LSST Communications Middleware Implementation

Dave Mills[a], German Schumacher, and Paul Lotz

LSST , 950 N Cherry Ave , Tucson AZ 85719, USA

Abstract

The LSST communications middle-ware is based on a set of software abstractions; which provide standard interfaces for common communications services. The observatory requires communication between diverse subsystems, implemented by different contractors, and comprehensive archiving of subsystem status data. The Service Abstraction Layer (SAL) is implemented using open source packages that implement open standards of DDS (Data Distribution Service[1]) for data communication, and SQL (Standard Query Language) for database access. For every subsystem, abstractions for each of the Telemetry data-streams, along with Command/Response and Events, have been agreed with the appropriate component vendor (such as Dome, TMA, Hexapod), and captured in ICD's (Interface Control Documents).The OpenSplice (Prismtech) Community Edition of DDS provides an LGPL licensed distribution which may be freely re-distributed. The availability of the full source code provides assurances that the project will be able to maintain it over the full 10 year survey, independent of the fortunes of the original providers.

Keywords : LSST, DDS, Middle-ware, EFD

1. Introduction

The Large Synoptic Survey Telescope (LSST) [2] is a revolutionary facility which will produce an unprecedented wide-field astronomical survey of our universe using an 8.4-meter ground-based telescope. The LSST leverages innovative technology in all subsystems: the camera (3200 megapixels, the world's largest digital camera), telescope (simultaneous casting of the primary and tertiary mirrors; two aspherical optical surfaces on one substrate), and data management[3] (15 terabytes of data nightly, nearly instant alerts issued for objects that change in position or brightness).

This paper describes the architecture for the Communications Middleware which will mediate the interactions between the numerous component subsystems of LSST.

a. dmills@lsst.org

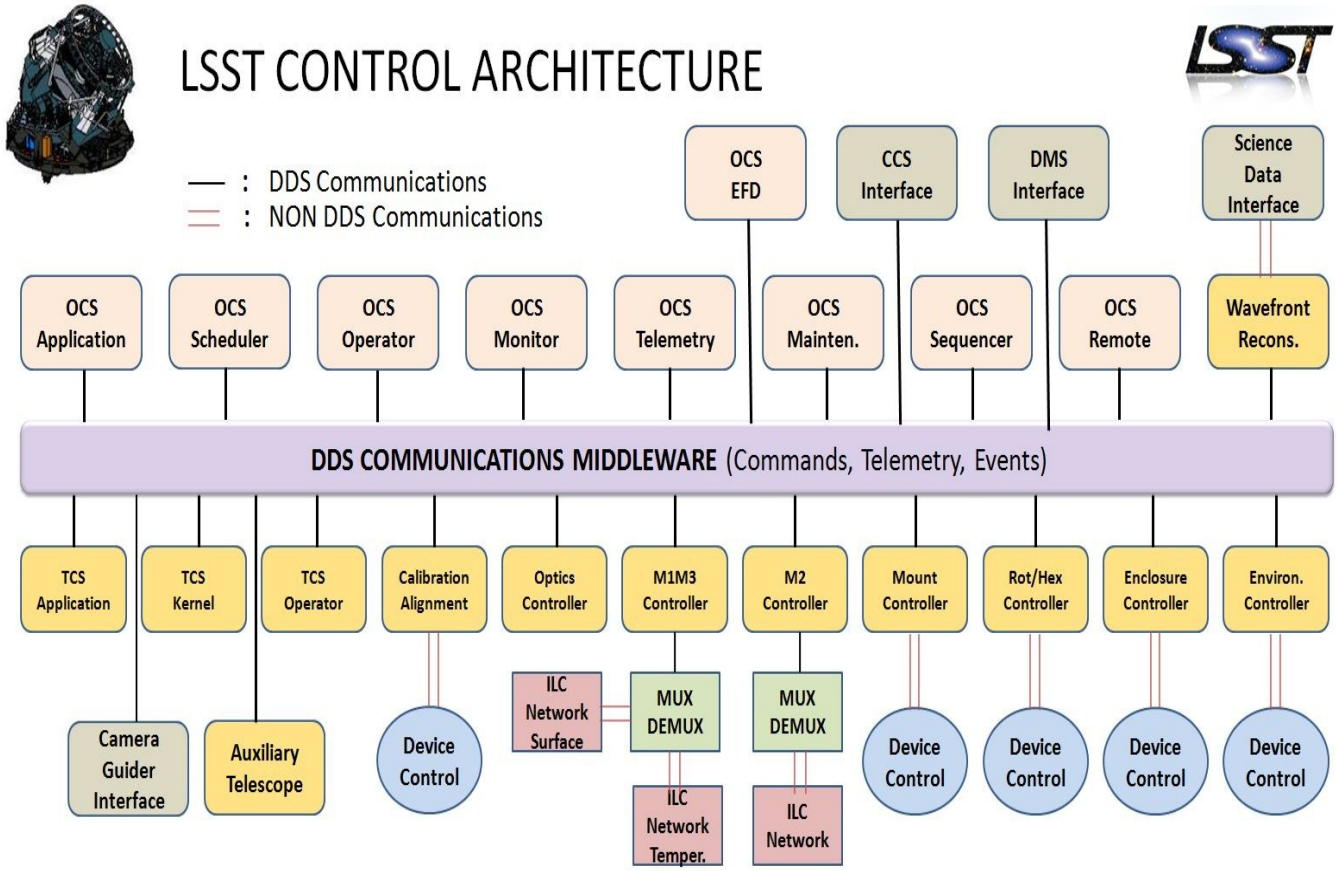

Figure 1 – LSST Control Architecture

Communications between all of the component subsystems of the observatory are mediated by a layer of middleware. This software provides Ethernet based communications using the Data Distribution Service (DDS) protocol. This is in turn layered on top of the Real Time Publish Subscribe (RTPS) protocol[4]. This is also an Open Standard, which permits transparent interaction between independent DDS implementations produced by different vendors.

A higher level package , the Service Abstraction Layer (SAL) , has been written to leverage the DDS services and make them easy for the application developers. The generation of the detailed specifications of the supported transactions, is facilitated by a database of syntactic elements know as the System Dictionary, which holds details of all the syntactic atoms which are defined and recognized in the application domain.

The middleware architecture has been modeled using the SysML[5] language, along with the detailed Telescope, and Observatory Control System designs.[6]

2. Service Abstraction Layer

Primary access to the communications services is provided by a set of core abstractions : Commands, Telemetry, and Events. For each abstraction an Extensible Markup Language (XML) schema is provided to assist in the definition, and maintain consistency throughout the system. For example, each Telemetry definition consists of basic meta-data such as subsystem, author, version etc, plus a set of strongly type data elements comprising the actual telemetry values. The structure of each abstract type is also reflected in the system wide Engineering and Facility Database (EFD), which records all the communications between the subsystems.

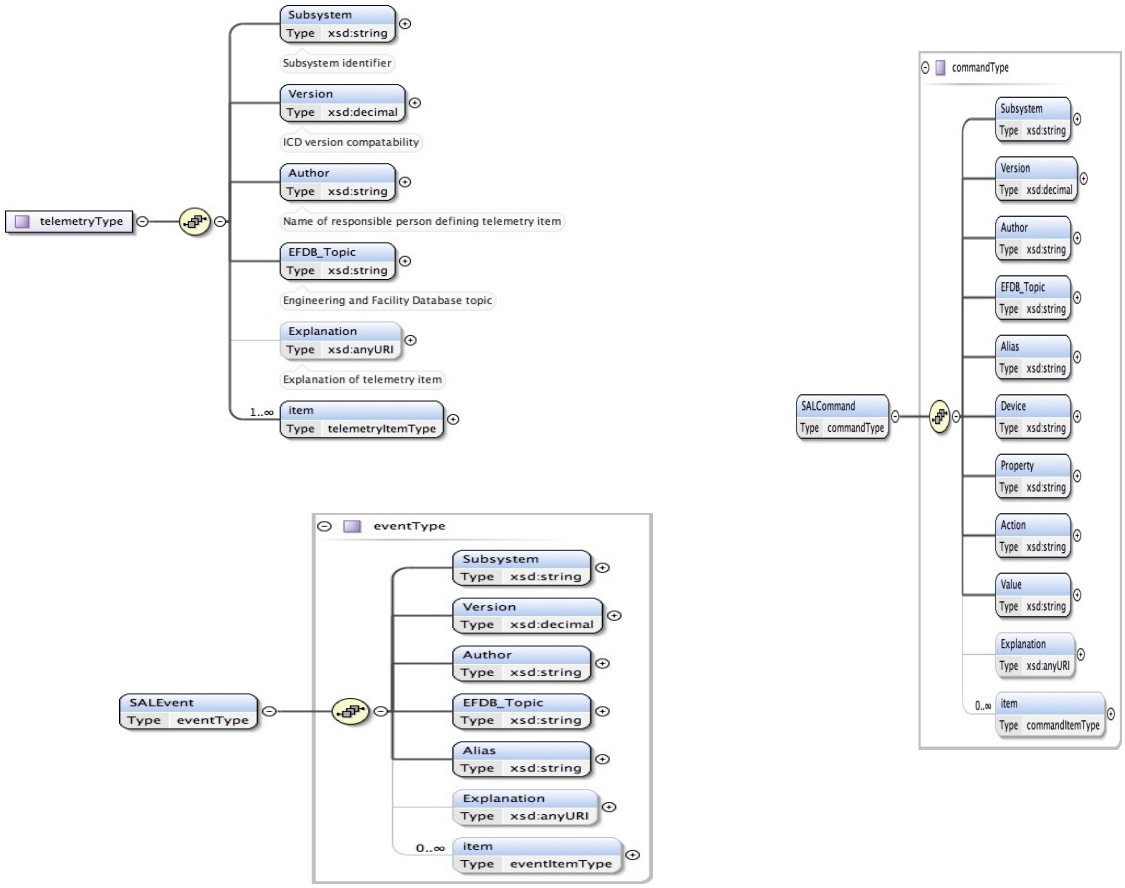

Figure 2 – SAL XML schema

Extensive prototyping has been done to calculate the total network bandwidth required to support operations, and the EFD clusters are sized to store complete records of the entire 10 year survey (currently estimated at 400TB). There are two identical live EFD's , one at the summit, and another at the base. The EFD is also replicated to the archive center.

The Service Abstraction Layer (SAL) is implemented using open source packages that implement the Object Management Group (OMG) open standards of IDL (Interface Definition Language), DDS (Data Distribution Service) for data communication, and SQL (Standard Query Language) for database access.

The SAL Software Development Kit (SDK) provides tools to automatically generate communications libraries based on the data-types defined using the SAL XML schema. Supported languages include C/C++, Java, Python, and LabVIEW. The current development platform is CentOS Linux x86_64 7.0. (a variety of other platforms can be supported if necessary, including Darwin/BSD, Vxworks, Windows). The code base is maintained in a local Git repository and mirrored to Github[b] for external access.

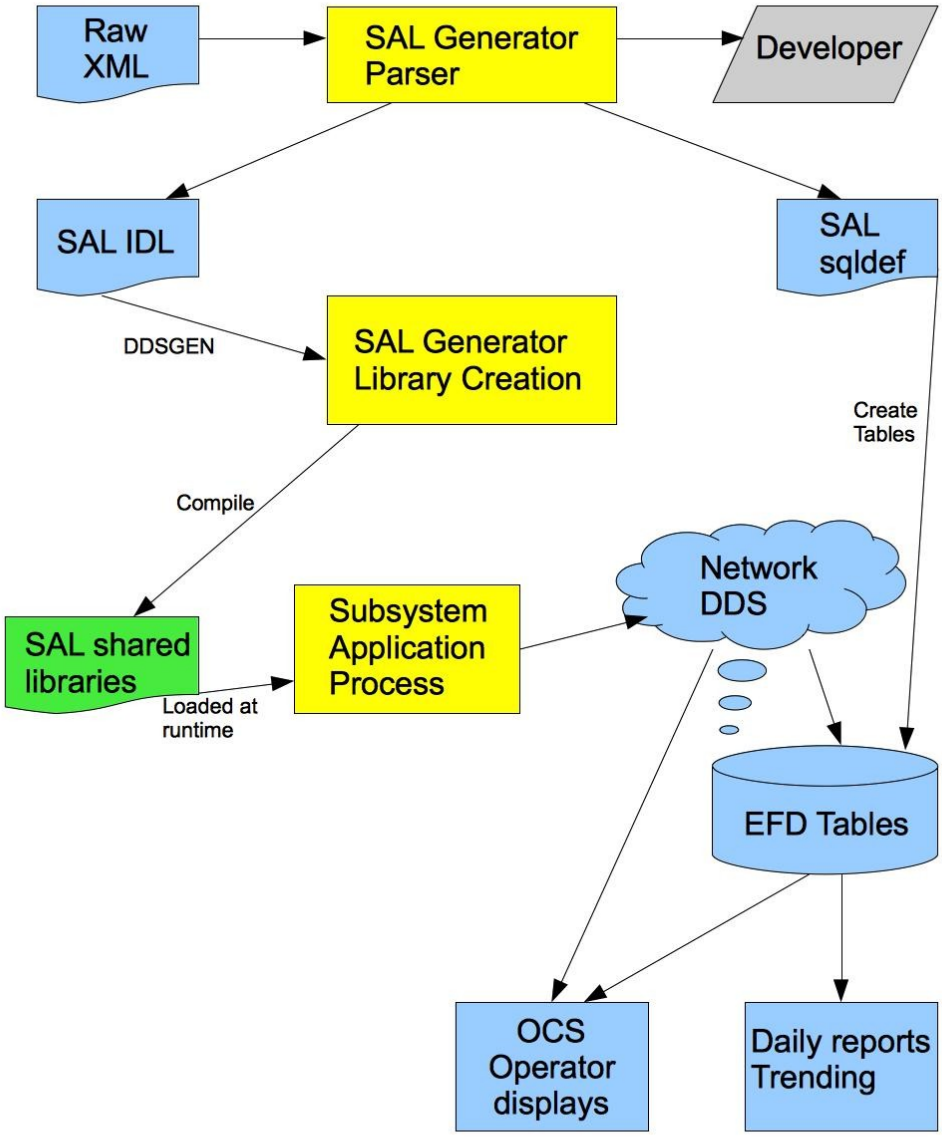

Figure 3 – SAL development workflow

b. http://github.com/lsst-ts/ts_sal

The per-subsystem XML definitions are also maintained in a Github repository. A Jenkins Continuous Integration server is used to monitor the development. Google Test is used to automate higher level integration and performance tests. Jenkins is also cross-platform as it is Java based.SAL interactions during tests are logged to the EFD for future reference

Figure 3 illustrates the flow of processing involved. We start with the input XML, which is parsed and syntax checked. The corresponding IDL is generated and some housekeeping meta-data is also added at this stage (SAL IDL). The type support library is then generated for the appropriate target language (the example uses C++), along with EFD table definitions. The application developer then writes code to the SAL API, and links it with the SAL generated shared library at run-time

In the case of Java, the SDK generates the type support and SAL Application Programming Interface (API) Jar files, and then bundles them into a Maven project.

For Python (2.7, 3.4) we wrap the C++ API using Boost::Python and generate a Python importable shared library (Simple Wrapper and Interface Generator (SWIG) wrappers are also available).

The process for LabVIEW is a little more interactive as the VI's must be generated from within the LabVIEW environment using the “Import Shared Library” wizard. An additional LabVIEW tool was also written to create “Cluster” types based on the SAL IDL data definitions.

There is also a web based interface to the code generator which can be used by non-developers to easily modify the basic types.

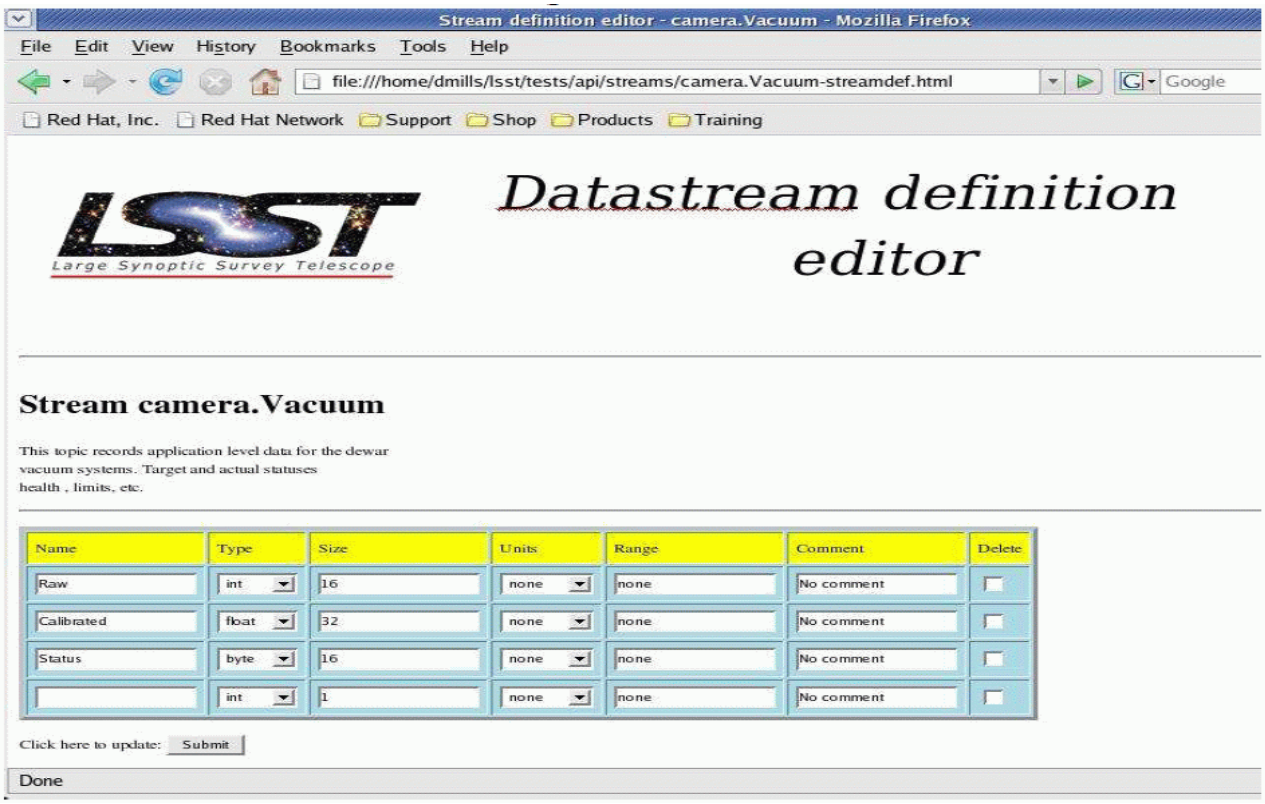

Figure 4 – SAL Datastream Definition Editor

3. Consistency checking and version control

Each transaction type which is defined has embedded in it's low level structure a UUID generated from the description. At run-time all transactions are consistency checked to ensure that the same definition is being used by both sender and receiver. Transactions are also automatically time-tagged so that any clock-skew between subsystems can be immediately identified. The system uses the Precision Time Protocol (PTP[8]) to maintain accurate synchronized system clocks throughout the observatory. The accuracy requirement is at the 1msec level, easily satisfied by PTP.

Identifier	**Description**
private_revCode	Crc of IDL source
private_sndStamp	System time of sender
private_rcvStamp	System time of receiver
private_SeqNum	Sequence number (process)
private_origin	IP addr and PID

Table 1 – Consistency check metadata

4. Transport Layer

The middle-ware transport layer is an implementation of an Open Standard, the DDS. The open-source OpenSplice package is being used, but other compliant DDS implementations could also be substituted (eg RTI NDDS).

DDS does not require a central server and implements a publish/subscribe architecture. Multiple actors may be present on multiple nodes. For the case of multiple actors within a single node, shared memory is used as the low level transport.

The interaction between DDS actors on multiple nodes in a distributed system is mediated by a discovery phase. New nodes can be added to the system at any time and DDS automatically manages their "discovery" and integration.

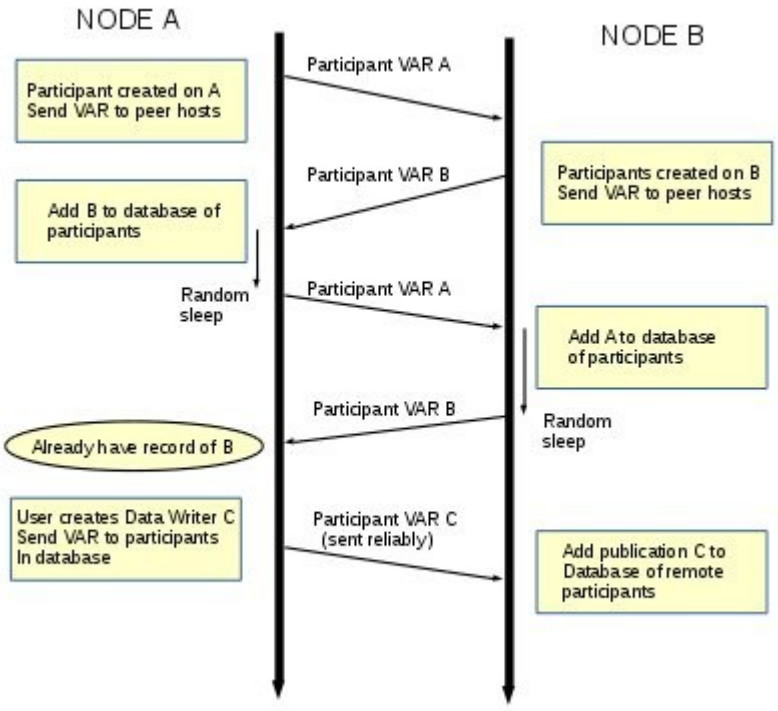

Figure 6 – DDS Discovery phase

5. Quality of Service (QOS)

As different subsystems publish data at different cadences, and there are differing priorities for different kinds of transactions , we use the DDS QOS mechanisms to tune the overall system behavior

QoS Policy	Description
DEADLINE A duration, 'deadline_period'	Indicates that a data reader expects a new sample updating the value of each instance at least once every deadline_period. Indicates that a data writer commits to writing a new value for each instance managed by the data writer at least once every deadline_period.
TIME BASED FILTER A duration, 'minimum_spearation'	Filter that allows a data reader to specify that it does not want to receive more than one value each 'mnimum_seperation' period, regardless of how fast changes occur.
CONTENT BASED FILTER A string, 'expression' and a sequence of strings, 'parameters'	Filter that allows a data reader to filter the data received based on the contents of the data itself. Syntax of 'expression' is like an SQL WHERE

	clause. Only the 'parameter' part may be changed.
HISTORY A kind (KEEP_LAST or KEEP_ALL) and an integer 'depth'	Specifies the behavior of the middleware in the case where the value of a data object changes (one of more times) before it can be successfully communicated to one or more existing subscribers. Controls whether the middleware should attempt to keep in its history only the most recent set of values (KEEP_LAST) or all values (KEEP_ALL)
RELIABILITY A kind (RELIABLE or BEST_EFFORT)	Specifies whether the middleware should use a reliable protocol to ensure each data sample in the publishers history is received by all subscribers.

` Table 2 – *DDS QoS policies*

6.Summary

The LSST SAL provides a mechanism to generate run-time support in a variety of languages. The only input required is a set of Interface definitions written using a simple XML schema. The generated code provides consistency and coherence all the way from application code (developed by a diverse set of globally distributed contractors), to the recorded history of all inter-subsystem interactions (Commands, Events, and Telemetry), in the EFD. This functionality is essential for the optimal analysis of the survey data. It enables the necessary corrections for environmental and procedural effects. The EFD will also be continuously monitored to detect early signs of hardware problems and ensure that all subsystems are operating within design specifications.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation through Cooperative Agreement 1258333 managed by the Association of Universities for Research in Astronomy (AURA), and the Department of Energy under Contract No. DE-AC02-76SF00515 with the SLAC National Accelerator Laboratory. Additional LSST funding comes from private donations, grants to universities, and in-kind support from LSSTC institutional members.

REFERENCES

[1] Object management Group, “*DDS Specification*”, http://www.omg/org/spec/DDS/1.4 (2015)

[2] Kahn, S., “*Final Design of the Large Synoptic Survey Telescope*,” in [Ground-based and Airborne

Telescopes VI], Hall, H. J., Gilmozzi, R., and Marshall, H. K., eds., Proc. SPIE 9906, in press (2016).

[3] Jurić M. et al , “The LSST Data Management System”, Proceedings of ADASS XXV, in press (2016)

[4] Thiebaut el al, "*Real-Time Publish Subscribe (RTPS) Wire Protocol Specification*", Internet Draft Document, Internet Engineering Task Force RFC 2326 (2002)

[5] Claver C et al, “Systems engineering in the Large Synoptic Survey Telescope Porject : an application of model based systems engineering”, Proceedings of the SPIE, Volume 9150, id. 91500M 13 pp. (2014)

[6] Object Management Group , “*The Real-time Publish-Subscribe Wire Protocol DDS™ Interoperability Wire Protocol (DDSI-RTPS™)*”, http://www.omg.org/spec/DDSI-RTPS (2014)

[7] Schumacher, G. and Delgado, F., “*The Large Synoptic Survey Telescope OCS and TCS models*,” in modeling, Systems Engineering and Project Management for Astronomy IV], Angeli, G. Z. and Dierickx, P., eds., Proc. SPIE 7738, 77381E (2010).

[8] IEEE 1588-2008 , “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems”, 2008.

[9] Lotz, P. J. et al., “*LSST control software component design*,” in [Software and Cyberinfrastructure for Astronomy], Chiozzi, G. and Guzman, J. C., eds., Proc. SPIE 9913, in press (2016).