Simulating the LSST OCS for Conducting Survey Simulations Using the LSST Scheduler

Michael A. Reuter[a], Kem H. Cook[b], Francisco Delgado[a], Catherine E. Petry[a], and Stephen T. Ridgway[c]

[a]LSST, 950 N Cherry Ave, Tucson, AZ USA
[b]Cook Astronomical Consulting, San Ramon, CA USA
[c]National Optical Astronomy Observatory, 950 N Cherry Ave, Tucson, AZ USA

ABSTRACT

The Operations Simulator was used to prototype the Large Synoptic Survey Telescope (LSST) Scheduler. Currently, the Scheduler is being developed separately to interface with the LSST Observatory Control System (OCS). A new Simulator is under concurrent development to adjust to this new architecture. This requires a package simulating enough of the OCS to allow execution of realistic schedules. This new package is called the Simulated OCS (SOCS). In this paper we detail the SOCS construction plan, package structure, LSST communication middleware platform use, provide some interesting use cases that the separated architecture allows and the software engineering practices used in development.

Keywords: LSST, simulations, observing strategy

1. INTRODUCTION

Both the research and development phase of the Large Synoptic Survey Telescope (LSST)[1] project prior to the construction award in August 2014 and the subsequent years afterwards saw the successful development and testing of the Operations Simulator (OpSim).[2–12] OpSim is a software application for simulating surveys for LSST. It contains an environment simulation using data from the actual LSST site, a fully detailed kinematic telescope and camera model, configuration parameters for controlling the science driven requirements, and the prototype version of the LSST Scheduler. OpSim has been tuned to produce the current LSST baseline survey and some alternate survey strategies.[13]

While OpSim has been successful at producing survey strategy scenarios, the Scheduler is entwined within the code base. When LSST begins operations, it will require the Scheduler to perform the duties of determining the next targets to observe while interfacing with LSST's Observatory Control System (OCS).[14] The Scheduler will use the LSST communication middleware[15] project which is a layer upon the Data Distribution Service (DDS) architecture. DDS is a publish/subscribe system in common use for communication between components of complex systems.

With the requirements stated above, the project decided to separate the Scheduler[16] into its own code base so that an effective deliverable can be provided. In order to continue running simulations with the refactored Scheduler, a new harness is being created that utilizes the same DDS infrastructure to communicate with the Scheduler. This new project is called the Simulated Observatory Control System (SOCS). SOCS is not a high fidelity simulation of the entire OCS, but just enough of it to get the Scheduler to complete an entire survey. The combination of SOCS and the Scheduler is still collectively called the Operations Simulator. This new pairing will be referred to as OpSim version 4 (OpSim4) whereas the older Operations Simulator is now referred to as OpSim version 3 (OpSim3). The schematic representations for both versions are shown in Figure 1.

Figure 2 is an exact replica of a diagram showing the interaction of the Scheduler and the OCS with SOCS simulating OCS functions. The job of SOCS is to provide all of the needed inputs to the Scheduler shown in

Further author information: (Send correspondence to M.A.R.)
M.A.R.: E-mail: mareuter@lsst.org

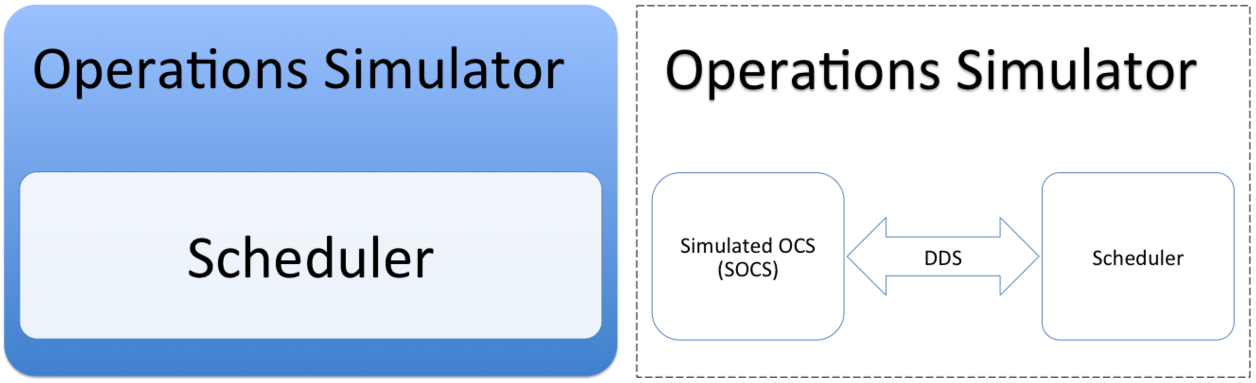

Figure 1. The left figure schematically shows the operational bounding box for OpSim version 3. This clearly indicates the buried nature of the Scheduler within the confines of OpSim. The right figure schematically shows the operational bounding box for OpSim version 4. This shows the newly separated nature of the two components which allows for development and usage flexibility.

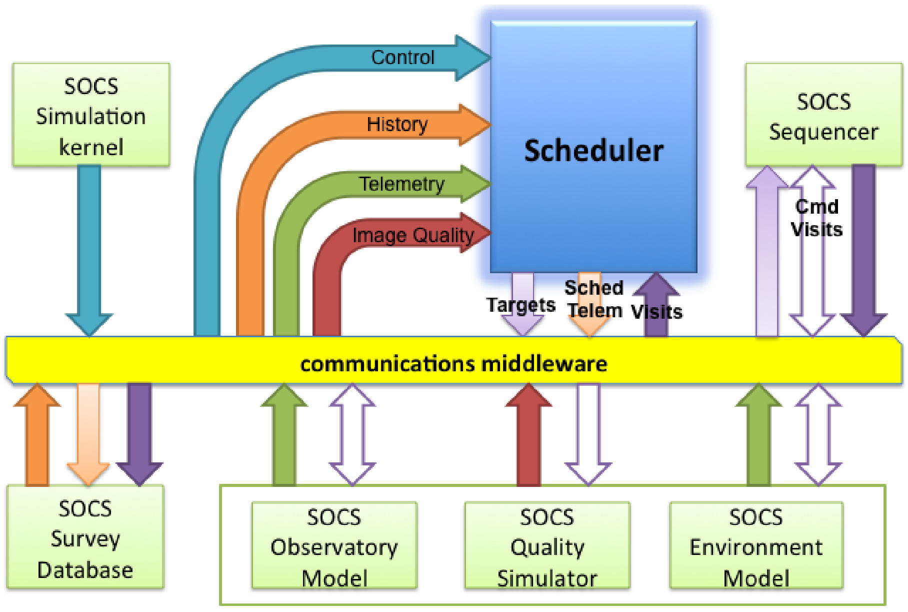

Figure 2. This diagram shows the DDS communication pathways between the SOCS and Scheduler systems.

this diagram. Central to this is the communications middleware via which information flows between SOCS and the Scheduler. Key to the design of SOCS is the ability to simulate a survey either deterministically or with stochastic variation in a variety of inputs. This is needed to understand the behavior of simulated surveys when the Scheduler is not provided completely accurate environmental data due to rapid variation in the environment, and also when the Scheduler's observatory model is not a completely accurate model of the real observatory.

2. CONSTRUCTION

Since SOCS and the Scheduler have interdependent functionality, the developed construction plans are tightly coupled. The schedules for both products are aligned so that software releases appear simultaneously. The first goal in the construction process is to create a version that replaces all of the functionality offered by OpSim3 with a few improvements added. The construction will proceed with adding more features to the system until the Scheduler is delivered to the project completing a major milestone. There will be continued development after the delivery to implement more features of the system that have been requested.

The following description provides the proposed release versions and the functionality required from each version. The version 1.0 release represents the OpSim3 replacement milestone and is slated for the fall of 2016. The version 1.5 release represents the milestone of the Scheduler delivery to the project.

v0.2 SOCS interface with DDS, Implement basic configuration system, Implement basic survey database system, Simulation kernel and time handling, sequencer with target handling, Inherit from Scheduler observatory model, Interface with Scheduler exchanging fixed list of targets

v0.3 Implement slewing and visit behavior for observatory model, Implement observing behavior in sequencer, Integrate storage of target and observation information in survey database, Track slew information in survey database, Implement configuration for area distribution proposals

v1.0 Implement configuration for time-dependent proposals, Configure Scheduler via DDS topics, Implement environmental model, Incorporate weather and seeing data, Implement downtime information, Implement filter swaps during new moon

v1.1 Implement configuration and support development of look-ahead for area distribution proposals, Implement non-deterministic downtime, Implement non-deterministic weather

v1.2 Implement configuration and support development of look-ahead for time-dependent proposals

v1.3 Evaluate and implement performance enhancements

v1.4 Implement warm start process for Scheduler, Implement image quality feedback simulation, Implement degraded operations mode simulation

v1.5 Implement and support dithering in Scheduler

v2.0 Implement publication of future targets from Scheduler

v2.1 Implement weather forecast data

v2.2 Implement configuration for alternate Scheduler optimization algorithms

3. DESIGN

The design of SOCS inherits from lessons learned in building OpSim3. Among those is a desire for a more modular approach to the software with some parts of the Application Programming Interface (API) available for use outside the main steering program. Figure 3 shows the top-level component diagram. It contains both module definitions as well as information flow between components and between SOCS and the Scheduler.

3.1 Modules

The code is divided into modules with each one characterized by a particular set of behaviors and responsibilities for the system. Descriptions of the modules are given in the following sections.

3.1.1 Simulation Kernel

The Simulation Kernel is responsible for the main control flow of the simulator. There is a Simulator class which handles the orchestration of the different steps within the survey simulation. The TimeHandler class is clock for running the survey simulation. The timestamps provided by this class are updated faster than real time since it does not have to wait for any subsystem to return information. However, the timestamps run from the beginning of the survey until its finish. Finally, there is the Sequencer which mimics some of the behavior of its OCS counterpart. This class is responsible for performing the observation cycle on the requested target that was given by the Scheduler.

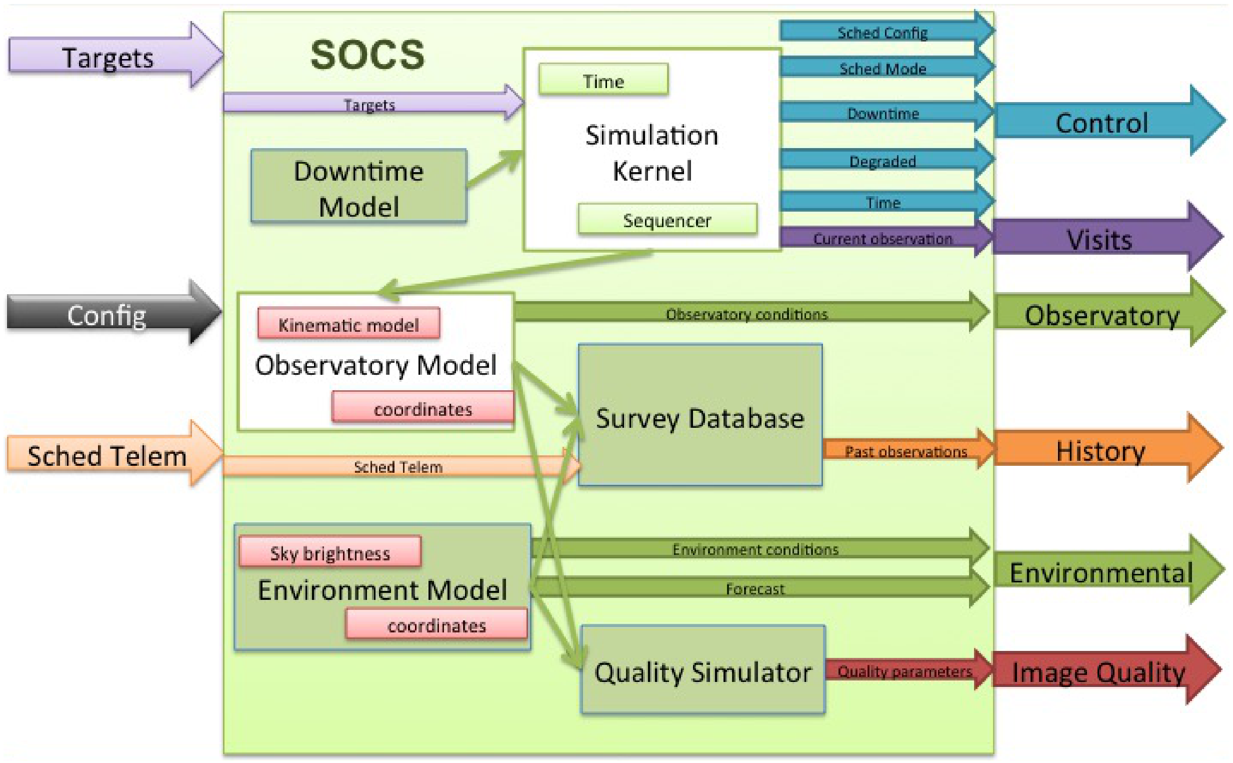

Figure 3. This diagram shows the block level components for the SOCS design as well as the information flow both internal and external to SOCS.

3.1.2 Observatory Model

The Observatory Model contains the representation of the "real" LSST observatory. It uses as an aggregate of the Observatory Model from the Scheduler. The Scheduler's model will always be an ideal version of the real observatory. However, the SOCS Observatory Model is constructed for allowing perturbations of the various model parameters. This allows for testing of engineering scenarios of degraded observatory performance. This module also houses container classes used for gathering information about the observatory for inclusion in the survey database.

3.1.3 Survey Database

This module handles the creation and interactions for the generated survey database. It contains functions that specify the information content of the different tables and then constructs the actual instances of those tables. There are functions that also aid in the ability to map data from DDS topics and other SOCS information structures onto the appropriate columns for a given table. The SocsDB class is the main orchestrator of survey database interactions. It is responsible for creating the database, gathering the data for output and actually performing the database writes. The system uses SQLAlchemy[17] to handle the different database backends and is currently configured to output to a MariaDB[18] or a SQLite[19] database.

3.1.4 Environmental Model

The Environmental Model will contain classes that handle digesting and distributing environment information, such as weather and astronomical sky conditions, to the Scheduler. This module is due to reach development status about the time this paper is presented at the conference, so specific details are not available. Weather and seeing data from the LSST site will be stored in a digested format. The code will also allow overriding of that data so long as it's in a similar format. The module classes will be responsible for consuming the digested data and converting it into topics that are then passed to the Scheduler.

3.1.5 Quality Simulator

While this module has not reached the development stage of the construction plan, it also has a known set of functionality. It will contain classes that mimic some the information from the Data Management Level 1 products.[20] The main focus is the image feedback on quantities like seeing, transparency and other quality metrics. This system will be engineered to provide a mechanism for occasionally making a recently completed visit violate image quality constraints. This information will be fed back to the Scheduler and the winning proposals for that visit will need to determine how the quality factors effect their completeness requirements.

3.1.6 SAL

While this module is not on the diagram, it is designed to help handle the interaction between other classes and the LSST middleware's Software Abstraction Layer (SAL). The main class, called SalManager, allows for easy creation and setting of publish and subscribe topics that are the backbone of the communication between SOCS and the Scheduler. This class also provides an easy mechanism for performing the actual publishing of topics and receiving the subscribed topics.

3.1.7 Configuration

This is another module not on the diagram, but one that keeps the baseline LSST survey configuration. It uses a configuration system that is available from the LSST Data Management software stack.[21] The classes contain the appropriate configuration parameters for the baseline survey and the configuration system provides a mechanism to override those values without changing the code. There are also classes which interface the configuration parameters to the configuration DDS topics that need to be passed to the Scheduler.

3.2 Information Flow

We will now describe the content of the external information flow for SOCS. Each arrow represents one or more DDS topics. For reference the Observatory and Environmental arrows in Figure 3 are bundled together as the Telemetry arrow from Figure 2.

3.2.1 Control

This arrow consists of many different topics. The main one is the exchange of a timestamp during the simulation. This timestamp represents the calendar time prior to the upcoming visit. Multiple topics are used to handle configuration of the various Scheduler components. Other topics include downtime notifications and possible degraded modes for the observatory

3.2.2 Visits

This arrow consists of an observation topic from the visit of the field requested via the Scheduler target. This topic contains information like the time the visit started, actual number and duration of exposures, total time of visit, actual field position observed (RA, Dec), filter used in observation, sky brightness, clouds (or transparency), field separation from moon, etc.

3.2.3 Environment

This contains multiple topics for exchange of environmental conditions. Clouds (or transparency), seeing, weather data (temperature, pressure, wind speed and direction, etc.) and weather forecasting data are provided. Topics like clouds and seeing are provided as a grid of information for the whole sky. The grid resolution is determined by the Scheduler requirements.

3.2.4 Observatory

This arrow contains a single topic that is exchanged prior to the upcoming visit. It contains the current pointing location, tracking state, telescope, dome and rotator positions and the camera filter state information.

3.2.5 Image Quality

This arrow contains a topic with image quality parameters from a recent visit. It will contain an overall image quality factor and calculated seeing and transparency for the visit field.

3.2.6 History

This arrow's main use is for setting up a new instance of the Scheduler based on the current running Scheduler. It contains a stream of observation topic messages that are read from a survey database. The contents of the topic messages are the same as the one from the Visit arrow.

3.2.7 Config

This arrow represents any configuration coming from the Scheduler that outside systems, like SOCS or the OCS, might require. Currently, this is limited to the configuration of the LSST observation fields. For SOCS, the LSST field information is used to link against the target field request from the Scheduler.

3.2.8 Targets

This represents the stream of targets requested by the Scheduler after considering all of the environmental, observatory and science requirements. The target topic contains information like the requested field, field position (RA, Dec), filter, number and duration of exposures.

3.2.9 Sched Telem

This arrow represents a topic containing all of the relevant information the Scheduler used to make its decision about which target to request for observation. It consists of information like sky brightness, clouds, transparency, seeing, target distance from moon, rank from science drivers, interested science drivers, etc.

4. SOFTWARE ENGINEERING

OpSim3 had a document detailing all of the functional and performance requirements for the developed system which included those for the Scheduler. The refactoring of OpSim3 into the SOCS and Scheduler components necessitated a reorganization of the requirements document into one for each resulting system. The SOCS requirements document was adjusted to account for the new role as the harness for driving the Scheduler and placed under LSST project level change control.

A development plan for SOCS was created containing incremental releases, with a specific set of capabilities and validation activities for each one. This plan is captured in JIRA[22] release epics and described at a high level in Primavera[23] which is a component of LSST's Project Management Controls System (PMCS).[24] The plan is coordinated with the release timeline of the Scheduler due to their heavy interdependence. The detailed construction plan is also described in JIRA epics relating to each release epic, keeping track of individual tasks that mark the way to each release. Tracking the development process uses the LSST variant of Agile software development.[24] The JIRA plan, releases, milestones and tasks are used periodically to report back to PMCS which computes earned value[24] for the project and to organize the work of the development team.

The SOCS code is written in Python[25] and is kept in a GitHub[26] git repository. It follows the coding standards[27] from LSST Data Management and LSST Systems Engineering Simulations about templates, interfaces, coding and unit tests. The SOCS code follows the basic tenants of the Test Driven Design[28,29] (TDD) philosophy of software development. TDD requires writing tests before implementing code, helping to focus the design of the system. Also, unit tests are run during the implementation process at a fairly high frequency to uncover issues quickly. The baseline science survey configuration is also kept within the SOCS GitHub repository.

More importantly, each SOCS release is integrated and tested with the LSST Scheduler and the combined efforts from both the Telescope and Site and Systems Engineering Simulations teams. The capabilities of SOCS to drive the Scheduler for implementing the LSST survey are validated with specialized tools, such as the Metrics Analysis Framework[30] and the participation of the science collaborations.

5. USE CASES

The separation of the Scheduler from the simulation harness allows for efficient development. This means only one code base has to be maintained to serve the separate input systems (OCS and SOCS). This separated approach and the use of a standard communication framework allow for some interesting use cases with respect to the SOCS/Scheduler combination. The main use case, the ability to run a full LSST survey, is not influenced by the system separation. The other cases are predicated off the idea that the Scheduler configuration running the LSST during operations can be injected into a new instance of the Scheduler. The new instance can be driven by SOCS to perform different scenarios while leaving the operating Scheduler unaffected. One scenario is advancing the Scheduler through a given time window in order to publish a list of targets that the LSST will visit within the caveats of environmental and instrumental conditions. Another scenario is to take the current Scheduler state and fast forward through the remaining survey to evaluate the efficiency based off the current progress. This mechanism can also be used to evaluate alternate scenarios for the survey, such as new proposals, proposals being finished or alternate configuration parameters.

6. SUMMARY

In this paper, we have shown the necessity for refactoring OpSim3 into the LSST Scheduler and SOCS which is known collectively as OpSim4. We presented the construction plan and architecture design for SOCS including information flow. We provided evidence of the software engineering practices applied during the code development of SOCS. Lastly, we presented some use cases that demonstrated the unique capabilities of this separated system.

ACKNOWLEDGMENTS

Financial support for LSST comes from the National Science Foundation (NSF) through Cooperative Agreement No. 1258333, the Department of Energy (DOE) Office of Science under Contract No. DE-AC02-76SF00515, and private funding raised by the LSST Corporation. The NSF-funded LSST Project Office for construction was established as an operating center under management of the Association of Universities for Research in Astronomy (AURA). The DOE-funded effort to build the LSST camera is managed by the SLAC National Accelerator Laboratory (SLAC).

REFERENCES

[1] Kahn, S., "Final design of the Large Synoptic Survey Telescope," in [*Ground-Based and Airborne Telescopes IV*], Hall, H. J., Gilmozzi, R., and Marshall, H. K., eds., *Proc. SPIE* **9906**, in press (2016).

[2] Delgado, F. and Schumacher, G., "The LSST OCS scheduler design," in [*Observatory Operations: Strategies, Processes, and Systems V*], Peck, A. B., Benn, C. R., and Seaman, R. L., eds., *Proc. SPIE* **9149**, 91490G (2014).

[3] Delgado, F. et al., "The LSST operations simulator," in [*Modeling, Systems Engineering, and Project Management for Astronomy VI*], Angeli, G. Z. and Dierickx, P., eds., *Proc. SPIE* **9150**, 915015 (2014).

[4] Saha, A. et al., "Advancing the LSST Operations Simulator," in [*American Astronomical Society Meeting Abstracts #221*], *American Astronomical Society Meeting Abstracts* **221**, 247.03 (Jan. 2013).

[5] Ridgway, S. et al., "Simulation of autonomous observing with a ground-based telescope: the LSST experience," in [*Observatory Operations: Strategies, Processes, and Systems III*], Silva, D. R., Peck, A. B., and Soifer, B. T., eds., *Proc. SPIE* **7737**, 77370Z (July 2010).

[6] Krabbendam, V. et al., "LSST Operations Simulator," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **42**, 217 (Jan. 2010).

[7] Cook, K. H. et al., "LSST: Cadence Design and Simulation," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **41**, 367 (Jan. 2009).

[8] Pinto, P. A. et al., "LSST: Cadence Design and Simulation," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **39**, 979 (Dec. 2007).

[9] Delgado, F. et al., "LSST operation simulator implementation," in [*Observatory Operations: Strategies, Processes, and Systems*], Silva, D. R. and Doxsey, R. E., eds., *Proc. SPIE* **6270**, 62701D (June 2006).

[10] Pinto, P. A. et al., "LSST Survey Strategy: Cadence Design and Simulation," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **38**, 1017 (Dec. 2006).

[11] Cook, K. H. et al., "LSST Operations Simulator," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **37**, 1206 (Dec. 2005).

[12] Cook, K. H. et al., "LSST Operational Cadence Simulation and Design," in [*American Astronomical Society Meeting Abstracts*], *Bulletin of the American Astronomical Society* **36**, 1528 (Dec. 2004).

[13] Cook, K. H. et al., "Exploring possible Large Synoptic Survey Telescope (LSST) surveys with the LSST operations simulator and delivering a reference simulated survey," in [*Modeling, Systems Engineering, and Project Management for Astronomy VI*], Angeli, G. Z. and Dierickx, P., eds., *Proc. SPIE* **9911**, in press (2016).

[14] Daly, P. and Schumacher, G., "LSST OCS status and plans," in [*Software and Cyberinfrastructure for Astronomy IV*], Chiozzi, G. and Guzman, J. C., eds., *Proc. SPIE* **9913**, in press (2016).

[15] Mills, D. et al., "LSST communications middleware implementation," in [*Ground-Based and Airborne Telescopes IV*], Hall, H. J., Gilmozzi, R., and Marshall, H. K., eds., *Proc. SPIE* **9906**, in press (2016).

[16] Delgado, F., "The LSST Scheduler from design to construction," in [*Observatory Operations: Strategies, Processes, and Systems V*], Peck, A. B., Benn, C. R., and Seaman, R. L., eds., *Proc. SPIE* **9910**, in press (2016).

[17] Bayer, M., "SQLAlchemy," (2016). https://www.sqlalchemy.org.

[18] "MariaDB," (2016). https://www.mariadb.org.

[19] "SQLite," (2016). https://www.sqlite.org.

[20] Jurić, M. et al., "LSST Data Products Definition Document." http://ls.st/LSE-163 (2013).

[21] Jurić, M. et al., "The LSST Data Management System," in [*Astronomical Data Analysis Software & Systems XXV*], Lorente, N. P. F. and Shortridge, K., eds., *ASP Conf. Ser.* **in press**, arXiv:1512.07914, ASP, San Francisco (2016).

[22] "JIRA Software," (2016). https://www.atlassian.com/software/jira.

[23] "Primavera," (2016). https://www.oracle.com/applications/primavera/products/project-management.htmlnote.

[24] Kantor, J. et al., "Agile software development in an earned value world: a survival guide," in [*Modeling, Systems Engineering, and Project Management for Astronomy VI*], Angeli, G. Z. and Dierickx, P., eds., *Proc. SPIE* **9911**, in press (2016).

[25] "Python," (2016). https://www.python.org.

[26] "GitHub," (2016). https://www.github.com.

[27] "DM Python Style Guide." http://developer.lsst.io/en/latest/coding/python_style_guide.html (2016).

[28] Beck, K., [*Test Driven Development: By Example*], Addison-Wesley Professional (2002).

[29] Astels, D., [*Test-Driven Development: A Practical Guide*], Prentice Hall (2003).

[30] Jones, R. L. et al., "The LSST metrics analysis framework (MAF)," in [*Observatory Operations: Strategies, Processes, and Systems V*], Peck, A. B., Benn, C. R., and Seaman, R. L., eds., *Proc. SPIE* **9149**, 91490B (2014).