Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

Keck Telescope Control System

W. Lupton

W. M. Keck Observatory, 65–1120 Mamalahoa Highway, Kamuela, HI 96743

Abstract. The two 10m Keck telescopes run identical control systems, implemented using the Experimental Physics and Industrial Control System (EPICS). The new Adaptive Optics (AO) and Interferometer projects are also partly based on EPICS. However, other observatory subsystems, including acquisition and guiding, primary mirror control, and all the instruments, do not use EPICS. To manage this heterogeneous mix, the high level keyword-oriented Keck Task Library (KTL) was implemented. KTL can run on top of multiple message systems and allows a common API to be used for control, coordination and monitoring of all observatory subsystems.

This paper describes several aspects of Keck observatory software, concentrating on interfaces, infrastructure and technologies rather than on technical details. First of all, KTL is introduced, using the telescope control system as an example and illustrating how it was possible to move from the old pre-EPICS Keck 1 control system to the new EPICS control system with hardly any changes to the KTL keyword interface. Next, software for AO and the Interferometer (which uses AO) is described, with an emphasis on automation and the management of complexity. Finally, some new developments are discussed, notably the increasing use of Java and the move towards the use of CORBA.

1. Introduction

The W. M. Keck observatory operates the two 10m Keck optical / infrared telescopes on the summit of Mauna Kea. Keck 1 and Keck 2 have been operational since May 1993 and October 1996 respectively. A natural guide star (NGS) AO system has been scheduled on Keck 2 since Summer 1999, and the Keck Interferometer, a collaboration with JPL, is due to come on-line in 2002.

Both telescopes now run identical control systems, implemented using the Experimental Physics and Industrial Control System (EPICS, Dalesio et al. 1991). However, the original Keck 1 control system was not based on EPICS; the EPICS control system was written for Keck 2 and then retrofitted to Keck 1 in March 1997.

2. Keck Task Library (KTL)

In addition to the telescope control system, the new Adaptive Optics (AO) and Interferometer projects are also partly based on EPICS. However, other observatory subsystems, including acquisition and guiding, primary mirror control, and all the instruments, do not use it. To manage this heterogeneous mix, the high level Keck Task Library (KTL, Conrad & Lupton 1992, Lupton & Conrad 1992) was implemented. KTL can run on top of multiple message systems and allows a common API to be used for control, coordination and monitoring of all observatory subsystems.

2.1. Keywords

Subsystem interfaces are what we call "keyword" interfaces, meaning that the information flowing across them is in the form of name/value pairs, where the value is a scalar or a small array. These keywords are very like FITS keywords; indeed the original keyword concept was based on the idea of a set of FITS keywords which could be used both to describe system state and to restore the system to that state. In order to use a keyword, you need to know only its name and its purpose.

We require each subsystem to provide access to all its functionality via keywords (possibly with the exception of low-level engineering access). Indeed, subsystem acceptance procedures are normally implemented as scripts which access the subsystem only via keywords.

Keywords are not used for large data arrays. Typical uses of arrays are for binning factors (two elements) or readout windows (four elements). Such arrays are small enough that the value can be encoded as a string in a FITS file if necessary. Larger arrays are occasionally used, e.g. the AO system uses several 349-element arrays for telemetry items (the AO deformable mirror has 349 actuators).

It is often necessary to pass structured data between subsystems. For example, an astronomical target specification includes an (RA, Dec), a coordinate frame, an equinox, an epoch and proper motions. Rather than defining a record structure which would then have to be known on both sides of the interface, we define a keyword for each scalar item and another keyword which, when written, tells the telescope to go to the new target.

2.2. Application Programming Interface

The KTL API was defined in 1991. It offers control, coordination and monitoring of all observatory subsystems (Table 1). Currently, KTL is mostly used for GUIs and scripting.

2.3. Implementation

KTL is implemented as a C shareable library. The `ktl_open()` routine has a string argument which is the name of the subsystem whose keywords are to be accessed. From this name it derives the name of a shareable library (a so-called "keyword library") which it then activates dynamically and in which it expects to find the routines which implement the driver interface. A given KTL

Table 1. KTL API features

Feature	Description
Read a keyword	Return the current value of the specified keyword
Write a keyword	Set the value of the specified keyword to the specified value
Monitor a keyword	Request a callback whenever the specified keyword's value changes
Inquire attributes	Return information, e.g. list of supported keywords or, for a specified keyword, type, units, description etc.
Driver interface	Allows dynamic plug-in of subsystem-specific code

application can activate several different keyword libraries, each of which may use a different message system to talk to its underlying subsystem.

2.4. Generic Clients

Generic clients are "configured" rather than "programmed". In simple cases, the necessary configuration information can be passed on the command line (Table 2). For example,

```
% show -s dcs telescop instrume utc
                      telescop = Keck II
                      instrume = LRIS
                           utc = 02:50:10.41 h

% modify -s dcs telfocus=12
setting telfocus = 12.000 (wait)
```

Table 2. Some simple generic KTL clients

Name	Description
show	Shows the values of one or more keywords (specifies the subsystem and keyword names)
modify	Changes the values of one or more keywords (optionally blocking on completion; optionally, if supported by the keyword library, in parallel)
xshow	Monitors one or more keywords, maintaining a simple X display of their names and current values
cshow	Like xshow but logs to standard output rather than creating an X display
waitfor	Monitors a keyword until it has the specified value or a timeout occurs

In more complex cases a configuration file must be created (Table 3). Use of generic clients can dramatically reduce development time (because configuration is quicker than programming) and increase reliability (because code gets more widely used and is therefore better tested).

Table 3. Some complex generic KTL generic clients

Name	Description
kezar	Monitors a list of keywords and writes a log file of time-stamped keyword names and values
electrolog	Maintains an X display showing a table of the values of a selected set of keywords; creates a new row of the table when a new exposure begins; keeps a record of the table on disk

2.5. Scripting

Most scripting applications simply invoke generic clients (*show*, *modify* etc.) from within *csh* scripts. For example, all of NIRC's user commands are implemented as *csh* scripts (NIRC is the Caltech Near-InfraRed Camera).

Use of *csh* is adequate for simple cases but for more complex or event-driven scripts there is increasing use of *ktcl* and *kidl*, which are, respectively, KTL's Tcl/Tk and IDL[1] (Interactive Data Language, not Interface Definition Language!) interfaces. Both *ktcl* and *kidl* are implemented as dynamically loadable shareable libraries and both support all KTL facilities, including monitors.

Many instrument specialists and observers write *csh* keyword scripts. This is encouraged by the simple keyword model, the generic clients, and general familiarity with shell programming.

3. EPICS

We have already said that EPICS is used for the two telescope control systems and for parts of the AO and interferometer. We will not discuss technical details of EPICS or of the control systems here (Lewis et al. 1996, Lupton et al. 1996), but will draw some comparisons with KTL and describe our experiences.

3.1. Why Did We Use EPICS for the Keck 2 Control System?

We had enough problems with the Keck 1 control system to justify a new approach for Keck 2. For example,

Tight system/application coupling There was no application-independent infrastructure; this created undesirable coupling between what should have been system and application modules; it also meant that writing applications was much harder than it should have been.

Difficulty of extension The system was very hierarchical; this is good from the point of view of hiding information, but in practice it meant that many tasks were just traffic managers for their sub-tasks; adding or extending commands required changes to many modules.

---

[1]IDL is a registered trademark of Research Systems, Inc.

Dependence on VMS The old system used Unix for user interfaces, VMS for middle-tier processing, and VxWorks for real-time processing; VMS was falling out of favor during system development; its use required an unnecessary and messy Unix/VMS interface which added considerably to the difficulty of extension.

This was early 1994. We knew that Gemini had adopted EPICS and that UKIRT were looking at it, so we decided to find out more. We were impressed by the system but were worried that its adoption would force our team to make too many changes in their ways of working. For example,

Database programming Much of the data flow within the real-time systems is implemented via links within an EPICS database; these links are not programmed via a conventional language but are instead configured in a data file or, more commonly, connected using a visual tool.

Sequencing Most sequencing is implemented using the EPICS sequencer; state transition diagrams are entered in a state notation language (SNL) which is then converted to C.

In the event, after an appropriate review process, we decided to proceed with EPICS for the Keck 2 control system. The rest is history.

3.2. Channel Access

EPICS provides its own message system, known as Channel Access (CA). Clients connect to channels by name (they do not need to know on which server a channel resides). This is sufficiently close to the KTL model that we were able to implement a generic keyword library for all EPICS systems.

Given an EPICS system, it is necessary only to create a configuration file in order to access it via KTL. The configuration file, which is read at run-time, defines the keyword names and indicates which EPICS channels they correspond to. Via such a configuration file we were able to define a KTL interface to the new EPICS telescope control system which was almost identical to the old non-EPICS interface. This meant that the KTL GUIs from the non-EPICS system were able to run almost unchanged with the EPICS system.

3.3. Generic Clients

EPICS provides a rich set of generic clients, some of which have KTL equivalents (Table 4). As is true of the generic KTL clients, use of these tools can dramatically reduce development time. EPICS contains several quite sophisticated tools for which there are no KTL clients, and these are particularly useful. We tend to use these in violation of our own "keywords for everything" rule.

It goes without saying that such generic tools are only possible if control system objects are accessed by name and if any structures used are known to both client and server (structures where members are accessed by name are not really structures, they are aggregations of scalars!).

Table 4. EPICS generic clients

Name	KTL equivalent	Description
caget	*show*	Gets and prints the value of an EPICS channel
caput	*modify*	Sets the value of an EPICS channel
probe	*xshow*	Monitors a channel, maintaining an X display of its name, current value, severity, time stamp etc.
camonitor	*cshow*	Monitors a channel or channels but logs to standard output rather than creating an X display
ezar *ar*	*kezar*	Monitor a list of channels (taken from a configuration file) and write log files of time-stamped channel names and values (NB, two different tools)
edd/dm *medm*		In edit mode, allow layout and customization of widgets; in execute mode, go "live" (NB, two different tools)
alh		Monitors a list of channels and displays a fault tree (structure taken from a configuration file); operator acknowledgment, enable/disable etc.

3.4. Experience

EPICS has been a success at Keck. This is not the case at all observatories which have used it (indeed it has quite a bad reputation with some) and it is interesting to consider why.

Steep learning curve At the start of our first EPICS project we paid for two members of the Los Alamos EPICS team to give an EPICS training course in Hawaii. Over the next few months, other members of the same team spent time with us, mostly helping with device drivers and giving general advice.

For at least the first year of our EPICS development, we made every effort to compartmentalize knowledge. One of us handled EPICS builds, another knew all about device drivers, a third was an expert in the Capfast database generation tool, and so on.

We had sufficient resources, and our group was of a sufficient size, to permit this approach. Getting started with EPICS can be particularly difficult for a very small group.

Poor documentation Like the curate's egg, EPICS documentation is "good in parts". The lack of overview documentation is a problem, especially when starting out.

The only practical solution to this (short of remedying the omission) is to have a local EPICS expert who can fill in the gaps for the rest of the group. This is what we did.

Too many choices There's always more than one way to solve a problem. Why are there two display managers? When should I use the database and when should I write a sequence?

This is really a fact of life for systems such as EPICS, which are created by volunteers at many sites. There are two display managers because they

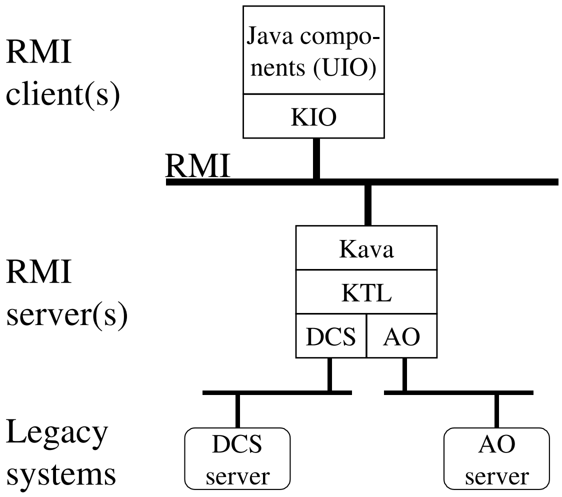

Figure 1. Three layer AO WUI model.

were written at two different sites and, when development started, neither met (nor promised to meet) the other site's needs.

The fact remains that the core system and many of the tools are of the very highest quality. They are used by many groups for many different applications on a wide range of Unix and VME hardware and are therefore much more reliable than one would expect from comparable tools produced for a single project.

4. New Developments

We will describe some new developments in the use of:

Java, which is now used for several projects, e.g. for the AO web-based user interface (WUI);

CORBA, which is being used to interface to the Keck Interferometer and to create a CORBA keyword service;

High-level sequencing, which is becoming more important, driven by interferometer requirements.

4.1. Java and the AO WUI

The AO WUI allows control of the AO system from a web browser. Java applets in the browser instantiate UIOs (User Interface Objects) which use KIOs (Keyword Interface Objects) to talk via Java RMI (Remote Method Invocation) to a Java keyword server. The Java keyword server uses the JNI (Java Native Interface) to interface to KTL and thereby communicate with legacy systems. This three layer model is illustrated in Figure 1.

4.2. Use of CORBA

Keck and JPL evaluated CORBA (using the free TAO ORB) and have decided to use it to interface to the Keck Interferometer and to implement a CORBA keyword service.

CORBA allows a client written in one language to communicate with a server written in another. If the AO WUI's use of RMI were replaced with CORBA, the server could be written in C++ and it would not be necessary to use Java native methods to interface to the legacy systems.

IDL and services CORBA applications use IDL (Interface Definition Language this time!) to specify interfaces in a language-independent way. Some interfaces are pre-defined and come as part of the CORBA implementation. These are referred to as "CORBA services".

One such service is the "Property Service", which allows a set of named properties and property values to be associated with an object. Such properties and property values are conceptually pretty close to keywords and keyword values.

COKE We are extending the Property Service to create a Keyword Service that we call "COKE" (**CO**RBA **KE**yword). We are using the Property Service at two levels:

1. To list the keywords supported by a given server (the property name is the keyword name and the property value is the keyword object reference),
2. To list a given keyword's attributes, e.g. range, units, value, severity, time stamp (the property name is the attribute name and the property value is the attribute value).

The Property Service does not support the delivery of events to clients when property values change. We are using the TAO Real-time Event Service to add event support to COKE.

The resulting three layer CORBA model is illustrated in Figure 2. Not surprisingly, the main difference from Figure 1 is the replacement of RMI with IIOP (Internet Inter-ORB Protocol) and the use of the COKE interface.

As should be clear from the tone of this section, the COKE Keyword Service is still under active development.

4.3. Sequencing for the Interferometer

Unlike most other modern observatories, Keck has no OCS (Observatory Control System). This is because our user community has never required one. High-level sequencing is restricted to various instrument-specific scripts.

The full-up interferometer system consists of the interferometer itself (delay lines, fringe tracker etc.), two 10m Kecks (each with an NGS AO system), and four 1.8m Outrigger telescopes (each with a fast tip/tilt mirror). It is far more complex than any of our existing systems, and its automated operation has always been a requirement. The big worry, of course, is that, with so many subsystems, we will not be able to achieve acceptable system reliability.

We are dividing the sequencing task between two different tools, described below.

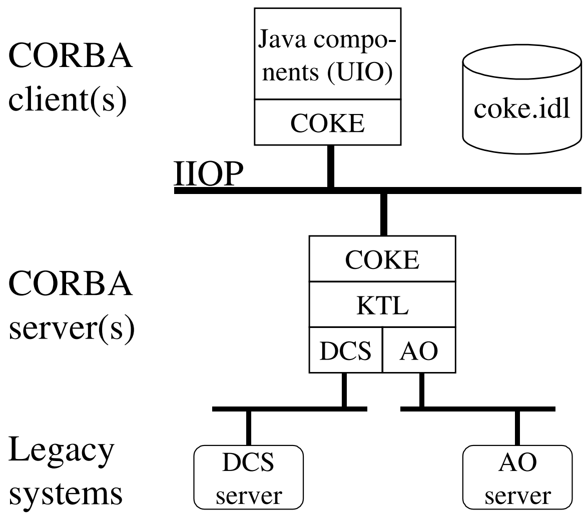

Figure 2. Three layer CORBA COKE model.

Interferometer sequencer This is JPL's responsibility. It understands about interferometer observing modes and calibration sequences. It talks to the telescope sequencers, one per telescope.

The interferometer sequencer will be implemented using NASA's "Remote Agent" (RA). This is a planner and sequencer which was originally developed to increase spacecraft autonomy, has recently flown on NASA's DS1 mission, and which will be used not only for the Keck Interferometer but also for NASA's future space-based interferometers.

Telescope sequencer This is Keck's responsibility. There will be an instance of this sequencer per telescope. It is responsible for a single telescope and its subsystems.

The telescope sequencer will be implemented using a Unix/KTL version of the EPICS sequencer. This is a natural extension of our use of the same tool for lower-level sequencing within the telescope and AO control systems. The telescope sequencer will also be useful for non-interferometry applications.

5. Conclusions

The main message of this paper is that KTL and its keyword model have served us well at Keck. KTL offers clean subsystem interfaces, encourages the use of generic clients, is simple and accessible to all, and is easily integrated into scripting languages.

We do not see this keyword model being superceded as we adopt new technologies such as Java and CORBA. As we have described, we are working on a CORBA Keyword Service (COKE). It is quite possible that COKE, rather than KTL, will become our interface standard.

Finally, we have attempted to explain why it is that EPICS has been a success at Keck.

References

Conrad, A., & Lupton, W. 1992, in ASP Conf. Ser., Vol. 52, Astronomical Data Analysis Software and Systems II, ed. R. J. Hanisch, R. J. V. Brissenden, & J. Barnes (San Francisco: ASP), 203

Dalesio, R., Kraimer, M., & Kozubal, A. 1991, in Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Machines, KEK, Tsukuba, Japan, 278

Lewis, H., Lupton, W., Tsubota, K., Honey, A., & Quady, S. 1996, in Optical Telescopes of Today and Tomorrow, ed. A. Ardeberg (SPIE), 2871, 970

Lupton, W., & Conrad, A. 1992, in ASP Conf. Ser., Vol. 52, Astronomical Data Analysis Software and Systems II, ed. R. J. Hanisch, R. J. V. Brissenden, & J. Barnes (San Francisco: ASP), 315

Lupton, W., Lewis, H., Tsubota, K., Honey, A., & Quady, S. 1996, in Optical Telescopes of Today and Tomorrow, ed. A. Ardeberg (SPIE), 2871, 977

Acknowledgments. The W. M. Keck Observatories are operated as a scientific partnership between the California Institute of Technology and the University of California. The observatories are made possible by the generous financial support of the W. M. Keck Foundation.