Astronomical Data Analysis Software and Systems XIII
ASP Conference Series, Vol. 314, 2004
F. Ochsenbein, M. Allen, and D. Egret, eds.

Pre-processing the INTEGRAL telemetry

Nicolas Morisset [1], Tomaso Contessi [2], M. T. Meharga, Mathias Beck, Roland Walter

INTEGRAL Science Data Centre (ISDC)[3], Chemin d'Écogia 16, CH-1290 Versoix, Switzerland

Abstract. The acquisition, pre-processing, analysis, archiving and distribution of the INTEGRAL data are performed by the ISDC system at the Integral Science Data Centre (ISDC) in Versoix, Geneva. The **PreProcessing** sub-system is one of the most critical components of the ISDC system. It continuously processes the entire INTEGRAL raw telemetry, science and housekeeping data. It sorts, decomutes, decompresses and reorganises data into time slices providing the first ISDC data level. The PreProcessing component implements all requirements defined by the scientists and it is able to face all artifacts that may occur in the telemetry such as telemetry gaps, duplicate packets, corrupted data and clock reset. The originality of PreProcessing is its design. Indeed, the Object Oriented approach makes the program's core very flexible and independent of the INTEGRAL telemetry (reusable software). The convenience of such a conceptual model is the ease of implementation of any new type of telemetry packet with no change to the architecture. It is easily and quickly done by adding a new parser (inheritance and overloading concepts). The good results and performance obtained with the processing of INTEGRAL data encouraged us to reuse the same software for the PLANCK mission with a minimum of changes.

1. Introduction

A part of the work at ISDC was to develop the ground software. PreProcessing is one component of the ISDC system. It processes the raw telemetry in order to build the ISDC first data level in FITS format. At the conception phase, the telemetry format was not known. Its core has been designed in order to be as independent as possible of the telemetry format, flexible and resusable. Object Oriented design helps us to satisfy such requirements.

---

[1]Apco Technologies SA, Vevey, Switzerland

[2]Nuove Idee sas, Milano

[3]http://isdc.unige.ch/

2. PreProcessing Overview

Another ISDC sub-system provides "continuously" the Telemetry Files (see Figure 1) which contain all the raw packets (nearly 100 packet types) coming from the INTEGRAL satellite at the rate of 120 kbits/s. PreProcessing gets data either from near Real Time or from a set of CD-ROMs (consolidated data). The main tasks of the PreProcessing sub-system is to read these telemetry packets and to decode, sort, decompress and reorganize them into the Science Windows FITS files (ScWin Raw Files on Figure 1). A Science Window gathers a set of data taken during a certain period. PreProcessing produces 2.3 Gbytes/day and its speed is nearly 4.4 times the telemetry rate on a SUN V100 computer at 500MHz.

3. PreProcessing Core

The **PreProcessing core** consists of the main classes (written in C++) described in Figure 1. They will be presented briefly in the following sections. The PreProcessing core has been designed in order to be flexible and as independent of the telemetry as possible.

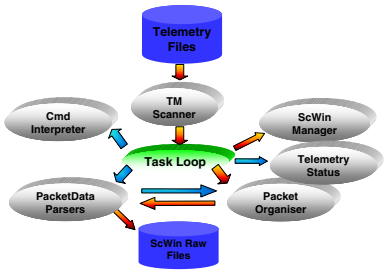

Figure 1. PreProcessing Core

3.1. TaskLoop Class

The **TaskLoop** class drives the activity of all the other components (see Figure 1). It gets a user command, a new packet or a new file ready to be processed. It does it by repeatedly calling the member function of the appropriate objects. If nothing happens during a loop, the TaskLoop sleeps for a while. Within each cycle the **Cmd Interpreter** (Command Interpreter) is called first. It looks for the Command File and takes the corresponding action, if any. The TaskLoop then calls the **TM Scanner** (Telemetry Scanner) claiming the next available packet. If there is one, a **Packet Object** is created and a pointer is returned to the TaskLoop. This pointer is passed to the **ScWin Manager** class and so on. TaskLoop will then activate all remaining components showed in the Figure 1, sequentially starting from the Cmd Interpreter up to the PacketDataParsers class.

3.2. TM Scanner Class

The TM Scanner class reads each **Telemetry File** created by the Data Receipt sub-system. It associates each telemetry packet to a **Packet** object. From this step, PreProcessing will manipulate only Packet objects. When the TM Scanner has completely loaded a Telemetry File or there is no Telemetry File available, it returns to the TaskLoop component and its task is over until the next activation by TaskLoop. PreProcessing is relatively independent from the Telemetry file format.

3.3. ScWin Manager Class

TaskLoop provides each Packet object available to the **ScWin Manager** Class. The ScWin Manager component has the difficult task of detecting a new Science Window from some information included in specific Packets. An INTEGRAL observation consists of many Science Windows. Data are reorganised into time slices named Science Windows. So if new Science Windows are detected, the ScWin Manager component creates them. This class uses a state machine based on the polymorphism concept.

3.4. Packet Organiser Class

TaskLoop provides each Packet object to the **Packet Organiser** class. The task of the Packet Organiser is to store data in the appropriate chain. The Packet Organiser consists of a set of chains of Packet objects. A chain is a set of Packets of the same type of data linked together based on on-board time from the oldest to the most recent Packet. This module also discards incorrect Packets like duplicates or old Packets. In fact, all bad Packets resulting from telemetry artifacts are removed.

3.5. PacketDataParser Class

TaskLoop activates the **PacketDataParsers** class to start the decoding of the Science or Housekeeping data. When PacketDataParsers is activated, it first requests the appropriated data to the **Packet Organiser** and provides them to the corresponding **Parser**, the **Code Driven Parser** for Science data or the **Data Driven Parser** for Housekeping data. After the Parser's call, the number of Packet Objects decoded will be returned to this class in order to clean up the correponding chain in the Packet Organiser class. A virtual function named **Parse** is redefined by the Data Driven Parser and the the Code Driven Parser classes for data decoding purposes. If a new type of packet is created, PreProcessing's structure will be not affected. A new Code Driven Parser class may be created, with no change to the core at all (see Figure 2).

3.6. Telemetry Status Class

All classes may access the **Telemetry Status** Class. This class logs messages of any types (warning, error, alert) to a log file giving the state of the processing at any point of the execution.

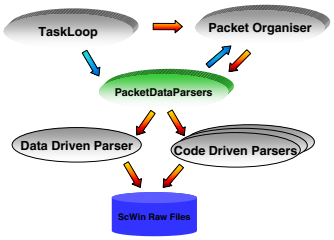

Figure 2. PacketDataParsers

4. Parsers

4.1. Data Driven Parser Class

As previously mentioned, the **Data Driven Parser** class decodes Housekeeping data using a C include file produced from a database which describes all Housekeeping packets with their parameters. Parameters coming from either cyclic Housekeeping or On-Request packet are stored in the same FITS table. Prior to this step that PreProcessing did not need to know the structure of the telemetry. This class decodes and reorganises all Housekeeping data for storage in FITS ScWin Raw files (see Figure 2).

4.2. Code Driven Parser Class

As explained in the above sections, the **Code Driven Parser** classes decode Science data or Housekeeping data if they are difficult to process in an automatically manner. A Code Driven Parser class is associated to every type of Science data. These classes decode, decompress and reorganises data for the storage in the FITS ScWin Raw files (see Figure 2).

5. Conclusion

The object oriented approach makes PreProcessing's core very flexible and independent of the INTEGRAL telemetry making the software reusable. The Packet object needs just a little information about the telemetry. The convenience of this concept is theease of implementation of new types of telemetry packets with no change at all to the architecture. This can be done quickly and easily by adding a new parser (inheritance and overloading concepts). It is also easier to maintain software. The good results and performance obtained with INTEGRAL data encouraged reuse of the same software for the PLANCK mission with a minimum of changes.

References

Beck, M. 2004 this volume, 436