Astronomical Data Analysis Software and Systems XII
ASP Conference Series, Vol. 295, 2003
H. E. Payne, R. I. Jedrzejewski, and R. N. Hook, eds.

The Raptor Real-Time Processing Architecture

Mark Galassi, Daniel Starr, Przemyslaw Wozniak, Konstantin Borozdin

Los Alamos National Laboratory, Los Alamos, NM, USA

Abstract. The primary goal of Raptor is ambitious: to identify interesting optical transients from very wide field of view telescopes in real time, and then to quickly point the higher resolution Raptor "fovea" cameras and spectrometer to the location of the optical transient. The most interesting of Raptor's many applications is the real-time search for orphan optical counterparts of Gamma Ray Bursts.

The sequence of steps (data acquisition, basic calibration, source extraction, astrometry, relative photometry, the smarts of transient identification and elimination of false positives, telescope pointing feedback, etc.) is implemented with a "component" approach. All basic elements of the pipeline functionality have been written from scratch or adapted (as in the case of SExtractor for source extraction) to form a consistent modern API operating on memory resident images and source lists. The result is a pipeline which meets our real-time requirements and which can easily operate as a monolithic or distributed processing system.

Finally, the Raptor architecture is entirely based on free software (sometimes referred to as "open source" software). In this paper we also discuss the interplay between various free software technologies in this type of astronomical problem.

1. Scientific Motivation

The January 23 1999 burst (sometimes referred to as the "Rotse Burst", Akerlof 1999; Akerlof & McKay GCN 205[1]) showed that we can detect the *prompt optical emission* of Gamma Ray Bursts (GRBs) with inexpensive wide field-of-view cameras.

The very recent October 4, 2002 burst from HETE (1) confirms this (it was seen by amateur telescopes), and it raises the stakes on the prompt optical emission.

Right now it could be said that the prompt optical emission is the holy grail of GRB science, and many satellite and ground-based experiments are being planned to help observe it.

1.1. How to Capture the Prompt Optical Emission

There are two main approaches that are being tried right now:

[1] `http://gcn.gsfc.nasa.gov/gcn/gcn3/205.gcn3`

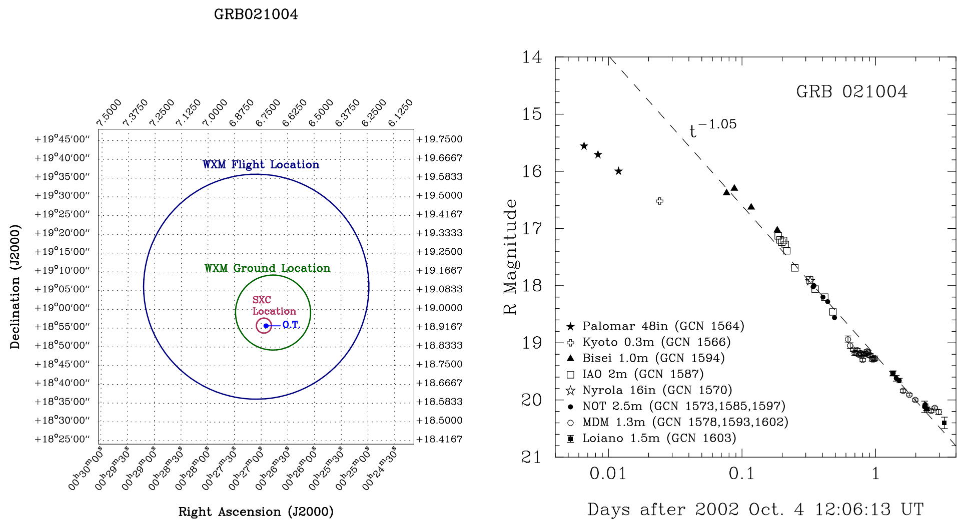

Figure 1. The 2002-10-04 GRB real time localization by HETE-2, and the decay of the optical intensity

- rapid robotic response to HETE or Swift events, and
- capturing the optical signal when it first appears, possibly without a gamma ray or X-ray trigger at all!

The first approach can be very effective for $t0 + 45$sec but it cannot reach $t0$ except by luck.

The second approach (the search for **orphan** optical transients) is difficult because GRB optical counterparts are relatively dim in a very cluttered optical sky.

1.2. Enter Raptor

Many systems (Rotse, Lotis, ...) are prepared to do rapid robotic response to high energy satellite triggers from HETE-2 and later on from Swift (both of these missions are dedicated to GRBs) as well as INTEGRAL and Agile.

Raptor (Vestrand et al. 2002) can do rapid robotic response, but it also has a *closed-loop self-triggering* system: it scans the optical sky (40×40 degree field of view) and triggers on potential orphan optical transients.

2. Details: How Raptor Tackles Science Goals

2.1. Hardware

Self-triggering requires wide field of view, thus each Raptor system has four 20×20 degree cameras (total of 40×40 degrees, 12.5th magnitude).

Optical self-triggering requires sifting through very many false transient events. Raptor uses both stereoscopic vision and intelligent back ends to sift through transients.

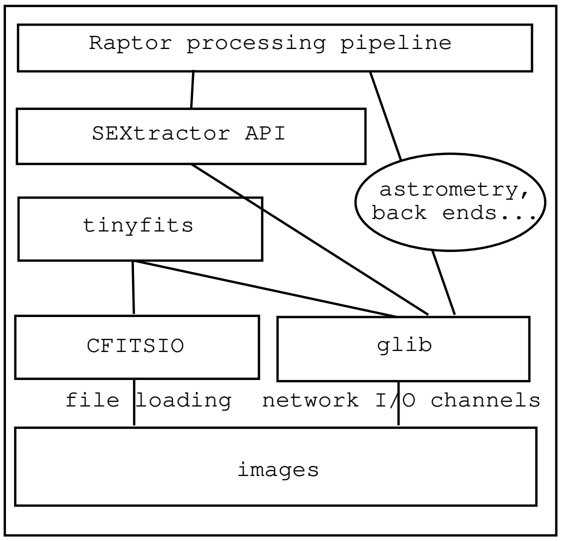

Figure 2. Raptor processing pipeline: the layering of the APIs.

Immediate follow up by deeper instruments is important. Raptor has a central *fovea* camera (4×4 degrees, 16th magnitude) which can slew to the desired location within seconds.

2.2. Software Pipeline

In designing the Raptor processing pipeline we face two main issues: software complexity and performance (images have to be processed and analyzed with intelligent back ends *in real time*).

Our main approach to both issues is through *abstraction* and *API design.* Each portion of the pipeline functionality is defined as an API (Application Programming Interface).

Programs are thin shells above the library APIs to move away from the classic clunky astronomy pipeline.

3. Abstractions

The main abstractions we use in the software pipeline are:

- Abstract memory resident images.
- API to manipulate these images provided by tinyfits library.
- Plugin architecture for image-feeding front-ends.
- Plugin architecture for smart and machine-learning back-ends.

4. Software Engineering

4.1. Approaches

We used a particular modern point of view in developing the infrastructure and programs for the Raptor pipeline.

The software is a collection of several small libraries, firmly based on free software, with all components released under the GPL. We follow the GNU coding standards and conventions throughout (standards compliance, configure/build, test suites). We use `glib` (from the Gtk+ toolkit) as a C Rosetta stone, offering some of the standardization that STL offers on top of the C++ standard library. A certain amount of eXtreme programming philosophy permeates our approaches (Beck 1999).

Development is currently hosted on `http://sourceforge.net` (although we might shift to the GNU project hosting sites).

4.2. The Tinyfits and SEXtractor APIs

The reference implementation of the FITS data format is the CFITSIO library. This library does not lend itself to the abstractions we used, especially API we used for SEXtractor. We implemented *tinyfits*, a very thin layer over CFITSIO.

Tinyfits provides the very few FITS file manipulation operations that are used almost all the time. It uses opaque data types for FITS images and tables, which allows it to be dropped in to other applications that have their own set of data types and abstractions.

One of the largest software efforts in the Raptor pipeline was to make the very widely used SEXtractor program (which extracts source lists from star images) work in a real time pipeline. Our modifications turn SEXtractor into an embeddable library which operates on memory-resident images and tables, which is crucial for performance and adaptability.

5. Some Results

Raptor is operational and searching for triggers, as well as responding to GCN alerts.

The versatility in our approach has allowed frequent rapid rewrites of the code (we consider this crucial in a significant software effort).

The use of APIs with thin application shells, which in turn makes obvious the use of memory-resident objects, gives Raptor the performance it needs to generate alerts in real time.

The principal timing improvements come from the faster SEXtractor runs and from tighter file I/O.

Acknowledgments. We are grateful to the entire Raptor team for the work in conceiving and bringing to life this project. We also gratefully acknowledge the internal Los Alamos LDRD grant which funded the project.

References

Akerlof, K. 1999, Nature, 398

Vestrand, T. et al. 2002, “The RAPTOR Experiment: A System for Monitoring the Optical Sky in Real Time”, astro-ph/0209300

Beck, K. 1999, Extreme Programming Explained: Embrace Change (Reading, MA: Addison-Wesley)