Fully robotic telescopes are carrying out a growing number of tasks. The majority of us are focused on Gamma-ray Burst (GRB) follow-up observations. A lot of this work is done through various astronomical calculation libraries, particularly Libnova. Libnova is an open-source celestial mechanics, astrometry and astrodynamics library. Libnova provides RTS2 with all its calculus machinery. Libnova is highly competitive. The GRB field is very fast-paced and requires a lot of computational power. Libnova is written in C and is usually compiled with a few thousand lines of source code. As we are developing our new WWW interface, we believe that converting the C Libnova library to a Java library by using a third-party library would be a good solution. Drafts of the project can be used to further study the GRB in question, he should be contactable. When a GRB observer is making a decision which instruments to contact, how many triggers are left. In an ideal world, this data must be presented in a quick, efficient, and clear manner. Thus, the development of our own GUI or sophisticated AJAX elements appears to be a potentially good solution. Drafts of the information dialog boxes are presented in figures 3 and 5.

3. Global RT planner and scheduler

GRBs and other target-of-opportunity (TOO) observations have an important common aspect – it is essential to minimize any delay in commencing observations. In the case of GRB observations, every minute can make a difference between obtaining data with a detection of an optical transient, or only an upper limit. Hence, it is crucial to trigger the TOO observing programmes on larger telescopes. When a GRB observer is making a decision which instruments to contact, how many triggers are left. In an ideal world, this data must be presented in a quick, efficient, and clear manner. Thus, the development of our own GUI or sophisticated AJAX elements appears to be a potentially good solution. Drafts of the information dialog boxes are presented in figures 3 and 5.

4. Web Interface for RT

The RTS2 web interface is based on the thin client idea. Data is taken from a PostgreSQL database used by RTS2. Display is provided by Java servlets running on an Apache Tomcat server and a couple of Java custom classes. Graphical user interface uses Java-Script client-side framework Ext. The graphical user interface uses the Java-Script client-side framework Ext. The communication between server and client side is asynchronous, using XML as the message format (AJAX). The web client can also offer limited control over the devices and servers controlled by RTS2. A SOAP server, part of RTS2, is used to pass commands from the client to the server.

5. Image processing library

Currently, image processing in RTS2 is performed by calling shell scripts or scripts of calls to various image processing routines, utilising either independent binaries or routines in IRAF, Octave or other image processing environments. While that approach works, it adds a significant processing load on the system when the program is initialised. As the script subroutines are independent programs, each must open a FITS file where the image is stored. Instead of passing routines, parse the FITS file, do some computation and store the resulting image. Even with the improved performance of modern operating systems, achieved by caching data in memory as much as possible to avoid accessing data from hard drives (or other slow I/O channel) unless absolutely necessary, significant computational costs associated with the parsing of image meta-data remain.

6. Java astronomical calculation library

RTS2 uses Lithovna, an open source celestial mechanics, astronomy and astrodynamics library. Lithovna provides RTS2 with all its calculus machinery. Precise calculations of lunar and planetary positions require thousands of lines of source code. As we are developing our new WWW interface as a Java servlets application, we would like to have a pure Java solution to compute various target properties, and display target information on the web page.

Lithovna can provide us with that information, but as it is written in C and is not object-oriented, it will be very difficult to use it in Java. We believe that converting the C Lithovna library to a Java library by using a Java Native Interface (JNI) will result in a pseudo-object-oriented library, which we think is not worth the effort.

Ideally we would like to have the precision and stability of Lithovna coded natively in an object-oriented language, such as Java. According to our estimates, this task would require one to two months of design, and about a month to perform the coding, and test the Java library. However, we believe that using C++ or some other higher level language (e.g. Python) would be a more realistic option.