A XML Data Dictionary and Document Representation Framework for Binary File Structure Description

Nuno Viana[1] and João Moura-Pires[2]

[1] Deimos Engenharia, Avenida D. João II, Lote 1.17, 8ºB, 1998-023 Lisbon, Portugal
nuno.viana@deimos.com.pt
http://www.deimos.com.pt
[2] CENTRIA/FCT, Quinta da Torre, 2829 -516 Caparica, Portugal
jmp@di.fct.unl.pt
http://centria.di.fct.unl.pt/~jmp

Abstract. System interface specification documents are hard to maintain updated, mainly due to the fact that the frequency of new improvements/ additions of new systems is high. To facilitate maintenance of existing interface specifications and enhance its usability, the development of a formal language specification, to address the definition and description of EUMETSAT's (European Organisation for the Exploitation of Meteorological Satellites) ground segment's system interfaces was envisaged. The language specification mechanism enables representation of binary message's structure, exchanged between EUMETSAT's systems. Additionally, the same definitions (derived from the specification language) can be embedded in automatically generated interface description documentation. In this manner, documents, become more than simple crystallized human-readable specifications. Document embedded data becomes "computable" and prone to be re-used by external tools, for generic data volume estimations and binary file structure validations. This paper focuses on the implementation details of a supporting XML Framework and its associated tools.

1 Introduction

The Operations Department at EUMETSAT[1] is currently responsible for the development and maintenance of several software and hardware infrastructures (also commonly known as facilities), which perform generation, archival and dissemination of meteorological products to all regional meteorological European member centres.

These facilities make extensive use of protocols based on binary messages for both control and transfer of meteorological raw/data products between systems (e.g. Atmospheric Motion Vectors, Clear Sky Radiances, Cloud Analysis, Cloud Top Height, Sea Surface Temperature, Total Ozone, Tropospheric Humidity – for a complete list of EUMETSAT available list of products, please refer to [2]). Specification of valid binary message structures and system interface descriptions is currently performed via creation of Interface Control Documents (ICD).

With the constant improvement of EUMETSAT's Ground Segment systems facilities, specifications have been subjected to successive updates. The complexity of the protocols (which rely on binary messages, reaching the Gigabyte order of magnitude) and the improvement of protocols (which are created to support new meteorological data products), generate high network traffic, thus making debugging, maintenance, development and deployment activities extremely difficult.

Nowadays, specifications for the EUMETSAT's system interfaces are maintained as interdependent Word documents (i.e. documents may contain references to specifications present in external documents), on which coherency and consistency is difficult to guarantee. Moreover, document contents are not computable (i.e. document data is generated for human-reading purposes only).

Enhancing the usability of documentation's contents, by transforming them into dynamically generated documents (containing "computable" specifications) represented in a suitable format, thus enabling the development of more generic document-driven tools, is one of the main objectives for this activity. Furthermore, migration of existing specification documents into the new format is also envisaged within the scope of this project.

This paper introduces a formalization approach for definition of a binary data messages' structures and definition of dynamically generated specification documents, which re-use the previously mentioned definitions. The first section (current) introduces the context of the activity. Afterwards, in the second section, the authors unveil the structure of the binary messages. In the third section the specification language used to describe the structure of the binary messages is presented. Further ahead, on section 4, the envisaged XML solution is described, with a special focus on the proposed model mechanism. Afterwards, the authors describe the operational tools which take advantage of this solution (section 5). Finally, on section 6 the authors discuss the conclusions and envisaged future work in the line of the present activity. Finally, the interested reader may find additional information concerning the work being described in this paper in section 7.

2 Binary Message Structure

In order to better understand the binary messages' structure we will follow a top-down descriptive approach. Firstly, the binary File/Packets' structure will be discussed. Afterwards, we will address the definition of Application Data Units (ADU) and finally the binary structure definition specification language will be presented as well as encoding/decoding rules.

2.1 Ground Segment Packets and Files

Binary messages exchanged between the different facilities in the EUMETSAT Ground Segment, use two types of transfer services (i.e. packet transfer and file transfer services) and are codified as binary data streams. Binary messages can therefore be divided into two sub-groups: "Packets" or "Files", which have a pre-defined struc-

ture built on ADU definitions, as described by the following Figs. 1 and 2 (optional fields are identified by dashed-line boxes):

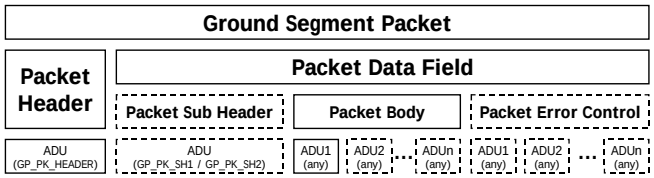

Fig. 1. Structure of a Ground Segment Packet ("Packet Header", "Packet SubHeader", "Packet Body" and "Packet Error Control").

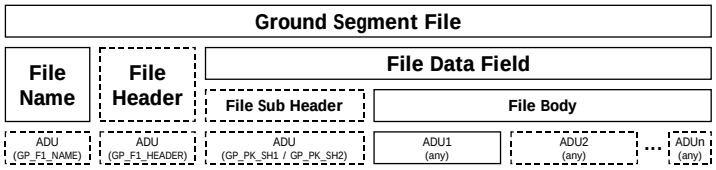

Fig. 2. Structure of a Ground Segment File ("File Name", "File Header", "File SubHeader" and "File Body").

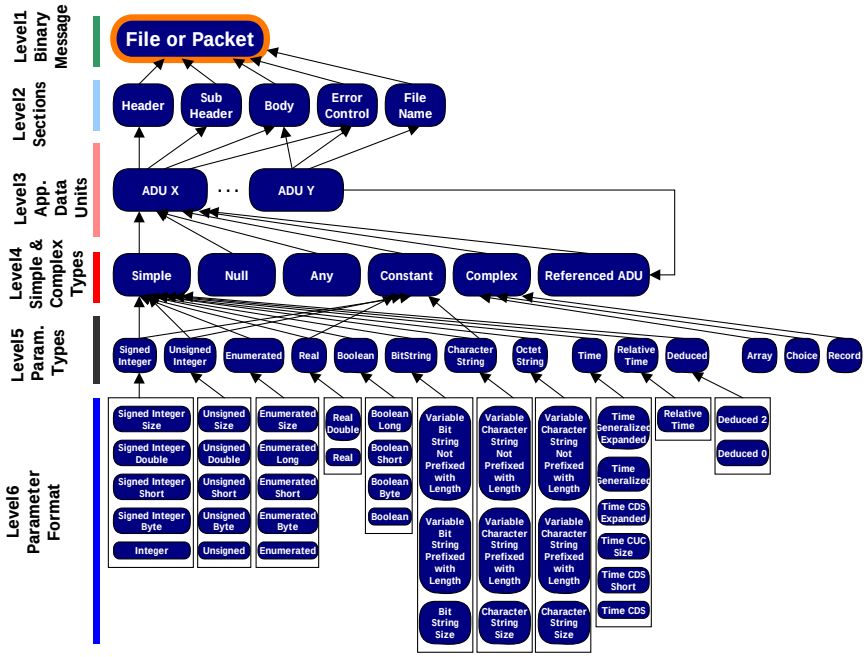

Fig. 3. Hierarchy of concepts showing how a Binary File/Packet can be specified using Application Data Unit which are defined using the formalized specification language using Simple, Complex and additional types (Null, Any, Constant and Referenced ADU).

As illustrated by the previous picture, a Ground Segment Packet is formed by a "Packet Header" (of type "GP_PK_HEADER"). Optionally, it can also be composed by an additional "Packet SubHeader" (of type "GP_PK_SH1" or "GP_PK_SH2"), a "Packet Body" (formed by any sequential collection of one or more ADU definitions). Finally, the "Packet Error Control" field can also be appended to the previous fields. In the same manner as the Ground Segment Packet definition, Ground Segment File definitions are also composed by a "File Header" (of type "GP_F1_HEADER"), an optional "File SubHeader" (of type "GP_F1_SH1") and a "File Body" (sequential composition of one or more ADU definitions). In addition, a Ground Segment File definition also contains the file naming convention to be used when naming files.

2.2 Application Data Units

Ground Segment engineers should define ADU (Level 3) types in order to form collections of sections for Binary File and Packet structure descriptions (Level 2). By its turn, each ADU makes use of an arrangement of simple and complex types (Level 4), including default and optional values.
Here is an example of an Application Data Unit definition ("GP_F1_SH1" – General Purpose File Sub Header type 1):

```
GP_FI_SH1 ::= RECORD
              {SubheaderVersionNo     UNSIGNED BYTE,
              ServiceType             GP_SVCE_TYPE,
              ServiceSubtype          UNSIGNED BYTE,
              FileTime                TIME CDS SHORT,
              SpacecraftId            GP_SC_ID,
              Description             CHARACTERSTRING SIZE (187)}
```

2.3 Encoding Rules

Previously, the hierarchy of concepts, which enable the specification of the binary File's and Packet's structure, has been described. We will now turn our attention to how the field values are in fact read and written from/to binary Files and Packets. In the frame of the EUMETSAT's Meteosat Second Generation Programme's Ground Segment systems, each binary value can be clearly identified by a set of attributes which define:

- The field type is represented under the form of a code, known as **PTC** (Parameter Type Code). E.g. Boolean(1), Integer(4), Unsigned Integer(3), Real(5), etc. This field is of type ENUMERATED_BYTE with a length of 1 byte.
- The representation format for the field is represented under the form of a code, known as **PFC** (Parameter Format Code). E.g. IntegerByte(4), Integer Short(12), Integer(14), Integer Double(16). This field can be of either ENUMERATED_BYTE (for fixed-length fields) or ENUMERATED_SHORT (for variable-length fields) with respective lengths of 1 and 2 bytes, respectively.

- The length of the parameter data field, known as **PDL** (Parameter Data Length), only valid for variable-length data types (variable string derived types such as CHARACTERSTRING, BITSTRING and OCTETSTRING). This value is of type SHORT_UNSIGNED and has a size of 2 bytes.
- The data value itself. Its type depends on the PTC and PFC values (when the field encoding type is "explicit" or if "implicit". In the context of our activity, binary data representation is possible through two methods for encoding simple parameter field values:

  a) "Explicit" method, where each value is preceded by an indication of the field type that follows, as well as the format for the type.
  b) "Implicit" method, where only the parameter value exists. That is the parameter type and format are assumed to be commonly agreed by the applications, which perform the encoding and decoding of the binary values.

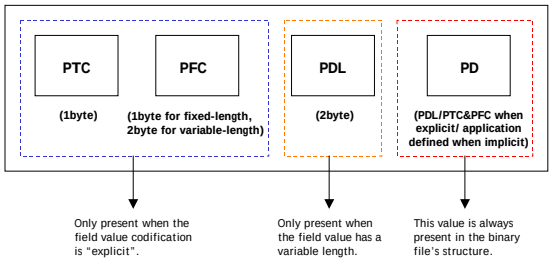

Fig. 4. Codification mechanism used for representation of a field value in the binary messages exchanged between the EUMETSAT's Ground Segment's systems.

On the previous figure, one may find the required and optional fields necessary to encode/decode binary values from the exchanged messages. When a value is "explicitly" encoded, the binary field (stored on the binary file) is prefixed with the PTC and PFC values. If in addition, the field has a variable length, the value is firstly prefixed with its size (PDL), resulting in the structure above (see Fig. 4). There is no explicit method for the complex types (i.e. no PTC and PFC fields). However, when a complex type is declared "explicit", its components (that is to say, all simple types) inherent this attribute's flag value. In short, when "explicit" encoding is used, this means that before reading the actual field value from the binary file, an application should first read its type code (PTC), format code (PFC) and its size fields from the binary file (if the field has a variable length specification). On the opposite side, when declared "implicit", applications assume a commonly agreed type and format for interpreting binary values read/written from/to files.

3 Specification Language

On the previous section (2), we have seen how the binary messages (packets and files) can be decomposed into simple component structures and how their identifications are encoded in the binary file.

In this section, the authors will address the formalization of a specification language, which was envisaged to characterize the binary structure for files and packets. The formalization of the specification language has been described using a BNF (Backus-Naur Form) notation and comprises “Simple Types” (the atomic types) and “Complex Types” (array, choice and record structures). Additionally, the specification language was enriched by the authors with new types such as “Null”, “Any” and “Referenced” types.

3.1 Simple Types

Simple types should be understood as the atomic type definitions, which exist in the specification language. The language comprises “Boolean”, “Enumerated”, “Unsigned Integer”, “Signed Integer”, “Real”, “Bit String”, “Octet String”, “Character String”, “Absolute Time”, “Relative Time” and “Deduced” types (see Level3 in Fig. 3).

3.2 Complex Types

In addition to the simpler atomic type definitions, the language also incorporates the notion of complex structures built on previous defined Simple Types:

1. Array – An ordered set of a fixed or variable number of fields (array elements) of the same simple or complex type (e.g. Array of Array).
2. Record – An ordered set of a fixed number of fields (components) of any simple or complex type (e.g. Record with component1=Array and component2=Integer).
3. Choice – A set of complex or simple fields, from which one can be selected based on the context and determined at application runtime (e.g. Choice with component1=Signed Integer and component2=Unsigned Integer).

3.3 Complementary Types

In addition to the simple and complex types, the authors decided to introduce into the specification language other high-level types such as “Constant”, “Null”, “Any” and “Referenced ADU” (due to representation requirements). “Constant” types (as the name implies) allow definition of environment constants, which can be of type Signed and Unsigned Integer, Real and Characterstring. “Any” allows definition of a field value without specifying its type or internal structure, while “Null” defines an element, which has no assigned type or field value. Finally, yet importantly, there is a special type, which was introduced in this specification language to allow re-usage of existing ADU definitions (“Referenced ADU”) inside other ADU. Below you can find three examples of variable definitions using the specification language:

Day ::= ENUMERATED SIZE (4)
{Mon, Tue, Wed, Thu, Fri, Sat, Sun, NotADay (OTHERS)}

```
NumericValue ::= EXPLICIT CHOICE
                 {UnsignedType      UNSIGNED,
                 IntegerType        INTEGER,
                 RealType  REAL}
Example ::= VARIABLE ARRAY SIZE (1..4) OF RECORD
                 {Date              TIME GENERALIZED,
                 DayOfWeek          Day,
                 Filler             BITSTRING SIZE (4),
                 ParameterName      CHARACTERSTRING SIZE (6),
                 ParameterValue     NumericValue,
                 Unit               CHARACTERSTRING SIZE (4)}
```

4 The XML Framework Solution

Previously (section 2), we have described the internal structure for the binary messages (files and packets) exchanged between the different EUMETAT system facilities. Moreover, a formalization of a specification language capable of describing the complex structure of the binary messages was presented in section 3. We are now going to describe the envisaged solution and its main functionalities.

4.1 Goals

Defining a specification language for description of binary messages, addressed part of the problem. Users have expressed their desire to be able to represent the language in such as way that it would be easily manageable and computable. XML was then selected as a natural candidate for representation of the both the specification language. XML is computable and there is a variety of XML manipulation tools freely available to users.

In addition to the formalization of the binary message specification language using XML, it was envisaged that the definition of files/packets including its internal structure (headers, sub-headers, body and so on), which are composed by ADUs would also be performed via XML. This would allow the full representation of files/packets under XML format. In order to maintain ICD (system interface specification documents) updated, it was also proposed to transform existing documentation into a XML based format. By this manner, documents would be able to embed binary file/packet “computable” definitions based on XML, capable of being used by externally developed tools, while minimizing necessary user intervention during document updates (references to files/packets and ADUs specifications are dynamically updated upon user request). This proposed approach, opens a path for development of more generic tools, in less time with less effort (e.g. computation of data volume estimations for generated network traffic between facilities and estimation of archival allocation requirements for generated data products).

4.2 The Proposed Model

Users wanted to be able to specify the collection of ADUs (Application Data Units) and the binary messages' structure (sequential composition of ADUs). Since ADUs are defined using the formalized specification language components in section 3, the XML-based framework model included the ability to firstly define the specification language. A language vocabulary for specification of the specification language was then designed and implemented (see a in Fig. 5 a) and b))

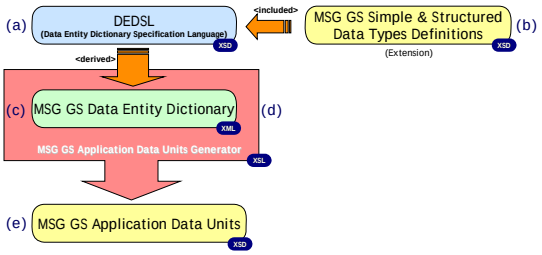

Fig. 5. Engineered solution to address the representation of binary message structures using XML based technology.

By using the rules expressed in this language specification vocabulary a) and b), under the form of a XML schema, users are able to define the full collection of Data Dictionary entities (where each Data Dictionary entity element represents an ADU definition in a data dictionary) which will be made available to users for definition of files/packet structures in interface specification documents. At this point we have a collection of Data Dictionary entities under the form of an XML file. In order to allow definition of binary packets and files (composed by collections of ADUs), we should first generate the library of ADUs. This library can be dynamically generated using a provided MSG GS ADU Generator (implemented as an XML Stylesheet file), which transforms the XML file into an ADU specification language (XML Schema file). As can be seen in Fig. 6, the example ADU is composed by a record with different components of type ("UNSIGNED BYTE", "TIME_CDS_SHORT", "CHARACTERSTRING_SIZE" and even of "REFERENCE_TYPE" – yellow coloured. This last type allows re-usage of previously defined ADU as types themselves).

The proposed model for the implementation of the XML Framework solution relies on "Schema" validation for ensuring that both the Data Dictionary types and the built ADU definitions follow the standardized implemented structure. The use of "Schematron"[3] validation would bring significant advantage for semantic validation (e.g. several field constraints are currently only checked afterwards, and not at XML document definition time – field sizes which depend on variable size definitions of child types).

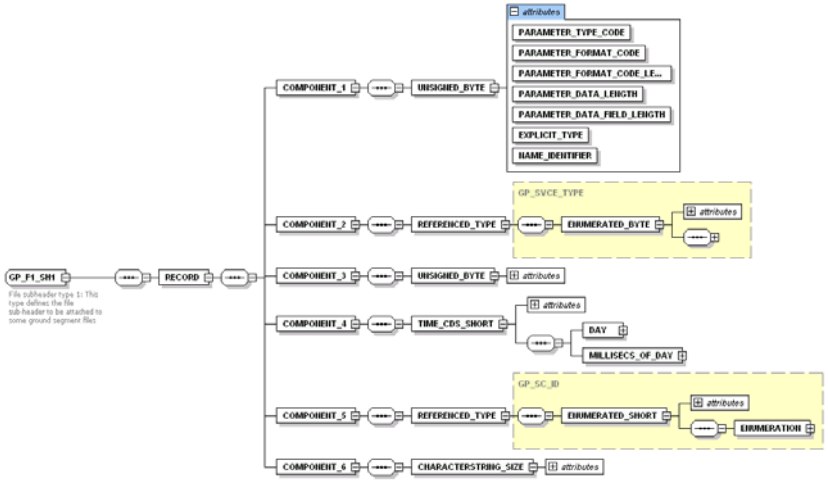

Fig. 6. ADU structure example ("GP_F1_SH1").

4.3 Interface Specification Documents

The paragraphs that follow, describe the mechanism for definition of interface specification documents and how the ADU definition language previously generated will be utilized.

In addition to the standardization of ADU definitions, users wanted to both standardize the contents for interface specification documents and be able to include the previously ADU language vocabulary in each of the generated document. This task was accomplished by defining a XML Schema language for structuring of the documents contents. Moreover, this document structure specification language takes advantage of the ADU library previously generated. This mechanism is described in the following illustration:

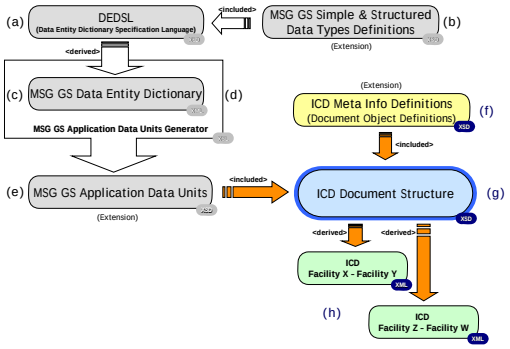

Fig. 7. ICD document structure (g), which includes the dynamically generated ADU specification (e).

4.4 Envisaged Deployment

After defining the collection of ADUs and the structure for interface specification documents, in order to that ensure users use the same language (not customized versions), a centralized repository was designed. This repository stores the vocabulary languages, as XML Schema files (for both the ADUs and interface specification documents). Furthermore, when defining the contents for the interface specification documents, users should insert references to ADUs (which are stored in this centralized repository). When users would like to retrieve any interface specification document (in XML format), the repository automatically replaces the references to ADUs, by its internal structure (used as input data for external tools).

It is also envisaged that for document updating purposes, the central repository will also be able to provide the original document versions in XML (containing references to ADUs), which should be re-uploaded into the central repository once updated.

Given that any major change on ADUs definitions has an impact on all documents which refer them, only administrator users should have access to the XML Framework (located into the central repository) – for updating ADUs definitions for instance.

5 XML Framework's Tools

With the XML Framework deployed in a centralized manner, administrators are able to enforce coherency of ADU definitions in interface specification documents, as well as standardizing of its internal structure across all platforms.

But the real gain comes from developing tools which use the binary files/packets definitions specified in each of the interface specification documents. Validation of binary files and packets (exchanged between the different EUMETSAT system facilities - and specified as collections of ADU definitions in the interface specification document) captured under the form of files can be accomplished via a Binary File Validation Tool, which is currently under development. This tool ingests a binary file or packet, an ICD in XML format and the chosen identification of the file/packet definition. The tool is then able to produce a compliance report containing the results from the field values extracted from the binary file according to its specified structure. Another tool, could take for instance, a specification document, a user specified binary file/packet definition and a given frequency for generation of that particular file/packet, for determination of daily-generated network traffic and/or allocation requirements. Besides the previously described tool, other tools will continue to provide browsable (HTML) and printable (PDF) versions of ICD documents with a format similar to existing documents in Word document format. This process is accomplished by making extensive use of XML Stylesheets for both transformation of specification documents from XML format into XHTML format, or into XSL:FO (Formatting Objects), which can be further transformed into PDF files via a "Formatting Objects Processor" engine such as the "Apache FOP Engine"[4].

Last but not least, a Data Format Extractor Tool is also nearly finished. This tool automates the process of extracting ADU textual definitions from original ICD documents and converting them into their XML representation according to the XML Framework for further integration in ICD documents in XML format. All these command-line tools are being developed using Java for added portability and will also be integrated and transparently made accessible via the centralized Server.

6 Conclusion and Future Work

The paper has focused on the descriptions of an engineering solution devised for standardization of data types representation through the formalization of a specification language, under the form of a Data Dictionary in XML format, re-usage of user-defined data entities specifications (ADUs) inside interface specification documentation (with added validation mechanisms provided by XML Schemas) also represented in XML format and also on tools which are able to ingest the specification language in XML, thus turning previously reading-oriented (document) data contents into "computable" data contents. The proposed model depends exclusively on XML Schemas for validation of XML file structures, nevertheless this solution could be greatly improved by using other validation mechanisms such as Schematron and/or XCSL[5, 6] (XML Constraint Specification Language). It also important to mention that similar work (specification of binary files using XML notation) has also been carried out by edikt since 2003. Their work has culminated in the development of BinX[7, 8], a library and associated editor tool for representation and manipulation of scientific data for grid applications. Unfortunately, the fact that bit types are not supported as well as the limited portability of the solution (library is written using C++, although porting to other languages such as Java is being envisaged), have rendered this approach unusable (for representation of EUMETAST binary data).

Besides the development of the XML Framework, several tools have been developed: the Binary File Validation Tool, an ICD Preview Tool, Data Format Extractor Tool and the server component which wraps all these command-line tools under a multi-user web interface for easy access. The current work has been partially inspired in previous and ongoing research work carried at the Uninova Research Institute[9], namely the development of metadata storage repositories and manipulation tools[10] for SEIS[11] (Space Environment Information Systems) and SESS[12] (Space Environment Support System) projects.

As future work (and as previously discussed), the possible use of Schematron and/or XCSL to improve the XML Framework should be investigated. Additionally, it has been also anticipated that a port of the tools will take place into C/C++ for added speed gains as well the enhancement of the XML Framework with the development of a generic API library capable of providing reading and writing capabilities with little integration effort with external applications.

Furthermore, we hope that it would be possible to broaden the applicability of the XML Framework and tools to other areas of EUMETSAT. This solution is prone to be re-used by any project on which the representation and manipulation of binary data is an issue.

7 References

1. EUMETSAT. *European Organisation for the Exploitation of Meteorological Satellites*. 2005 [cited 31/10/2005]; Available from: http://www.eumetsat.int.
2. EUMETSAT. *Eumetsat Access to Data - Product List*. 2005 [cited 2005/11/06]; Available from: http://www.eumetsat.int/idcplg?IdcService=SS_GET_PAGE&nodeId=522&l=en.
3. *Schematron - A Language for Making Assertions About Patterns Found in XML Documents*. 2006 [cited 2006.01.21]; Available from: http://www.schematron.com/.
4. Apache. *The Apache XML Graphics Project*. 2005 [cited 2005/11/06]; Available from: http://xmlgraphics.apache.org/fop/.
5. José Carlos Ramalho, et al. *XCSL : XML Constraint Specification Language*. 2006 [cited 2006.01.21]; Available from: http://www.di.uminho.pt/~gepl/xcsl/.
6. Ramalho, J.C. *Constraining Content: Specification and Processing*. in *XML Europe'2001*. 2001. Berlin, Germany.
7. *BinX - A Library for Representation of Scientific Data*. 2006 [cited 2006.01.21]; Available from: http://www.edikt.org/binx/index.htm.
8. Rob Baxter, et al. *BinX – A tool for retrieving, searching, and transforming structured binary files*. in *All-Hands Meeting 2003*. 2003. Nottingham.
9. UNINOVA/CA3. *UNINOVA - Centre For the Development of New Technologies/Soft-Computing and Autonomous Agents*. 2005 [cited 2005.11.20]; Available from: http://www.uninova.pt/ca3.
10. R. Ferreira, et al. *XML Based Metadata Repository for Information Systems*. in *EPIA 2005 - 12th Portuguese Conference on Artificial Intelligence*. 2005. Covilhã, Portugal.
11. Pantoquilho, M., et al. *SEIS: A Decision Support System for Optimizing Spacecraft Operations Strategies*. in *IEEE Aerospace Conference*. 2005. Montana, USA.
12. ESA. *Space Environment Support System for Telecom/Navigation Missions*. 2005 [cited 2005/11/06]; Available from: http://telecom.esa.int/telecom/www/object/index.cfm?fobjectid=20470.