New control system software for the Hobby-Eberly Telescope

Tom Rafferty (rafferty@astro.as.utexas.edu), Mark E. Cornell, Charles Taylor III, Walter Moreira
McDonald Observatory, University of Texas at Austin, 1 University Station C1402, Austin, TX, USA 78712-0259;

Design Principles
- Component-oriented design
- Object-oriented programming techniques
- Physical world is composed of components and modules that are connected with "wires"
  - Each wire represents an interface (TCP/IP, etc.)
  - Components are "wired" together
- Simultaneous development of all objects in systems
- Similar hardware drivers implement the same power supply interface
- Shared algorithms, routines, utilities, etc
- Suitable interface built at device level
- Controls low-level device drivers to provide I/O to the hardware
- Provides abstraction interface to the device so changing hardware manufacturer is simple
- Defines API using ADO/COM with a well-defined interface to the action processing system
- Based on common, well-defined interface to manipulate the hardware subsystems
- State machine driven

Control System Architecture
- Used as basic architecture for any system
- Configurable modules to existing design
- Expandable down the lower level (hardware)
- Freeform (not wired to special hardware)
- Provides hardware abstraction using interfaces

Development Environment
- Primary Platform: Windows, Unix
- Compiler: C, C++, Python, SQL
- Tools: emacs, gn-icon, build, tcpdump, Wireshark, adb
- Technology: TCP/IP, database (MySQL, Postgres), GUIs, Python libraries, Protobuf, MAVEN/GRADLE, Java GUIs, GUI, Spring, Django, Swagger, Python, SQL
- Continuous integration: Jenkins setup using Azure
- Version control: git with a central "Locker" repository
- Other: Automated testing of anomaly detection and high-level logic for log monitoring

Distributed GUI
- Data model in control systems, not GUI
- Each control system has only one GUI (server)
- Simplifies debugging
- Each GUI has a common dashboard at the top, but the remaining components can be customized according to the role of the user

User Interface
- GUIs for control, monitoring, and analysis
- Each GUI has a common dashboard at the top, but the remaining components can be customized according to the role of the user

Central Logging
- Helps manage system behavior
- Integration with other systems

Scripting Engine
- All devices, servers, and actions available to scripting engine through the "magic" of WSGI wrapper
- CAN be compiled into Python snippets and run anywhere within the system
- Allows the possibility of completely changing the configuration of your system simply by using a different or modified configuration script
- Testing and debugging simplified because you can easily isolate any driver or change it's properties

Network Server
- Using well-known port numbers
- Database engine can be changed to run on a thin layer above a defined interface
- Currently uses MySQL but allowing high-level protocol stack and easy integration with other systems
- Use of SVN for further testing and debugging
- Database is not a part of the system, but can be used in cases of overloading. Each of the components can be run on separate machines if necessary

Database Backend
- Handles the logic between network and database backend
- Does not store any historical data
- Provides a link to the database
- Leverages the power of Python and its libraries for development

Data Flow
- Data model in control systems, not GUI
- Each control system has only one GUI (server)
- Simplifies debugging
- Each GUI has a common dashboard at the top, but the remaining components can be customized according to the role of the user

Sample GUI showing Dashboard at top, and specific data below