AIAA 2010-2015

XTCE at GSOC - First experiences adopting a new standard

Armin Braun[1],Yi Wasser[1], Michael Schmidhuber[1], Harald Hofmann[1], Martin Wickler[1]
German Space Operations Center, DLR Oberpfaffenhofen, 82230 Wessling, Germany

Simon Maslin[2]
InSyEn AG, 82234 Wessling, Germany

On the level of data protocols and their usage we have come along a long path in the last decades having standardized the transport mechanisms, data protocols and packet layers; first missions are applying the packet utilization standard. But still the content of telemetry and command data is as varied as the missions and payloads themselves. Project by project one had to agree on a common "language", i.e. an exchange format for the description of telemetry and command data-bases, which turned out more cumbersome than the eventual communication with the space segment. Given complete freedom when designing on-board software, developers would find a myriad of clever solutions, resulting in different "user interfaces" for satellites and even more possibilities on documenting these. Different operation entities, user centres, bus and payload controllers have to co-operate. They are equipped with their own systems, coming from different technical heritages and company cultures. However all project-participants are dealing with the same technical subject: a common spacecraft. It is obvious that transforming databases for no other purpose as to make it readable for the organization-owned software is a waste of time and resources. Following popular examples of today's information technology, it was proposed to use a formal mark-up language could build a bridge between the diverse systems. For this purpose a formal language has been developed, which is XTCE. This evolving standard shall facilitate the task of describing the telemetry and command data and communicating it between operation entities and spacecraft manufacturers. This paper will report on the experiences of the first implementation of XTCE for the TM/TC database management at GSOC.

Acronyms

API	=	Application Programming Interface
ASCII	=	American Standard Code for Information Interchange
CCSDS	=	Consultive Comitee for Space Data Systems
CSV	=	Comma Separated Values
DOM	=	Document Object Model
DTD	=	Document Type Definition
EGSE	=	Electronic Ground Support Equipment
ESA	=	European Space Agency
ESOC	=	European Space Operations Centre, Darmstadt
GSOC	=	German Space Operations Center, Oberpfaffenhofen
ISO	=	International Organization for Standardization
JAXB	=	Java Architecture for XML Binding
MCS	=	Monitoring and Control System
MIB	=	Mission Information Base (the TM/TC Database of SCOS 2000)
.NET	=	is a Microsoft product (runtime framework and development environment)
OMG	=	Object Management Group
PUS	=	Packet Utilization Standard, ECSS

[1] Mission Operations Department, Münchner Str. 20, 82234 Wessling
[2] Senior System Engineer, Münchner Str. 20, 82234 Wessling,

SAX = Simple API for XML
S/C = Spacecraft
SCOS = "S/C Operation System" is an ESA provided tool used at GSOC
TC = Telecommand
TM = Telemetry
UML = Unified Modeling Language
W3C = World Wide Web Consortium
XML = Extendable Markup Language
XSD = XML Schema Definition
XSL = XML Stylesheet Language
XSL = XSL Transform
XTCE = XML Telemetric & Command Exchange

I. Introduction

Telemetry and tele-commands are the only means of communication with an operational spacecraft. TM/TC therefore provides the only operator-interface by which casual on-board problems can be detected and handled. So it is mandatory that this part of the control centre's software is well tested and reliable. A modern spacecraft comprises several thousand sensors and software parameters, so the decomposition and interpretation of binary data-streams has to be managed. Typical monitoring and control systems, which are the ground-side of the communication, are basically generic and have to be "missionalized" for each satellite, which means configured to interpret that specific data-streams, locate and calibrate all parameters. This is nominally done by configuration files containing all that information, coming from a data-source which may be referred to as "TM/TC database".

As the spacecraft consist of many modules and components originating from various suppliers, TM/TC databases will have to integrate also portions from different sources. Finally the S/C is handed over to a customer, also probably to a LEOP control center and for that purpose TM/TC databases have to be passed on previously. Surprisingly enough there has been no standard format for such an important interface item. XTCE now fulfils that need.

In most of the latest projects at GSOC the transformation of TM/TC databases was circumvented by exploiting the common use of the monitoring control system as core of the EGSE or vice versa. The last projects where databases had to be built up by the control centre date back to 1990s. Unfortunately in a newer, upcoming project - an optical earth observation mission - a common basis for monitoring and control could not be found, as both, control centre and spacecraft manufacturer, insisted on re-using inherited tools. Again the database format agreed within this project will be proprietary and tool oriented; a deficiency that was believed being overcome. So the project is facing exactly the cost, schedule, and error risk, for the solution of which XTCE was invented.

II. XML in general

For those readers not familiar with XML and related technologies a short overview is provided in the appendix section. This was included not to presuppose to much previous knowledge.

III. TM/TC databases

A. Documentation

The TM/TC database, being the interface to the satellite, always forms a central product within the satellites documentation. Commands and TM-Parameters are referred to in all parts of the spacecraft documentation. Updates and changes are quite common during the course of spacecraft integration and testing. Consistency with the rest of the documentation is important. So this information must be up to date and easily accessible during project preparation. That is why it is typically maintained using database tools. From there the information can be retrieved, filtered and extracted in almost any format.

B. Related Tools

Besides the purely spacecraft related part, database tools can be used for supplementing information, like procedures or definition of display pages. Relations to the TM/TC descriptions within the databases are always needed for this.
The TM/TC database content is a prerequisite for defining the following

- Display Pages

- Flight procedures
- Test Data
- Offline Analysis
- and Simulators.

As one can imagine there may exist several tools for above mentioned items which have to be adapted with every update of the TM/TC database. The database content is never stable during mission preparation, so the all the related tools might need maintenance. It can not be assumed that all tools are able to understand the native database format of the MCS so, at best a lot of special database reports or file conversions have to be done, not to speak of manual updates and coding.

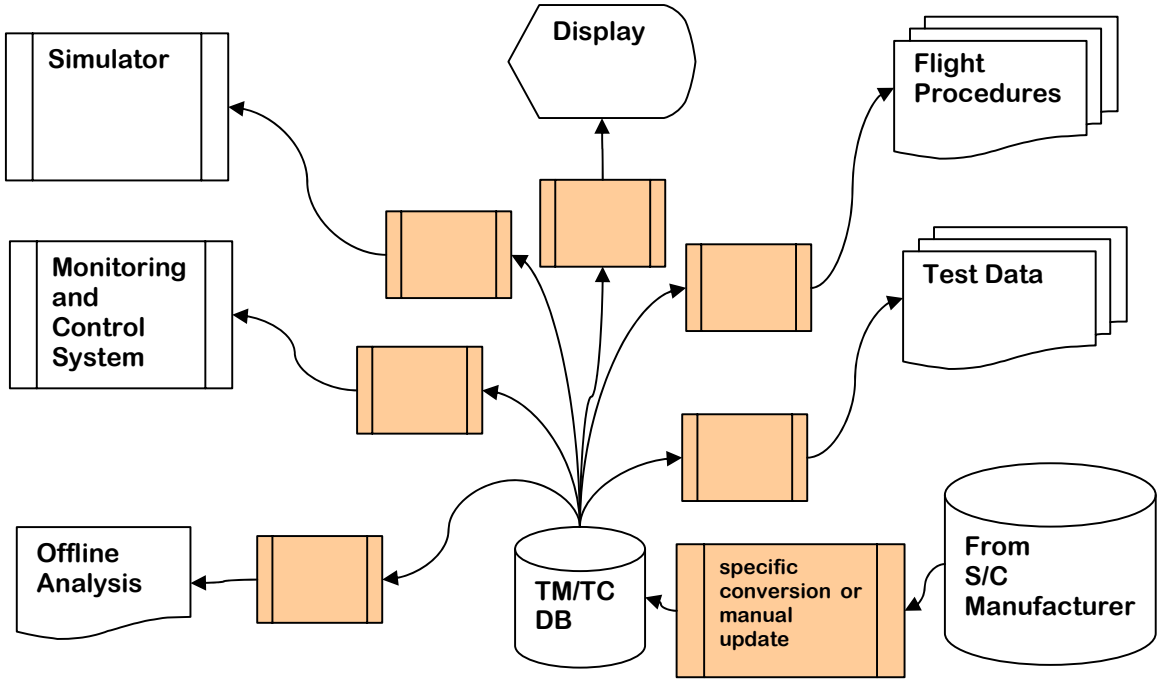

Figure 1 Update process using tool specific formats

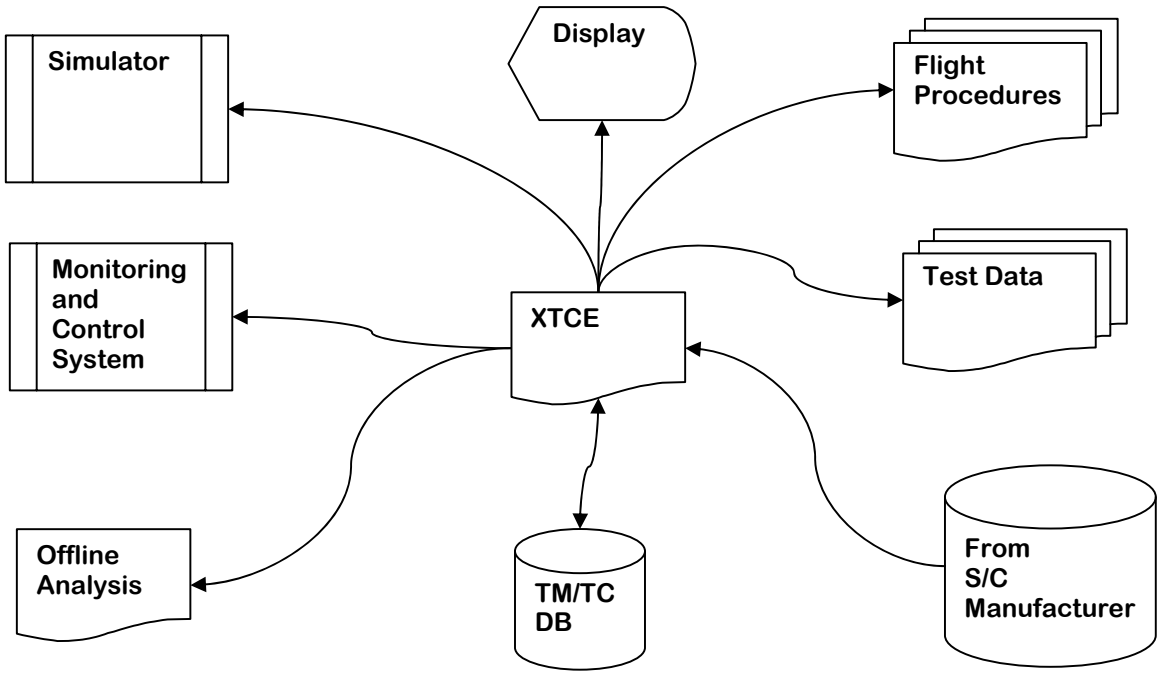

Figure 2 Simplification of update process using XTCE as a common format

C. Exchange Formats

The most crucial conversion process is the import of the TM/TC database delivered by the S/C manufacturer. The efficiency and correctness of this strongly depends on the agreed format and structure.

1. *Paper Docs*

It can safely be assumed that this way of exchanging TM/TC databases is no longer exercised, if not driven by contract lawyers who preferred a written signature on a deliverable rather than an electronic exchange format. All TM/TC databases deliverables are now "machine readable".

2. *Tabular*

The simplest form of exchange consists of listings in form of tables. This could be a set of files or a workbook with several spreadsheets. Using flat files there are no means for enforcing consistency, correctness, and completeness except care and caution.

3. *Databases*

Compared to flat tables, databases have better means for accessing data randomly. They usually provide a programming interface on which extracts and conversion processes, as well as consistency checks can be based. The concept of relations between data-tables supports the consistency. Depending on the tool data is not necessarily stored in simple file. In this case replication and export needs special tools.

Database engines that work file oriented offer the possibility to exchange the database as a whole. Typical file sizes are still manageable even with wide-spread office tools. Though this sounds like common denominator that can easily be found, commercial tools very often show the problem of incompatible versions and irreversible conversion between those.

Both methods have a common drawback that the columns can be chosen rather arbitrarily. Compatibility between sender and recipient is unlikely. In future XTCE will provide a standard interface.

IV. XTCE

XTCE is a XML dialect for telemetry and commands exchange. It was started as an interagency study hosted by the Object Management Group (OMG). It now reached a certain level in the CCSDS recommendations and has been accepted as an ISO project by TC20/SC13. The latest version of XCTE is currently 1.2, just passing a review.

XTCE is capable of describing all transmission layers down to physical streams. Therefore there may be different recipients of an XTCE file. For example: XTCE could be passed to a ground station and to the control centre. Whereas the ground station may use only the definition of physical streams and virtual channels and not care about the content, the control center may be interested in packets not needing to know in which frames and segments they arrive.

V. Specific project at GSOC

As monitoring and control system ESA's SCOS 2000 is used in the multi-mission environment. As mentioned in the introduction there is the necessity to convert an externally delivered database from a proprietary format into something that can be transformed into a SCOS mission information base (MIB). The solution envisaged attempts to satisfy two parallel objectives. Firstly, the conversion process shall be automated as far as possible, as it will have to be performed repeatedly during the mission preparation. All structural incompatibilities will have to be identified, handled and if necessary iterated with the manufacturer. Secondly, XTCE shall be used as an intermediate format, in order to establish and exercise the use of XTCE as future exchange format.

Starting point of the work was a two-way conversion tool, the experimental use of which was presented by M. Merri and J. Müller on SpaceOps 2006. The tool transforms XTCE to a set of tab separated text files that make up an instance of the SCOS MIB. The technology used for the conversion process is XSLT. The key characteristics of XSLT is described below. The most important of which is that the transforms must have XML input. For the "forward" conversion – XTCE to MIB – this is of course the case, but not so for the reverse transformation. The inverse transformation needs an intermediate step of building two XML files, with a one-to-one relation ship of MIB data fields to XML structures. By inspection of the code it was found that more than half of it served the construction of the intermediate structures.

Several implementation strategies have been considered and are described below.

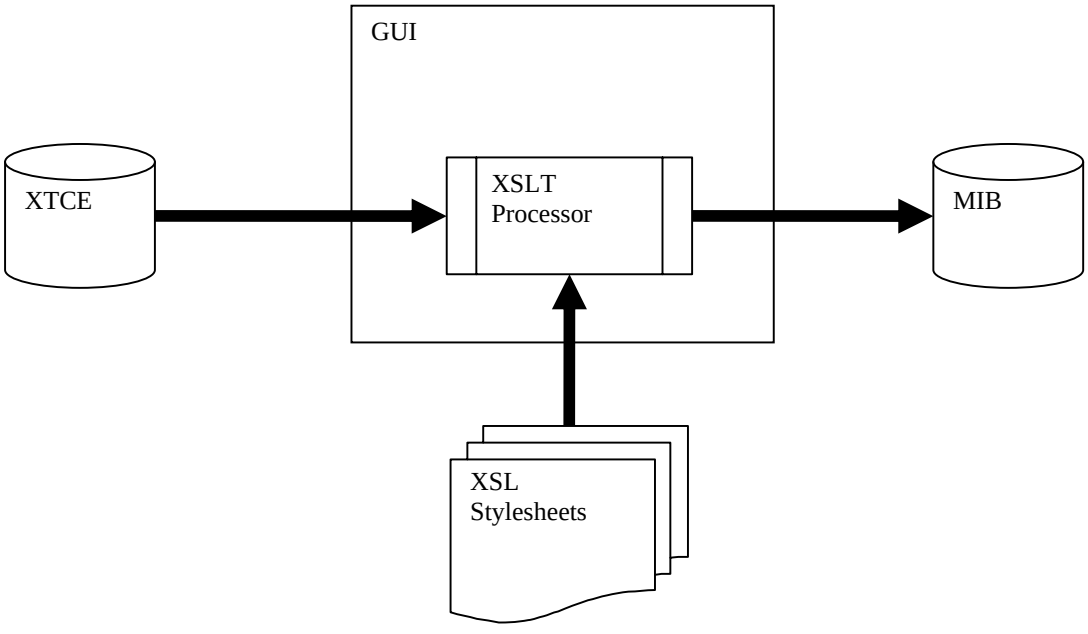

Figure 3. Forward conversion - XTCE to MIB

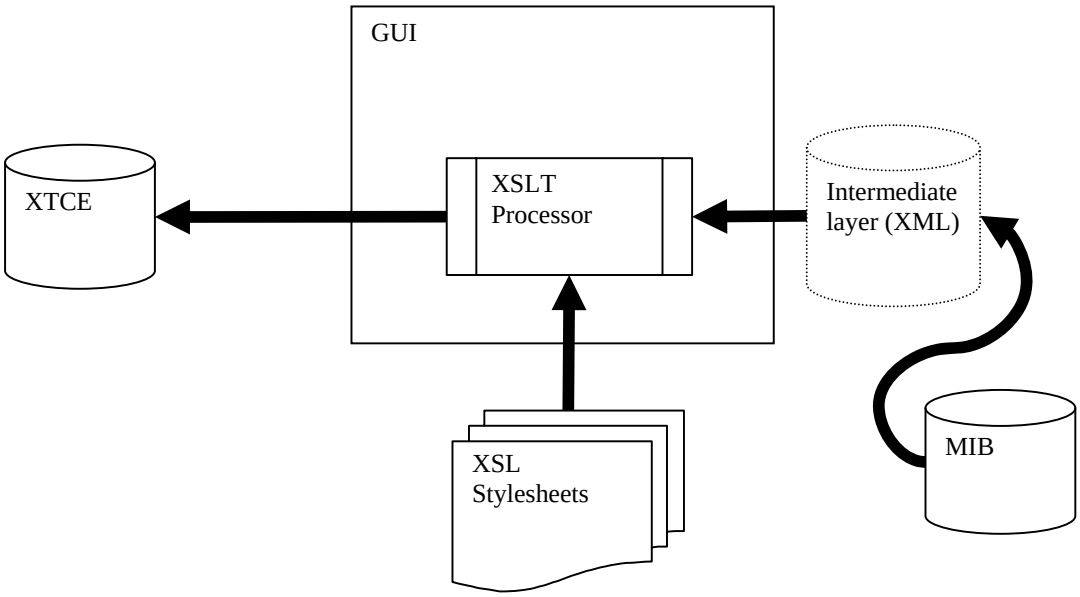

Figure 4. Backward conversion via intermediate format

A. .NET

As XSL are not language specific the transforms may be implemented on any system and in any language. Because the common development environment at GSOC mission planning team is Microsoft ® .NET, a port of the ESA tool was attempted. However this development branch was suspended, as it is foreseen to feedback the enhancements to ESOC and keep the tool platform-independent. However successful reduction of the code size could be achieved.

B. Abstraction of intermediate layers

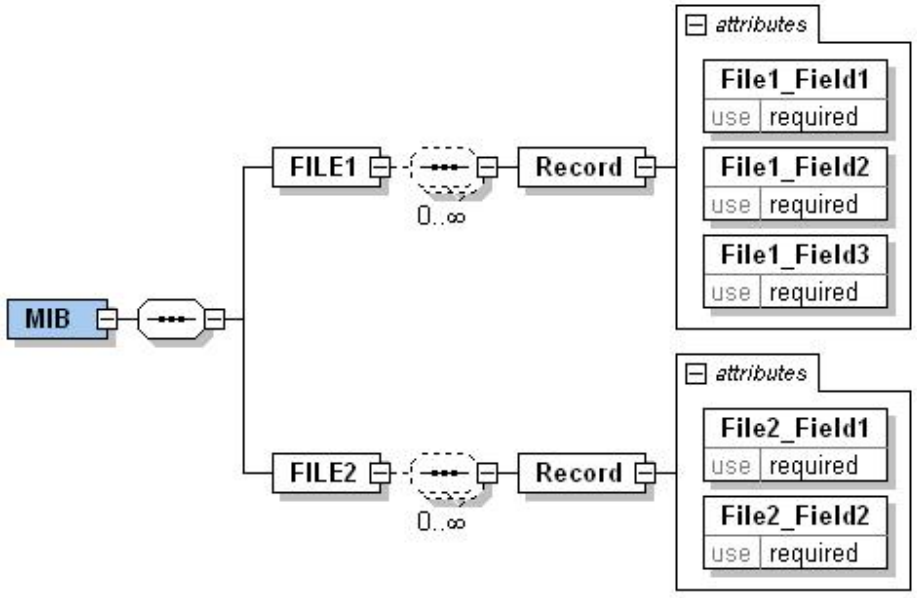

Figure 5. Structure of the intermediate layer

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSpy v2008 sp1 (http://www.altova.com)-->
<MIB>
   <FILE1>
      <Record File1_Field1="text" File1_Field2="text" File1_Field3="text"/>
      <Record File1_Field1="text" File1_Field2="text" File1_Field3="text"/>
   </FILE1>
   <FILE2>
      <Record File2_Field1="text" File2_Field2="text"/>
      <Record File2_Field1="text" File2_Field2="text"/>
   </FILE2>
</MIB>
```

Figure 6. Corresponding form of the XML

As can be seen the file structure is repetitive and simple. In order to reduce code, in the attempt reducing the porting effort to .NET this regularity was exploited. An example XML file is used as a basis. By convention the element names are identical to the file names. Each element is a specimen for XML elements originating from that specific file and is replicated zero to n times, depending on the number of lines in the file. The list of attributes is identical to the names of the data columns. A simple routine reads the MIB files line by line and fills the attribute values correspondingly.

The example files are generated by a XSL transform out of a description of the MIB in XML-Schema syntax. This was generated by copy and paste directly from the SCOS MIB ICD. The schema can also serve the purpose of validating the intermediate files; for example checking the existence of mandatory entries.

This streamlining of the code was successful in the .NET, but showed a problem in Java because the sequence of the attributes where not preserved in the document object model (DOM, see appendix). According to the W3C guideline, DOM implementation should sort the attributes according to their names. This feature did not match our

specific purpose where the preservation of the sorting order was essential for mapping the (nameless) data columns of the MIB file to the XML attributes. Finally the problem was solved in JAVA after replacing the DOM parser used in the converter by a SAX parser.

C. Generation of XTCE from external inputs

1. Use of XSLT

Initially an attempt was made to re-use the "backward" part of the tool, which does the "table to XTCE" conversion for the purpose of doing the first conversion step. The java code of building up temporary XML can be re-used after implementing the abstraction mentioned above. "Only" the appropriate XSL sheets have to be written. Bearing in mind that this part of the conversion is project internal and needs to be neither platform-independent nor re-used in another project, there are no obligations to stick to that approach. Meanwhile another attempt, initially being purely experimental, turned out to give much quicker results.

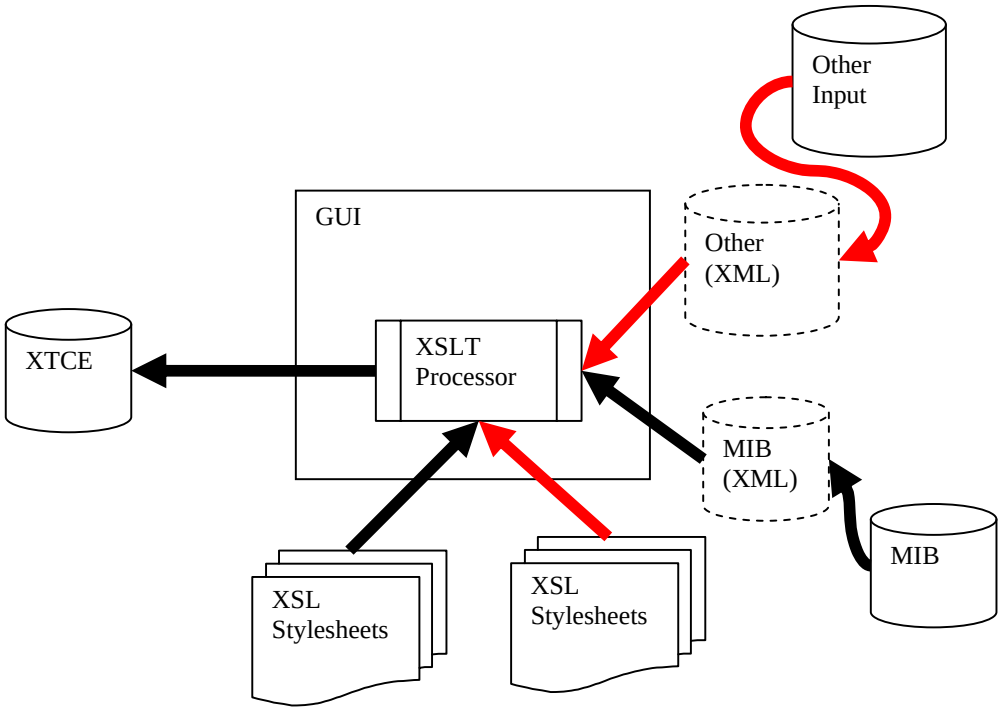

Figure 7. Re-using backward part for external input

2. Serialization of classes

Based on a code, which resembles a class-representation of the XTCE, the data is directly stored in objects. Using available serialization functions, the XTCE is directly written from the content of the objects. This approach will most likely replace the concept depicted in Figure 7.

D. XTCE representing classes

How to obtain code for appropriate classes that permit the XTCE serialization? Several options were exercised and are now in different states of completion.

1. Manual implementation

A first version of the class hierarchy was written on the basis of an UML representation of the XTCE elements. Starting from the level of detail of XTCE documentation the code was gradually enhanced. Special care was taken, to ensure that there was a special unit test for every XTCE element. Each test comprised of a serialization of the example, validation of the resulting XML and de-serialization. Of course the development process is slow, but well protected against errors and sometimes revealed idiosyncrasies in the XTCE schema itself. The approach proved to be robust at the time when the new 1.2 version of XTCE had to be adopted. It only took a number of hours to successfully re-run all the unit tests. Advantage of this method is full control over the developing API, including the mapping to generic data-types. Disadvantage of this method is the time required to implement the code. The

completion status of March 2010 is about 50% at less than 6000 lines. We assume that this will lead to the leanest code version of all alternatives.

2. Microsoft® XSD.exe

"XSD.exe" is a tool belonging to the Microsoft .NET® development environment. It is able to convert XML Schema files to C# or visual basic code, representing classes or "datasets" (which are classes marshalling database structures). The tool was very successfully used in another project at GSOC, TerraSAR-X, where all interface file handlers could be generated without problems. This resulted in a lean API using native classes. However the XTCE schema uses features that cause problems. The reasons for this remained a bit obscure, but it seems to be a problem with XML attributes being "complex types". As explained below attributes can have only values that can be represented as a string of characters, with no further sub-elements. But any deviation from a "build-in" – better: in XML-Schema pre-defined – type unavoidably renders it a "complex type". A "string" for example is accepted by the tools as valid type for an attribute, a string consisting of capital letters only is not. Unfortunately no number can be given for the resulting code size, but it would have been comparable to that which was achieved manually.

3. XML Spy®

XML SPY is able to derive code from a Schema in C#, C++ or Java. The code does not rely on language specifics and is kept portable, which on the other hand means Schema specific data types are generated as own classes instead of being mapped to generic types of the language where possible. The generated code as tested so far seems to be performing very fast and consists of approx. 25000 lines, not counting several libraries provided by Altova.

4. Java Architecture for XML Binding

Besides DOM and SAX the method of de-serializing into domain specific classes is also available for the Java programming language by means of the JAXB framework; here it is called "un-marshalling". The tool used is named XJC and available in the Java developers toolkit. When used it generated 116 java classes with 24200 lines of code.

E. Difficulties of the specific project

1. No PUS implementation

SCOS 2000 is intended for use with spacecrafts that comply with the packet utilization standard. The definition of parameters is organized by PUS type and sub-type, which are part of a special implementation of the CCSDS secondary header. Unfortunately this is used differently in our reference project.

In the whole conversion process, XTCE is the most generic component. The CCSDS packet TM/TC is only one instantiation of all the possible data stream formats that can be described by XTCE. So many specific data structures can be successfully described in XTCE. However the converter in its current state now relies on certain contents and conventions that need to be included in the XTCE file.

2. Data Types

SCOS decommutation is based on the assumption that any packet is a chain of pre-defined data-types. So the successful database conversion process depends on the ability to map every used parameter encoding on to an existing PUS parameter type

VI. Proposals for future enhancements

A. SCOS MIB

The MIB file format is simple and can be imported quickly. On the other hand, the content can not be read without the knowledge of the column number containing the data. The use of XML, as a self-describing and context safe data-structure could enhance readability and facilitate cross conversion to other XML dialects. The use of a single file would guarantee completeness and consistency of the database. Looking to the future it would be preferable if SCOS were able to read in XTCE directly.

B. XTCE discussions

1. Types and Parameters

Discussions are on-going to define whether calibration curves and alarms are properties of a TM parameter or are features of a parameter type. Another issue for larger databases is whether parameters and their types should be topologically separated in the tree structure. It even may be asked if the differentiation of parameter types and parameters is necessary. For example: is an alarm limit a property of the type or of the individual parameter? In order to give the user a choice, one could imagine that an example (abstract) parameter serves as a specimen for others.

2. Scope of XTCE

For good reason XTCE explicitly does not deal with documentation and explanation of TM parameters and commands and lacks features for describing GUIs or display pages. The first would duplicate user manuals; the latter would violate the intended tool independence. As a consequence it must be complemented with other databases or data products to form the whole picture.

3. Command sequences

Projects may choose different granularity of commands and associated parameters; some may operate command per command, others consider complete (validated) procedures as the smallest unit to send to the spacecraft. Sometimes a chain of commands is compiled as a ground or on-board macro and later handled as if it was a single command. So the boundaries between commands and command sequences are not clearly defined. As far as we are aware of XTCE can not currently create command sequences representing flight procedures. Another standardization effort, making use of the XTCE achievements to handle aspects of procedure management would seem a natural progression.

VII. Next steps

The following identifies a number of tools that would appear useful in the context of XTCE implementation.

A. Test Data generator

Presuming XTCE contains enough information to de-commutate and interpret binary data-stream, it should be possible to synthesize data-structures that can serve as test data.

B. XTCE file management

XTCE supports a rather freely structured set of files, but tools might require a single file as input. So probably a tool will be needed which is able to

- merge XTCE files
- extract or split XTCE in several files
- handle different versions and partial deliveries
- harmonize separate deliveries of XTCE files from different sources,
    - e.g. identify identical parameter types named differently and combine them to a common type,
    - resolve name conflicts

C. Generic XTCE editor

Although XTCE, as every XML file, could be written by something as simple as a text editor, a syntax aware editor with capability of validating the file against a schema is helpful. Tools like Stylus Studio® or XML Spy® could support this feature. However, a specialized schema-aware editor with fewer features is also worth considering.

D. Validator

Syntax and valid types and ranges are sufficiently tested by the above-mentioned schema validation. Certain combinations could pass a pure syntax check; but on-top of that there may be a couple of less obvious semantic checks. A tool may search heuristically for those mismatches and issue warnings.

VIII. Conclusion

XTCE has the potential of replacing existing exchange and configuration file formats. By providing a formal input to all components of the ground segment it could contribute to effectiveness and cost saving. Adoption of the widely distributed XML makes use of a great variety of available tools. Development of APIs is straight forward and tool-supported so the establishment of interfaces to all tools in scope is simply a matter of time and determination. GSOC has taken first step to bring XTCE into action in a concrete mission.

IX. Acknowledgments

I would like to thank M. Merri, M. Koller and J. Müller of ESOC for their previous work on the converter prototype they provided to us. Thanks to Kevin Rice for his work on the standards and his help on XTCE matters.

Appendix

XML in general

A. History, purpose, syntax

1. ASCII

Everybody is familiar with character encoded text files, formerly known as "ASCII", i.e. symbols represented by a binary pattern that make up data or written language. Only a small subset of all possible file contents can be unambiguously interpreted by software for further processing. To be able to do so, files have to follow certain convention, which is called syntax. A "comma separated file" (CSV) for example has the simple convention that every record is separated from the preceding one by a new-line and every data field separated by a comma. In most cases this is sufficient to exchange data and it is the common denominator for many programs.

2. XML

On the first glance XML seems to be a complication, w.r.t. to text or CSV, containing a lot of duplicated key-words. The advantages are:

- files are both: machine readable (fulfil the requirements to a formal language)
- and human readable at the same time (character encoded meaningful keywords)
- not limited (no fixed line length, number of columns, arbitrary complex structures)

It should be noted that every (well formed) XML file in the world obeys the same syntax, which only uses half a dozen single character symbols. Every XML processing software module can open and parse every XML file. Of course that does not mean it can make sense out of every file, and most likely the application will not run smoothly, but once you have an XML parser at hand in your libraries, you never need another one, no matter what application you will use it in.

B. XML in a nutshell

An XML file consists of a tree structure of elements. To be precise only one element is allowed per XML file, which is called the "root" or "document element", but it may have an arbitrary number of sub-elements. Each Element and the subordinate tree structure are completely enclosed by a start and an end tag. The tags are case-sensitive identifiers embraced by "<" and ">" the end tag version using the identical identifier starting with "</" instead. There is a short-hand writing of omitting the end tag, if there is no further content closing a single element tag by "/>". In between the brackets the elements may carry further attributes, which are key-value pairs notated as: *key* = "*value*". To resolve ambiguities, element names may be further specified by prefix string preceding the element identifier and separated by a colon. The prefixes are referring to a namespace, which must be defined by an URL typed attribute of the root element.

Processing instructions are bracketed by "<?" and "?>", comments are led in and out by "<!-- " and "-->". A special element <![CDATA[Content]]> serves for inline insertion of any binary data.

C. Dialects, Schemas

On top of the standardized simple syntax, the format is very versatile and extendable to any needs. It is recommended to use speaking identifiers that briefly describe what the contents of the elements are. Thus a self documented readable file is obtained. Adaption to certain application is done by using appropriate hierarchy and repetition of named elements. The sum of these conventions makes up a special XML dialect.

A deprecated way of defining a dialect was the document type definition (DTD), using a special XML syntax, which now is commonly replaced by "XML Schema" which itself is a XML dialect. XML schema definitions typically carry the extension XSD. Schema allows strong-typing of element contents, restricting element contents to numerical ranges or a limited set of valid values. The process of checking file contents against all the constraints defined in a schema is called *validation*.

D. Validation

It is necessary to check any input data before it can crash the application; when using XML files this can be done by "validation". Validation includes checking of the file being "well formed", that means following the standard XML-Syntax. Further it is validated against a schema. Validation ensures that the data is correct and complete. As stated earlier the whole usable content is enclosed by the root element tags, therefore an incomplete or corrupted file can be easily detected. Validation is usually done by existing run-time classes, so for the tedious process of validating inputs there is no further programming needed, not even if there are later changes. Separating the parsing/validating

code from the definition of valid inputs saves a lot of coding and lets developers concentrate on content and purpose of data. In other words: all the input checks need not to be hardcoded, which simplifies maintenance of the software.

E. Parser Technologies

What is the view on the data transported by an XML file from inside a program? What programming interfaces are exposed?

1. Xml Document Object Model

The file is ingested at once, the tree of elements is mapped on a tree of XML nodes, attributes are stored in a key-value list and identifiers are available by a "name" property. Therefore all elements look the same from the interface point. This has the advantage that the whole document is stored in memory. The program has full control over it – searching, modifying, etc. Literally every XML file can be read in. But this advantage is paid for by the necessity to implement the parsing of the data in more inner stages of the software.

2. Sax

This concept presumes that the data is received as a sequential stream. The parser resembles a kind of state engine, at the occurrence of certain elements in a well defined context; call-back functions or event-handlers are invoked and react on the data as wanted. This concept is advantageous if only fractions of a file have to be used. SAX consumes less memory space than DOM because it does not store the whole structure in the memory, and it is faster since it is event-driven, it does not need to traverse the whole tree to search for a piece of information.
Negative: the process is one-direction only. Information is read and dropped. The original input can not be restored.

3. Marshalling Classes

Element types as defined in a schema are mapped to classes in the object oriented programming language. Every occurrence of an element corresponding to that type instantiates an object. This way data is directly mapped to classes that resemble or approximate the "business model" of the application. These classes can be complemented by any necessary code. The process of writing XML files containing the data of the class hierarchy is called "serialization", the reverse reading of files and exposing objects to the application is called de-serialization.
As the structure of the XML file is unambiguously and completely defined by the XML schema, the task of generating code for the classes can be done by software as well. There is a multitude of tools available for that purpose. In our concrete case we collected some experience with different tools and approaches which are summarized further below.

F. XSL Transform

Though XML is versatile and multi-purpose, there is a most common usage for it: the separation of website content, from the website's layout. As this should be possible with least possible programming overhead, the layouts are provided in form of a website template, where parts of it have to be substituted by content coming from the XML file. These templates are expressed in the XML style-sheet language, the processing and generation of final output is a so called XSL-Transform. In principle the output can be any textual representation of the data and is not restricted to an output as html, though this is the most common case. The data input is always an XML file – let's ignore the trivial case of an XSL without any data substitution, where the output of the XSL is the content of the XSL template itself.
Because XSL has to

- query and filter data content coming from an XML file
- switch between distinct cases and loop over lists
- do simple calculations and text processing functions,

XSL sometimes resembles or duplicates features of a query- and a programming language. Consequently XSL lacks the clarity and conciseness of special purpose languages. The general concept as being a template often reveals the sequence and logical interdependencies in which the XSL is evaluated. This is a major drawback that makes debugging of XSL sheets difficult and time-consuming.

References

Proceedings

M. Merri and J. Muller., "The XTCE Standardization Approach of Telemetry and Telecommand Databases: The ESA Example", AIAA-2006-5582, *SpaceOps*, Rome, Italy, 2006