NGAO Software Architecture
Review Committee Report

A. Conrad (chair), R. Dekany, K. Tsubota, D. Gavel

Sep 22, 2009

1. Introduction

The *software control architecture* for the Keck Next Generation Adaptive Optics (NGAO) system has been reviewed by the authors. This design is documented in Keck Adaptive Optics Note #679 and was discussed during a half-day presentation on Monday, August 24, 2009. This report contains the recommendations and detailed comments of the committee.

2. Recommendations

The development team clearly understands the power of this new approach. The reviewers feel that this approach is technically credible. Closer alignment with user issues and the formal requirements are, however, recommended.

Specific Recommendations:

1. Provide details of why CFS was chosen over the considered alternatives (ACS, RTC, TANGO, and EPICS).
2. As planned, re-write the requirements document. As part of this process:
   - Tie selected requirements to use cases in the Observing Operations Concepts Document (OOCD).
   - Include quantitative performance requirements taken from the timing efficiency budget.
   - After the requirements document has been completed, provide a compliance matrix that indicates which sections in the design document address each requirement.
3. Work with budget manager to make sure this effort is staying within the original cost estimates for NGAO software.
4. Identify metrics that can be used to determine how well the system is performing, then instrument the control system to provide those metrics.
5. Provide feedback regarding the question of whether this system could be used in a phased implementation scheme (possibility necessitated by incremental funding). For example, would it be well suited for an upgrade to add an infrared wave front sensor to the existing system?
6. Regarding the mini-review process: Continue with this approach, keep it to a half day, but consider the following changes:
   - Cut down the presentation time by approximately 30 minutes.
   - Provide a summary of the reviewer's comments.

Risks and concerns: Reviewers expressed concern that:

1. CSF appears to be in its infancy, and, although used in several prototyping exercises, has not yet been proven in an operational environment.
2. We may be stranded with CSF. Would that be a problem?

3. Detailed Comments

This section provides detailed comments (primarily taken from the notes recorded by the chair during the review).

a) The mini-review process is a welcome change to a more heavy-weight review process for two reasons: (a) it is less disruptive to the design team's efforts completing the project; and (b) it allows for more frequent reviews so that recommended changes can be considered before design decisions are 'cast in concrete.'

b) Maintaining start-up configuration information in a relational database is the right thing to do.

c) It is good to hear that KCSF will provide support for synchronous task development (wait mechanisms and callback-on-completion mechanisms). This lacking in CA has made programming motion control more difficult and prone to race conditions. A system like KCSF (and KTL) that provides both peer-to-peer and publish-subscribe is best.

d) The need to support existing KTL clients (instrument fits writers, generic status displays like facsum and LUI that use KTL to obtain status from other subsystems, etc) is a requirement and should be formally stated in the requirements document.

e) The KTL interface required for the NGAO architecture (see d above), does not necessarily require a CA server and CAKE. What example of a direct CA client like that shown in figure 16 can be cited? The KTL model is better matched to KCSF (i.e., it includes methods for implementing synchronous tasks via wait mechanisms and callback-on-completion mechanisms; unlike CA). Inserting CA between CA and KTL will introduce an unnecessary and difficult translation.

f) Collaboration with ATST on the maintenance and continued development of CSF is compelling, but beware of shared code repositories. Releasing software in lockstep with an external facility (with its own set of user demands and scheduling constraints) has, historically, bogged down software development.

g) Python seems to be emerging as the common scripting mechanism that is accessible to both software engineers and scientists; however, it would be inefficient to expect scientists, observers, and astronomers to drop their traditional scripting methods (IDL, CSH, perl, and TCL) to adopt a new method immediately. As long as the KTL interface is faithfully provided, there is no problem offering python as the preferred alternative (so that we can methodically move toward more modern methods); but we don't want to rely on extensive re-training to maintain efficient operations.

h) The question of whether to adopt DDS or ICE as the underlying mechanism is critical. Would it be worth it to design some burn-in tests to characterize the reliability and performance of each?

i) It is great to hear that, for both DDS and ICE, "… ready to use graphical applications allow users to graphically display data coming from devices without the need for any programming…," but please do not subject these graphical interfaces to end-users (e.g., the DM-screen equivalents). Please provide user-friendly graphical interfaces by both (a) providing well designed GUI's with the delivered system and (b) providing a mechanism for scientists and astronomers to, post-delivery, extend the GUI systems themselves (presumably via python).

j) It was good to hear that the NGAO architecture intends to maximize the benefits of teaming software engineers with scientists. Again, python may be the common mechanism for this.

k) It would be good to see more use cases. In particular, small KCSF clients written in python (and KCSF+legacy-system clients written in python).