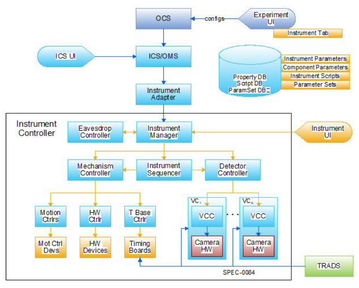
SOFTWARE QUALITY ASSURANCE FOR THE DANIEL K. INOUYE SOLAR TELESCOPE CONTROL SOFTWARE

A. Greer, A. Yoshimura, Observatory Sciences Ltd
B. Goodrich, S. Guzzo, C. Mayer, DKIST

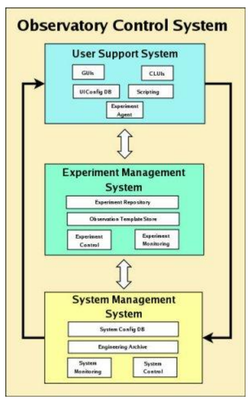
Introduction To DKIST Software

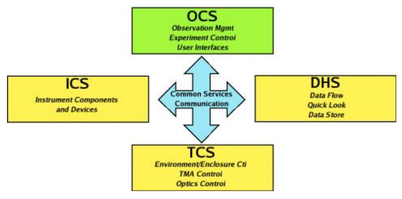
During construction DKIST elected to use a Common Services model as a basis for the standard distributed software infrastructure used to build the control subsystems. The Common Services Framework (CSF) was developed as a result of this decision, providing a standard framework supported in three programming languages (Java, C++ and Python).

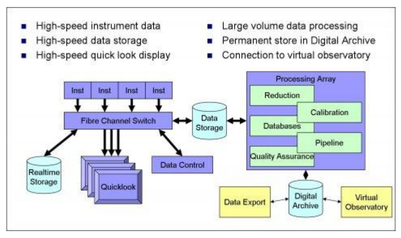


Services offered by the CSF Framework:

- Deployment Support
- Communications Support
- Persistence Support
- Application Support
- Utility Libraries

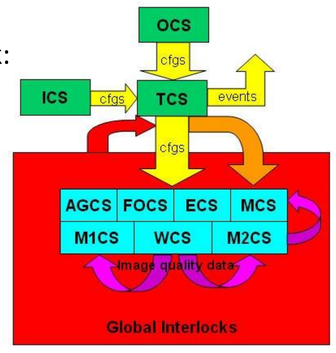






Systems built on top of the CSF:

- Observatory Control System (OCS)
- Instrument Control System (ICS)
- Data Handling System (DHS)
- Telescope Control System (TCS)
- TCS Subsystems



Ensuring the quality of such a large software project requires dedicated resource, which DKIST have committed to providing by running a Quality Assurance (QA) program, re-using existing and developing additional tools as required.

Software Quality Assurance

The Problem:

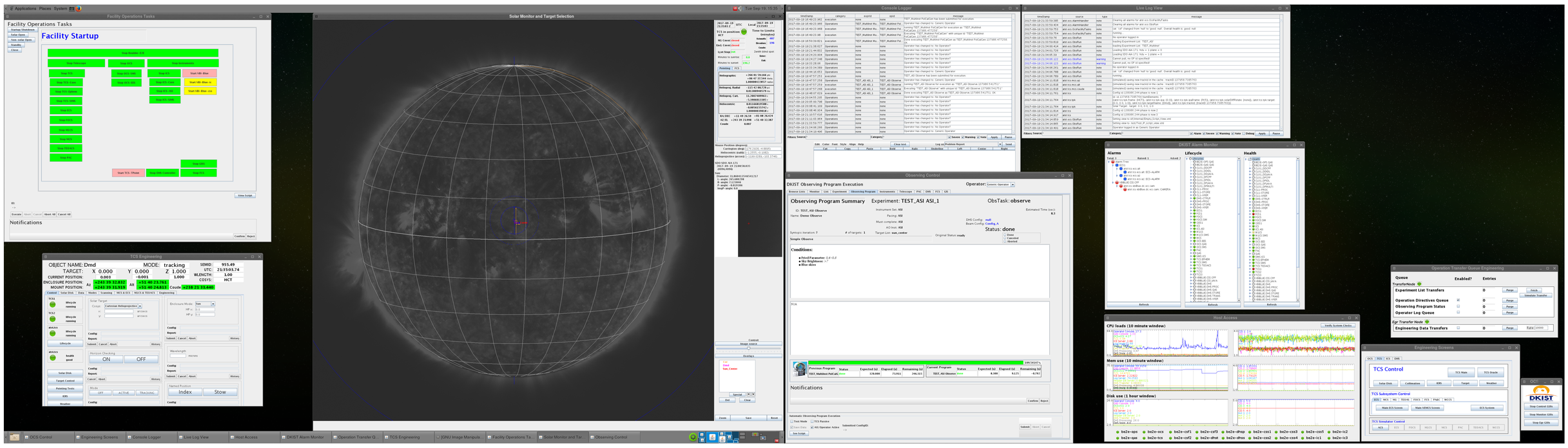
- DKIST Control Software supported in three languages.
- Middleware independent infrastructure developed in-house.
- Many TCS subsystems developed by external commercial companies.
- Any change to in-house software may impact delivered subsystems.
- Test matrix is large.

The Requirements:

- Consistent test reports for each language.
- Unit test framework for each framework.
- Use existing tools and frameworks where possible.
- Use existing virtual machine software approved by DKIST.
- Provide a simple method for installing software infrastructure, systems and subsystems.
- Provide the ability to group test types and languages.
- Continuous execution of tests.
- Notification of test results.

The Solution:

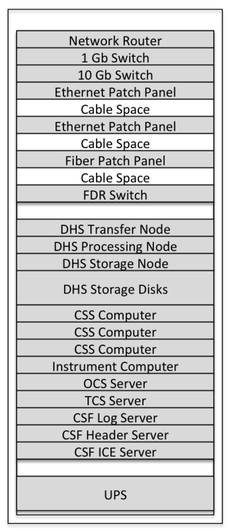
Existing test frameworks were used for C++ and Java (CxxTest and Junit). The framework test runners were extended to provide a consistent report style for all tests. An XML layout was selected for the test reports to make the reports human readable but also easily digested into larger reports. For python no unit testing was required, and so test classes were created to assist with system tests and to produce the same report format. For installation of the software and management of test execution and reporting the Testing Automation Framework (TAF) was developed. The TAF reads a configuration file, installs all necessary software, executes tests or groups of tests and supervises reporting of tests. VMWare is used for execution of isolated test suites within virtual environments as this product was already used at DKIST. An API provides full control of the virtual machine from the host, which allows automated testing and monitoring of the tests to take place. Detailed reports are compiled from the reports produced by individual virtual machines.

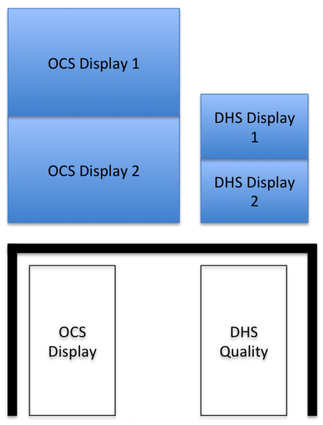


Screen shot taken from the E2E simulator

Development And Test Setup

The DKIST project offices in Tucson, Arizona and Boulder, Colorado are equipped with the necessary control hardware to be capable of running the entire control software stack. This includes the Data Handling System (DHS) servers, the Camera System Software (CSS) servers, the CSF, TCS and subsystem servers, as well as UPS and an additional network hub. The diagram to the right shows the basic layout. Below is a photo of the rack.





This hardware installation is called the End To End (E2E) simulator. It is possible to either run the whole DKIST software (in simulation) natively on the hardware or virtual machines can be spawned to execute subsystems in isolation. For the QA a utility known as the E2E Test Executor (ETE) was developed. This utility coordinates execution of virtual machines and manages TAF instances to execute tests, compile reports and notify people of the results.

Enhancements

Benchmark Tests:

Recently the QA software was updated to allow a test to be marked as a benchmark test. Once marked in this way the TAF will execute the test a number of times, collecting the results. Instead of the results being treated as separate they are combined and benchmark statistics are calculated.

Enhancements Awaiting Implementation:

- Ignoring specific (and understood) failures.
- Disk space monitoring.
- Permanent storage of benchmarks to raise warnings for significant changes.
- Standard test templates constructed to simplify creation of similar tests.

Conclusion

DKIST have fully invested in the software QA programme for project control software. As a result of this investment, a QA infrastructure tailored to the needs of the DKIST project has been developed, leveraging on the power and flexibility of existing virtualisation and testing tools. Many hundreds of tests are automatically executed on DKIST servers throughout every day, with key project members receiving detailed email reports of the results. DKIST control software developers can commit new features with the confidence that all existing requirements will be re-verified and that any failures will be reported.