A Java GUI Client for Remote Interactive Observing: a Front-End to the Telescope Automation and Remote Observing System (TAROS)

Dione F. Smith, Adam Czeźowski, Mark A. Jarnyk, William H. Roberts, Greg M. Wilson, Peter J. Young, Anthony C. Green, Gary R. Hovey, Jon G. Nielsen, Kim M. Sebo, Annino L. Vaccarella

Mount Stromlo Observatory, Research School of Astronomy & Astrophysics, The Australian National University, Canberra, Australia

Abstract. TAROS is a system that will allow The Australian National University telescopes at Siding Spring Observatory to be operated automatically or interactively with authenticated control via the Internet. In the context of remote interactive observing, TAROS is operated by a Java GUI front-end which incorporates JSky components and relies on two key Java technologies: Java Message Service and Java Native Interface. This paper, the third in the TAROS series, describes the TAROS client application for remote interactive observing and its underlying GUI framework.

1. Introduction

The Research School of Astronomy and Astrophysics (RSAA) at the Australian National University (ANU) is upgrading and extending its software support at Siding Spring Observatory (SSO) to allow its telescopes to be operated automatically or interactively with authenticated control via the Internet. In particular, the observatory’s existing 2.3-m telescope will be supported for interactive use, with selected new and old instrumentation, and its new SkyMapper telescope (which is designed to carry out an automated wide field imaging survey of the southern sky) will be supported for automated operation. TAROS is the software being engineered to provide the support.

As part of the TAROS initiative, a Java GUI client application is being developed for astronomers to conduct their observing runs from remote locations on the Internet. The application relies on two key Java technologies employed in TAROS: Java Message Service (JMS), for communicating between the remote observer and the TAROS server; and Java Native Interface (JNI), for integrating both newly developed Java classes and extensions to Java GUI components.

1http://www.mso.anu.edu.au/
2http://www.anu.edu.au/
3http://java.sun.com/products/jms/
4http://java.sun.com/docs/books/jni/
from the JSky\textsuperscript{5} collection with existing instrument control and data acquisition software (CICADA)\textsuperscript{6} written in C++ (Young, Roberts & Sebo 1999).

The TAROS architecture (Wilson et al. 2005) and its communications infrastructure for coordinating the operation of distributed components and interactions with remote (and local) client applications (Cze\'zowski et al. 2006) have been presented at previous ADASS conferences.

2. TAROS Client/Server Communication

A TAROS client communicates with the TAROS back-end via the system’s key API: \texttt{ClientConnection}. This is a high-level interface designed for all user-related control and monitoring of the telescope, instrumentation, operating environment and data acquisition. Note that, although JMS is the message transport mechanism used for all TAROS client/server communication, the abstraction provided by \texttt{ClientConnection} means that a TAROS client can be implemented without referring to JMS code. The API allows a TAROS client to make an authenticated connection to the TAROS back-end, with username, password and observing run identifier; disconnect from the TAROS back-end; send a command to the TAROS back-end and get an acknowledgement in return (indicating acceptance or otherwise of the client’s command); and subscribe to status information updates regarding the telescope, instrumentation, operating environment and data acquisition.

3. Key Application Windows

Upon starting the TAROS GUI client for remote interactive observing, the user is presented with the \texttt{MainWindow}, configured for operating a specific telescope and science instrument. The \texttt{MainWindow} serves as the application hub: from it all the other primary windows in the application are accessible. Those representing key objects that the observer would want to control and/or monitor during an observing run include: \texttt{WeatherWindow} for displaying live meteorological data from the telescope site; \texttt{DomeWindow} for controlling, monitoring the telescope’s operating environment; \texttt{TelescopeWindow} for controlling and monitoring the telescope; \texttt{InstrumentWindow} for controlling and monitoring the science instrument; \texttt{GuideWindow} for performing object acquisition and guiding; \texttt{DataWindow} for performing science data acquisition; and \texttt{DisplayerWindow} for displaying quick-look images of acquired science data.

4. TAROS GUI Framework

A conceptual GUI framework has been evolved and is evolving from generalising specific GUI window needs encountered during the development of the TAROS GUI client for remote interactive observing. The framework is based on Java’s

\textsuperscript{5}http://archive.eso.org/JSky/rootpage.html

\textsuperscript{6}http://www.mso.anu.edu.au/computing/cicada/
Swing GUI components, which form part of the Java Foundation Classes (JFC), and relies on Java 1.5 language features, most notably enums and generics. It provides a structured basis for coding individual GUI windows. As far as possible an attempt has been made to separate the “application logic” from actual GUI elements; facilitate customisation of GUI text; systematically manage the application’s user-initiated tasks; and coordinate interactions between the application’s top-level windows.

The remainder of this section lists, with brief explanations, current features in the TAROS GUI framework.

**Abstraction over `javax.swing.JInternalFrame` and `javax.swing.JFrame`** allows windows to be coded independently of the chosen window scheme.

**User-initiated tasks represented as action/functor pairs** An action is a `javax.swing.AbstractAction` subclass and a functor is an object representing a single, generic function. A specific action is paired with a specific functor; the functor knows how to execute the action. The approach effectively separates application logic from actual GUI elements and allows polymorphism to be used when a window handles its user-initiated tasks, of which there are typically many.

**Codified notions of primary and secondary windows** streamline development of all application windows; `JInternalFrame/JFrame` abstraction and action/functor-based support for user-initiated tasks are incorporated into the relevant base classes.

**Single and multi-action GUI controls** are supported via `JComponent`’s client properties. A number of distinct GUI elements with associated actions, where relevant, can be treated as a unit. GUI elements presenting status information updates are attached to a GUI control via the same technique. This feature grew out of the need to control and monitor a science instrument’s configuration.

**Ability to classify actions as connection-dependent, privileged** is required by the TAROS client application for remote interactive observing. The framework supports this need.

**A primary window hub** is provided by the framework to create an application’s primary windows and coordinate interactions between them.

**GUI text sourced from `java.util.PropertyResourceBundle`** Each action, window has a `ResourceBundle`. Sourcing GUI text from it means that there is no need to recompile code if only changes to the GUI text are made; this feature also leaves the door open for an application’s internationalization and corresponding localization.

**Fallback exception handler** minimally logs an exception (using Apache’s `log4j` logging) but can also propagate it (unchecked) or cause the application to abort, in which case there is an opportunity for the application to attempt a graceful exit.
Integrated JSky components The DisplayerWindow in the TAROS client application for remote interactive observing presents quick-look images of acquired science data, using an extension of JSky’s DivaMainImageDisplay to do so\textsuperscript{7}. This is made possible because the framework incorporates an extension of JSky’s ImageDisplayControl class.

5. Conclusion

The TAROS client application for remote interactive observing is a work-in-progress; as of October 2006, implementations exist for MainWindow, DataWindow, DisplayerWindow, InstrumentWindow and WeatherWindow, with work on an initial TelescopeWindow to commence in November 2006. TAROS is scheduled for commissioning mid-2007.

Although the TAROS GUI framework has been engineered to service remote interactive observing in an RSAA context, its design lends itself to being further refined and applied in other contexts.

References


\textsuperscript{7}as will the GuideWindow to present guide camera images