Requirements for OAIS Structure Representation Information

Stephen E Rankin [1], David Giaretta [2]

[1] ***STFC***
Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot, OX11 0QX, UK
EMail: stephen.rankin@stfc.ac.uk

[2] ***STFC***
Rutherford Appleton Laboratory, Harwell Science and Innovation Campus, Didcot, OX11 0QX, UK
EMail: david.giaretta@stfc.ac.uk

ABSTRACT

OAIS Structure Representation Information (Structure RepInfo) specifies the mapping of the bits of a digital object to primitive data types such as integers, floating point values etc. Additionally, it also allows more complex logical structures to be described such as the grouping of data values into related structures (records). The requirements for Structure RepInfo have been acquired by studying many data formats and several formal data description languages. These requirements are stated and then compared against existing structure description languages such as Enhanced Ada Subse T (EAST) and Data Request Broker (DRB), highlighting their ability to be used to formally describe complex digital structures and file formats. It is found that for the more complex file formats the exiting structure description languages cannot be used to create adequate Structure Repinfo, but in some simpler cases the EAST language can be used to provide a perfectly adequate description of the structure that can be used for preservation purposes. Finally the need for formal descriptions of file formats is discussed, including topics such as their advantages for data validation, preservation and reuse.

Keywords: OAIS, Representation Information, Structure, File Format

INTRODUCTION

This paper serves several purposes. Firstly there is a need to give a practical meaning to OAIS[1] Structure RepInfo by defining an abstract description of what should be contained within it. Secondly, by doing so, we enable the reader to identify, in their own existing documentation, what would be "good" Structure RepInfo to curate for their data. The term "good" Structure RepInfo is somewhat difficult to quantify and depends on many factors, the three main factors are (which incidentally apply to any type of RepInfo, and not just to Structure RepInfo):

- What does a piece of RepInfo allow someone to do with the data - what is its use case? Alternatively, what do you expect people to do with the data, and what information about the data will allow them to do it?
- How long into the future do you expect the data and RepInfo to be used before further RepInfo is needed?
- Who is supposed to be using the RepInfo and data, and what is their expected background knowledge i.e. in OAIS terms, what is the Designated Community?

Inevitably there can never be a complete set of definitions for Structure RepInfo about data. This is simply due to the fact that data is so varied and complex. But this paper at least highlights, albeit in some detail, the most important characteristics, as judged by studying many data formats and several formal data description languages.

We introduce first an abstract notions of Structure RepInfo, then existing tools and standards can also be described that may be helpful in creating it if one discovers that existing Structure RepInfo is inadequate or non-existent. Most of these tools are limited in what they can do, and we will highlight exactly what they can and cannot be used to describe. Most of the tools generate Structure RepInfo in accordance to some formal standard and format, sometimes with a well defined encoding. This has the significant advantage that the Structure RepInfo can then be used in automated tools, as opposed to traditional documentation which requires human interpretation.

STRUCTURE REPRESENTATION INFORMATION

Structure RepInfo can be broken down into levels, the first level being the structure of the bits and how they map to data values. This involves the exact specification of how the bits contain the information of a data value and involves the definition of several generic properties. This bit structure will be referred to as the Physical Data Structure, and often is dictated by the computing hardware on which the data was created and the programming languages used to write the data. Data values are then grouped together in some form of order (that may or may not have meaning) which will be described as the Logical Data Structure.

The background knowledge required to understand the Structure RepInfo is assumed to be that of someone with a reasonable knowledge of computing, such as a software developer. It is not assumed that the user has any background knowledge about the data that the Structure RepInfo describes.

Physical Structure Information

The Bits

Digital data is just a sequence of bits, which, if the structure of those bits is undefined, is meaningless to hardware, software or human beings. Bits are usually grouped together to encode and represent some form of data value. Here we will use the term "Primitive Data Type (PDT)" as the description of the structure of the bits and "Data Value (DV)" as an instance of a given PDT in the data. The exact nature of the structure of the different PDTs will be discussed in the following sections, but for now we can summarise the ten PDTs as Integer, Character, Array, String, Boolean, Real Floating Point, Enumeration, Marker, Record or Custom. All other PDTs can that can be found in digital data can be derived from these types.

One other important organisational view of data is viewing the data as sequence of octets (eight bit bytes - bytes have varied in bit size through the history of computing but currently eight bits is the norm). PDTs are composed of one or more octets and the order in which the octets are read is important. This ordering of the octets is usually called byte-order and is a fundamental property of the PDT. There are two types of byte-order in common use (others types do exist), big-endian and little-endian.

Characters

Characters are digital representations of the basic symbols in human written language. Typically they do not correspond to the glyph of a written character (such as an alphabetic character) but rather are a code (code point) which can be uses to associate with the corresponding glyph (character encoding) or some other representation.

One of the most common character encodings is ASCII. ASCII was extended to use octets with the development of ISO/IEC 8859 giving a wider set (255) character encodings. ISO/IEC 8859[2] is split over 15 parts where the first part is ISO/IEC 8859-1 is the Latin alphabet no. 1. Each part encodes for a different set of characters and so a given encoding value (158 say) can correspond to different characters depending on what part is used. Typically a file containing text encoded with say ISO/IEC 8859-1 would not be interpreted correctly if decoded with ISO/IEC 8859-2, even though they are both text files with eight bit characters. The encoding standard used for a text file is thus very important Structure RepInfo.

Recently a new set of standard have been developed to represent character encodings, these new standards are called Unicode[3]. Unicode comes with several character encodings, for example UTF-8, UTF-16 and UTF-32. UTF-8 is intended to be backwards compatible with ASCII, in that it needs one octet to encode the first 128 ASCII characters.

Integers

Integers come in a variety of flavours where the number of bits composing the integer varies or the range of the numbers the integer can represent varies. Typically there are 8, 16, 32, 64 and 128 or more bits in integer types.

Integers can also be signed. Usually the most significant bit is the sign bit (but can be located elsewhere in the octets), zero for positive and one for negative. The rest of the bits are used to represent the decimal values of the number.

The interpretation of the bits is done using some interpretation scheme, for example, two's compliment interpretation. There are other ways of interpreting integers, such as sign-and-magnitude, one's compliment etc. This method of interpretation is a fundamental property of digital integers.

Real Floating Point Numbers

Floating point numbers draw their notation from the fact that the decimal point can vary in position. Their notation is usually the same as the scientific notion for real numbers i.e., $1.49243\text{x}10^{-3}$, where there is a base (which in this case it is base 10), an exponent (which in this case is -3) and a significand (mantissa) which is the significant digits 149243 having a precision of 6 digits. The decimal point is assumed to be directly after the left most digit when reading left to right. But in data and in a computer system the representation of floating point numbers is done in binary, for example, $1.011\text{x}2^{1011}$. Here the base is 2 and the exponent value has a binary representation along with the significand. Usually the number is normalised in that the decimal point is assumed to be directly after the left most non-zero digit reading left to right, as this digit is then guaranteed to be 1. This digit can then be ignored and the significand reduced to 011 (what is actually stored in the data).

The significand, as with integer values, can be interpreted as a two's compliment number, one's compliment number or some other interpretation scheme. The exponent is also usually subject to some interpretation scheme to get a signed integer value, typically this is a bias scheme where the number is first treated as an unsigned integer and then some bias is deducted from it. Also there will be a sign bit to apply to the final number where a 0 may represent a positive number and a 1 a negative number.

Sometimes some bit patterns in the exponent and the significand are reserves to represent floating point exceptions.

The exact location of the bits that correspond to the significand, exponent and sign bit also needs to be known.

The IEEE 754[4] standard is good RepInfo for data files that contain IEEE 754 floating point values and it should be expected that and Structure RepInfo describing data should give the type of floating point values being used, i.e. via a reference to the IEEE 754 standard or other documentation describing the bit structure of the values if they are not IEEE 754. Not all data uses IEEE 754 floating point values. For example data produced from VAX systems have a very different floating point format. A list of floating point formats and their respective structure can be found in the CCSDS green book[5], though this is not a comprehensive list.

Markers

In some instances it may be necessary to terminate a sequence of DVs in a data file with a marker. This allows the number DVs to be variable. The marker could be a DV of any of the PDT that have a size grater than zero and can be made unique (a value that other DVs are guaranteed not to take), such PDTs are usually Integer, Real Floating Point, Character, and String. An important marker is the End of File (EOF) marker. Although there is no specific value held in data representing the EOF, the operating system usually provides some indication to software that the EOF has been reached. This can be used by some data reading software to find the end of a particular structure. For example, you may need to keep reading DVs from a file until the EOF has been reached.

Enumerations

Enumerations are simply a lookup table, also named a Hash Table. Usually it consists of two columns of values where each column has values of a single PDT. The first column is referred to as the "keys" while the second column is referred to as the "values". When a data structure in the data file is indicated to contain values that are to be "looked up" (enumeration type) the enumeration is used to find the correct value by reading the DV from the file and then finding the corresponding value in the enumeration. So here the DVs in the data file are "keys" and its corresponding values in the enumeration are the "values". Enumerations can be used where data has only a fixed number of values, say ten names of people in a family (Strings). The names can then be represented as 8bit integer values (1 to 10 say). Here the 8bit value would be stored in the data, and when reading the data the enumeration would be used to "look up" the name as a string. This also results in a reduction of the number of octets used in the data as a name as a string will be composed of a number of 8bit characters, but the stored data is only one 8bit integer.

Records

Records are purely logical containers and do not have any size. More shall be said about record later when talking about the logical structure.

Arrays

Arrays are simply sequences of DVs that can have one or more dimensions (a one dimensional array is just a list of values). The dimensions of an array are a significant property and may be static (defined in the Structure RepInfo) or dynamic. If the dimensions are dynamic then there will be a DV in the data file that will give the value of the number of dimensions, i.e. an integer or a numerical expression to calculate the number of dimensions from one or more DVs. Restrictions may also exist on the number of dimensions, i.e. the maximum or minimum and also if there are only a fixed number of dimensions allowed (for example, fixed dimensions of 1, 3, 6 and 10). Another significant property of arrays is the ordering of the values – either row order or column order.

The size of each dimension (the number of array values in that dimension) will either be static (stated in the Structure RepInfo) or dynamic. If the size of each dimension is dynamic then there will be a DV in the data file that will give the value of the size i.e. an integer or a numerical expression to calculate the size from one or more DVs. Restrictions may also exist on the size of the dimensions, i.e. maximum and minimum values and also if there are only a fixed number of sizes allowed (for example, fixed sizes of 1, 3, 6 and 10). If the arrays are dynamic then it may also be possible that a marker of a given type is used to represent the end of a dimension or the array.

Strings

Strings are simply a one dimensional array of characters. They can be mixed with other PDTs in binary data or they can exist on their own usually in text files. The most important basic characteristic is that of the character PDT used in the string (ASCII, UTF-8 etc).

Strings can be structured or unstructured. When a string is structured it means that it contains a known set of sub-strings each of which may or may not contain a limited set of characters. The most common way of defining the structure of stings is using a variant of the *Backus Naur Form* (BNF). One standard exists for BNF and that is Extended Backus Naur Form (EBNF) - ISO-14977 [6]. Most text file formats, for example XML[7], use their own definitions of BNF. BNF is used as a guide to producing parsers for a text file format, BNF is not machine processable and has not been used to automatically generate code for parsers. Usually a parser generator library is used to map the BNF/EBNF grammar to the source code which involves handwriting code using the grammar as a guide. Some common libraries are Yet Another Compiler Compiler (Yacc)[8] and the Java Compiler Compiler (JavaCC)[9]. They are called compiler compilers due to the fact that they are used extensively in generating compliers for programming languages. The source files for programming languages are usually text files where their syntax (string structures) is defined in some form of BNF, see for example the C language standard [10].

BNF is not the only way of defining the structure of a string. Regular expressions can also be used. Regular expressions can be thought of in terms of pattern matching where a given regular expression matches a particular string structure. One advantage of regular expressions over BNF is that the regular expression can be use directly with software APIs that handle them. The Perl language for example has its own regular expression library that takes a specific form of regular expression and a string and outputs the locations in the string of the matching cases. Other languages such as Java also have their own built-in regular expression libraries. The main disadvantage of regular expression is the variability of their syntax (usually not the same for all libraries the support them). The Portable Operating System Interface (POSIX) [11] does define a standard regular expression syntax which is implemented on many UNIX systems. The other disadvantage is that the expressions themselves can increase considerably in complexity as the string structure complexity increases making them very difficult to understand and interpret.

Boolean

Boolean data values are a data type that represent true or false only. Boolean data values can have many different representations in data. The simplest is to have a single bit which can be either zero or one. But also a string could be used such as "true" or "false", and also an integer (of any bit size) could also be used as long as the values of the integer that represent true are false are specified. This makes the Boolean data type potentially a derived data type, but with restrictions on the values of the data type it is derived from.

Custom

Some data formats can take advantage of the fact that software languages allow the manipulation of data values at the bit level. In some data formats, particularly older data formats, bit packing was the norm due to memory and storage space constraints. The fact remains that a set of bits can be used to represent any information and so a requirement exists for custom data types to be defined from the bit level up.

Restrictions/Facets on Data Types

Restrictions to data types are important, for example, the length of a string in a file format may be restricted to a fixed number of characters. The characters in a string may also be restricted. The values of integers and real floating point numbers can have a minimum, maximum (or both) or fixed value.

Logical Structure Information

Strings and text files have been discussed above and their structure can, in the case of structured strings, be broken down into sub-structures (sub-strings). Similarly any binary file can be broken down into sub-structures ending in individual DVs of a given PDT. We will now concentrate on the logical structure of binary files. But binary files can also contain strings which are usually a fixed number of characters of a given character set. These strings may also have structure which can be further described by a BNF type description or regular expressions.

We can view binary data as just a stream of DVs of a given PDT. But this simple view is not usually helpful as it does not allow us to locate DVs that may be of particular interest, nor does it allow us to logically group together DVs that belong together such as a column of data values from table of data. With binary data DVs or groups of DVs can usually be located exactly if the logical structure is known in advance. The next sections show the common methods used in binary data that facilitate the logical structuring of DVs.

Location of Data Values

Numerous data file formats use offsets to locate DVs or sub-structures in binary data. For example, TIFF image files[12] contain an octet (byte) offset of the first Image File Directory (IFD) sub-structure, where an IFD contains information about an image and further offsets to the image data. The offset in this case is a 32 bit integer which gives the number of octets from the beginning of the file. Offsets are usually expressed in data as integers but the actual value may correspond to the number of bits, octets or some other multiplier to calculate the location exactly. Offsets may also be calculated from one or more DVs in the data, which requires the expression for the calculation to be stated in the structure RepInfo. In NetCDF[15] the location of the DVs for a given variable (collection of DVs) are calculated from a few DVs in the file, i.e. the initial offset of the variable in octets from the start of a file, the size in bits of the DVs and the dimensions of the variable (one, two or three dimensional array etc.)

Makers may also be use to locate DVs or sub-structures and to also indicate the type of sub-structure. The FITS file format[14] uses a markers to indicate the type of a given sub-structure, for example a FITS file can contain several types of data structure such as table data, image data etc. Each of these sub-structures is indicated with a marker, in the case of table data the marker is an ASCII string with the value "TABLE ". The end of the data sub-structure corresponding to the table data is also marked with the ASCII string value "END". Note, the data itself is in fact stored in binary format where additional "header" information is contained in fixed width ASCII strings.

Data Hierarchies

It is common to think of the structure of a data file as a tree of DVs and sub-structures. XML is a classic example of storing data in a tree like structure where an element may contain other child elements and they to may have children, and so on - see Figure 1. Viewing data in such a way gives logical view of the data as a hierarchy. More importantly, it also gives you a way of calculating the locations of DVs and sub-structures and a way of referencing them.

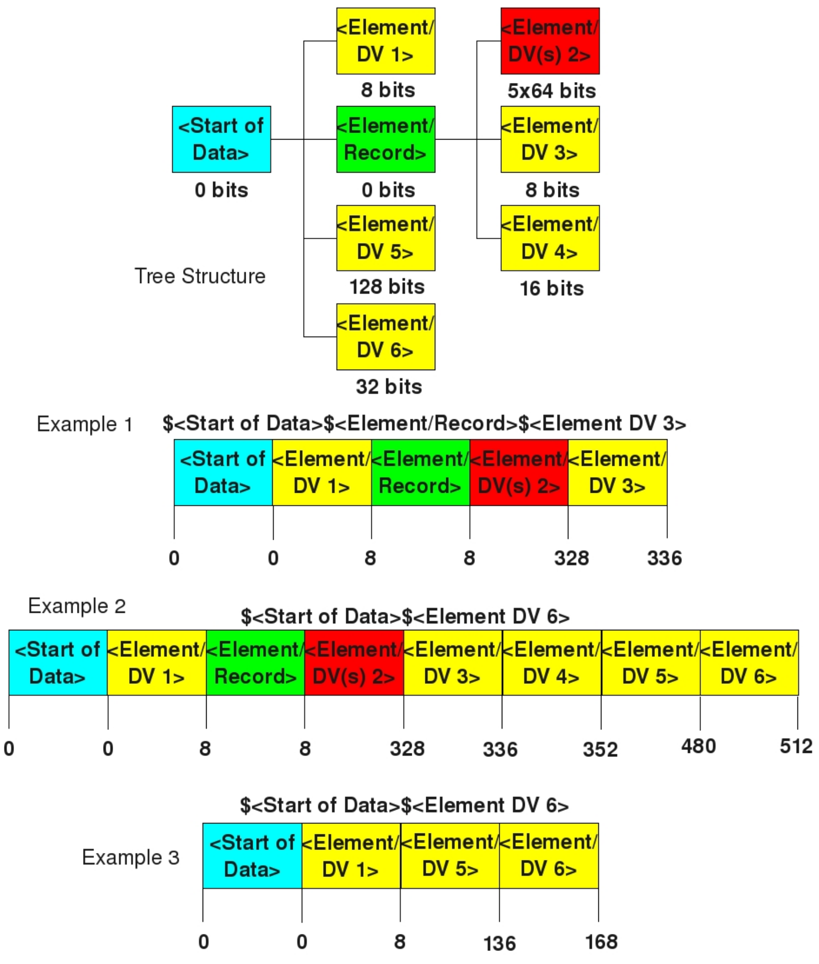

Figure 1: Data Hierarchies

DVs in a binary data file are in a sequence (one after the other), but the intended structure is usually a logical tree. Figure 1 shows a tree structure of several DVs, here only the size in bits of the DVs is important but for clarity sake we have indicated that the blue element is the start of the data file (at 0 bits and zero size and can also be considered as a record), yellow elements are individual values, green elements are containers or records (zero size) and a red elements are arrays of values.

We can think of walking through the tree starting at the location <Start of Data> and then going directly to <Element Record> and then to <Element DV 3>. Using this information it is possible to provide a simple statement (path statement) that represents this walk-through by separating each element name with a $ sign, so for this example (Example 1 in Figure 1) the path statement would be $<Start of Data>$<Element Record>$<Element DV 3>. Given the tree structure and the path statement you can reference a data element uniquely.

This path statement can be related to the exact location of the DV in the data file. To do this we first have to realise that elements in the same column in the tree (vertically aligned) that appear above the element we are trying to locate are located directly before it in the data file (as long as they are part of the same record). In this case <Element DV(s) 2> is in the same column and record in the tree as <Element DV 3> but it above it and so appears before it in the data file. <Element DV(s) 2> is actually an array of values and so there are in fact five 64bit DVs before it.

Adding a predicate to the path statement can allow the selection of an individual element of the array, for example, $<Start of Data>$<Element Record>$<Element DV(s) 2>[2], where the predicate represented as [2] indicated that the second element of the array should be selected.

Conditional Data Values

Elements or records in the logical structure may be conditional, which means that they may or may not exist, depending on the result of a logical expression (true if it exists or false if it does not exist). There may also be a choice of elements or records in the data from a list, where only one of the choices exists in the data.

A logical expression may consist of one or more DVs combined using the logical operators AND, OR, NOT etc. Typically the DVs in the expressions are either a boolean PTD or an integer data type that is restricted to have the values 0 or 1, they could also be the string “true” or “false”. The result of evaluating the expression will either be true or false (0 or 1) and will indicate whether the value exists (true) or not (false).

Another type of logical expression could be the identification of an element with a specific DV. For example, in the FITS format there are several different structures where each is identified by a keyword (String), so here an expression must exist that compares the value of the string against a lists of possible values, if it matches one then the appropriate structure is selected. Integer values are another possible DV that can be used for selecting structures.

Formal Structure Representation Information

Formal Structure Representation Information (FSRI) in this case takes the form of a formal language for capturing Structure RepInfo. There are three main languages that capture formally some of the properties outlined as required for Structure RepInfo above. Not all of these languages have been developed with preservation in mind. Their main purpose is to provide a framework for current data interoperability.

Enhanced Ada Subse T (EAST)

EAST[16] was developed by the CCSDS[27] to provide a language for data interchange specifying syntactic (structure) information about data. There are tools for producing EAST descriptions[21][22] and to use them to read and write data. EAST is an ISO standard.

EAST is fully capable of describing the physical structure of integer, real floating point and enumerations. It does not support boolean data types. The exception bit patterns of real floating point values are not supported. The byte-order for the data can be specified globally, but not on individual DVs. Characters are restricted to 8bit and the code points are specified in the EAST specification. String made up of 8bit characters are allowed with a fixed length. The appropriate restrictions and facets for strings are supported. The lack of ability to define dynamic offsets for the logical structure is the main restriction, file formats life TIFF cannot be described with EAST. No path language is specified in the EAST standard.

Data Request Broker (DRB)

DRB[19] is based on XML Schema[23] and XQuery[25] and uses some additional non-standard extensions to deal with binary data. The main restriction is that the physical structure of data types cannot be defined explicitly as with EAST. Byte-order can be specified for each DV, but the interpretation scheme for integers is restricted to two’s compliment and real floating point data types are assumed to be IEEE 754. XPath[24] can be used as a path language, and the XQuery API is also implemented for more complex data queries. Using XQuery complicates the language unnecessarily, potentially making the descriptions difficult to understand and software difficult to maintain or re-implement in the long-term.

Data Format Description Language (DFDL)

DFDL[20] is being developed (version one of the specification is under development) by the Open Grid Forum[26], with the main objective of providing syntactic data interoperability. In its current state it is difficult to assess due to the lack of implementations. It is similar to DRB in that it is based on XML

Schema and uses the XPath language. It is simpler than DRB in that it does not use XQuery. It cannot yet provide a complete physical description of real floating point types and integers like EAST, which appears to be its biggest restriction, though it does allow you to indicate that the data type is IEEE etc.

CONCLUSION

There are a number of benefits of having a formal description for the Structure RepInfo, these are:

- Machine readability of the FSRI, allowing analysis and processing.
- Common format for FSRI that can apply to may data formats giving a common (single) software interface to the data.
- Higher probability of future re-use due to having a single software interface.
- Easy validation of the data against the FSRI and also easy validation of the FSRI against its formal grammar.
- Ensures that all the relevant properties of the structure have been captured.

Machine readability of the FSRI is important as information about the structure can be easily parsed making the implementation of data access routines that use them easer to programme. This has the added benefit of a reduction in cost of producing software implementations now and in the future. Being able to process the FSRI also gives rise to the possibility for automating some aspects of data interoperability. For example, PDT of DVs and sub-structures such as arrays and records can be automatically discovered and compared between FSRIs which can allow the automatic mapping and conversion between different data formats.

Software can be produced that takes the FSRI and the data and produces a common software interface to the DVs and sub-structures. In effect you have one software interface that reads the DVs from many data files with different structures (formats). Having many FSRIs for many different data formats (XML Schema for example) increases the likelihood that an implementation will exist in the future, or if one does not exist, then the likelihood and motivation to produce one will be increased. Basically this is due to the value and amount of data that has been described (consider the vast number of XML schemas that exist for XML data). Currently though, binary data is not usually accompanied with FSRI, and their structure is usually described in a human readable document. But the relatively recent development of formal languages to describe binary data structures may change this if they are adopted more widely. Such an adoption would be highly beneficial for data preservation.

The current set of FSRIs are themselves formally described, for example, EAST and DRB are both described with a form of BNF as they are structured text based formats. This allows and instance of the FSRIs to be validated to ensure its structure and content follow the formal grammar. Having FSRI for data also allows you to automatically check that the data is written exactly in accordance with the FSRI, i.e. each instance of the data has the correct structure. This ability is important for data preservation for the following reasons:

- It can be used to check the valid creation of a data structure.
- It can be used to periodically check the data structure for errors or corruption (also useful in authenticity to check for deliberate structure tampering).
- It can be used to identify a data file accurately – it is accurate because knowledge about the whole data structure is used as opposed to simple file format signatures[17][18].

Properties that the FSRI highlights guide a person in capturing the relevant structure RepInfo that is required to read the DVs. Having a well thought out FSRI which ensures that all the relevant structure information is captured is possibly the most important thing for the preservation of data. The current set of FSRIs are good but are still incomplete. DRB and DFDL, for example, have been developed with data interoperability in mind. Developing these languages for data interoperability results in most of the requirements for Structure RepInfo, but a few important requirements can be neglected. Structure

RepInfo is intended to preserve the access to the data values in a digital data for the long-term. When doing this, care must be taken as to not neglect structure information about data that may be well known or “obvious” now, but may or may not be in the future. For example, the current FSRIs either restrict the types of logical data structure that can be described or fail to provide sufficient generality to describe the physical data structure (or both). EAST for example has most of the properties defined to provide an adequate description of the physical structure, but can be quite restrictive of the logical structures it can describe. But if one can describe a data file format with EAST then it is believed that it will provided a perfect FSRI for that data in terms of providing all the information required for long-term preservation of the structure.

REFERENCES

[1] - Reference Model for an Open Archival Information System (OAIS). Blue Book. Issue 1. January 2002. ISO 14721

[2] - Information technology -- 8-bit single-byte coded graphic character sets ISO/IEC 8859

[3] - UNICODE - http://www.unicode.org/standard/standard.html

[4] - IEEE Standard for Floating-Point Arithmetic – IEEE 754-2008

[5] - The Data Description Language EAST—List of Conventions. Green Book. Issue 1. May 1997 - http://public.ccsds.org/publications/archive/646x0g1.pdf

[6] - Extended Backus Naur Form (EBNF) 1996 ISO 14977 - http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[7] - Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation 16 August 2006, edited in place 29 September 2006 - http://www.w3.org/TR/2006/REC-xml11-20060816/

[8] - Yacc: Yet Another Compiler-Compiler - http://dinosaur.compilertools.net/

[9] - Java Compiler Compiler (JavaCC) - https://javacc.dev.java.net/

[10] - Programming languages — C ISO/IEC 9899:1999

[11] - Portable Operating System Interface (POSIX) - IEEE Std 1003 http://standards.ieee.org/announcements/opengroup.html

[12]- Tagged Image File Format (TIFF) V6 Specification - http://partners.adobe.com/public/developer/tiff/index.html

[13] - Matlab 4 Format as used for EISCAT data – http://e7.eiscat.se/groups/Documentation/UserGuides/matlab4.pdf

[14] - Flexible Image Transport System (FITS) - http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html

[15] - Network Common Data Form (NetCDF) - http://www.unidata.ucar.edu/software/netcdf/

[16] - The Data Description Language EAST Specification (CCSD0010). Blue Book. Issue 2. November 2000 ISO 15889 - http://public.ccsds.org/publications/archive/644x0b2.pdf

[17] - Digital Record Object Identification (DROD) - http://droid.sourceforge.net/

[18] - UNIX File - http://www.opengroup.org/onlinepubs/009695399/utilities/file.html

[19] - Data Request Broker (DRB) - http://www.gael.fr/drb/

[20] - Data Format Description Language (DFDL) V1.0 Draft - http://forge.gridforum.org/projects/dfdl-wg

[21] – BEST EAST Tools - http://debat.c-s.fr/index.html

[22] – CNES EAST tools - http://east.cnes.fr/english/index.html

[23] - XML Schema - http://www.w3.org/XML/Schema

[24] - XML Path Language (XPath) Version 1.0 - http://www.w3.org/TR/xpath

[25] - XQuery 1.0: An XML Query Language W3C Recommendation 23 January 2007 - http://www.w3.org/TR/xquery/

[26] - Open Grid Forum - http://www.ogf.org/

[27] - Consultative Committee for Space Data Systems (CCSDS) - http://public.ccsds.org/default.aspx