (e.g. ScriptQueue, TCS, ATCS, OCS, etc in Fig. **??**).

LOVE is responsible for capturing SAL messages and displaying them in a useful way for general users, providing some basic interface to query and analyze data from the EFD, an interface to issue pre-defined commands to a set of components and user interaction with the ScriptQueue (see Sect. **??**).

The SAL processes XML based definitions of the Commands, Events, and Telemetry for each CSC. Using this information, it creates runtime objects which support the messaging required. These take the form of shared libraries (C++, Python, LabVIEW) or Jar archives (Java) which implement consistent namespaces and API's. Other assets such as Simulated data, Sql table definitions, and web based documentation, may also be generated. On top of these low level APIs, developers have access to two higher-level set of frameworks; Python SalObj[3] library and the LabVIEW component template. No higher level framework is supported for implementations in Java or C++.

Overall, the system architecture can be divided into three main namespaces; Observatory, Main Telescope (MT) and Auxiliary Telescope (AT). The Observatory is the highest level and encapsulates both the Main Telescope, Auxiliary Telescope and global components such as the script queue, scheduler, environment awareness system, etc (see Fig. **??**).

2.1. *SalObj - Python and scripting*

[2] But note that the EFD, for instance, will also be listening for commands, though it will not acknowledge them.

[3] https://github.com/lsst-ts/ts_salobj

SalObj is a Python library that provides a pythonic and object-oriented interface for SAL components such as CSCs and SAL scripts (see Sect. **??**). The library defines two sets of base classes that are mirror to each other, Remote and Controller. A Remote will send commands to and receive telemetry and events from a specific component whereas a Controller will receive commands and publish telemetry and events. SalObj also provides BaseCsc, a subclass of Controller that handles the standard state transitions and is intended to be used as a parent class for CSC.

Internally, SalObj uses the python library asyncio[4] to handle the inherently asynchronous nature of the SAL messaging system.

2.2. *Hardware interface components*

Probably the most critical or sensitive components of the LSST system architecture are those that directly control hardware. Some of these components are going to be delivered directly by external vendors, such as those that will control the main telescope mount (MTMount) and the main telescope secondary mirror (MTM2). There are also those that are developed in house, e.g. the main telescope M1M3 (MTM1M3).

In some special cases, where fast real time response is required, it is highly desirable that the control software and hardware are part of an integrated system. For those systems, the components are developed either using the LabView component template, which is part of the LSST infrastructure or in C++ developed using the low level SAL API.

In most other cases, the hardware comes with control software that can be easily interfaced by using standard protocols (such as TCP/IP or serial ports), and there is no special need for the component software to reside close to the low level hardware controller. In those cases, the components are written in Python using the SalObj library which is also part of the LSST infrastructure. By writing these components using a unified language and library (Python+SalObj) we allow a high level of flexibility and maintainability of the software and considerably decrease the development cycle.

2.3. *Pure software components*

In the LSST System Architecture there are a number of components that, even though they do not control hardware directly, dictate what hardware components are supposed to do. Some of these components are responsible for heavy computational routines, such as the Optical Feedback Control (MTOFC), which is responsible for applying corrections to both M1M3, M2 and hexapod components for the main telescope or even the Scheduler, which is responsible for processing an entire set of observatory telemetry information and history of observations to compute an observing queue.

[4] https://docs.python.org/3/library/asyncio.html

These pure software components are mostly written in Python using SalObj library. There are three special cases of these components that form the basis of the LSST System Architecture; the ScriptQueue (Sect. **??**), Control Systems (Sect. 4.1) and the Watcher (Sect. 3.4). Together, they provide the tools needed for integration, commissioning and operation of the observatory.

2.4. *Configuration Management*

During commissioning and operations, LSST will have a large number of running software components under the purview of DM, Camera, and TSS. In general, the behavior of each of these components is modifiable through configuration information which is read in during startup of the component, or possibly changed while the component is running. Careful management of this configuration information is crucial to reliable functioning of the Observatory, and to the analysis of its data products.

There are basically two ways configurations are managed, using a configuration database or version management, for ascii-file based configuration. Configuration databases can use any type of database technology as long as versioning control is properly implemented and verified.

For file-based configuration, version control is done using git repositories Git is already a standard in industry as a software management tool and has becoming increasingly used to manage general documents and files as well, not to mention that it is already readily available and broadly adopted by the project. Therefore, each component must be capable of handling a git-based configuration repository. These configuration repositories will be hosted on a configuration server at the summit so that, even if communication with the base or the internet is not available, components still maintain access to their configuration repositories.

Several options for configuration file format, and their associated software tools, have been considered. Each of the available options naturally has its strengths and weaknesses, and none stand out as being particularly useful for all LSST use cases (and/or available for all the project adopted programming languages). The standard adopted for LSST software components is YAML (https://yaml.org). If a specific component is developed in a language without support to YAML, a waver may be granted.

3. MONITORING AND INTERACTIVE SYSTEMS

This section contains a description of the monitoring and interactive systems.

3.1. *LSST Operations and Visualization Environment (LOVE)*

Description of LOVE interface.

3.2. *User interface servers*

Description of jupyter lab server for control and data analysis. Reference LSP/nublado

3.3. *Script Queue*

Description of the Script Queue.

3.4. *Watcher*

Description of the Watcher.

4. MAJOR COMPONENT NAMESPACES

4.1. *Observatory Control Systems*

Description of OCS components

4.2. *Main Telescope Control Systems*

Description of Main telescope components

4.3. *Auxiliary Telescope Control Systems*

Description of Auxiliary Telescope components

5. CONCLUSIONS

APPENDIX

A. REFERENCES

REFERENCES

B. ACRONYMS

Acronym	**Description**
API	Application Programming Interface
DAQ	Data Acquisition System
DM	Data Management
DMCS	Data Management Control System
EFD	Engineering and Facility Database
EPO	Education and Public Outreach
LOVE	LSST Operations Visualization Environment
LSP	LSST Science Platform
LSST	Legacy Survey of Space and Time (formerly Large Synoptic Survey Telescope)
M1M3	Primary Mirror Tertiary Mirror
M2	Secondary Mirror
OCS	Observatory Control System
SAL	Service Access Layer
TCS	Telescope Control System
TSS	Telescope and Site Software
XML	eXtensible Markup Language
YAML	Yet Another Markup Language