A New Meta-Data Structure For Telecommand and Telemetry Transmission in Space Missions*

Uzay Görevlerinde Uzkomut ve Uzölçüm İletimine Yönelik Yeni Bir Meta-Veri Yapısı

Fatih İleri[1]

Turkish Aerospace Industries Inc., Ankara, Turkey
fatih.ileri@metu.edu.tr

Abstract. Increasing complexities in satellite missions require the telecommand (TC) and telemetry (TM) transmission and the data structures within the telecommands and telemetries to be designed to meet the complex mission requirements. Building complex data structures yields a heavy work load on the development stage, while transmission of such data frames consumes a significant amount of time period of the available space link interval. In order to respond these challenges, a metadata model which is able to be generated through XML formation, together with the TC encoder, TM decoder (on the ground station), TC decoder and TM encoder (on the flight SW (FSW)) is developed in this work. This metadata model enables easy production of complex telecommands and telemetries composed of various combinations of field types and nested structures.

Keywords: metadata · telecommand · telemetry · encoder · decoder · space missions.

Özet. Uydu görevlerindeki karmaşıklıkların artması, uzkomut ve uzölçüm iletiminin ve uzkomut - uzölçüm veri yapılarının karmaşık görev gereksinimlerine de cevap verebilir şekilde tasarlanmasını gerektirmektedir. Karmaşık veri yapıları inşa etmek yazılım geliştirme sürecine ağır bir yük getirmekteyken, bu tür veri paketlerinin iletimi de mümkün olan uydu bağlantısı süresinin ciddi bir kısmını tüketmektedir. Bu zorluklara cevap vermek üzere XML düzeni aracılığıyla kodu üretilebilen bir meta-veri modeliyle birlikte yer istasyonunda uzkomut kodlayıcı, uzölçüm çözücü, uçuş yazılımında uzkomut çözücü, uzölçüm kodlayıcı geliştirdik. Bu meta-veri modeli, veri tiplerinin çeşitli kombinasyonlarından ve iç içe yapılardan oluşan karmaşık telekomutların kolayca üretilmesini mümkün kılmaktadır.

Anahtar kelimeler: meta-veri · uzkomut · uzölçüm · kodlayıcı · kod çözücü · uzay görevleri.

* Supported by Turkish Aerospace Industries Inc.

1 Introduction

Space missions are getting more and more complex in the recent years, such that, space missions pose hard challenges reflected through a multitude of dependent parameters [1]. Besides the transmission challenges due to the increasing size of telecommands and telemetries from the bandwidth point of view, decomposing the telecommands and the telemetries also brings extra load at the corresponding receivers as the variety of these data frames increases. Defining and tailoring mission-oriented telecommands and telemetries is also a hard task from the development point of view, such that a modular metadata design is an emerging need.

Satellite communication system SW design and implementation is a huge work load, which should be based on modularity as much as possible for the SW to be easily tailored for new satellite systems. The required modularity should enable easy tailoring the SW system through the configuration files stored in databases rather than making modifications on the code itself [2]. Architectures which have the capability of self-adaptation in the runtime with respect to the modified requirements are named as "meta-architectures" [3]. Thomé et. al. implemented a such an architecture for controlling (monitoring and commanding) multiple satellites dedicated for different missions through a common infrastructure [3]. They designed different metadata for each of the satellites to be controlled. Each metadata set composes a different part of a single object model. A satellite operator is permitted to utilize only the part of the object model which is related to the satellite system he/she is authorized to control. The authors do not provide details about the metadata they used but the work they did is important from the usage scenario point of view.

In 2004, Simon et. al. published the XTCE - an XML based standard for the mission operation databases - which constitutes a basis for our meta-data model [4]. In XTCE, the output bit stream is composed of containers. A container may contain other containers. The useful data packaging unit is named as "message". A message together with the identifier key is encapsulated by the "key container" entity. The key is used by the receiver unit in identifying the related message. The atomic data unit in the XTCE context is the "parameter". A parameter has different features such as type, size and bit order. All the types of parameters such as integer parameters, float parameter, array parameters and etc. are inherited from the parameter base class. All the telemetry and telecommand data structures can be defined in XTCE format as discussed here.

In this work, we propose a tailored version of the XTCE approach, which has a hierarchical architecture enabling code generation, to define the telecommands and the telemetries together with all the subcomponents. Once a TC or TM is defined in XML format with respect to our metadata model, then the metadata code for the TC or TM can easily be generated. The modularity of this metadata structure enables quickness and easiness in populating different versions of telecommands and telemetries. This metadata model can easily be integrated with the TM and TC encoders and decoders to be used in FSW and

ground station software (GSW). Telecommands and telemetries are called as "data frame" in next chapters for simplicity.

2 Methodology

The metadata is built for all the data frame (TC and TM) structures in a hierarchical manner in XML formation for them to be programmatically populated. All the metadata are stored in both the onboard computer and the ground station. The metadata representing a TC is utilized by the encoder module of the GSW, which generates the bit stream. The generated bit stream is passed to the next layer to be transmitted to the satellite. Same methodology is applied for the TM delivery: The metadata representing a TM frame is used by the TM encoder module of the FSW, for the corresponding bit stream to be generated. Once the TM bit stream is generated, it is forwarded to the next layer to be transmitted to the ground. Decoding of a TM by the GSW, and a TC by the FSW are handled in a similar manner.

2.1 Metadata Model Components

Field descriptions constitute the biggest part of the metadata. Field descriptions contains all the information related to the component it refers to. Each description consists of the following:

- **descriptionType**: *field* or *array*
- **isConditional**: The flag that represents whether the existence of the related field is dependent to a condition or not.
- **conditionDescription**: Details of the condition on which the existence of the field in question depends.
- **fieldData**: Includes the decription of a single element (field).
  - **offset**: Number of bytes before this field in the encapsulating structure in the metadata.
  - **nativeFieldType**: Enumerator representing the primitive type of the field in the output structure of the decoder or input structure of the encoder.
  - **encodedFieldType**:Represents how the native field is encoded (e.g. unsigned, twos complement, and etc.)
  - **fieldBitLength**: Size of the fields in bits
  - **fieldEndianness**: Represents the bit ordering of the field.
  - **polynomialCoefficientsIndex**: The index of the polynomial coefficients in the polynomial coefficients multidimensional array, to be applied on the raw field value to obtain the corresponding engineering value.
  - **rangeIndex**: The index of the range description in the range descriptions array, which enables range validation on the field in question. Negative value is assigned to rangeIndex if there is no need for a range check.
- **arrayData**: Includes the decription of an array.

- **arrayStartOffset**: Number of bytes in the encapsulating structure before the first element of this array
- **isSizeFieldConstant**: The flag which shows whether the array size is constant or located in a position in the metadata
- **constantArraySize**: If the *isSizeFieldConstant* flag is "True", then the array size is written in the constantArraySize field.
- **arraySizeFieldOffset**: If the *isSizeFieldConstant* flag is "False", *arraySizeFieldOffset* shows from where to read the array size information in this metadata. *arraySizeFieldOffset* is relative to the array level in which this array is nested, or absolute (relative to the start of the upper layer data frame structure) if there is no nesting.
- **arraySizeFieldOffset**: The enumerator which represents the type of the array size field. It is the same enumerator with the *nativeFieldType* of the *fieldData*.
- **arrayElementSize**: Number of bytes allocated in the output structure of the decoder or input structure of the encoder per element of the array in question.
- **arrayDescriptionSpan**:Number of component descriptions for the array in question. It is used to understand where the descriptions of the next element of the data to be transferred begins in the metadata.

Only one of the components *fieldData* and *arrayData* can exist in a component description, since a component in the data frame can either be an array, or a single field. But each element of array may contain another array together with a field, which is called "nested array".

2.2 Conditional Fields, Arrays

When the *isConditional* flag of a description is "True", the described element may or may not exist in the data frame bit stream depending on the condition. *conditionDescription* field in the metadata describes how to apply a conditional check on the related field or array. A *conditionDescription* is composed of the following components:

- **navigationParameters**: A set of parameters through which the encoder or decoder module is supposed to compute the address of the field value on which the condition check is going to be applied.
- **conditionFieldType**: Primitive type of the condition field
- **conditionValue**: The value which will be used to determine the existence of the field in question

The pathway in applying a condition check on a field is as follows:

1. Calculate absolute offset of the field which is going to be checked,
2. Retrieve value from the field,
3. Apply equality check between the field value and the *conditionValue*,

4. If values are equal, then the described component (array or field) exists in the data frame bit stream,
5. If not, the described component is not present in the data frame bit stream.

An important detail about retrieving a value by using the absolute and relating offset values is that the offsetting should be related to a field which is already parsed from the data frame bit stream (for decoding), or written to the data frame bit stream (for encoding).

2.3 Calibration

Calibration is the conversion of a raw value to its corresponding engineering value. This is generally needed when it is more efficient to send the raw value together with the calibration parameters when compared to sending the resulting engineering value [5]. In our metadata model we implemented polynomial calibrations, such that the calibration parameters are the polynomial coefficients.

All the coefficients are stored in a single array. The field which will be calibrated has the index (stored in *polynomialCoefficientsIndex*)for the polynomial coefficients table. If no calibration is to be applied, *polynomialCoefficientsIndex* is set to a negative number. The polynomial coefficients table is an array of *polynomialCoefficients* elements. Each *polynomialCoefficients* element is composed of:

- A variable for the number of coefficients,
- A pointer for the reaching the first polynomial coefficient on the coefficients array.

Once the calibration parameters are accessed, then the engineering value is computed simply as in the following:

$$e = \sum_{i=0}^{n-1} c[i] r^i \tag{1}$$

where;
n is the number of polynomial coefficients associated with the field in question,
c is the array of polynomial coefficients,
r is the raw value,
e is the engineering value.

2.4 Range Check

Range check is a protective process before encoding or decoding a data frame. Range check is applied on a field in the data frame if its description indicates a range check necessity (i.e., *rangeIndex* has a non-negative value in the related field description). Range descriptions array is composed of *RangeDescription* objects. A *RangeDescription* object has the following attributes:

- **rangeType**: Type of the range check to be applied (e.g. lower limit check, upper limit check and etc.)
- **rangeOperationParameters**: The parameters of the range check operation like the lower and upper limits, or the primitive type of the limit parameters.

A field, value of which is 20, should pass the range test given on Table 1:

Table 1. A range description example

"lowLimitCheck"	int8	-100	Null	0	Null
rangeType	*valueType*	*minVal*	*maxVal*	*permittedValuesCount*	*permittedValuesStartAddress*

When the range test of any field of the data frame fails, the overall data frame is discarded and some alert mechanisms specific to the application should be initiated.

2.5 Array Description

An array description is followed by its elements' description(s) in the metadata. Elements of an array may be of primitive types, structured types, or combinations of primitive and structured types. Figure 1 demonstrates a single level (array composed of non-array elements) array description. Each element of the given sample array is a structured type, which is composed of two primitive type fields. It is also worth emphasizing that any integer element of the array which is encoded in the data frame bit stream with n bits, may be represented with m bits signed integer position in the upper layer structure (output of the decoder or input of the encoder), where m is larger than n.

2.6 Nested Arrays

If any sub-field of an array is also an array, this architecture is called "nested array". The metadata structure we propose provides a flexible nested array definition. Before giving an example of nested array description, it is needed to mention the "array level info" table. When decoding or encoding an array, the information necessary to use with the offset values to compute the absolute location of any field in the higher level structure are kept in a table called "array level info". Number of entities in this structure gives the array level in which the encoding or decoding process is being carried on.

A graphical representation of a nested array description is given in Figure 2. Each element of the outer array consists of a 16 bits signed integer and an array. The inner array consists of unsigned integers (encoded with 8 bits in upper layer structure).

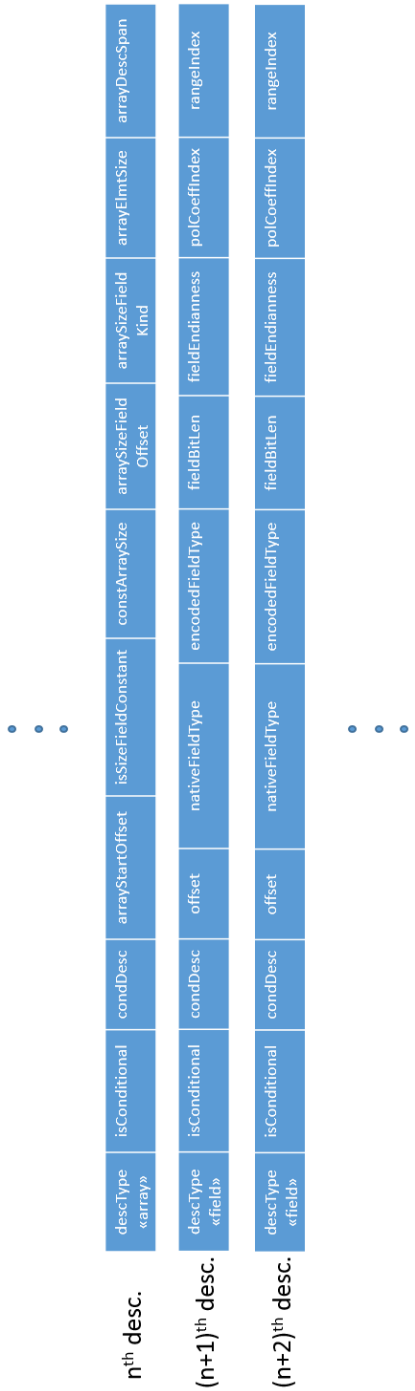

Fig. 1. Nested array description example

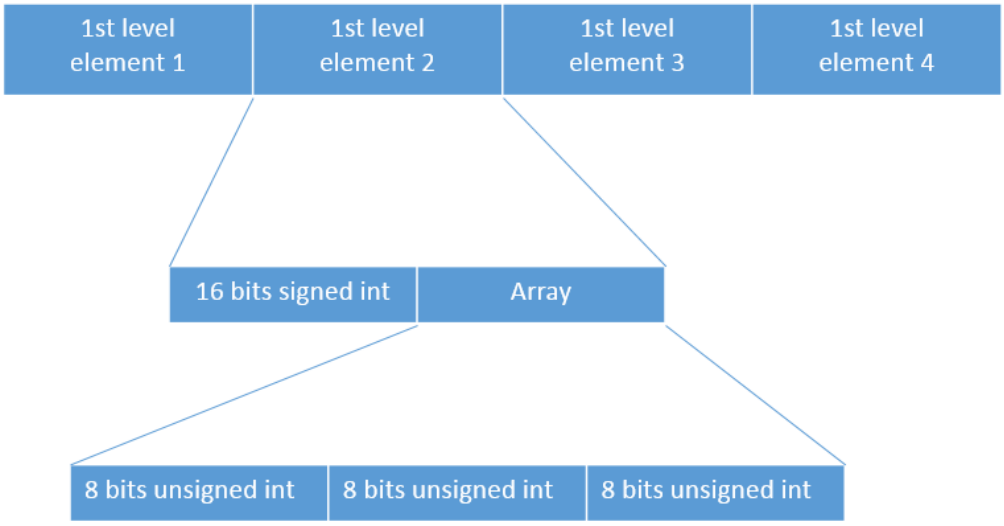

Fig. 2. Illustration of a nested array

2.7 The Codec Table

The field descriptions metadata is provided to the encoder or decoder within a codec table. One can define the codec table as the metadata of the field descriptions metadata. Components of the codec table is given below:

- Number of field descriptions & field descriptions array,
- Number of range descriptions & range descriptions array,
- Number of polynomial coefficients & array polynomial coefficients,
- Byte parsing direction.

Finally, considering the TC transmission; a TC bit stream is transmitted from GS to the FSW together with the codec table for the FSW to decode the TC bit stream. Similarly, GS utilizes the codec table to encode the upper layer TC structure to the TC bit stream with the minimum number of bits to represent the whole data. Block diagrams of TC encoding in GS, and TC decoding in FSW are given in Figure 3 and Figure 4, respectively. The opposite direction of the same pathway is applied in TM transmission.

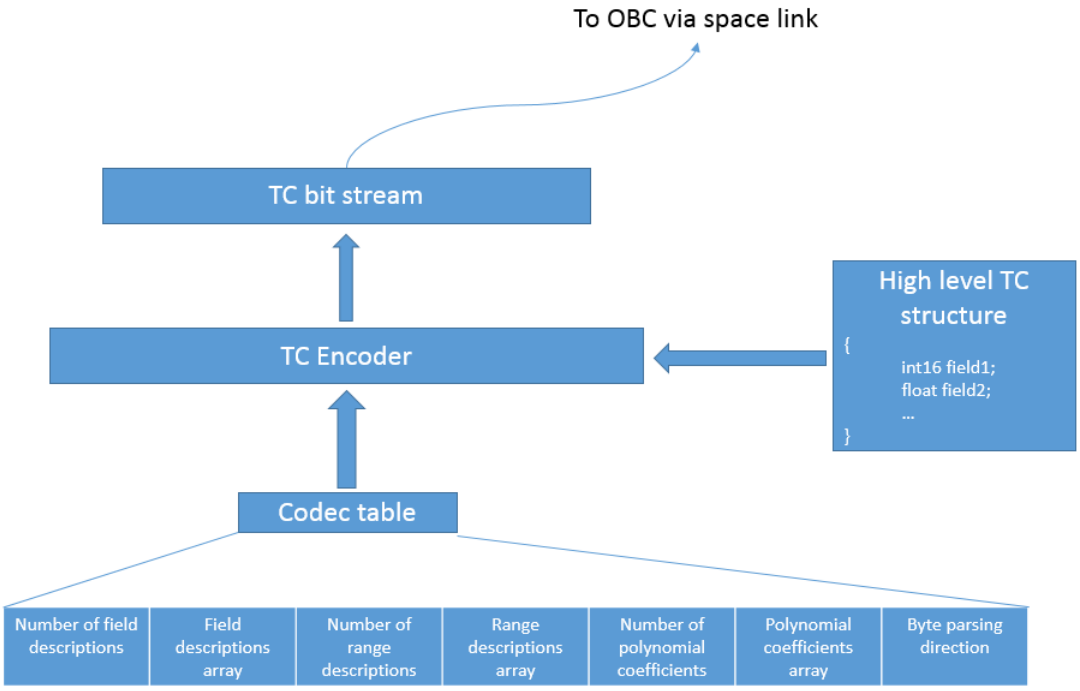

Fig. 3. Block diagram for TC encoding in GS

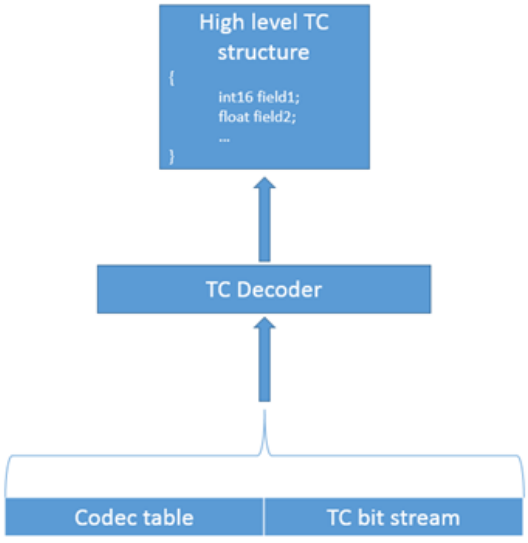

Fig. 4. Block diagram for TC decoding in FSW

3 Conclusion and Future Work

In this work, we developed a metadata structure which guides the GSW and the FSW through the encoding and decoding of data frames. The proposed metadata structure minimizes the bit length of the transmitted data frame, reducing the bandwidth needed per data frame, which is a critical parameter especially for the LEO satellites [6]. Since the metadata architecture is hierarchical, any TM or TC data structure can be defined in XML format, so that the metadata code for that data frame can be generated. This also enables easy tailoring of TM and TC structures. The encoder and decoders within this work are designed in iterative fashion, while it is possible to utilize recursion. Recursive implementation decreases the code size and complexity of the encoder and decoder. On the other hand, it is not considered safe to implement on the onboard computer, since it may lead to fatal errors when in erroneous codec table cases. Besides all these; considering the approximate code size of TM/TC interfaces in space projects, the proposed metadata format may reduce the TM/TC interface development effort significantly. As the future work; we are planning to build a specific language together with an editor to generate XML based metadata models for telecommands and telemetries. This is supposed to prevent syntax errors as well as architectural errors in producing XML files representing data models.

References

1. Balint, T.S.: Nuclear systems for Mars exploration. IEEE Aerospace Conference Proceedings 2004, vol. 6. IEEE (2004). https://doi.org/10.1109/AERO.2004.1368102
2. Yoder, J. W. et al.: Architecture and Design of Adaptive Object-Models. ACM Sigplan Notices, Vol. 36, pp. 50–60 (2001)
3. Thomé C. et al.: SICSDA: An adaptive configurable distributed software architecture applied to satellite control missions. (2010)
4. Simon, G. et al.: XTCE: a standard XML-schema for describing mission operations databases. IEEE Aerospace Conference Proceedings 2004, IEEE (2004). https://doi.org/10.1109/AERO.2004.1368138
5. CCSDS Secretariat: Telecommand Data Management Service. NASA, Washington, DC 20546, USA (2001)
6. Belokonov, I. et al.: The Technology of LEO Satellite Communication Systems Utilization for the Rapid Exchange of Data with the Low-altitude Spacecraft: Scientific Technological Equipment “Kontakt-MKA” On the Small Spacecraft “AIST-2”. Procedia Engineering, vol. 204, pp. 147–156 (2015)