Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

A Radical Approach to Software Modularity in Telescope Control Software

M. H. Clark

National Radio Astronomy Observatory, Green Bank, WV 24944-0002, USA

Abstract. A difficulty in writing control software for research telescopes is designing sufficient flexibility to handle continually changing requirements. The primary goal of the control software architecture for NRAO's new 100-meter radio telescope in Green Bank, West Virginia is flexibility through the construction of a radically modular software system.

1. Introduction

The original motivation for the design used in the monitor and control system for the Green Bank Telescope (GBT) (Lockman 1998) was to generate a radically modular "straw man" in order to better understand the limits of modularity[1] when implementing a control system for a general purpose telescope such as the GBT. However, to our surprise, repeated scenarios did not break the model, so it was decided to adopt the design for the GBT. There were two other influences affecting the choice of design. We were given the requirement to think in terms of a virtual telescope, i.e., to design a system that was not telescope specific. This was important not so much because there were plans to use the system on additional telescopes, but rather the fact that a telescope does, in fact, become a different telescope over time because of changes in instrumentation. The second influence was that, at the time, I was first becoming impressed with object-oriented methods, and my inclination was to design everything as objects. No regrets or apologies.

The basic design[2] (Clark 1998) is simple. A radio telescope is defined as a laboratory rather than an instrument, i.e., it is a set of devices (of which the antenna is only one) which needs to be configured in novel ways to accomplish observations (or scans). Each device is an autonomous subsystem which can be fully configured prior to a scan through a finite set of control parameters in order to run the scan in coordination with other devices. For purposes of configuration and observing, each device requires no more than four interfaces (control, monitor, message/alarm, and data) which are identical for all devices. The same control interface for user programs is also used to recursively build a tree of control modules whose root is a generic "scan coordinator". Any

[1] See Meyer (1988) for a definition of software modularity.

[2] `http://info.gb.nrao.edu/~mclark/article/article.html`

control module of the tree may be used to initiate a scan for the sub-tree under its control because each control module is derived from the same base class (called a Manager) and therefore inherits all of the control protocols required to coordinate and initiate a scan.

2. Advantages

In the GBT, a device is implemented using two sources of code: generic code from base classes which is used across all devices, and code written specifically for the device either by writing virtual methods inherited from a base class or by implementing new classes. Because the four device interfaces are identical for all devices, a larger proportion of the functionality can be designed into the base classes, and even much of the device-specific code is implemented as virtual methods. This approach allows many features to be handled wholly in the base classes.

An example of the use of base classes is the integration of the message/alarm system into the control system. A Manager's scan parameters can generate two possible errors: one is the result of user input (illegal value) and the other is the result of failures in digital interfaces (activate fault). Since some Managers may require a large numbers of scan parameters in order to define the operation of the device, specifying individual messages for illegal values and activate failures can be a non-trivial effort. In addition, there is the possibility of inconsistency of implementation across devices or even in the wording of the messages. It would be an advantage if these messages could be wholly implemented in the base Manager. Though the methods to check and activate scan parameter values are written for specific parameters in the derived Manager, these methods are called in the base class. The base Manager class uses the return value of the scan parameter virtual methods to generate and report the appropriate messages for all parameters.

Another example of integration of the control and message/alarm systems is the generation of a summary status for each device. Each message is assigned a severity level (i.e., information, notice, warning, error, fault, or fatal). The base Manager class computes a status based on its most severe message currently active. The control hierarchy of Managers that make up the control system recursively passes these summary statuses up the hierarchy so that the status of each device and subsystem reflects the most severe level of all of its messages. From this it becomes trivial to create a display for indicating statuses for the entire system.

Each individual monitor value and scan parameter is referenced by an object of the class DataDescriptor which contains a complete description of the value including name, explanation, data type, size, units, and array count (if a vector). These DataDescriptors allow more functionality to be handled by the base classes than otherwise would be possible. For example, the user programs do not have to be rewritten (or even recompiled) to handle new devices and their associated scan parameters and monitor values since the reading, writing, communicating, and displaying of their values depends on code based on the DataDescriptors, not on compiled data types. Also, the displays can be embellished with explanations and units since the information is readily available. In fact, the DataDescriptor is complete enough that headers for FITS binary tables (Cotton, Tody, & Pence 1995) can be generated automatically. This allows the engineers to select any monitor point, whose values are sent to a logging program which generates files in FITS format. Likewise, the observer can select any monitor point's values to be stored as part of the observational data along with traditionally sampled values such as antenna positions and weather.

3. Pitfalls Avoided

The project's highly modular approach has allowed us to avoid many problems which are traditionally encountered while developing a telescope control system.

The GBT control system is a distributed system. Rarely anymore does one find a large telescope control system where all devices are controlled and monitored from one computer. Even if one tries to avoid building a distributed system, the continuing introduction of new devices often with their own control computers will create a range of computers that need to communicate with each other. If one designs a control system so that each device acts as an autonomous unit, then one can remove the need for real-time dependencies between computers. Because each device in the GBT is almost wholly independent of other devices, the GBT monitor and control system is able to use "vanilla" ONC RPCs to handle those communications which are needed. So, though it is a distributed system, the development effort is greatly reduced.

As in any system, the devil is in the details, and we have expended a significant amount of effort in debugging code as devices undergo increasing amounts of use. However, integration per se, i.e., problems resulting from adding devices to the system as they come on line, has been almost painless. The interfaces between devices are so few and small that system integration has simply not been an issue.

Related to system integration is device testing and maintenance. In early reviews of the requirements and design for the system, a recurrent request from engineers was the ability to run and test a specific piece of equipment while the rest of the telescope was being used by other engineers, was shut down, or was being used for observing. Specifically, they wanted to know what they had to do to get the device to work, e.g., how to run a backend with no antenna which traditionally initiates the "go" signal and generates antenna positions. Because of the autonomy of each device, the answer is "nothing."

Some software engineers like to stress that they are "real-time" programmers, almost as if it were a badge of honor. And it is certainly true that the constraints of deadlines in code adds an entirely new dimension to every aspect of a problem. In large real-time systems the problems become very complicated very quickly. Many large real-time systems are tours de force and the brain children of some extremely talented people. However, by defining a telescope control system not as a real-time system, but as a computer system that contains small pieces that are real-time, the problem becomes greatly simplified. Because each device is an autonomous unit, one can design devices so that real-time loops are closed within the same processor. And perhaps most importantly, because real-time dependencies are local to a device, the implementation of these real-time programs can be handled by a single engineer.

One of the most gratifying aspects of dealing with a system where modularity has allowed a single set of interface to be developed for all devices on the telescope, is the ease of integrating this interface into interpreters which are used both as a user-interface and a tool to generate graphical user interfaces. We currently have two interfaces for the system: Glish (Schiebel & Paxson 1998) and Tcl/Tk (Ousterhout 1994). The original implementation of a Glish client for the antenna was a two day effort, and the current client which works for all devices took three weeks. The enhanced Tcl interpreter for communicating with all telescope devices took less than a month. As a device comes on line, we expend no effort toward interfacing the new device to our interpreters, though the appearance of the graphical user interface per se for a specific device does take significant thought and planning.

Acknowledgments. My original concepts (Clark 1992)[3] underwent extensive change and enhancement because of the experiences gained and the close-knit team effort during development. I especially want to thank J. Richard Fisher and Geoff Croes, without whose confidence and invaluable insights, the project would have fallen far short of its goals.

The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

References

Clark, M. H. 1992, GBT M&C Requirements Analysis Model (GBT Memo Ser. 88), (Green Bank: NRAO)

———, 1998, in SPIE Proc., Vol. 3351, Telescope Control Systems III, ed. H. Lewis (Bellingham: SPIE), 287

Cotton, W. D., Tody, D. & Pence, W. D. 1995, A&AS, 113, 159

Lockman, F. J. 1998, in SPIE Proc., Vol. 3357, Advanced Technology MMW, Radio, and Terahertz Telescopes, ed. T. G. Phillips, 656

Meyer, B. 1988, Object-oriented Software Construction, (New York: Prentice Hall)

Ousterhout, J. K. 1994, Tcl and the Tk Toolkit, (Reading: Addison-Wesley)

Schiebel, D. & Paxson, V. 1998, "The Glish 2.7 User Manual", National Astronomy Radio Observatory, `http://aips2.nrao.edu/aips++/docs/reference/Glish/Glish.html`

[3] `http://info.gb.nrao.edu/GBT/memos/GBTmemos.html#1992`