USING BINARY FILE FORMAT DESCRIPTION LANGUAGES FOR DOCUMENTING, PARSING, AND VERIFYING RAW DATA IN TAIGA EXPERIMENT

I. Bychkov [1,2], A. Demichev [3], J. Dubenskaya [3], O. Fedorov [4], A. Hmelnov [1,2], Y. Kazarina [4], E. Korosteleva [3], D. Kostunin [5], A. Kryukov [3], A. Mikhailov [1, a], M.D. Nguyen [3], S. Polyakov [3], E. Postnikov [3], A. Shigarov [1,2,b], D. Shipilov [4], D. Zhurov [4]

[1] *Matrosov Institute for System Dynamics and Control Theory, SB RAS, Irkutsk, Russia*

[2] *Irkutsk State University, Irkutsk, Russia*

[3] *Skobeltsyn Institute of Nuclear Physics, Lomonosov Moscow State University, Moscow, Russia*

[4] *Applied Physics Institute, Irkutsk State University, Irkutsk, Russia*

[5] *Institute for Nuclear Physics, Karlsruhe Institute of Technology, Karlsruhe Germany*

E-mail: [a] mikhailov@icc.ru, [b] shigarov@icc.ru

The paper is devoted to the issues of raw binary data documenting, parsing and verifying in astroparticle data lifecycle. The long-term preservation of raw data of astroparticle experiments as originally generated is essential for re-running analyses and reproducing research results. The selected high-quality raw data should have detailed documentation and accompanied by open software tools for access to them. We consider applicability of binary file format description languages to specify, parse and verify raw data of the Tunka Advanced Instrument for cosmic rays and Gamma Astronomy (TAIGA) experiment. The formal specifications are implemented for five data formats of the experiment and provides automatic generation of source code for data reading libraries in target programming languages (e.g. C++, Java, and Python). These libraries were tested on TAIGA data. They showed a good performance and help us to locate the parts with corrupted data. The format specifications can be used as metadata for exchanging of astroparticle raw data. They can also simplify software development for data aggregation from various sources for the multi-messenger analysis.

Keywords: data format description language, binary data, astroparticle physics, data lifecycle management

1. Introduction

The current trend in science is that the researchers from all over the world can immediately get access to research data as soon as they are published. An important topic for modern science in general and astroparticle physics in particular is open science, the model of free access to data (e.g. [1]). It declares that scientific data should be accessible not solely to collaboration members but to all levels of an inquiring society. This approach is especially important in the age of Big Data, when a complete analysis of the experimental data cannot be performed within one collaboration.

Some experiments in astroparticle physics have already adopted this fascinating idea. They have involved their scientific data in electronic publishing, such as KCDC (KASCADE Cosmic ray Data Centre) [2]. KCDC is a web portal where KASCADE-Grande [3] scientific data are made available for the interested public. KCDC is driven within KASCADE-Grande experiment, which is already dismantled. However, TAIGA [4], an operating experiment in Russia, is producing the data for more than ten years. Obviously, various activities should be performed continuously across all stages of the data life cycle in these experiments: collection and storage of data, its processing and analysis, refining the physical model, publication and share, as well as archive and reuse of the data in the future.

One of the important issues is how to efficiently curate raw binary data to support their availability and reuse in future. TAIGA has five unique binary file formats for representing raw data: TAIGA-IACT, Tunka-HiSCORE [5], Tunka-133 [5], Tunka-Grande [6], and Tunka-REX [7]. The long-term preservation of raw binary data as originally generated is essential for re-running analyses and reproducing research results. To be accessible for the scientific community the raw data should be documented in details and accompanied by open and freely available software for accessing to these data. The neglect of this issues may lead to the need for a reverse engineering of their formats.

The state-of-the-art toolsets for formal describing binary data formats provide a satisfactory solution for the issues of raw data documenting, parsing and verifying. This work demonstrates applicability of binary file format description languages to specify, parse and verify raw data of TAIGA experiment. The formal specifications implemented for five formats of the experiment gives possibility for automatic generation of source code of data accessing libraries in target programming languages (e.g. C++, Java, and Python). These libraries were tested on real data. They demonstrated a good performance and helped us to locate files with corrupted data. This result shows ways for describing binary file formats for astroparticle raw binary data share and reuse. It can be interested in other experiments where raw binary data formats remain weakly documented or some parsing libraries for contemporary programming languages are required.

2. Binary Data Format Description Languages

There are several alternatives for formal specification of a binary data format. Some of them allow one to generate program libraries for reading binary data in specified formats. Here we consider some of them to choose ones for describing raw data of astroparticle experiments.

Some tools for specifying network protocols (e.g. NetPDL [8], NetPDLFltr [9] and BinPAC [10]) can serve for describing binary file formats. Since the nature of network protocols requires the sequential reading of data, these languages are appropriate only for formats with a sequential form of information storage (i.e. without pointers). Some of them provide also a program code generation for processing of binary data. NetPDLFltr generates binary code for network packet filtering. BinPAC allows one generating C++ code for reading packets. The listed tools are very specialized for the network protocols. HUDDL [11], an XML-based language, serves for specifying hydrographic data formats. It is intended to describe streams of binary data. HUDDL specifications can be used for generating a source code in C, C++, and Python language for data reading. Typically, hydrological files contain some sequences of measurements. Therefore, the generated code performs a sequential reading of the data blocks.

The parser generators, namely ANTLR [12] and Bison [13], can be used for automatically building of program code for data reading. However, the use of a binary file format specification as a grammar imposes severe limitations on its capabilities. The parser generators require presenting a binary file format specification as a grammar with fragments in a target language. As a result, the specification is not declarative. In DataScript [14] data format specifications are used to generate libraries for reading data in Java language. Instead of pointers, DataScript uses labels that contain expressions with file fragment addresses. A specification is considered as a set of data type definitions. A separate set of simple bit data types (bit fields) allows one describing bit-oriented data. However, the project has not been updated since 2003 [15].

FlexT [16], adeclarative language, is intended for presenting specifications of binary data formats. Its syntax allows one expressing the specifications in a neat and well understandable form. FlexT is accompanying by a code generator that can produce data reading source code in the imperative languages: Pascal and C++. Now, it implements the code generation for the most widely used data types, but some complex types like that used in specifications of machine instruction encoding are not supported yet. Kaitai Struct [17] toolkit suggests a declarative language for describing formats of binary data and network packets. The language is primarily designed to describe communication protocols and container data formats. Specifications presented in this language can be translated into a source code for reading files in the one of the supported programming languages: C++, Java, JavaScript, Perl, PHP, Python, Ruby, and Go. The toolkit implements a set of standard methods, which implement reading data from the stream in accordance with the type from the description. Its approach to code generation is similar to DataScript: all the data from the stream are sequentially loaded into the fields of the data structures.

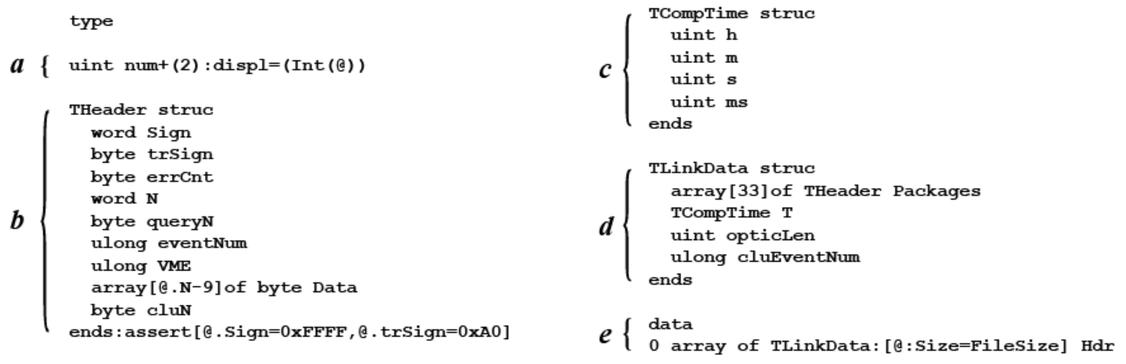
Among the tools listed above, FlexT and Kaitai Struct are the most suitable to be used in our case. Both provide the declarative languages for presenting file format specifications. Similarly, they consider a specification as a set of data type definitions. They support bit-oriented data (bit fields) and variant blocks. Both allow one generating source code of reading libraries for the raw data formats from the specifications. FlexT language is more expressive, but Kaitai Struct language is based on well-known format, namely YAML. Moreover, Kaitai Struct supports more programming languages for the source code generation. We used both of them, for formally describing the raw data formats of TAIGA experiments. As a result, we generated reading libraries for each file format in the widespread programming languages including C/C++, Java and Python.

3. Using Binary File Formats Specifications for Astroparticle Experiments

The considered raw data are generated and transmitted as packages by the facilities using the TCP protocol. Their file formats implement containers with simple structure. The developed format specifications consequentially match byte streams against data structures interpreting them. First, a specification optionally introduces format metadata and then it defines the section of data types and the data section containing definitions of variables.

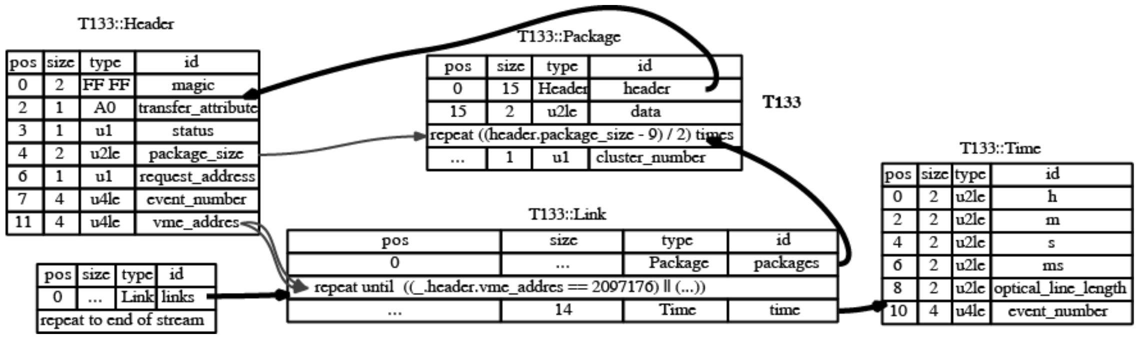
A specification expressed in FlexT consist of some definition blocks. The main blocks are the following: **const** defines constants that are results of calculations with variables of data blocks; **type** contains data type definitions; **data** is the data block with definitions of variables; **code** presents code block including address (shifts from the beginning of a file) and names of code parts. Figure 1 shows the simplified specification of Tunka-133 format specification expressed in FlexT language: (a) **— unit**, a type for displaying time as two digits of integer; (b) **—** THeader, a header of data package; (c) **— TCompTime**, a computer time of a registered event; (d) **— TLinkData**, a data container using the defined types (**TCompTime, THeader**); (e) **— data** section specifies that data are an array of **TLinkData** structures.

A specification presented with Kaitai Struct includes the following blocks: **meta —** metadata (section describing format name, version, file extension); **doc** — a description of regular fields; **seq** sequentially lists definitions of variables used in the described format; **instances —** a description of fields that require additional processing; **enums** matches integer constants and some names; **types** contains user-defined data types. Figure 2 demonstrates the auto-generated diagram for Tunka-133 file format specification presented in Kaitai Struct. It shows the defined sections of the specification and the relations between their definitions.



```
    type

a { uint num+(2):displ=(Int(@))

  { THeader struc
  {   word Sign
  {   byte trSign
  {   byte errCnt
  {   word N
b {   byte queryN
  {   ulong eventNum
  {   ulong VME
  {   array[@.N-9]of byte Data
  {   byte cluN
  { ends:assert[@.Sign=0xFFFF,@.trSign=0xA0]

  { TCompTime struc
  {   uint h
c {   uint m
  {   uint s
  {   uint ms
  { ends

  { TLinkData struc
  {   array[33]of THeader Packages
d {   TCompTime T
  {   uint opticLen
  {   ulong cluEventNum
  { ends

e { data
  { 0 array of TLinkData:[@:Size=FileSize] Hdr
```

Figure 1. Simplified specification of TUNKA-133 file format expressed in FLEXT language



T133::Header

pos	size	type	id
0	2	FF FF	magic
2	1	A0	transfer_attribute
3	1	u1	status
4	2	u2le	package_size
6	1	u1	request_address
7	4	u4le	event_number
11	4	u4le	vme_addres

T133::Package

pos	size	type	id
0	15	Header	header
15	2	u2le	data
repeat ((header.package_size - 9) / 2) times			
...	1	u1	cluster_number

pos	size	type	id
0	...	Link	links
repeat to end of stream			

T133::Link

pos	size	type	id
0	...	Package	packages
repeat until ((_.header.vme_addres == 2097176) || (...))			
...	14	Time	time

T133::Time

pos	size	type	id
0	2	u2le	h
2	2	u2le	m
4	2	u2le	s
6	2	u2le	ms
8	2	u2le	optical_line_length
10	4	u4le	event_number

Figure 2. Auto-generated diagram for TUNKA-133 file format specification presented in KAITAI STRUCT

We implemented the specifications for all file formats of the considered experiments in both FlexT and Kaitai Struct languages. It allowed us generating automatically source code of program libraries represented in the target programming languages (C++, Java, Python, etc.) for parsing and verifying the raw experiment data.

6. Conclusion

The best practices of scientific data maintenance recommend keeping raw (unprocessed) data [18-19]. This allows facilitating future re-analysis with some improved analytical and data processing techniques as well as analytical reproducibility of published results. Astroparticle experiments involve with accumulating and processing a big volume of raw data. Each experiment uses some specific file formats for representing raw data. The considered case with raw data TAIGA experiment showed that such file formats could be insufficiently and weakly documented. Only a few experts involved in the experiments can interpret these data.

The paper considers our research experience on describing formally the binary file formats used in TAIGA experiment. We used FlexT and Kaitai Struct toolsets to specify, parse and verify raw data of these formats. The implemented format specifications allowed us to generate source code for parsing and verifying the raw binary data in the target languages. The libraries were evaluated on real data. Tunka-133, Tunka-Grande, and Tunka-REX formats were tested on about 89K files.Thanks to these libraries, we found that about 1.2% of these files contain corrupted data. TAIGA-IACT and Tunka-HiSCORE formats were tested on about 120K files and 0.6% of them contains corrupted data. We suggest to use this approach for exchanging of astroparticle raw data. They can also simplify the software development for data aggregation from various sources in the case of multi-messenger analysis. We plan to share our experience of exporting raw data with other scientific collaborations.

Acknowledgments

This work was financially supported by the Russian Scientific Foundation (Grant No 18-41-06003).

References

[1] David P. Understanding the emergence of `open science' institutions: functionalist economics in historical context // Industrial and Corporate Change. 2004. V. 13, N. 4. P. 571-589.

[2] Haungs A. et al. [KASCADE-Grande Collaboration]. The KASCADE Cosmic-ray Data Centre KCDC: Granting Open Access to Astroparticle Physics Research Data // ARXIV:1806.05493. 2018.https://arxiv.org/abs/1806.05493

[3] Apel W.D. et al. [KASCADE-Grande Collaboration]. The KASCADE-Grande experiment // Nucl. Instrum. Meth. 2010. V. 620, Issues 2-3, P. 202-216.

[4] Budnev N. et al. [TAIGA Collaboration]. The TAIGA experiment: from cosmic ray to gamma-ray astronomy in the Tunka valley // Nucl. Instrum. Meth. 2017. V. 845, P. 330-333.

[5] Prosin V. et al. [TAIGA Collaboration]. Results from Tunka-133 (5 years observation) and from the Tunka-HiSCORE prototype. EPJ Web Conf. 2016. 121. 03004.

[6] Monkhoev, R.D. et al. [TAIGA Collaboration]. The Tunka-Grande experiment: Status and prospects. Bull. Russ. Acad. Sci. 2017. 81. 468–470.

[7] Bezyazeekov P. et al. [TAIGA Collaboration]. Measurement of cosmic-ray air showers with the Tunka Radio Extension (Tunka-Rex). Nucl. Instrum. Meth. 2015. A802. 89–96.

[8] Risso F., Baldi M. NetPDL: An extensible XML-based language for packet header description // Comput. Netw. 2006. V. 50, N. 5. P. 688-706.

[9] Morandi O., Risso F., Baldi M., Baldini A. Enabling flexible packet filtering through dynamic code generation // Proc. IEEE Int. Conf. on Communications. 2008. P. 5849-5856.

[10] BinPAC. Available at: https://www.bro.org/sphinx/components/binpac/README.html.

[11] Calder B.R., Masetti G. Huddler: A multi-language compiler for automatically generated format-specific data drivers // Proc. U.S. Hydrographic Conference. 2015.

[12] Parr T. ANTLR. Available at: http://www.antlr.org.

[13] GNU Bison. Available at: https://www.gnu.org/software/bison.

[14] Back G. DataScript - A Specification and Scripting Language for Binary Data // Proc. 1st ACM SIGPLAN/SIGSOFT Conf. on Generative Programming and Component Engineering. 2002. P. 66-77.

[15] Back G. DataScript. Available at: http://datascript.sourceforge.net.

[16] Hmelnov A., Bychkov I., Mikhailov A. A declarative language FlexT for analysis and documenting of binary data formats // Proc. ISP RAS. 2016. V. 28, N. 5. P. 239-268.

[17] Kaitai Struct. Available at: http://kaitai.io.

[18] Wilson G., Bryan J., Cranston K., Kitzes J., Nederbragt L., Teal T.K. Good enough practices in scientific computing. PLOS Comp. Biology. 2017.

[19] Hart E.M., Barmby P., LeBauer D., Michonneau F., Mount S., Mulrooney P., Poisot T., Woo K.H., Zimmerman N.B., Hollister J.W. Ten Simple Rules for Digital Data Storage. PLOS Comp. Biology. 2016.