Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

OAPI: the OPUS Application Programming Interface

Walter Warren Miller III

Space Telescope Science Institute, SESD/DPT, 3700 San Martin Drive, Baltimore, Maryland, 21218

Abstract. The OPUS pipeline processing environment at the Space Telescope Science Institute (STScI) supports the automated conversion of gigabits of raw telemetry from the Hubble Space Telescope (HST) to useful science and engineering products each day. OPUS provides a flexible, fully concurrent, distributed control system in which a set of processes operate. The applications that form the HST data processing pipeline communicate with the blackboard architecture employed by OPUS through a library of C functions. Currently, that library is being ported to C++ as part of an effort to establish and document a standard programming interface to OPUS. The OAPI will facilitate OPUS application development for future HST instruments as well as other spacecraft and ground-based data processing projects.

1. OPUS: An Operational Pipeline System

The OPUS[1] pipeline processing environment at the Space Telescope Science Institute supports the automated conversion of gigabits of telemetry from the Hubble Space Telescope to useful science and engineering products each day. OPUS provides a flexible, fully concurrent, distributed control system in which a set of processes operate (Rose 1998; Rose et al. 1995; Rose, Choo, & Rose 1996; Boyer & Choo 1997; Swade & Rose 1998a). Applications in an OPUS pipeline specify, through simple text files, the criteria under which they are to be triggered by OPUS to perform some action. OPUS takes care of polling the necessary information sources (termed "blackboards") given the application's trigger conditions, and generating the appropriate events when those conditions are met.

The interaction between pipeline applications and OPUS differs depending on whether the application was developed for use with OPUS or is an off-the-shelf software product– OPUS does support both OPUS-savvy and off-the-shelf pipeline applications. In the latter case, a shell process acts as a proxy for communicating with OPUS by executing the "external" application when an event is delivered to it. With the release of the OPUS Sample Pipeline on CD-ROM last year (see Swade & Rose 1998b), it became possible for groups outside of STScI to construct complete OPUS-driven pipelines using external

[1] `http://www.stsci.edu/opus`

applications (numerous groups have done so or are in the process of developing OPUS pipelines on the supported platforms: Solaris, Digital Unix, and Linux).

Without a public application programming interface (API), however, it has not been possible for outside groups to develop software that communicates with OPUS directly (that is, through library function calls). The benefits of designing pipeline applications that communicate with OPUS through library function calls instead of through indirect means may prove critical for certain projects; for example, those where efficiency is a primary concern or where complex interaction with OPUS blackboards is required.

The Data Processing Team[2] recently defined as a high priority goal the full documentation of the programming interface to OPUS. Besides fostering a better environment in which to engineer OPUS-based applications, both in terms of development effort and software support, it provided us an opportunity to standardize the objects which form the core of OPUS in a way that improves the extensibility of the design itself. OPUS already has proven itself to be highly extensible in terms of the variety of pipelines that can be constructed with it (e.g., Rose et al. 1998), but this effort enabled us to abstract the existing object-oriented design in terms of C++ classes that can be used to extend the present capabilities of OPUS while maintaining a common interface to client applications. A well-documented API also makes it possible to offer the library to groups outside of STScI so that they may develop OPUS-savvy pipeline applications and enhanced libraries built upon the base classes included in the library.

2. OAPI Design

The OAPI design aims to satisfy the needs of two groups of software developers. On the one hand, it must serve the programmer who wants to develop OPUS-savvy processes for whom the implementation details of the OPUS library are not important. On the other hand, it must be easily maintainable and offer the flexibility to meet future requirements of OPUS pipelines through expansion or modification of the implementation underneath of the interface. Implicit to both of these needs is the availability of good documentation.

Standard C++ offered us the best means by which to meet the project goals both because of its support for object-oriented design and its ability to coexist with ANSI C at the translation unit level. In particular, since all of the existing software is written in ANSI C, it is trivial to offer the exact functionality for these routines in the OAPI. More importantly, however, is the time gained by not having to translate existing C code to a syntactically different language; this is time better spent enhancing existing algorithms to capitalize on C++'s improvements over C. The choice of C++ as the core programming language for the OAPI is reinforced by the recent adoption of the ANSI/ISO Standard for C++. Along with providing valuable new features to the language, notably the Standard Template Library (STL), developing the OAPI code to this standard

[2] `http://www.dpt.stsci.edu/`

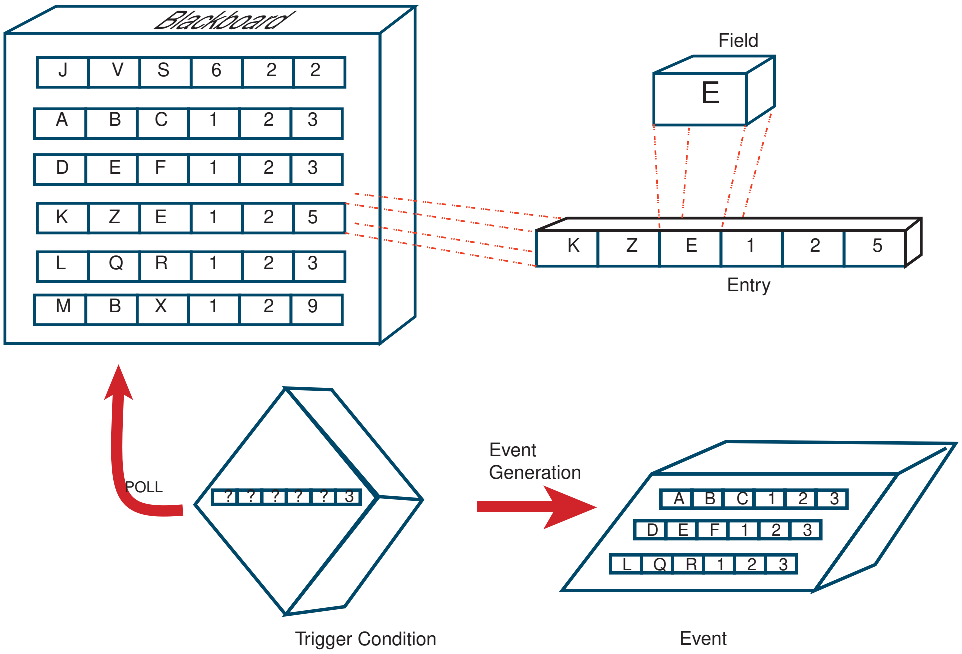

Figure 1. A cartoon representing the relationship between the basic objects in the OAPI. Blackboard objects contain Entry objects, which themselves are composed of Field objects. Polling a blackboard involves constructing a Trigger with an Entry condition, then searching with that condition on the appropriate blackboard. If matches are found, then an Event object is generated.

will ensure the same portability across different platforms as is enjoyed by the current ANSI C implementation.

The existing OPUS software modules are well developed, well tested, and offer a good degree of data hiding. Porting these packages to C++ purely for the added type checking and data hiding that C++ encourages is not a valid reason for undertaking such an effort. What is invaluable to this project is the match between the present software design and the preferred development environment of C++. Object-oriented design (OOD) tends to maximize the separation of interface from implementation. To a large extent, OOD concepts are employed by the original OPUS design and C implementation, but it is left to the developer to initially construct then maintain this philosophy throughout a large number of software packages. C allows the developer to build such a system, but it fails to enforce the rules of OOD and this leads to buggy software and maintenance difficulties. By porting OPUS software to standard C++, we adopt a system that enforces the rules under which we want to develop software. This makes it much easier to expand the capabilities of OPUS while preserving compatibility and to maintain existing software.

The essence of the OAPI design is reflected in a set of base classes that describe the objects that form the core of OPUS (see Fig. 1)– the Blackboard, Entry, and Field classes. More specialized classes are derived from these base classes to form objects that encapsulate different data structures or implement objects on different storage media. In general, access to derived types in the OAPI takes place through a pointer to the base class. Taking advantage of the polymorphic behavior of C++ class hierarchies in this way is a powerful tool to help preserve a high degree of separation between implementation and interface. Separating implementation from interface allows the use of generic algorithms to process different implementations of an object through a common interface thereby reducing code duplication and development effort. On a larger scale, it permits evolution of the OAPI with minimal impact on the clients of the library– in most cases, only a relink will be necessary to incorporate new features and enhancements to the OAPI in client software.

For example, current OPUS software uses the file system (specifically, file names) to store and communicate run-time state information for processes and data in the pipeline. Using the file system greatly simplifies interprocess communication and automatically leverages the robust, fault tolerant nature and near universal compatibility of NFS. However, it can sacrifice performance and extensibility in certain environments. Plans exist to implement new storage schemes for these blackboards in the near future– one using a database to store the entries and another to use network servers interconnected with CORBA. When such implementations become available, OAPI client code automatically will be able to use them without revising any of the algorithms.

Acknowledgments. OAPI design benefited from the insight provided by DPT staff Chris Heller, Jim Rose, Mike Swam, John Schultz, Mary Alice Rose, Daryl Swade, Pete Hyde, Steve Slowinski and Lisa Sherbert. Thanks also go to Allen Farris (STScI) for valuable help with C++ issues, and to Rosa Izela Diaz-Miller for production of the graphics in this article.

References

Boyer, C. & Choo, T. H. 1997, in ASP Conf. Ser., Vol. 125, Astronomical Data Analysis Software and Systems VI, ed. G. Hunt & H. E. Payne (San Francisco: ASP), 42

Rose, J. 1998, in ASP Conf. Ser., Vol. 145, Astronomical Data Analysis Software and Systems VII, ed. R. Albrecht, R. N. Hook, & H. A. Bushouse (San Francisco: ASP), 344

Rose, J., Choo, T. H., & Rose, M. A. 1996, in ASP Conf. Ser., Vol. 101, Astronomical Data Analysis Software and Systems V, ed. G. H. Jacoby & J. Barnes (San Francisco: ASP), 311

Rose, J., et al. 1995, in ASP Conf. Ser., Vol. 77, Astronomical Data Analysis Software and Systems IV, ed. R. A. Shaw, H. E. Payne, & J. J. E. Hayes (San Francisco: ASP), 429

Rose, J. F., Heller-Boyer, C. Rose, M. A., Swam, M., Miller, W., Kriss, G. A., Oegerle, W. R. 1998, in SPIE Proc., Vol. 3349, Observatory Operations to Optimize Scientific Return, ed. P. J. Quinn, (Bellingham: SPIE), 410

Swade, D. A. & Rose, J. 1998a, Proc. of the AIAA/USU Conf. on Small Satellites, (Logan: Utah State Univ.), in press

Swade, D. A. & Rose, J. F. 1998b, this volume, 111