11150

31970 p. 8

OOD/OOP Experience in the Science Operations Center part of the Ground System for X-ray Timing Explorer Mission

Abdur Rahim Choudhary
Hughes STX, Technology Applications Group, 7701 Greenbelt Road, Greenbelt, Md-20770, USA. (301-441-4229), rahim@rosserv.gsfc.nasa.gov

1.0 Introduction

The Science Operations Center (SOC) for the X-ray Timing Explorer (XTE) mission is an important component of the XTE ground system. Its mandate includes:

- Command and telemetry for the three XTE instruments, using CCSDS standards.
- Monitoring of the real-time science operations, reconfiguration of the experiment and the instruments, and real-time commanding to address the targets of opportunity (TOO) and alternate observations.
- Analysis, processing, and archival of the XTE telemetry, and the timely delivery of the data products to the principal investigator (PI) teams and the guest observers (GO).

The SOC has two major components: the science operations facility (SOF) that addresses the first two objectives stated above and the guest observer facility (GOF) that addresses the third. The SOF has subscribed to the object oriented design and implementation; while the GOF uses the traditional approach in order to take advantage of the existing software developed in support of previous missions.

This paper details the SOF development using the object oriented design (OOD), and its implementation using the object oriented programming (OOP) in C++ under Unix environment on client-server architecture using Sun workstations. It also illustrates how the object oriented (OO) and the traditional approaches coexist in SOF and GOF, the lessons learned, and how the OOD facilitated the distributed software development collaboratively by four different teams. Details are presented for the SOF system, its major subsystems, its interfaces with the rest of the XTE ground data system, and its design and implementation approaches.

2.0 Distributed Development

SOF development is distributed from following points of view:

- Development by a team with components distributed at Hughes STX and the three PI team locations at Goddard Space Flight Center (GSFC), University of California at san Diego (UCSD), and Massachusetts Institute of Technology (MIT). It also implies development under heterogeneous management structures, as each team component has its own management.
- Development on computer systems distributed at above team component locations, and internetworked using TCP/IP. This also includes development on heterogeneous types of machines.

SOF development uses the incremental build approach, with builds roughly six months apart. It employs the philosophy that the system software will be so modularized that the modules can be developed by the components of the team that has best expertise for them. Thus the software development related to a particular instrument is allocated to the corresponding PI team. These include the instrument health and safety, instrument commands, instrument telemetry unpacking algorithms, and algorithms to construct physically meaningful data partitions from the telemetry.

The rest of the system development is performed by Hughes STX. This includes the overall system engineering, development of abstract classes and base classes, integration of the total software system, testing of the system and the subsystem components, and integration of the SOF with the rest of the XTE ground system. The overall responsibility for the SOF remains with Hughes STX. This includes coordination with the various teams, clear definition of the development interfaces, and meeting the software build schedules.

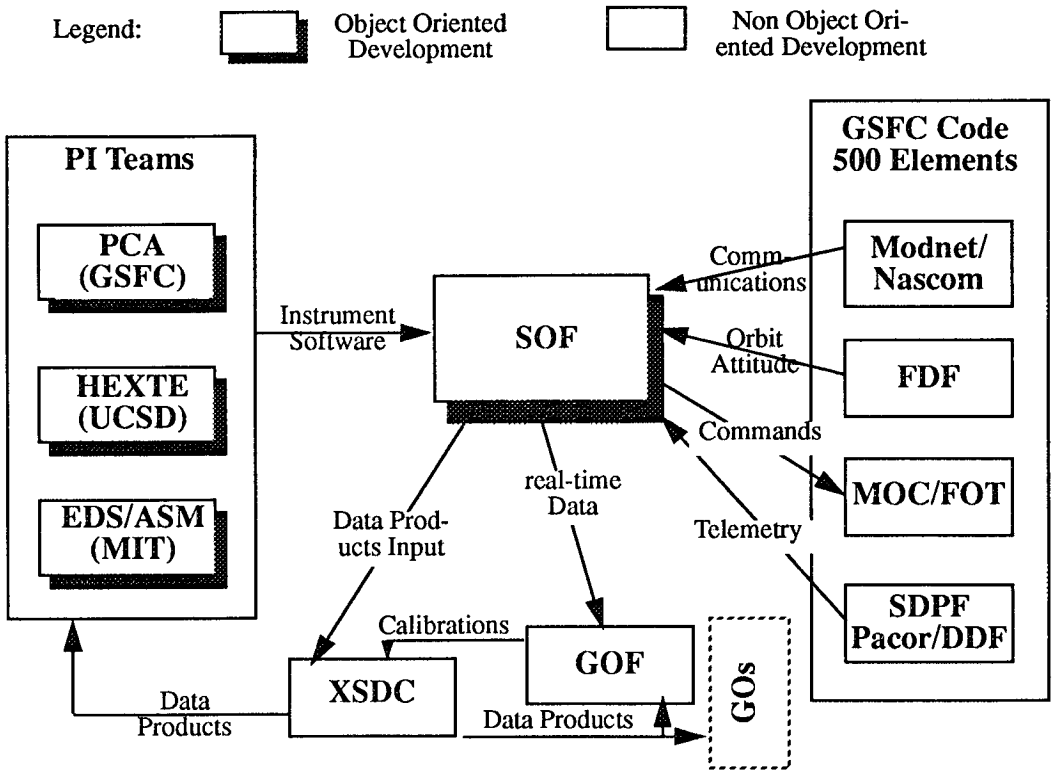

Figure 3-1: SOF interfaces and context within the XTE ground segment

3.0 SOF Context and Interfaces

Figure 3-1 shows the distributed parts of the SOF development effort together with the relationship of SOF with the rest of the XTE ground system. The SOF box shown in the center represents the net result of object oriented development by the PI teams and the Hughes STX. It has important interfaces with other ground system elements which are not object oriented. The GOF is not object oriented, but it needs to retrieve telemetry data

products from the SOF generated objects in order to generate the Flexible Image Transportation System (FITS) files. This interface is provided by data management subsystem of the SOF (see Fig. 4-1) that communicates with the XTE FITS Formatter software of the GOF using a set of data descriptors formulated according to a data description language (DDL) defined by the GOF for this purpose.

SubsystemConfig

This is the base class for the `DesiredConfig` and `PredictedConfig` classes.

Subclasses
`DesiredConfig`
`PredictedConfig`

Attributes
`RWCString configurationName;`
The configuration name.

`RWCString description;`
A descriptive string for the configuration.

Public Constructors
`SubsystemConfig ();`
Constructs a configuration with no description or configuration name.

`virtual ~SubsystemConfig ();`
Destructor.

Public Member Functions
`void setConfigurationName (const char* name);`
Sets the configuration name.

`const char* getConfigurationName() const;`
Returns a pointer to the configuration name.

`void setDescription (const char* name);`
Sets the descriptive text field.

`const char* getDescription() const;`
Returns a pointer to the description,

Virtual Member Functions
`virtual const char* getSubsystemName() const;`
Returns the name of the subsystem.

`virtual CommandScript* getCommandScript() const;`
Returns the command script.

`virtual TelemRate* getTelemRate (const Source& source) const;`
Returns a telemetry rate.

`virtual void printShort (ostream& ostr) const;`
Prints a description of the configuration.

`virtual void print (ostream& ostr) const;`
Prints a description of the configuration.

`virtual void printLong (ostream& ostr) const;`
Prints a description of the configuration.

Figure 3-2: An example of detailed class prototype from command generation subsystem.

The non Object Oriented interfaces are defined in traditional sense. All the data to be exchanged between SOF and an element of the ground system were identified; their formats were specified; the frequency and mode of each data transfer and the corresponding data volume was determined; and the standards to be adhered to were noted. A separate

ICD was concluded between SOF and the corresponding ground data element (as opposed to a single ICD between SOF and all other elements). This approach allowed the logistic complexities to be minimized and updates to these ICDs manageable by keeping the number of involved parties small.

The interfaces between the SOF and the components of the SOF to be developed by the PI teams were necessarily object oriented. The traditional methods for the interface treatment could not be employed in this case. To define the object oriented interfaces, first the class hierarchy was developed. The base classes were all allocated for development by the Hughes STX team. The subset of derived classes to be implemented by the PI teams were specified. The interfaces were defined in terms of the public member functions that these classes were required to support. As part of the interface definition, all such classes were prototyped; and those public member functions of each class were also prototyped upon which the other party depended for the implementation of their code. This set of prototype classes and public member functions were formulated early in the development and documented in an ICD. An example of such prototype class and its methods with their signatures is given in Fig. 3-2.

Separate ICDs were developed with each PI team. Further, the commonality between the ICDs with PI teams was explicitly acknowledged to facilitate their development, to avoid reinventing the parts already developed, and to manage the configuration of the common interfaces. This further facilitated the interface implementation, since the commonality explicitly formulated in the ICDs allowed the re-use of the corresponding software development approach among the PI teams.

4.0 Analysis and Design Approach

The book "Object-Oriented Modelling and Design" by Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorenson, W. (Prentice Hall 1991) was used by the SOF team to follow the Object Modeling Technique (OMT) advocated by these authors. The following procedure was found useful and worked for the SOF team, even though the various steps described below were often concurrently analyzed and subsequently refined via iterations.

1. The SOF team started with the usual requirements analysis. The requirements are sourced from the customer, domain experts, and the users.
2. The requirements were allocated to a set of high level functions. These functions were grouped into the subsystems, shown in Fig. 4-1. A lead engineer was appointed for each subsystem. The analysis described below was performed on subsystem basis.
3. The nouns used in the requirements allocated to a subsystem were potential objects. After the redundancy was weeded out and the overlap between the objects was minimized, the team had a fairly good starting set of the objects.
4. The associations between the objects can be indicated by the verbs in the requirements definition. This led to some objects being identified as the class attributes. The dynamic modelling scenarios were used to identify the objects that potentially form the member functions.The initial objects set was thus grouped into a set of classes, their attributes, and member functions.

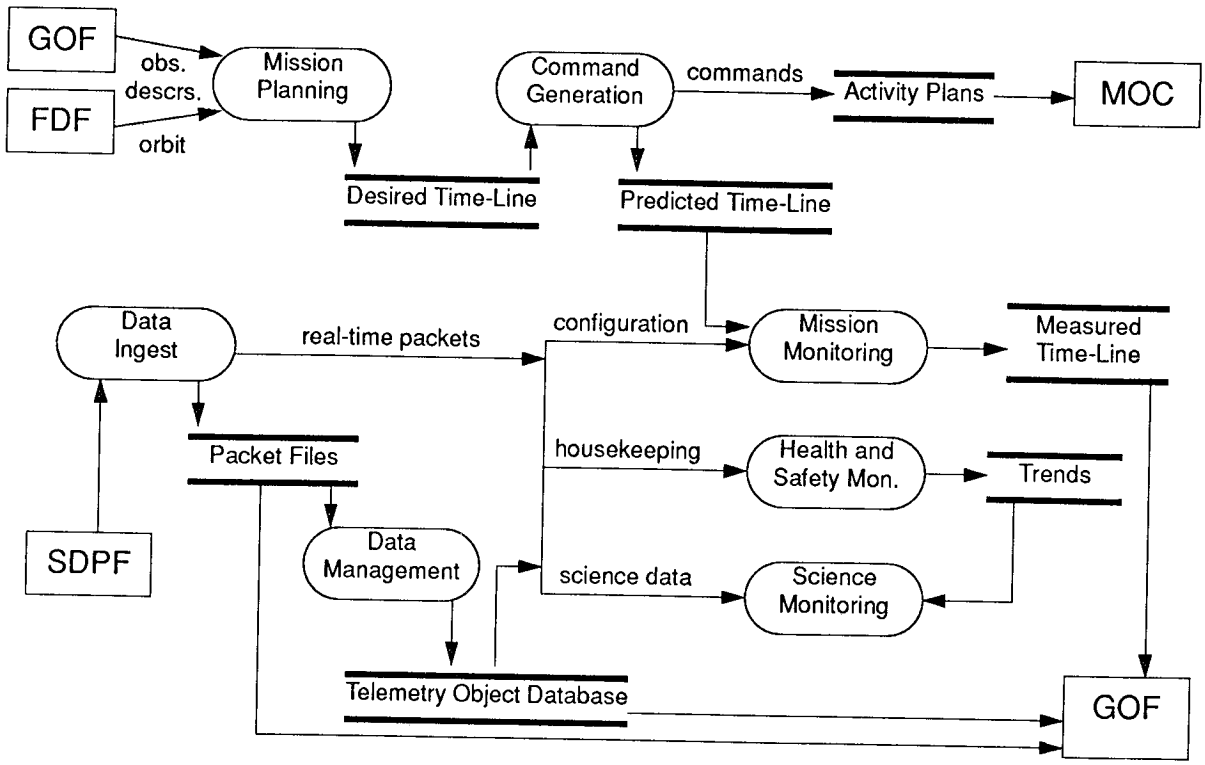

Figure 4-1: SOF software subsystems

5. A further analysis of these classes based on the bottom up and top down approaches was used to develop inheritance relationships between classes. The classes were then generalized to form the abstract classes; various specializations of which led to the derived classes. Some classes in each subsystem fell in the domain of expertise of the PI teams. Those were allocated for development by the PI teams. Such allocations however were not rigid so that they were reviewed as the design progressed and during the implementation phase of various builds.

Figure 4-2 shows an example of the object model for the command generation subsystem. The SOF design document has such object models for each subsystem and additional information as follows:

1. Subsystem introduction
2. Applicable requirements
3. Operating scenarios
4. Outstanding issues
5. Major design features
6. External interface
7. Subsystem interfaces
8. Subsystem object model
9. Subsystem class hierarchy
10. Detailed class design
11. Review comments and responses

The detailed class design is similar to the example presented in Fig. 3-2.

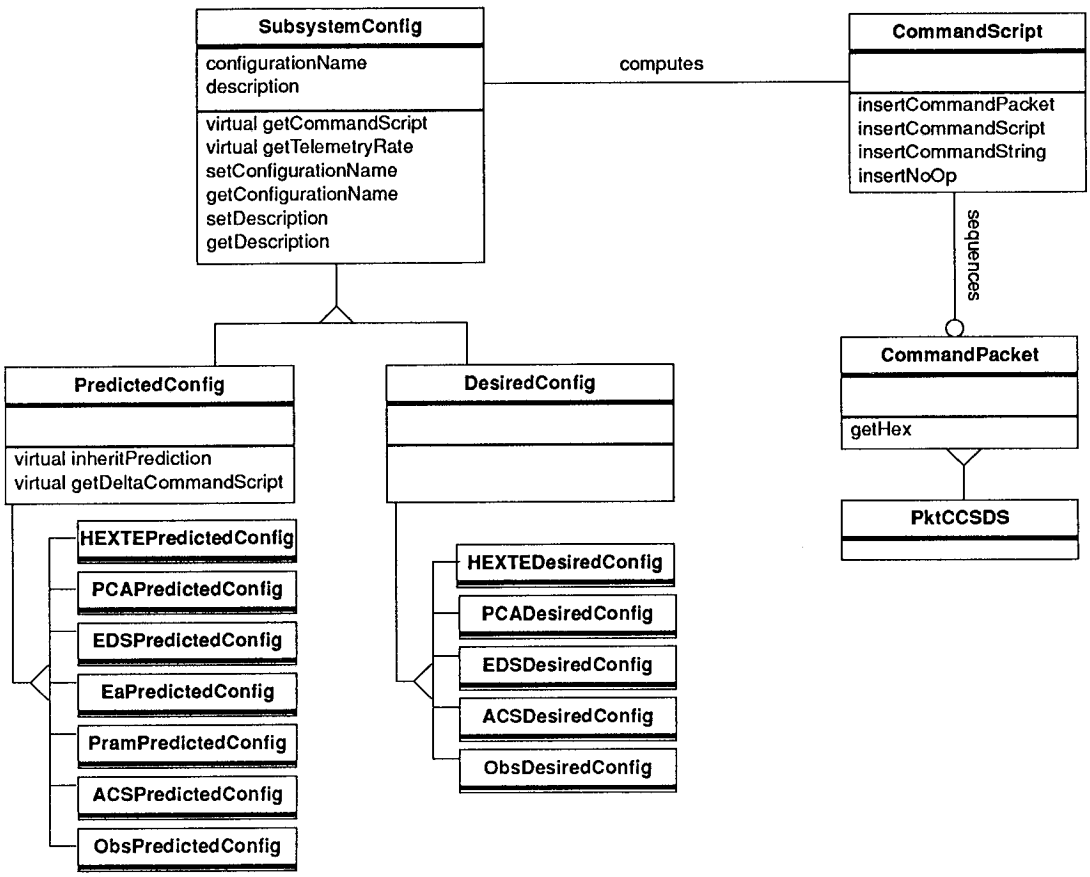

Figure 4-2: An example of Object Model from command generation subsystem

5.0 Development Environment

SOF decided for a client-server architecture using SunSparc workstations. However MIT wanted to use their existing DEC Ultrix workstations for their part of the SOF development. This meant that all development standards and the tools needed to be available on these two machines. To keep the development away from specific features of these two machines, a SGI IRIS Indigo was acquired to test that the software built on a third platform. The software development environment of the SOF are summarized in Table 5-1.

The internet connectivity between the computers on the four sites facilitated the distributed software development by the three PI teams and the Hughes STX. This allowed the developers to collaboratively debug problems on each others' computers using remote logon. It also allowed the periodic deliveries of the software and documentation from the PI teams to the Hughes STX for the SOF builds. Monthly meetings of all four components of the team were held. Other collaborations were ongoing using electronic mail. Each item in table 5-1 and all upgrades were discussed using these forums and kept in a standards document. Copies of all XTE SOF documents were available via an anonymous ftp account.

TABLE 5-1: Software development environment of SOF

Software Tool	Starting Version	Current Version	Comments
SunOS	4.1.3	4.1.3	Sun Operating System
Motif	1.1.4	1.2	GUI
ObjectCenter	2.0	2.0.6	C++ Debugger
CFront	3.0	3.0	AT&T C++ translator
Linpack/Lapack .h++	Linpack. 1.2	Lapack 1.0	Math.h++ and Maix.h++ supersets
xmgr	2.10	2.10	Oregon grad. inst. analysis package
RogueWave tools.h++	5.2	6.0	C++ library
RCS	5.6.0.1	5.6.0.1	Revision control system
TAE+	5.2	5.3	GUI Builder
X11	R5	R5	X Windows
FrameMaker	3.1x	4.0	Wordprocessor plus graphics
genman/genman++	2.0	2.0	Unix style man pages
GNU Make	3.68	3.70	make utility
Purify	2.1	2.1	check memory leaks/corruption
xteprob (home grown)	1.0	1.1	descrepancy tracking system

6.0 Object Persistence

Commercial object oriented data base management systems (OODBMS) were initially investigated for use in SOF. Ontos was selected for detailed evaluation. A pathfinder analysis showed that in the SOF context Ontos had several difficulties: presence of memory leaks, the performance limitations (SOF is required to ingest at an average rate of 64 kilo bits per second and a peak of one mega bits per second), and the fact that Ontos persistence mechanism required modifying those class library header files which must be persistent.

SOF's main data management needs are object persistence and persistent object retrieval. The more advanced features of an OODBMS such as sophisticated query capabilities or the transaction commit mechanisms are not required. The RogueWave (RW) Tools.h++ class library offers a means of making objects persistent. The UNIX file system together with the Dictionary classes in RW offer a means of accessing persistent objects; i.e. a way to simulate a global namespace. A prototype of the archival portion of the Ingest subsystem using Rogue Wave Tools.h++ was roughly ten times faster than the equivalent Ontos version. SOF therefore decided to develop internally the mechanisms it needed to satisfy many of its data management requirements.

7.0 Object Oriented Implementation

Some practical experiences during SOF implementation are presented in this section. The development was facilitated by early implementation of the object oriented interfaces. As can be seen from the example in Fig. 3-2, these interfaces were defined in terms of the method pro-

totypes in C++. The crucial parts of the code were therefore developed and scrutinized early in the process. As illustrated in Fig. 3-2, many of the interfaces were defined as virtual methods. This was very helpful in developing software with complete reliance on the PI teams for their instrument expertise and without the need for the Hughes STX engineers to also acquire such expertise. In fact the virtual method interfaces were often identically defined with each of the three PI teams; the specific instrument expertise were encapsulated in the way these interface methods were overridden by an individual PI team. At the same time the formulation presented a uniform interface to the Hughes STX engineers that were independent of the intricacies of the individual instrument subsystems. This approach is taken in many important instances including instrument configurations specification for command generation and mission monitoring, the telemetry unpacking to recover CCSDS packets, to assemble CCSDS packets into physically meaningful partitions, and to access that information from the persistent objects. This is a remarkable advantage of polymorphism in object oriented approach. The class hierarchy in such cases is illustrated in Fig. 4-2 for the case of instrument configurations. In this case the subclasses of PredictedConfig and DesiredConfig (except those for ACS and Obs) are developed by the PI teams while the rest are developed by the Hughes STX. The PI teams are free to derive their own sub-hierarchy.

The C++ templates were helpful. The real-time data ingest subsystem has a real-time server that passes CCSDS packets to the real-time clients. A real-time client template was developed that proved useful for the PI teams, the health and safety subsystem, the science monitoring subsystem, and the mission monitoring subsystem to write their own real-time clients.

The RW object oriented libraries of Tools.h++ proved very useful in saving the development effort on mundane things. The RW persistence and retrieval mechanism however was sometimes difficult for new developers to grasp.

8.0 Conclusion

Our OOD/OOP experience in SOF can be summarized as follows:

- Initial analysis and design activity took a while (the team was also passing through a learning phase); but the implementation proceeded pleasantly fast (couple of experienced C++ programmers later came on board, and the example of their work was helpful for the rest).
- Our decision not to use an OODBMS proved right.
- The COTS object oriented libraries saved SOF time and cost.
- Design changes due to the management decisions and requirements evolution were gracefully accommodated.
- The total SOF team is 11 persons, which is modest compared to similar past missions. XTE launch is scheduled for August 1995; the OOD/OOP approach has so far allowed SOF development on schedule and within cost.