APPLICATION OF XML TECHNOLOGIES TO TELEMETRY DATA MANAGEMENT IN TEST EQUIPMENT FOR SCIENTIFIC SATELLITE MISSIONS

Enrico Franceschi[1,2] *, Andrea Bulgarelli[1], Fabio Grandi[2], Fulvio Gianotti[1], and Massimo Trifoglio[1]

[1] IASF.BO, Space Astrophysics and Cosmic Physics Institute – Section of Bologna, CNR, Via P. Gobetti 101, I-40129 BOLOGNA, Italy
Email: franceschi@bo.iasf.cnr.it
[2] IEIIT.BO, Electronics and Engineering of Information and Telecommunication Institute – Section of Bologna, CNR and *DEIS, University of Bologna, Viale Risorgimento 2, I-40136 BOLOGNA, Italy*
Email: fgrandi@deis.unibo.it

ABSTRACT

The development of a space mission requires the implementation of several Test Equipments which simulate data flows for a specific payload. This paper describes a project involving the adoption of XML-related technologies in the management of those data flows arranged into telemetry packets. The project is mainly made up of three parts: the creation of an XML database containing the packet descriptors (requiring the definition of an XML Schema); the development of an interface between preexisting software modules and the new XML database; the implementation of a QuickLook module which processes packet data converted into the XML-based FITSML format developed at NASA/GSFC. The final result is a set of interacting software modules that implement a demonstrative but fully operational prototype.

Keywords: Metadata, XML, DTD, XML-Schema, XSLT, XPath, XDF, FITS, FITSML, Scientific Satellite Mission.

1 INTRODUCTION

Before the *flight ready status* can be achieved in scientific space missions, an extensive testing stage is needed to evaluate the functioning of the payload (i.e. detectors or instruments placed on a satellite) and also to investigate potential problems. The space mission for which the present work has been developed is called *AGILE* (*Astro–rivelatore Gamma a Immagini LEggero*: Light Imaging Detector for Gamma-Ray Astronomy; AGILE, n.d.). AGILE is an Italian Small Scientific Mission – currently planned to be operational in 2005 – devoted to high-energy astrophysics (Tavani, Barbiellini, Argan, Auricchio, Caraveo, Chen, et al., 2001): it will operate in the 30 MeV – 30 GeV range with imaging capabilities also in the 10–40 KeV range. The AGILE mission is supported by the ASI (Italian Space Agency) and scientifically developed in CNR (Italian National Research Council) and INFN (Italian National Institute for Nuclear Physics) laboratories.

The functional tests and the scientific calibration for the AGILE instruments (Trifoglio, Gianotti, Stephen, Celesti, Labanti & Traci, 2000) are carried out during the Assembly, Integration and Verification [AIV] stage by the workgroup at the IASF (the Space Astrophysics and Cosmic Physics Institute of the CNR), Section of Bologna (IASF.BO, n.d.).

The final configuration planned in the AIV phase is shown in Figure 1. The block on the left functionally corresponds to the payload, that is the four AGILE detectors, with their Data Handling System. In particular, the AGILE payload includes Super-Agile, Silicon Tracker, MiniCalorimeter

and Anticoincidence System detectors (Tavani et al., 2001; AGILE, n.d.). The payload is connected to the Spacecraft Interface Simulator in the Electrical Ground Support Equipment [EGSE] block through the same module that in the flight configuration will be the interface with the Ground Segment systems. In fact, at the AIV level, all the issues related to the in-flight data flow handling are ignored, and the data flows within the EGSE are instead simulated (with a TCP/IP channel, in the case of AGILE). These include the telemetry data [TM], which are the scientific and housekeeping data output from the satellite, and the telecommand data [TC], which are input to control the satellite. All these streams are organized according to the ESA telemetry and telecommand packet specifications (PTS, 1988; PTS, 1992). The Instrument Central Checkout [ICC] module has the task of sending the TC data (generated with the support of a specific database) and of receiving the TM data. The TC and TM data are then forwarded together, in order to preserve the mutual cause-effect

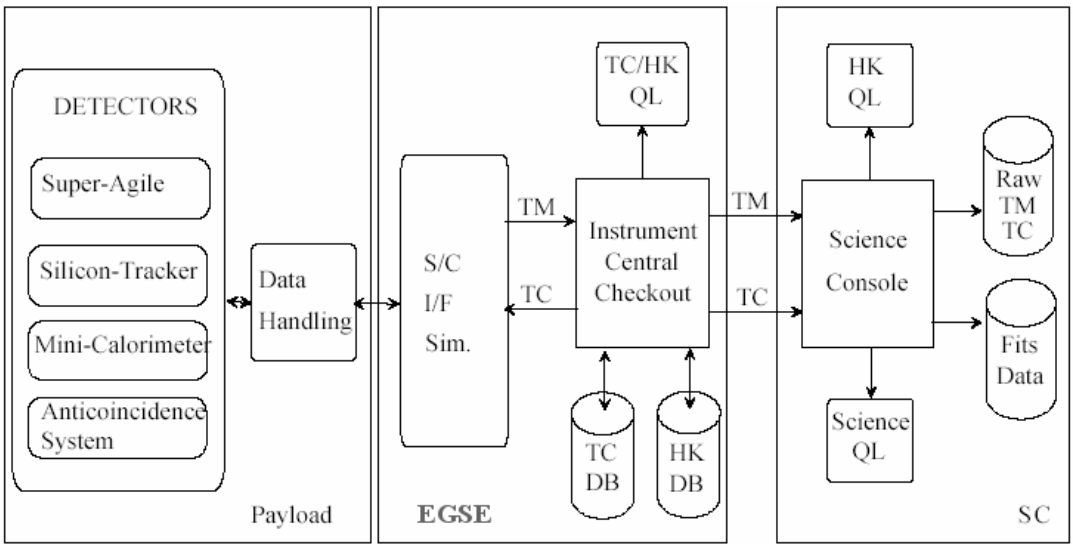

Figure 1. Assembly, Integration and Verification phase: the final testing configuration

connection, to the block which follows: the Science Console [SC], which receives and stores all the data exchanged both as raw data and in a suitable FITS format. FITS is the acronym of Flexible Image Transport System, that is a file format typically used to store and exchange astronomical data (FITS, n.d.). The ICC can also act as a monitor of all the service data by filtering the data flow (with the support of another database), where service data consist of TC and housekeeping [HK] packets. With respect to the space mission goals, HK packets have no scientific meaning: they may contain only information related to the state of health of a certain instrument or of the whole spacecraft. However, their scientific and engineering importance is crucial for the satellite design and running activities. The monitoring module, that operates in near real-time, is called QuickLook [QL]. In this framework, the term QuickLook is used to denote a monitoring action involving the display of log data, which has been generated by a computation based on acquired data. A QL module is intended to highlight particular features in a log section. Other specialized QL modules are also usually present in the SC to process housekeeping and scientific data.

During the test stage, in order to tune all its components, every payload module must be considered from different points of view: physical (e.g. mechanical, electrical, magnetic, thermal), functional, and also performance-related. To this purpose, a specific Test Equipment [TE] has to be set up to simulate the operational environment of each module in isolation and evaluate its behaviour, possibly with reference to a particular data flow. The layout of a generic TE (Figure 2) includes the tested module and a block that manages the data flows (here called the Host Computer). The data flows are then

forwarded to an external host where all the data are collected and processed, which plays the role of the SC here. The payload module, optionally solicited, is equipped with suitable front-end electronics and placed in an environment which simulates its real working conditions.

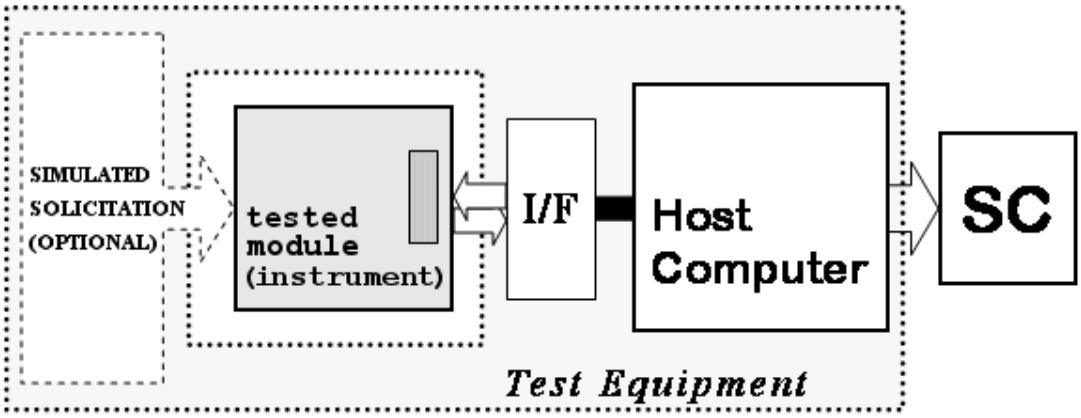

Figure 2. The layout of a generic Test Equipment

In this context, our project is concerned with the introduction of XML-related technologies (XML, n.d.) into the data management capabilities required of the SC, which is thus assumed to work with a *generic* TE. The final aim is the implementation of a QuickLook module for housekeeping data [HK-QL], which will specialize as a *Limit Checking* device that highlights out-of-range parameter values. The introduction of XML-related technologies is motivated by the adoption of an established standard for data processing and exchange on the Internet, which implies, for instance, the immediate remote availability of housekeeping data and their utilization by means of state-of-the-art Web browsers used as front-ends. This makes the sharing of scientific data between research groups easier and improves collaborative work between the different design teams involved in the space mission planning.

The rest of the paper is organized as follows. Section 2 contains an overview of the overall project scheme which has guided our design work. Section 3 is devoted to the design of the XML database for Packet Descriptors, whereas the design of the new output format required for the Packet Processor is described in Section 4. The redesign of the Packet Processor module is the subject of Section 5, while the implementation of the QuickLook module is described in Section 6. Conclusions can finally be found in Section 7.

2 THE OVERALL PROJECT SCHEME

In order to complete the design of the HK-QL module, several sub-modules deploying different technologies have to be set up at different stages of the packet management process. Moreover, also in order to reduce the design effort, one of the project specifications was the reuse of software modules (C++ libraries) developed in the context of previous space missions. As a result, the AGILE GSE Team at the IASF Section in Bologna has developed the final design scheme shown in Figure 3.

The shadowed box schematically represents all the preexisting modules: among these, the only one that needs amendments (viz. the addition of C++ classes corresponding to the new XML features) is the processor. All around the shadowed box, the Figure shows the new modules that have had to be specifically developed for this project. Notice that the grayed out ones are just anticipated possible add-ons.

The first component that we consider is the "Packet Descriptors database", which is an XML database that must contain the description of the structure (down to the bit level) of all the data packets that the system must be able to deal with. The component is shown as HK DB in the Figure, since the only type of packets we consider in this phase is HK data.

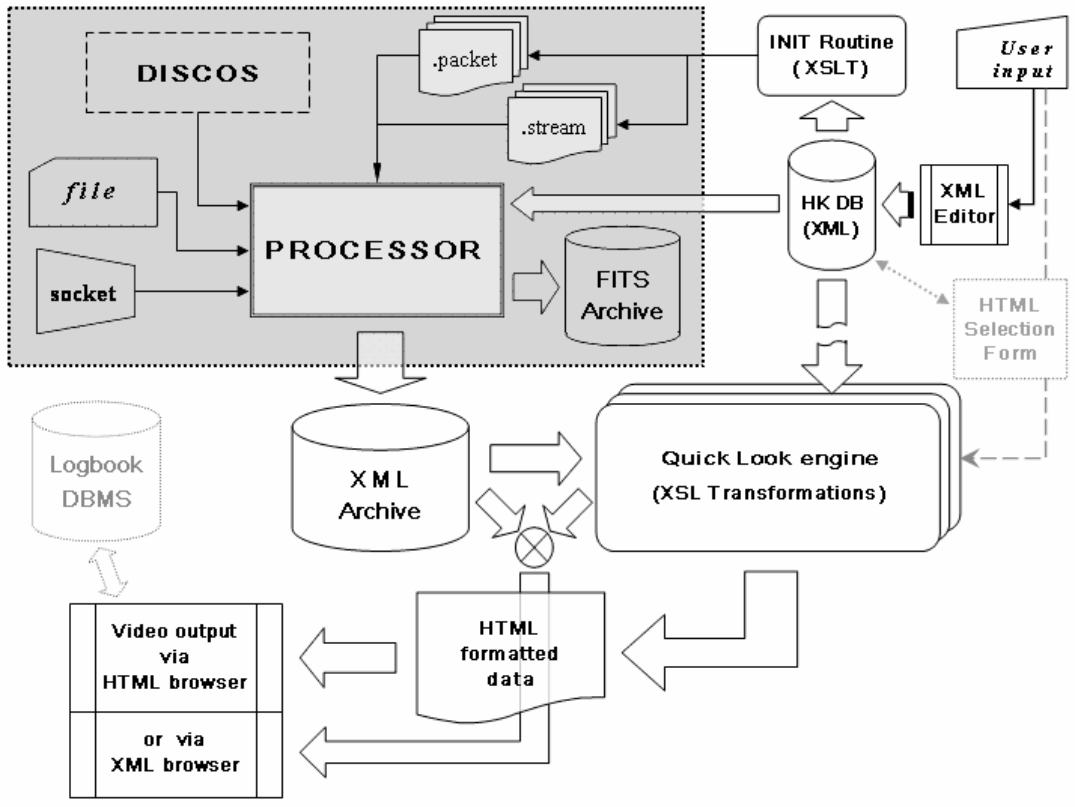

Figure 3. The project scheme (with the preexisting modules shown in the shadowed box)

Obviously, the correct processing of a packet relies on the presence in the database of a corresponding consistent description and of some additional information which is needed for the control of the data carried. Moreover, the design of the HK DB must be flexible enough to allow for different kinds of packets, also taking into account, for compatibility reasons, the generic packet layouts expected by the existing C++ libraries. After the schema design and implementation, the HK DB must be manually loaded with actual packet description data. To this end, a simple commercial XML editor has so far been used, but we are also considering the development of a suitable input module, customized for the special XML documents used, in the future.

The task of the second component we consider is the initialization of the acquisition system, which is done on the basis of the HK DB data. The initialization uses several configuration files (a pair of `.packet` and `.stream` files for each packet processor instance), which are generated by an initialization routine after querying the XML database to extract the required information. The initialization process could also be limited to a subset of all the available packets, relevant for the testing, which is selected by a human user from a list compiled via a preliminary HK database scan.

Thirdly, the processor module must be upgraded in order to produce output HK data formatted as suitable XML documents, containing all the information that was previously represented in the FITS files. Issues related to the *recalibration* of data (which is a form of data transformation, usually of the linear sort but occasionally of the non linear sort, used to decode telemetry data encoded before

satellite transmission) or advanced QL monitoring features (e.g. *delta checking* or *status checking* activities) have not been considered at this stage of the design. In fact, at a test stage, the *limit checking* action that the QL engine is required to perform can be done independently from recalibration, which can thus be dealt with in a future design phase.

The final task concerns the development of the QuickLook component. The QL module, which must monitor the HK data stream nearly in real-time, can be implemented by means of a standard Web browser, owing to a simple XML to HTML transformation. For the display of the HK parameters, a simple tabular layout has been judged sufficient to properly highlight out-of-range values, although a more sophisticated user interface could be adopted in the future. With this solution, the QL engine can easily be exported from a local host (the SC) to a remote and generic one. In this way, advanced querying and/or browsing facilities, including the possibility of selecting a particular subset of parameters and/or a specific temporal range, could also easily be integrated into a full-fledged QL or Logbook module. The Logbook module, which has not been implemented yet, will make it possible to analyze archived QL data, both in the off-line and the on-line mode, locally as well as remotely.

Although the adopted solution is currently aimed at monitoring HK data only, the developed infrastructure could easily be adapted in a future stage to process scientific data as well. Obviously, in such a case the importance of the developed infrastructure goes beyond the boundaries of the AGILE project, since the possibility of remotely accessing the data on the Web would be of great value to the whole scientific community.

3 THE PACKET DESCRIPTORS DATABASE

The packet descriptors database (hereinafter called HK DB) is, from a certain point of view, the core of the whole project. The initialization of a generic packet processor needs configuration files built based on the data stored in the HK DB. Once started, each specific processor built with reference to a particular TE must directly retrieve from the HK DB some support data which are used, along with the HK data, to feed the QL module. Finally, the QL engine must be aware of the HK DB contents to properly implement the required limit-checking feature.

The HK DB must provide an XML description of the structure of all the HK packets that the system is able to handle, in addition to information about the parameters that each of the packets may contain. Therefore, we have designed a specific XML document structure which is as flexible and reusable as possible, which can be easily shared by different modules, and which also allows users to input the (sometimes repetitive) description data in a quick and user friendly way.

The baseline adopted for the HK descriptive record corresponds to the (generic enough) set of fields provided for by the housekeeping documentation (IBIS.HK, n.d.) concerning the IBIS (Imager on Board of the INTEGRAL Satellite) instrument, developed for the INTEGRAL (INTErnational Gamma-Ray Astrophysics Laboratory) ESA mission with the long-term involvement of the IASF Section of Bologna. A sketch of this field sequence is shown in Figure 4, with a brief explanation of each item.

In order to define the structure of the required XML documents, we first considered the adoption of a DTD (Document Type Definition), according to the specification included in the current XML W3C Recommendation (v1.0, 2nd Edition). However, the DTD syntax (see "2.8 Prolog and Document Type Declaration" in Bray, Paoli, Sperberg-McQueen & Maler, 2000) is too poor from a semantic viewpoint and barely allows us to build a grammar that is a simple translation of the structure shown in Figure 4. Moreover, very few data types are supported in a DTD and these are too generic with respect to our application needs.

An initial extension of the DTD mechanism, which provides for a datatype declaration mechanism, is given by the DT4DTD (Datatypes for DTDs) specification (Buck, Goldfarb & Prescod, 2000). However, the DT4DTD approach is rather complicated, and must be seen mainly as a temporary solution

for extending already developed DTDs with datatypes, without requiring a great rewriting and re-structuring effort.

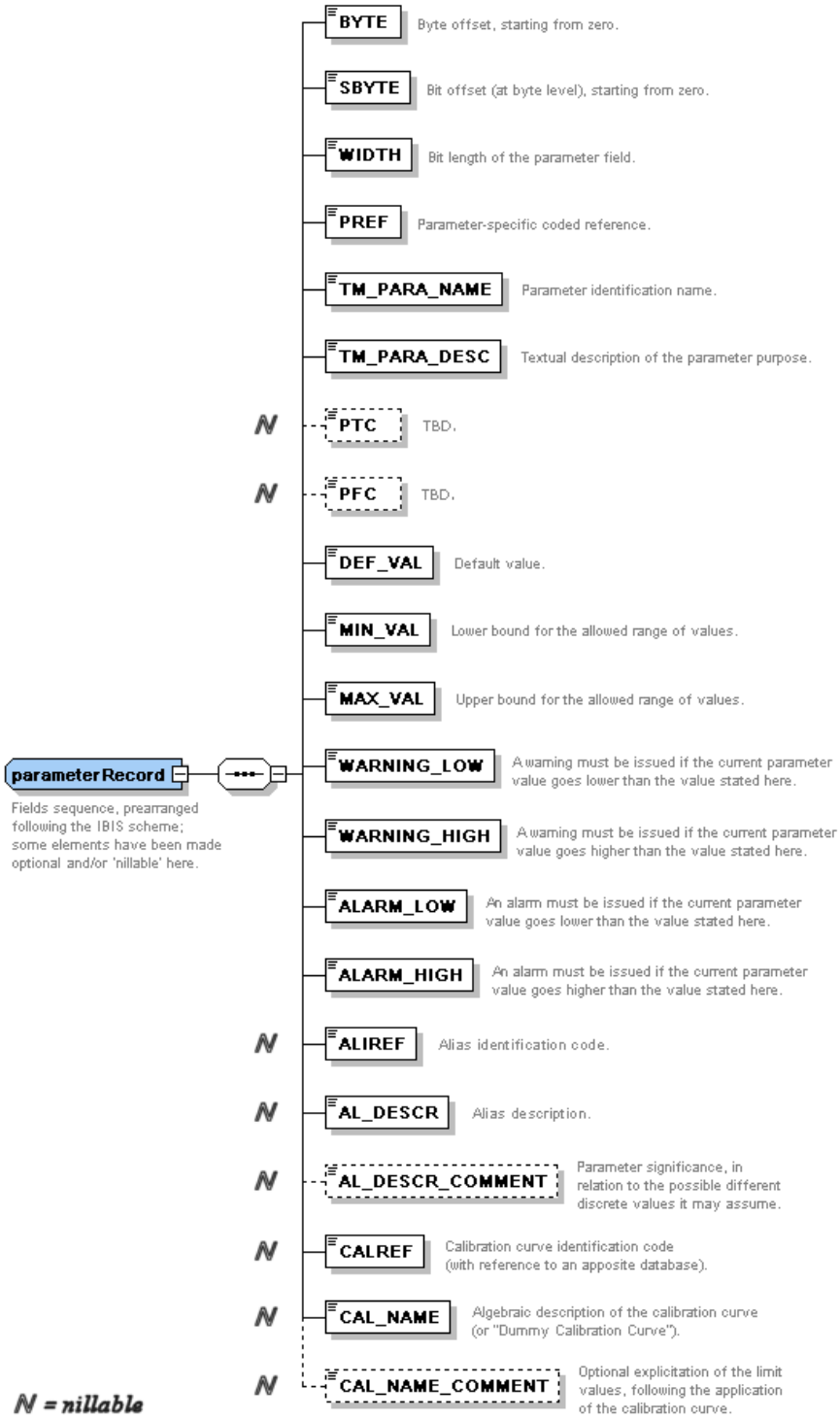

Figure 4. The `parameterRecord` datatype. A "nillable" element indicates an item which can have a null value, regardless of its type.

Looking for a more powerful way to describe the required data structures, we then considered two other alternatives: the XML-Data Reduced [XDR] language, and the XML-Schema Definition [XSD] language. The XDR language, which is essentially a subset of the ideas described in the former XML-Data specification (Layman, Jung, Maler, Thompson, Paoli, Tigue, et al., 1998), is not a W3C Recommendation but a proprietary standard proposal submitted to the W3C Consortium and promoted by Microsoft (XDR-doc, n.d.). In contrast, the XSD language is an established standard, endorsed by W3C since 1999, which has been designed to take into account XDR and other proposals. XSD supports, with an XML-like syntax, strong data typing when defining expressive and flexible XML document structures.

Hence, we decided to follow the XSD approach, according to the currently available final Recommendation dated May 2nd, 2001 (Fallside, 2001; Thompson, Beech, Maloney & Mendelsohn, 2001; Biron & Malhotra, 2001). Therefore, in our XML-Schema definition, we were actually able to profitably use several peculiarities of the XSD approach such as, for example, the possibility of specifying *identity-constraints* and defining structured datatypes inclusive of alternative elements.

An "identity-constraint" introduces into the definition of an XML-Schema (thanks to the use of XPath expressions; see section 3.11.1 in Thompson et al., 2001) a semantic tool which we have found useful because it enables a unique name to identify each packet description in the database at validation time. In this way, the selection of data related to each particular operating context becomes unambiguous, and the data retrieval process is made easier.

Furthermore, the uniqueness constraint already supported by XML through the ID and IDREF attribute types (see sec. 3.3.1 in Bray et al., 2000), included in the XML-Schema among the "derived datatypes" (see sec. 3.3.8 and 3.3.9 in Biron & Malhotra, 2001), allowed us to introduce a cross-referencing mechanism which we exploited, for instance, to avoid repetition when inputting descriptive data in the HK DB. Hence, in order to exclude wrong links between uncorrelated fields, the values of these attributes have been constrained to be not only unique but also to conform to a suitable specific "pattern", thus taking advantage of another XML-Schema peculiarity. Moreover, for a correct implementation, the cross-referencing mechanism must:

- not be available for elements with a null content;
- not interfere with the "nillability" concept;
- inhibit any explicit content definition, when a reference is used;
- ensure that the pointed element has a non-null content (unless exceptions are explicitly introduced, for specific cases).

In order to meet these requirements for all the elements for which the cross-referencing attributes were needed, we introduced a structured datatype that provides for alternative elements and thus makes it possible to define a database record with interchangeable fields. In particular, we decided to use the "element substitution groups" supported by XML-Schema, representing a structure that supports substitution of one named element for another, which is more powerful than a simple "choice" structure (see "2.2.2.2 Element Substitution Group" in Thompson et al., 2001). Figure 5 illustrates a pair of such structures, showing the relevant attributes.

Finally, after implementation we thoroughly tested the defined XML-Schema and, in particular, the cross-referencing mechanism, by storing the experimental HK packets specifications related to the TEs developed for the MiniCalorimeter Detector Front-End (MCAL-DFE) and the Super-AGILE Front-End Electronics (SA-FEE) in the HK DB.

3.1 The Packet Processor initialization routine

The software development group involved in this project has so far developed and used (for previous space missions, but also for the preliminary TEs of AGILE) packet processors based on a library of C++ classes written for Linux/Unix platforms. This solution requires two configuration files for each packet processor instance: the `.packet` and `.stream` files, which must strictly conform to a specific syntax corresponding to the C++ classes defined in the software library. In our new

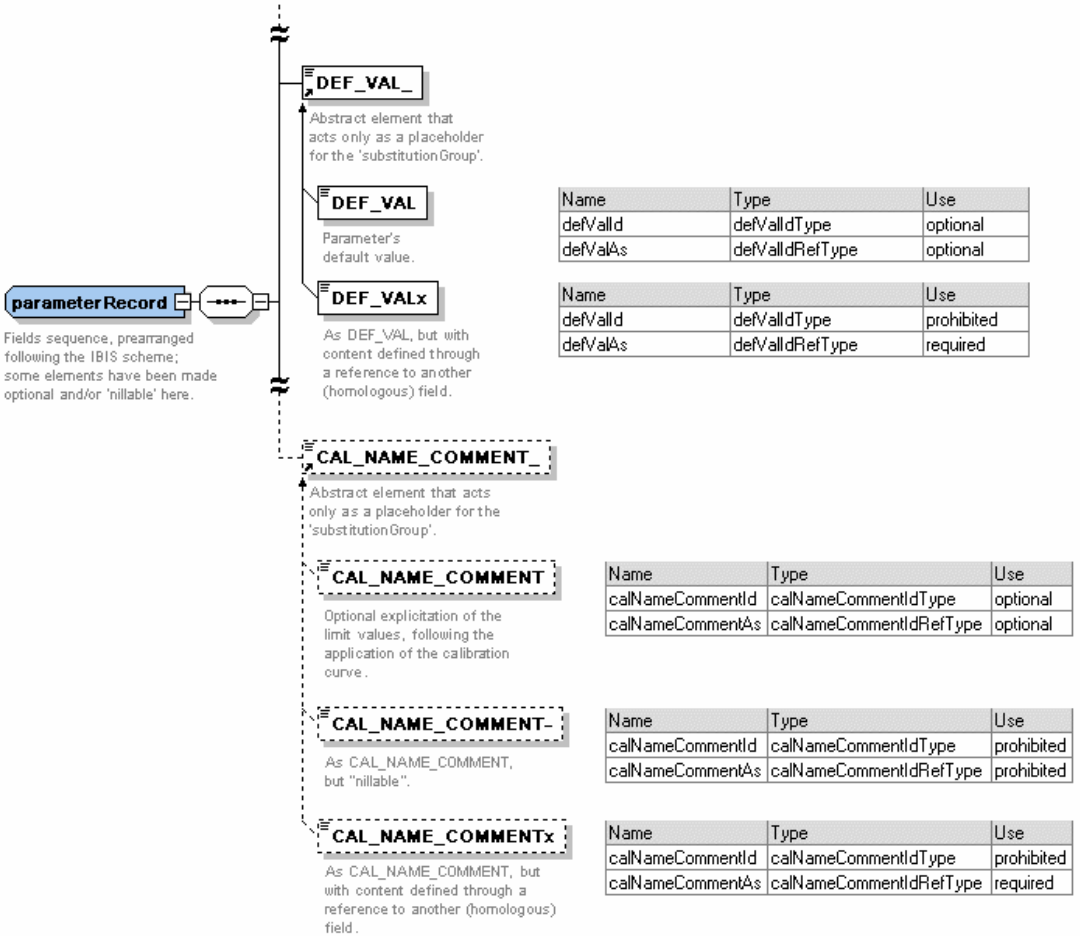

Figure 5. Two sample "substitution groups" (with the relevant cross-referencing attributes)

architecture, the initialization routine that generates the configuration files is quite a simple C program, called INIT routine in Figure 3. The initialization routine first retrieves the necessary information by querying the HK DB XML database, and then formats the retrieved data according to the required syntax before writing the configuration files.

Among the available XML querying technologies, we decided to use XSLT (Extensible Stylesheet Language Transformations) and XPath (XML Path Language) as query languages, in their 1.0 version published as a W3C Recommendation on November 16, 1999 (Clark & De Rose, 1999; Clark, 1999). Other options, like XSLT and XPath version 2.0 or XQuery (XML Query Language) v1.0, were discarded since though more "advanced" such languages are still (as of now, their last specifications are dated November 12th, 2003) at the *Working Draft* level and, thus, not stable enough. On the other hand, the consolidated XSLT v1.0 appeared to be sufficient for our needs.

The stylesheet we have thus developed is quite simple: it is basically made up of two nested loops (embodied in the two `<xsl:for-each>` XSLT elements), where the outer loop scans packet-related nodes of the XML document tree, and the inner loop scans parameter-related nodes (corresponding to the `parameterRecord`-type element of Figure 4). We only had to pay attention to the required ordering of output data, which must reflect the physical order of the parameters in the packet. An `<xsl:sort>` element (producing a primary and secondary order on the BYTE and SBYTE fields, respectively) has been added to the inner loop for this purpose. Finally, an `<xsl:choose>` element

has also been added to the outer loop to adapt the structure of the output configuration file to the classification of the processed packet layout, owing to some extra information available in the HK DB for each packet (at the same level as the parameters sequence).

In order to execute the XSL transformation, the INIT routine simply makes use of the Xalan executable taken from the Xalan/Xerces environment (Xalan-Xerces, n.d.). In particular, Xalan is an open-source high-performance XSL stylesheet processor, representing a really robust implementation of the W3C XSLT and XPath recommendations, supported by the "Apache XML Project". At the same time, the Xalan execution can also be used to make a validation check of the whole HK DB, which is particularly useful for implicitly testing the consistency of the internally used cross-reference mechanism. The transformation is carried out only once on the whole database, in order to avoid the overhead due to repeated processing of a rather large XML file. In such a way, the contents of all the required `.packet` files (one for each packet descriptor in the DB) are sequentially generated and appended to a single output file. Afterwards, individual `.packet` files can easily be extracted, using suitable delimiters inserted in the output file by the stylesheet.

The stylesheet also introduces many other extra lines (containing information taken, directly or indirectly, from the HK DB), which subsequently act as verification data: for example, during the XSL output file post-processing, a check on the completeness of the packet-descriptive data is actually carried out. If this check is successful, the current `.packet` file is generated, together with a provisional minimal `.stream` file, with the same name. Otherwise, the `.packet` file generation is aborted and the INIT routine ends with a message indicating the name of the corrupted packet description and the offset of the missing data. The checks performed by the INIT routine are not particularly sophisticated but they have been shown to be adequate to correctly detect all the error situations which may occur in this phase.

4 THE PACKET PROCESSOR OUTPUT FORMAT

The output data produced by the generic processor must also be in XML format. The question is which particular XML document structure should be used: a new one, created specially like the one used for the HK DB; or rather an already available one (with the necessary adjustments), that can be searched and taken somewhere else.

The experience gained in dealing with the packet descriptors showed that developing an XSD schema according to pre-defined format specifications could be a very time-consuming task. Moreover, the results may suffer from a lack of generality which eventually leads to difficulty of reuse. Therefore, it seemed advisable to choose the latter option, although even that choice is not risk-free. Basically, the main difficulty lies in finding an XML document structure that is really suitable for the application's purpose, that is able to adequately represent all (or nearly all) the information which was formerly kept in a FITS file.

As a preliminary candidate solution, our choice fell on XDF (eXtensible Data Format for scientific data – Shaya, Thomas & Cheung, 2001), which is a language designed by the XML Group of the NASA Goddard Space Flight Center [GSFC]. XDF is a mark-up language conceived as a reference standard for the representation of scientific data, which could allow an unrestrained exchange of information from one scientific field to any other. The XDF definition may then be "extended" and adapted for a specific context, in order to better fulfil the specific data representation and processing needs of a particular scientific field. However, XDF does not meet all the requirements of the ongoing project, mainly because it does not adequately support the representation of *metadata*, which are typically added to data files (e.g. as special headers in FITS files), and are strictly dependent on the application field as to form and content.

Actually, an XDF-based proposal for the representation of astronomic data already exists: the FITSML (FITS Mark-up Language – Thomas, Shaya & Cheung, 2001) language, which has been defined by the same group that developed XDF. FITSML is simply an XML version of the FITS

standard. It is not aimed at redefining the original specifications, but rather at *remapping* the FITS format onto the XML syntax, thanks to the inclusion of an XDF schema. The rationale of such a design choice is to avoid confusing the users, who for more than twenty years have been accustomed to the FITS format, and therefore favor migration to FITSML, as a potential new standard. The notes and definition files regarding the FITSML language can be found on the XDF home page (XDF, n.d.), in the section entitled "Markup Languages that Inherit from XDF". Hence, since FITSML can cover all our design requirements including the representation of metadata, our final choice fell on FITSML.

The version of FITSML we adopted is that labeled "v0.04", which is still under development. During our work, we have indeed found several inconsistencies between the proposed DTD and XSD correlated schemas. Moreover, the latter one, which was just introduced as a "Proposed Recommendation", was also discovered to be not fully compliant with the current XML-Schema specifications. Anyway, thanks to the cooperation of Edward J. Shaya of the GSFC XML Group who took into account our comments, a stable and consistent XML-Schema for the FITSML v0.04 has been released, along with a revised version 0.17 of the correlated XDF schema. The specification documents published by the GSFC XML Group on its web repository have also been accordingly updated and are now publicly available. The repository contains schemes, samples, and also some documentation, related to several languages that the GSFC XML Group has developed in the last few years (GSFC.XML, n.d.).

4.1 Mapping of the FITS format for use within the AGILE mission

Having settled the basic XML document structure needed to archive the data produced by the packet processors, we had to define a suitable mapping for the special extension of the FITS format which is used within the AGILE mission.

In general, a FITS file is made up of one or more Header and Data Units [HDUs], that is units made up of a descriptive text header optionally followed by a data block. Actually, only the first HDU – called *Primary Array* – is mandatory and its Data Unit, if present, complies with a given layout corresponding to an N-dimensional array of pixels, named IMAGE array. All the subsequent (optional) HDUs – called *extensions* – may include data blocks of several types, which are generally of three standard types: IMAGE, ASCII TABLE, and BINARY TABLE.

In practice, the Primary Array usually contains no data as it only serves as a global file header, while all the data are put in the subsequent extensions with the most suitable type (e.g. BINARY TABLE for HK data). Therefore, a FITS file can be schematized, for our purposes, with a sequence of three parts, as shown in Figure 6. The first two parts are headers, structured as sequences of fixed-length (80-byte) lines – called *Card Images* – containing information organized as < keyword, value, optional comment > tuples, whereas the third part is a stream of data in binary format. The former header (viz. Primary Array) contains general information whereas the latter, in particular, contains all the information which is necessary to reconstruct a logical tabular organization of the data contained in the third part, where each table column corresponds to a parameter, and each line represents a telemetry packet prefixed with a timestamp.

Although XDF supports the notion of binary data, which must be interpreted by a suitable *reader* (cf. the `binaryFile` Notation discussed in the "*white paper on XDF"*; "General XDF Documentation and Samples" in XDF, n.d.), we preferred to have raw data converted into a textual form (CDATA type) in order to be processed by an ordinary XSL stylesheet.

In this phase, we tested the translation process with some sample FITS files previously generated by an already developed packet processor, related to the Test Equipment for the AGILE's MCAL-DFE. An in-depth description of the MCAL instrument, and a discussion of several tests carried out on prototypes, can be found in Auricchio, Celesti, Di Cocco, Galli, Gianotti, Malaspina, et al. (2002). In the translation of the headers, all those FITS *keywords* that were judged redundant or useless for the new format, according to the notes at the end of the FITSML v0.04 XSD document (FITS, n.d.), were

rejected. Then, all the filtered information was marked up with XML tags in conformity with the FITSML format.

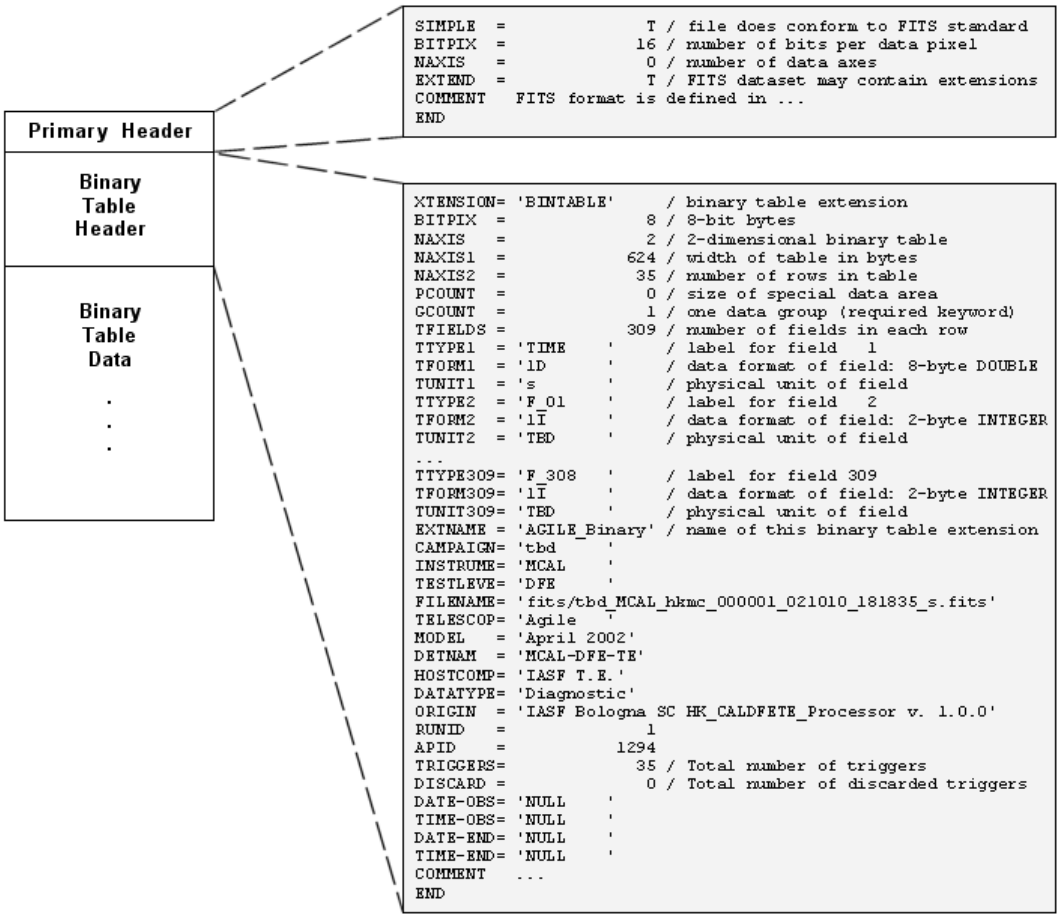

Figure 6. Sample FITS header (for the AGILE mission)

In general, the resulting FITS-FITSML mapping was non univocal, and was heavily influenced by the XDF structure that was chosen to represent the real data. We have also taken into account how the chosen solution could facilitate the QuickLook module that must subsequently process the resulting XML document. For instance, the way that the information extracted from the packet descriptors database is attached to the real processor output data, and their XML encoding, may influence the complexity of the XSLT code and of the XPath expressions needed to detect out-of-range values.

In particular, the selected organization complies with the suggestions given by the authors of FITSML, and aims to fully exploit the potential of XDF. In practice, the whole data set contained in the source FITS file is also organized in matrix form in the target FITSML document, inside a single `<array>` element. The `<array>` element includes a `<fieldAxis>` sub-element, which contains all the data-descriptive information (with a `<field>` element for each of the parameters), and a `<data>` subelement containing the actual data. Data rows, corresponding to packets, are delimited by `<d0>` … `</d0>` tags, whereas data items within rows, corresponding to parameter values, are delimited by `<d1>` … `</d1>` tags within a `<d0>` element. The exact correspondence between the FITS table data and the contents of the FITSML `<data>` structure is encoded by the "`read/tagToAxis`"

structure's contents (Shaya et al., 2001) in a rather complex way. For the sake of simplicity, we omit the technical details here.

As far as other information contained in the FITS headers is concerned, it is encoded by means of pre-defined tags with the same name as the keyword (if present in the FITSML definition) or by means of generic `<note>` elements having the original keyword as the value of the `mark` attribute. In this way, for instance, the value associated with the `ORIGIN` keyword can be referenced in XPath as "`note[@mark='ORIGIN']`" that denotes the value delimited by `<note mark='ORIGIN'> … </note>` tags in an XML document. Furthermore, we have to distinguish between the information concerning individual packets and the one that is a global constant for the instrument or for the whole mission. Hence, the corresponding `<note>` element is inserted, for the former, in the `<notes>` sub-element of the `<array>` element, or, for the latter, directly in the root element `<FITSML>` of the whole document.

```
SIMPLE  =           T      IGNORE   (*)
BITPIX  =          16      IGNORE, redundant   (*)
NAXIS   =           0      IGNORE   (*)
EXTEND  =           T      IGNORE, not needed in XDF   (*)
COMMENT   FITS (Flex...    -> /FITSML/note[@mark="COMMENT"]   (**)
END                        IGNORE, not needed in XML   (*)

XTENSION= 'BINTABLE'       IGNORE, not needed in XDF   (*)
BITPIX  =           8      IGNORE, redundant   (*)
NAXIS   =           2      IGNORE   (*)
NAXIS1  =         624      IGNORE   (*)
NAXIS2  =          35      IGNORE   (*)
PCOUNT  =           0      IGNORE, not needed in XML   (*)
GCOUNT  =           1      IGNORE, not needed in XML   (*)
TFIELDS =         309      IGNORE, no longer needed   (*)
TTYPE1  = 'TIME    '       -> /FITSML/array/fieldAxis/field/@name
TFORM1  = '1D      '       -> /FITSML/array/fieldAxis/field/dataFormat
TUNIT1  = 's       '       -> /FITSML/array/fieldAxis/field/units
TTYPE2  = 'F_01    '       -> /FITSML/array/fieldAxis/field/@name
TFORM2  = '1I      '       -> /FITSML/array/fieldAxis/field/dataFormat
TUNIT2  = 'TBD     '       -> /FITSML/array/fieldAxis/field/units

(--(--(-data-)--)--)       -> /FITSML/array/data/d0/dl

EXTNAME = 'AGILE_Binary'   -> /FITSML/array/@name
CAMPAIGN= 'tbd     '       -> /FITSML/array/notes/note[@mark="CAMPAIGN"]
INSTRUME= 'MCAL    '       -> /FITSML/array/observation/instrument
TESTLEVE= 'DFE     '       -> /FITSML/array/notes/note[@mark="TESTLEVE"]
FILENAME= 'fits/tbd_M...   -> /FITSML/array/notes/note[@mark="FILENAME"]
TELESCOP= 'Agile   '       -> /FITSML/array/observation/telescope
MODEL   = 'April 2002'     -> /FITSML/array/notes/note[@mark="MODEL"]
DETNAM  = 'MCAL-DFE-TE'    -> /FITSML/array/notes/note[@mark="DETNAM"]
HOSTCOMP= 'IASF T.E.'      -> /FITSML/array/notes/note[@mark="HOSTCOMP"]
DATATYPE= 'Diagnostic'     -> /FITSML/array/notes/note[@mark="DATATYPE"]
ORIGIN  = 'IASF Bolog...   -> /FITSML/array/observation/history/origin
RUNID   =           1      -> /FITSML/array/notes/note[@mark="RUNID"]
APID    =        1294      -> /FITSML/array/notes/note[@mark="APID"]
TRIGGERS=          35      -> /FITSML/array/notes/note[@mark="TRIGGERS"]
DISCARD =           0      -> /FITSML/array/notes/note[@mark="DISCARD"]
DATE-OBS= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionDate[@keyword="DATE-OBS"]
TIME-OBS= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionTime[@type="start"]
DATE-END= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionDate[@keyword="DATE-END"]
TIME-END= 'NULL    '       -> /FITSML/observation/datesAndTimes/observa-
                              tionTime[@type="end"]
COMMENT   ...              -> /FITSML/array/notes/note[@mark="COMMENT"]
END                        IGNORE, not needed in XML   (*)

 (*) = see the notes at the end of the XSD file related to FITSML v0.04
(**) = with an equivalent characters string introducing the FITSML format
```

Figure 7. The FITS to FITSML mapping adopted in the AGILE context

The resulting FITS to FITSML mapping is summarized in Figure 7. A preliminary test of the mapping was eventually carried out by manually deriving FITSML documents from some sample FITS files and then checking their validation successfully. Once the data output specifications had been established, we started the development of the planned demo processor, which must implement the

XML database query facilities and also support the XML format for the data output. The development required a partial redesigning of the code of a pre-existing packet processor, based on the PacketLib and ProcessorLib libraries.

5 THE PACKET PROCESSOR MODULE

In this section, we describe the amendments effected on the Packet Processor module source code in order to support the XML format.

5.1 PacketLib and ProcessorLib: a quick overview

The PacketLib (Bulgarelli, Gianotti & Trifoglio, 2003a) is a C++ open-source middleware library developed for Linux/Unix platforms with the aim of providing a common reference for applications that have to manage telemetry source packets that are compliant with the ESA Telemetry and Telecommand Standards. Its main purpose is to allow a strategic software reuse and, consequently, a faster development of TEs and EGSE applications based on an object-oriented paradigm.

The classes of this library give the programmer an abstract representation of the packets, which are further classified, in the current release, into three different typologies. Owing to this abstraction, the upper layer (see Figure 8) can then ignore the byte stream issues and provide the high-level telemetry management functions required by the application layer.

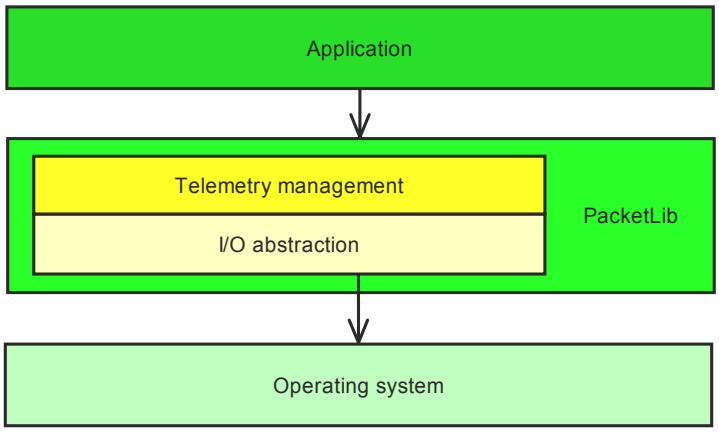

Figure 8. Abstraction layers provided by the PacketLib C++ library

The ProcessorLib (Bulgarelli, Gianotti & Trifoglio, 2003b) is the first example of an application based on the framework given by the PacketLib. It contemplates as input three packet sources, file, socket, and shared memory (see the outline in Figure 3), that are then considered as data providers in the subsequent packet processing stage. Shared memory is the mechanism used by another preexisting module, called DISCoS (see Gianotti & Trifoglio, 2001), to externally provide the data. The selection of the input packet stream used for each specific processor is made on the basis of an associated configuration file (i.e. the `.processor` file). The functional organization of the ProcessorLib package is illustrated in Figure 9. The classes defined in the ProcessorLib implement both the general purpose functions shared by all the processors (part A in the diagram in Figure 9) and all the elements needed to customize them (part B), according to the particular instrument and data flow to be processed.

5.2 Upgrading the Packet Processor module

The ProcessorLib was not designed to manage input or output data in XML format. Therefore, in our project, the ProcessorLib package had to be upgraded with the addition of new classes devoted to XML data management, in order to extract data from an external XML database and to produce output data in FITSML format.

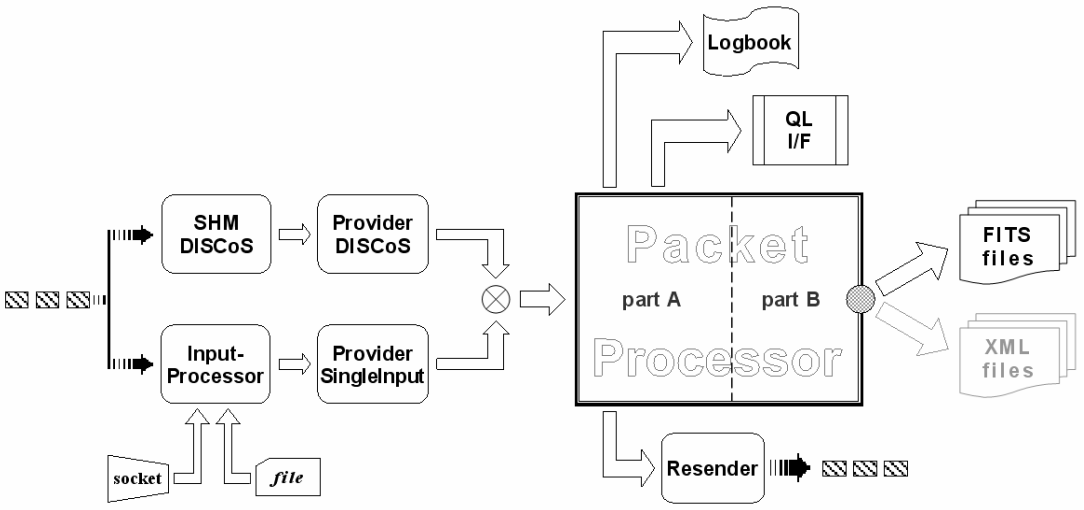

Figure 9. The functional scheme of ProcessorLib

The ProcessorLib upgrade was carried out starting with the particular case of HKCALDFETE, that is the TE related to the HK data flow coming from the above-mentioned MCAL-DFE. In particular, such an upgrade involved the revision of two classes making up the beta version of the previous test processor. These classes are the `HKCALDFETEProcessor` class – which originates from the `Processor` class (see Figure 10) defined in the ProcessorLib (Bulgarelli et al., 2003b) – and the `HKCALDFETE_FITS` class. Although the latter has been renamed `HKCALDFETE_FITSML`, it remains an extension of the abstract class `FITSBinaryTable`, which is actually generic and not conditioned by the adoption of the FITS format.

In order to speed up the upgrade process, we decided to simply add the new functions rather than totally revise the processor software design. Nevertheless, some improvements at the source code level have been introduced as well. First of all, the `MCAL_HK_PACKET` structure, representing the HK packet considered, has been simplified in order to make the packet data acquisition and the output generation processes (implemented by the `setValue()` and `writeData()` methods) more efficient. Then, the arrays used to implement the communication channel between the two classes were totally redesigned, in order to include in the generated XML output all the packet field names retrieved from the `.packet` configuration file, which can easily be accessed through suitable methods and properties of the PacketLib classes. Incidentally, the software fault-tolerance was also improved by means of more extensive exception handling, which also covers all the new FITSML output generation features.

The FITSML-related methods (easily recognizable by the prefix in Figure 10) have been built in order to encapsulate only atomic events, so that they could easily be identified and used (and also, in the future, reused). In order to execute the XML queries needed to retrieve support data from the HK DB, we decided to use two open-source C++ APIs: Xalan-C++ (version 1.4.0) and Xerces-C++ (version 2.1.0), from the already mentioned Xalan/Xerces platform (Xalan-Xerces, n.d.). These libraries currently seem the best solution – at least on a Linux machine – for parsing, validating and transforming XML documents.

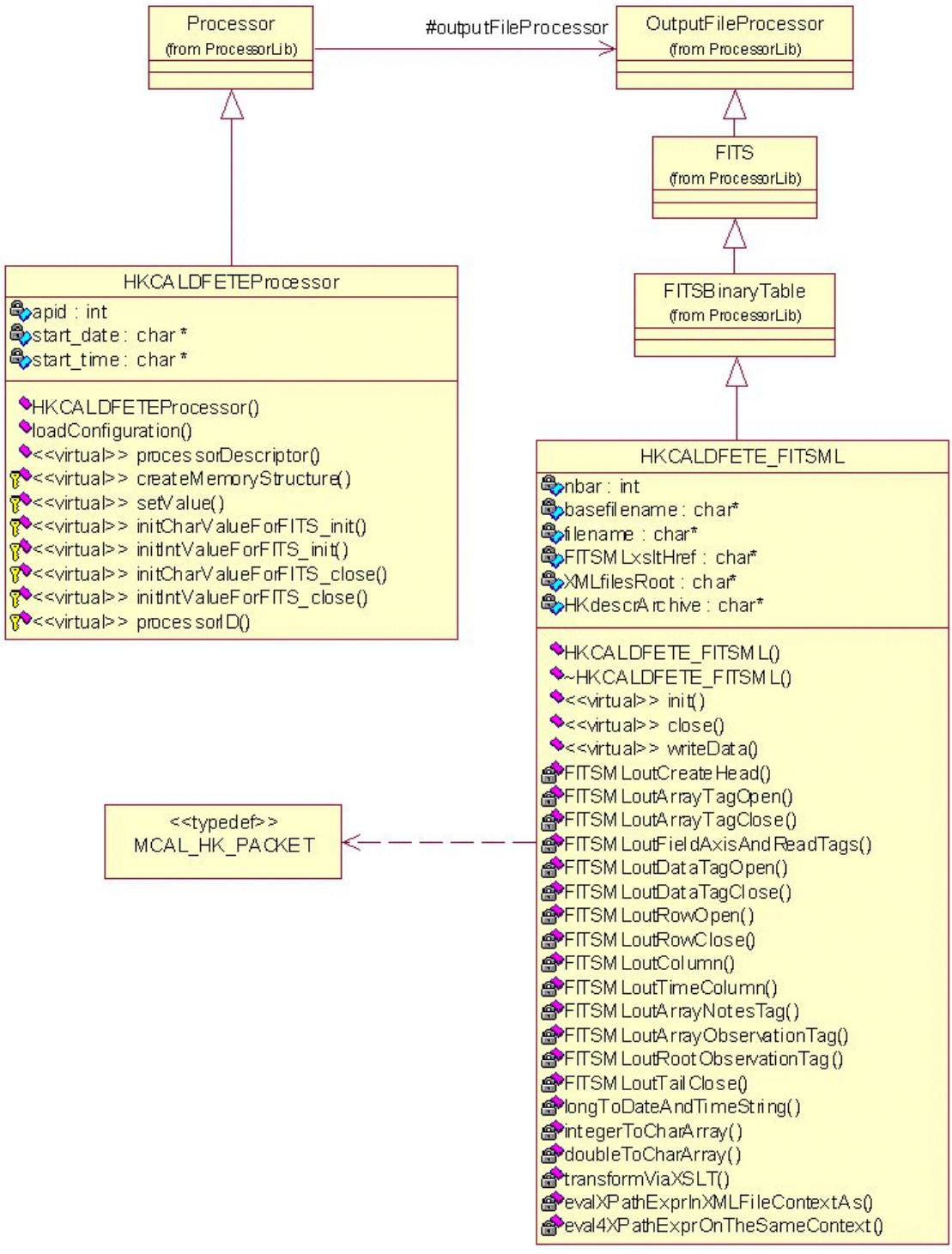

Figure 10. The HKCALDFETE classes

The methods that we implemented using the Xalan/Xerces APIs are the last two listed in Figure 10. They both provide (as character strings) the desired data, given a database name, an XPath expression, and a context in which to evaluate it. The second method allows up to four queries to be issued at the same time on the same context, in order to obtain – with a negligible programming

overhead (i.e. replication of source code lines) – an important performance improvement. Actually, verification of the context consistency is a time-consuming activity, whose execution time increases rapidly with the database size. Since the packet descriptors database easily grows very quickly, this can become a problem. Further optimization, working in a more general case, could most likely be obtained by executing the required parsing only once per XML document (generating an appropriate `XalanParsedSource` object) and then using its binary representation for further processing.

As to the inherited methods (i.e. `init()`, `writeData()`, and `close()` in Figure 10), they had to be completely redesigned. The `init()` method, formerly devoted to producing the Primary Header and the Binary Table Header of the FITS file (see Figure 6), is now used to generate the first portion of the new `.xml` file, containing the common XML header processing instructions, the root tag and, inside the `<array>` container, the `<fieldAxis>` with the related `<read>` elements. In order to compile the table of the parameter-descriptive data within the `<fieldAxis>` element, repeated calls to one of the above-mentioned methods using the Xalan/Xerces APIs are used. Each call requires the construction of a suitable context string, where special care is needed for the specification of correct *namespace* prefixes. The `writeData()` method feeds, as before, the real data, including the associated timestamps, into the output file. However, its new code is more compact, thanks to only a single loop being needed to scan the simplified packet representation used. The `close()` method, finally, has become much more significant than before, also because of the changes (requiring header fields at the end of the FITSML files) introduced by the GSFC XML Group in the FITSML format due to the adoption of the XML-Schema in place of the DTD previously used.

6 THE QUICKLOOK MODULE

The HK-QL module, specialized as a *Limit Checking* device, has simply been implemented by means of an XSLT stylesheet (Clark, 1999) that effects an XML to HTML transformation. The XSLT transformation produces a tabular output of the HK data, which highlights the out-of-range parameter values by means of different colours.

We briefly outline here the design of XPath expressions (Clark & De Rose, 1999) which have been used to connect the parameter values with their admissible bounds during the stylesheet application. The implemented solution relies on the fact that the order imposed on the configuration data appended to the `.packet` file by the initialization routine is the same as that for the parameters processed by the Packet Processor module. As a consequence, within FITSML files, the limit values of the different parameters are stored in the same order as their data in each of the packets, regardless of their actual physical order in the HK DB.

Hence, the XPath expression needed to establish the connection between the data and the corresponding boundaries can safely rely on the one-to-one correspondence between the position of the description of each parameter (`<field>` element) within the `<fieldAxis>` element and the position of its values (`<d1>` element) within all the `<d0>` elements. For instance, when processing the *i-th* value in a given row, the alarm and the warning values to which the stylesheet must refer can be found in the `<note>` subelements of the *i-th* `<field>` element. The positional correspondence is managed by means of the XPath `position()` function, which returns the ordinal number of the current XML node in the list of children of the parent node. Therefore, the data table is processed by means of two XSLT loops, nested as follows:

```
<xsl:for-each select="xdf:FITSML/xdf:array/xdf:data/xdf:d0">
<!-- lines (packets) scan -->
      <xsl:for-each select="xdf:d1">
      <!-- columns (parameter values) scan -->
             ...
      </xsl:for-each>
</xsl:for-each>
```

Within the inner loop, the ordinal of the current `<d1>` element is saved in an XSLT variable `p` as follows:

```
<xsl:variable name="p" select="position()"/> ,
```

such that it can be used in XPath expressions to select limit values, as in the following example concerning the ALARM_LOW boundary:

```
<xsl:when test=". &lt; ../../../xdf:fieldAxis/xdf:field[$p]/xdf:note[@mark='alarmLow']">
```

The whole XSLT stylesheet that generates the HTML code for the QuickLook chart basically carries out three different scan loops. The first loop is executed on all the `<field>` elements (except the first, that corresponds to the timestamp), in order to generate four preliminary lines indicating the maximum and minimum values of warning and alarm limits for each parameter; the second loop scans the fields in the `<fieldAxis>` element once more to identify the parameter-descriptive labels to be used as column headers; the third loop scans the data and sets their color in the HTML representation according to the result of the tests effected on the corresponding limit values by means of XPath expressions as described above.

A sample output, involving only three parameters, `F_01`, `F_02` and `F_03`, as rendered by a Web browser, is displayed in Figure 11. During the loop which generates the headers, a test on the consistency of the limit values themselves (`alarmLow`, `warningLow`, `warningHigh`, `alarmHigh`) is also carried out, and produces, whenever an error is detected, a header with a different, strikingly colored background (e.g. see the parameter `F_03` in Figure 11, having `warningLow` = `warningHigh`). In this way, the QuickLook user will at least realize that the alarm/warning signals he/she reads for a given parameter are not really meaningful. Actually, a strict consistency check is made only between the two warning limits, to ensure that the range of admissible values is not null. The alarm interval must contain (or coincide with) the warning interval.

MCAL-DFE-TE Packet Processor Output - 'type'= HK

'alarmLow' limit	255	1023	255
'warningLow' limit	1023	1023	5678
'warningHigh' limit	64512	64511	5678
'alarmHigh' limit	65280	65281	65281
timestamp	**F_01**	**F_02**	**F_03**
11-12-2002 23:58:02	255	1024	5678
12-12-2002 00:01:24	20	1023	5678

Figure 11. A QuickLook sample output

In order to make the timestamp information in the first table column useful at the QL stage, a final minor change has been introduced in the final release of the QL stylesheet (which also required a minor revision of the FITSML format specification for the processor output files). The change was necessary in order to have timestamps in a human-readable format in the QuickLook chart. As a matter of fact, the numeric timestamp inherited from the FITS files is expressed as a count of seconds starting from a time chosen as zero, whereas XSLT, in its current version, does not supply functions capable of interpreting and/or translating a timestamp in such a format. XPath *Extension* Functions for the management of time and date expressions exist, but would require data to be in the alphanumeric format provided in the XML-Schema specification (Biron & Malhotra, 2001). Hence, we included timestamp conversion in the functionalities of the packet processor (by simply

using the C language conversion functions available in the `ctime` library), so that also human-readable timestamps are included in the FITSML file generated by the processor. Such an addition required a minor revision of the FITSML file format to make room for the alphanumeric timestamps and an upgrade of the QL stylesheet for the correct management of such timestamp values.

7 CONCLUSIONS

In this work, we described the introduction of XML-related technologies into the development of Test Equipments for the AGILE scientific satellite mission. In particular, such technologies were employed for the representation and management of telemetry packets involving, in this phase, housekeeping data. For this purpose, packet descriptors and packet data XML databases were designed and implemented, and software modules were developed for the management of the XML databases, for interfacing with preexisting packages and for the release of a QuickLook module which uses a standard Web browser as (remote) front-end.

After implementation, extensive testing was performed to verify that each developed module functioned properly and, finally, the correct operation of the whole prototype. All the exploited XML technologies have confirmed their effectiveness: the XML-Schema grammar, used to define the new "centralized" packet descriptors database (also exploiting XDF and FITSML); the XPath queries, used to extract the required information from the descriptors database; the XSLT stylesheets, used to implement the INIT routine as well as the QuickLook module. The XDF and FITSML languages – in their revised versions – have been proven to meet all the project specifications. Also the interface between the Xalan/Xerces APIs and the already developed C++ libraries has not shown any particular problems, even if several of the features provided by that platform are still to be further investigated.

```
01 | Port of socket: 30000
02 | Sync memory created at key 10000
03 | ProcessorLib: createOutputFileNameBase: tbd_MCAL_hkmc_000000_030212_190356_s
04 | ProcessorLib: Start measurement
05 | Connection accepted
06 |
07 | APID=1294  Num fields=309
08 | FITSML: XML header written
09 |
10 | FITSML: 1 packet written
11 |
12 | FITSML: ALL written!
13 |
14 | XALAN running on tbd_MCAL_hkmc_000000_030212_190356_s.fits.xml, with XSLT =
   | proMozilla-v3'1.xslt, and out sent on tbd_MCAL_hkmc_000000_030212_190356_s.fits.html
15 | Validation status is now = FALSE
16 | Proceding to XSLTransform...
17 | ...XSLTransform DONE!
18 | DONE: FITSML converted to HTML!
19 |
20 | ProcessorLib: 1
21 | ProcessorLib: Stop measurement
22 | ProcessorLib: createOutputFileNameBase: tbd_MCAL_hkmc_000001_030212_190412_s
23 | ProcessorLib: Start measurement
24 | ...
```

Figure 12. Console output from a test session

A print-out of the console output, taken from one of the final test sessions based on a socket input stream (synthetic data), is shown in Figure 12. The messages in lines 14-18 show a call to another (experimental) method we built, named `transformViaXSLT()`, which makes use via the Xalan platform of an XSLT stylesheet to transform the processor output into an HTML document. In this way, the QuickLook function can be realized already within the processor module. The same method could also be used to validate the XML output; however, such a feature was only used during pre-

liminary tests and then disabled in order to avoid a significant and unnecessary overhead. The FITSML output was externally tested and proved to be *well-formed* and *valid* by means of several (also commercial) tools. FITSML output files have also been sent to an XML-aware Web browser (i.e. Mozilla) in order to be processed *on-the-fly* with the QL stylesheet, and in an *on-demand* fashion, that is only when a display of the QL data is actually requested. In fact, such an approach allows an even more complex stylesheet to be used, in order to let the QL user *interactively* select, on the client-side, the columns and/or the range of rows to be viewed. A final design choice on how and when to perform the XML to HTML transformation needed to achieve the QL function will be made at a later stage of the overall project, and will also take into account the overall performance of the whole system.

As far as performance is concerned, there are indeed several issues to be considered, including the waste of storage space caused by the XML format and the time-consuming XPath/XSLT processing. To solve the first problem, solutions based on XML-specific *compression* methods could be used, for example, by integrating already available tools at parser level. Compression leads to obvious benefits for network transfer, and storage of large files. Moreover, available query processing techniques (e.g. Arion, Bonifati, Costa, D'Aguanno, Manolescu & Pugliese, 2003; Buneman, Grohe & Koch, 2003) that work directly on compressed (or partially decompressed) XML files could also be used to improve the efficiency of their management. To address the second problem, an emerging class of hardware devices, generically named XML *accelerators* (Salamone, 2002), could be used to relieve, in practice, the application/database server from the XSLT processing, with a significant improvement in the conversion time. Future work will also consider options involving the introduction of such technologies.

In any case, we think that the chance to manage telemetry packet data flows (along with the metadata involved in their management) using XML-related technologies is worth further investigation and evaluation, and could also bring substantial advantages to scientific space missions in the long run. Long-term objectives also include the extension of the approach to the management of scientific data, for which the advantages would have an even greater impact (e.g. as to availability and dissemination).

8 REFERENCES

AGILE (n.d.) *Homepage of the AGILE Mission*. Retrieved January 1, 2004 from the IASF CNR Web site: http://agile.mi.iasf.cnr.it/

Arion, A., Bonifati, A., Costa, G., D'Aguanno, S., Manolescu, I., & Pugliese, A. (2003) XQueC: Pushing Queries to Compressed XML Data. *VLDB Proceedings (Demo session)* (pp. 1065-1068), Berlin, Germany.

Auricchio, N., Celesti, E., Di Cocco, G., Galli, M., Gianotti, F., Malaspina, M., Labanti, C., Mauri, A., Rossi, E., Stephen, J.B., Traci, A., & Trifoglio, M. (2002) MiniCalorimeter of the AGILE satellite*, SPIE Proceedings 4497* (pp. 187-198), San Diego, California.

Biron, P.V., & Malhotra, A., (Eds.) (2001) *XML Schema Part 2: Datatypes, W3C Recommendation, 02-05-2001*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/xmlschema-2/

Bray, T., Paoli, J., Sperberg-McQueen, C.M., & Maler, E., (Eds.) (2000) *Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation, 06-10-2000*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/REC-xml

Buck, L., Goldfarb, C.F., & Prescod, P., (Eds.) (2000) *Datatypes for DTDs 1.0, W3C Note, 13-01-2000*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/dt4dtd

Bulgarelli, A., Gianotti, F., & Trifoglio, M. (2003a) PacketLib: a C++ library for scientific satellite telemetry applications, *ADASS XII Proceedings* (pp. 473-476), ASP Conference Series 295, Baltimore, Maryland.

Bulgarelli, A., Gianotti, F., & Trifoglio, M. (2003b) *ProcessorLib Programmers Guide*. Retrieved January 1, 2004 from the IASF CNR Web site: http://www.bo.iasf.cnr.it/~GSE/ProcessorLib/

Buneman, P., Grohe, M., & Koch, C. (2003) Path Queries on Compressed XML. *VLDB Proceedings* (pp. 141-152), Berlin, Germany.

Clark, J., & De Rose, S., (Eds.) (1999) *XML Path Language (XPath) Version 1.0, W3C Recommendation, 16-11-1999*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/xpath

Clark, J., (Ed.) (1999) *XSL Transformations (XSLT) Version 1.0, W3C Recommendation, 16-11-1999*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/xslt

Fallside, D.C., (Ed.) (2001) *XML Schema Part 0: Primer, W3C Recommendation, 02-05-2001*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/xmlschema-0/

Gianotti, F., & Trifoglio, M. (2001) DISCoS - a detector independent software for the on-ground testing and calibration of scientific payload using the ESA Packet Telemetry and Telecommand Standards, *ADASS X Proceedings* (pp. 245-248), ASP Conference Series 238, Boston, Massachusetts.

FITS (n.d.) *The FITS Support Office Page at NASA/GSFC*. Retrieved January 1, 2004 from NASA, Goddard Space Flight Center Web site: http://fits.gsfc.nasa.gov/

GSFC.XML (n.d.) *The XML Group Resources Page at NASA/GSFC*. Retrieved January 1, 2004 from NASA, Goddard Space Flight Center Web site: http://xml.gsfc.nasa.gov/

IASF.BO (n.d.) *The Space Astrophysics and Cosmic Physics Institute of the CNR - Section of Bologna Homepage*. Available from: http://www.bo.iasf.cnr.it/

IBIS.HK (n.d.) *User Manual for the IBIS Instrument: TM Housekeepings*. Retrieved January 1, 2004 from the IASF CNR Web site: http://www.bo.iasf.cnr.it/Research/INTEGRAL/Documentation/IBIS_UserManual/IBIS_UM5-1_Vol3_PII_HKs.PDF

Layman, A., Jung, E., Maler, E., Thompson, H.S., Paoli, J., Tigue, J., Mikula, N.H., & De Rose, S. (1998) *XML-Data, W3C Note, 05-01-1998*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/1998/NOTE-XML-data

PTS (1988) *PSS-04-106 Packet Telemetry Standard, Issue 1, January 1988*, Noordwijk, The Netherlands: ESA Publications Division – ESTEC.

PTS (1992) *PSS-04-107 Packet Telecommand Standard, Issue 2, April 1992*, Noordwijk, The Netherlands: ESA Publications Division – ESTEC.

Salamone, S. (2002) Should XML Traffic Get Special Treatment?, *Networking Under the Microscope newsletter*. Retrieved January 1, 2004 from the World Wide Web: http://www.bio-itworld.com/archive/microscope/091102.html

Shaya, E., Thomas, B., & Cheung, C. (2001) Specifics on a XML Data Format for Scientific Data, *ADASS X Proceedings* (pp. 217-220), ASP Conference Series 238, Boston, Massachusetts.

Tavani, M., Barbiellini, G., Argan, A., Auricchio, N., Caraveo, P., Chen, A., Cocco, V., Costa, E., Di Cocco, G., Fedel, G., Feroci, M., Fiorini, M., Froysland, T., Galli, M., Gianotti, F., Giuliani, A., Labanti, C., Lapshov, I., Lipari, P., Longo, F., Massaro, E., Mereghetti, S., Morelli, E., Morselli, A., Pellizzoni, A., Perotti, F., Picozza, P., Pittori, C., Pontoni, C., Prest, M., Rapisarda, M., Rossi, E., Rubini, A., Soffitta, P., Trifoglio, M., Vallazza, E., Vercellone, S., & Zanello, D. (2001) Science with AGILE, *Proceedings of the 5th Compton Symposium* (p. 746), AIP Conference Proc. 510(1), Portsmouth, New Hampshire.

Thomas, B., Shaya, E., & Cheung, C. (2001) Converting FITS into XML: Methods and Advantages, *ADASS X Proceedings* (pp. 487-490), ASP Conference Series 238, Boston, Massachusetts.

Thompson, H.S., Beech, D., Maloney, M., & Mendelsohn, N., (Eds.) (2001) *XML Schema Part 1: Structures, W3C Recommendation, 02-05-2001*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/TR/xmlschema-1/

Trifoglio, M., Gianotti, F., Stephen, J.B., Celesti, E., Labanti, C., & Traci, A. (2000) Ground support equipment for scientific tests and calibration of the AGILE instrument, *SPIE Proceedings 4140* (pp. 478-485), San Diego, California.

Xalan-Xerces (n.d.) *Home page for the* Xalan-C++ *stylesheet processor and the* Xerces-C++ *validating parser*. Available from the *Apache XML Project* Web site: http://xml.apache.org/

XDF (n.d.) *eXtensible Data Format Homepage*. Retrieved January 1, 2004 from the NASA, Goddard Space Flight Center Web site: http://xml.gsfc.nasa.gov/XDF/

XDR-doc (n.d.) *XDR Schema Reference*. Retrieved January 1, 2004 from the *MSDN Library*: http://msdn.microsoft.com/library/en-us/xmlsdk/html/xmrefxdrschemareference.asp

XML (n.d.) *eXtensible Markup Language Homepage*. Retrieved January 1, 2004 from the W3C Web site: http://www.w3.org/XML/