

An Automated System for Processing and Distributing SIRTf Telemetry Data

Elmain Martinez, Myche McAuley, and Alice Stanboli

*Multi-Mission Image Processing Laboratory (MIPL)
Jet Propulsion Laboratory, Pasadena, CA USA*

Some of the primary business objectives of JPL's Multi-mission Image Processing Lab (MIPL) are to process spacecraft telemetry and distribute the resulting data products to the science community in a timely fashion. Our recent development and operational experience with the Space Infrared Telescope Facility (SIRTf), which is part of NASA's Origins program designed to study the evolution of the universe, is the focus of this paper.

Our role in this mission began in the era of 'faster, better, cheaper', which required that we adhere to a strict schedule, and maximize developer productivity while minimizing development and operational costs. To meet these objectives we stepped out of the box and implemented a software system novel for MIPL. From the onset we decided that the operational concept needed to focus on autonomous operations, especially since operations costs for a 2.5-7 year mission would be substantial. But solely automating the system was insufficient given that 2GB of data had to be processed in 4 hours; any data processing problems would immediately overwhelm the operator. Therefore, to minimize costs and maximize operability, the software design focused on automated error recovery, performance, and information management.

With the automation goals identified, specific features came to light. For example, error identification and recovery, as part of any good design and programming practice, became critical for managing the telemetry on-board the spacecraft. Another feature that has proven to be key for autonomous operations is the real-time continuous service, which can go quiescent after periods of inactivity. This and a number of other features have contributed to various aspects of operating autonomously, but the design also had to integrate a number of project and ground system interface requirements.

These varied and sometimes orthogonal requirements, along with creeping requirements levied by the project, required that the software architecture be flexible and easily adaptable. In trying to minimize development costs, we leveraged off C++ code developed for the Deep Space 1 mission. But once development was well underway and we began supporting tests, we quickly ran into issues concerning thread and memory leaks that started to impact the development schedule. After much debate and concern about performance, we decided to abandon C++ and migrate everything to Java. This decision turned out to be one of the key factors that contributed to a flexible and adaptable design. This approach improved developer productivity and system performance.

Now that we have some operations experience under our belt, we have learned a few lessons. First, assume that multi-mission interfaces are not reliable and implement an automated recovery strategy.

Second, be sure to implement software that generates metrics – managers will want this. Lastly, we learned that designing an autonomous system for “management by exception” required that troubleshooting information and recovery tools become the operator’s keystone.