XDTM: The XML Data Type and Mapping for Specifying Datasets

Luc Moreau,[1] Yong Zhao,[3], Ian Foster,[2,3]
Jens Voeckler,[3] and Michael Wilde[2,3]

(1) Univ. of Southampton, (2) Argonne National Laboratory, (3) Univ. of Chicago

Abstract. We are concerned with the following problem: How do we allow a community of users to access and process diverse data stored in many different formats? Standard data formats and data access APIs can help but are not general solutions because of their assumption of homogeneity. We propose a new approach based on a separation of concerns between logical and physical structure. We use XML Schema as a type system for expressing the logical structure of datasets and define a separate notion of a mapping that combines declarative and procedural elements to describe physical representations. For example, a collection of environmental data might be mapped variously to a set of files, a relational database, or a spreadsheet but can look the same in all three cases to a user or program that accesses the data via its logical structure. This separation of concerns allows us to specify workflows that operate over complex datasets with, for example, selector constructs being used to select and initiate computations on sets of dataset elements—regardless of whether the sets in question are files in a directory, tables in a database, or columns in a spreadsheet. We present the XDTM design and also the results of application experiments with an XDTM prototype.

1 Introduction

In large open environments, users need to be able to access data stored in many different formats. An answer to this problem is the standardization of data formats and associated APIs. For example, XML is playing an increasingly important role in data exchange; FITS, the Flexible Image Transport System, is a standard method for storing astronomical data; and HDF5, the Hierarchical Data Format [5], is a file format (and associated software library) for storing large and complex scientific and engineering data.

Data and format standards are helpful in this context but are not general solutions in today's heterogeneous networked environments. Open environments must deal with legacy data, coexisting and evolving standards, and new data formats introduced to meet the needs of new applications or devices. Above all, open environments must be able to cope with evolution in data formats, so that ideally (for example) computational procedures do not require major changes to work with new formats. Format neutrality is hard to achieve, however, and we often encounter computational procedures that are no longer operational simply because their target data format no longer exists. The problem is that format dependency prevents code reuse over datasets that are *conceptually* identical, but *physically* represented differently.

BFD [11] and DFDL [2] offer partial solutions to this problem based on binary format descriptions: BDF offers converters from binary to some XML-based universal representation (and vice versa), whereas DFDL provides uniform access APIs to binary. They do not address the variety of existing datasets formats, however, since they focus on legacy binary formats. Moreover, they incur potentially expensive conversion cost to XML (both in time and space), and they do not promote reuse of computational procedures for logically equivalent datasets.

We propose a new approach to this problem based on a separation of concerns between logical and physical structure. We use a type system for expressing the logical structure of datasets, and we define a separate notion of a mapping to describe their physical representations. For example, a collection of environmental data might be mapped variously to a set of files, a relational database, or a spreadsheet but can look the same in all three cases to the user who accesses the data via its logical structure.

To explore the feasibility and applicability of these ideas, we define (and introduce here) XDTM, the *XML Dataset Type and Mapping* system. XDTM provides a two-level description of datasets: a *type system* that characterizes the abstract structure of datasets, complemented by a *physical representation description* that identifies how datasets are physically laid out in their storage. We adopt XML Schema [13, 3] as our type system; this technology has the benefit of supporting some powerful standardized query language that we use in selection methods. XDTM allows the specification of computational procedures capable of selecting subsets of datasets in a manner that is agnostic of physical representation.

Working within the GriPhyN project (www.griphyn.org), we have produced an implementation of XDTM and used it to manipulate datasets produced by the Sloan Digital Sky Survey (SDSS) [12]. Throughout this paper we use SDSS examples to demonstrate that the separation of concerns enabled by XDTM does indeed allow us to specify workflows that operate over complex datasets with, for example, selector constructs being used to select and initiate computations on sets of dataset elements—regardless of whether the sets in question are files, directories, or XML structures.

This paper is organized as follows. Section 2 reviews applications that use complex datasets and examines limitations that prevent their easy manipulation. Section 3 discusses the distinction between structure and format, while Section 4 describes how a type system, and specifically XML Schema, can be used to specify datasets structures. Section 5 focuses on a mapping that allows us to specify how a dataset's structure can be mapped to a physical representation. Section 6 overviews our prototype implementation. Section 7 discusses related work.

2 An Application Manipulating Complex Datasets

We introduce an application that has strong requirements for complex multi-format datasets. The Sloan Digital Sky Survey [12] maps one-quarter of the entire sky, determining the positions and absolute brightnesses of more than 100 million celestial objects, and measures the distances to more than a million galaxies and quasars. Data Release 2, DR2, is the second major data release and provides images, imaging catalogs, spectra, and redshifts for download.

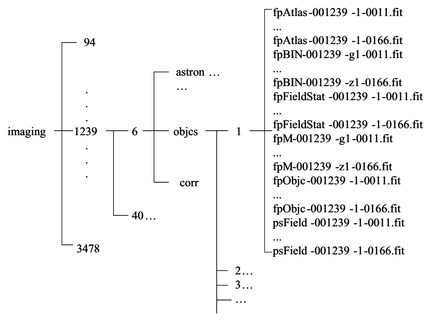
imaging
94
1239
3478
6
40…
astron …
…
objcs
corr
1
2…
3…
…
fpAtlas-001239 -1-0011.fit
...
fpAtlas-001239 -1-0166.fit
fpBIN-001239 -g1-0011.fit
...
fpBIN-001239 -z1-0166.fit
fpFieldStat -001239 -1-0011.fit
...
fpFieldStat -001239 -1-0166.fit
fpM-001239 -g1-0011.fit
...
fpM-001239 -z1-0166.fit
fpObjc-001239 -1-0011.fit
...
fpObjc-001239 -1-0166.fit
psField -001239 -1-0011.fit
...
psField -001239 -1-0166.fit

Fig. 1. Excerpt of DR2 (`http://das.sdss.org/DR2/data`)

We focus here on the directory `imaging` appearing at the root of the DR2 distribution, which we display in Figure 1. Inside it, we find a collection of directories, each containing information about a *Run*, which is a fraction of a strip (from pole to pole) observed in a single contiguous observing pass scan. Within a Run, we find *Reruns*, which are reprocessings of an imaging run (the underlying imaging data is the same, but the software version and calibration may have changed). Both Runs and Reruns directories are named by their number: in this paper, we will take Run 1239 and Rerun 6 as an example to focus the discussion. Within a Rerun directory, we find a series of directories, including `objcs`, *object catalogs* containing lists of celestial objects that have been cataloged by the survey. Within objcs, we find six subdirectories, each containing a *Camcol*, that is, the output of one camera column as part of a Run. In each Camcol, we find collections of files in the FITS format, the Flexible Image Transport System, a standard method of storing astronomical data. For instance, the *fpObjc* files are FITS binary tables containing catalogs of detected objects output by the frames pipeline. For these files, we see that the Run number (1239) and other information are directly encoded in the file name. From this brief analysis of a DR2 subset, we can see that DR2 is a complex hierarchical dataset, in which metadata is sometimes directly encoded in filenames, as illustrated for Run, ReRun, and CamCol directories and for all FITS files. Furthermore, DR2 is available in *multiple forms*: some institutions have DR2 available from their file system; it can also be accessed through the rsync and http protocols; alternatively, sometimes, astronomers package up subsets directly as tar balls, such as specific Runs, for rapid exchange and experimentation.

3 Distinguishing Structure from Format

XML Schema, standardized by the World Wide Web Consortium [13, 3], provides a means for defining the structure, content, and semantics of XML documents. We assert in this paper that XML Schema can also be used to express the structure complex datasets. Increasingly, XML is being used as a way of *representing* different kinds of information that may be *stored* in diverse systems. This use of XML implies not that all information needs to be converted into XML but that the information can be viewed *as if* it were XML. Such an overall approach is also supported at the programming level. The Document Object Model (DOM) [9] is an application programming interface for well-formed XML documents: it defines the logical structure of documents and a programmatic way of accessing and manipulating a document.

Against this background, we decided to adopt a multilevel explicit representation of information and propose to make the *physical representation* (i.e., the format) of datasets an explicit notion, to coexist with the *abstract structure* of the dataset (i.e., its type). We strive to specify format separately from type, so that a dataset of a given type can be encoded into different physical representations. Programs that operate on datasets can then be designed in terms of dataset types, rather than formats, but can be executed regardless of the physical representations.

4 Type System for Describing Datasets Structure

We now introduce the type system that we use to specify the abstract structure of datasets. As in programming languages, we consider two forms of aggregations: *arrays* make their contents referenceable by indices, whereas *records* (or structures) make them referenceable by their name. As far as atomic types are concerned (i.e., types that cannot be decomposed into smaller elements), we consider the usual primitive types found in programming languages, such as int, float, string, and boolean. Furthermore, there exist numerous formats for encoding well-defined datasets, such as FITS files in astronomy or DICOM for medical imaging. We want such existing formats to be reusable directly, which means that such datasets should also be viewed as primitive types.

The type system must be language independent so that it can describe the structure of datasets independently of any programming or workflow language. All the features we discussed above (and more)—namely, array and recordlike aggregation, type naming, and attributes—are supported by XML Schema [13, 3]. We therefore propose to adopt the XML Schema, since it brings along other benefits by its standardized nature.

We show in Figure 2 an excerpt of the XML Schema for DR2. (For conciseness, we keep namespaces implicit in this paper.) The schema specifies that DR2 is composed of a single element named `imaging` of type `Imaging`. The complex type `Imaging` is a sequence of elements `run`, of type `Run`, with a variable (and possibly unbounded) number of occurrences. The complex type `Run` is a sequence of elements `rerun`, of type `Rerun`. Note that a mandatory attribute is being specified for type `Run`, with name `number` and natural value.

```xml
<xs:schema targetNamespace="http://www.griphyn.org/SDSS"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.griphyn.org/SDSS">
  <xs:element name="imaging" type="Imaging"/>
  <xs:complexType name="Imaging">
      <xs:sequence>
          <xs:element name="run" type="Run" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
  </xs:complexType>
  <xs:complexType name="Run">
      <xs:sequence>
          <xs:element name="rerun" type="ReRun" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="number" type="xs:nonNegativeInteger" use="required"/>
  </xs:complexType>
</xs:schema>
```

Fig. 2. XML Schema Excerpt for DR2

We note from this example that the type system provides a uniform mechanism for representing the structure of datasets both "within" and "outside" files. In fact, at the level of the type system, no construct distinguishes between what is inside a file and what is outside. Such a uniform approach allows us to express workflows and programs that operate both on datasets and file contents. Note that while XDTM makes it possible to express the structure of a file's content in the type system, this level of detail is not required. In particular, it may not be desirable to describe the structure of binary formats, and we may prefer to consider such types as opaque.

This use of XML Schema to describe the structure of datasets allows us to take the *conceptual* view that there is an XML document that represents a dataset encoded into a physical representation. Navigating a dataset can thus be expressed by using the conceptual XML view. We say that the XML document represents a conceptual view of a dataset because we do not expect every dataset to be translated into XML: to do so would defeat the purpose of what we are trying to achieve here, namely, to deal with complex data formats, as currently used in everyday practice.

Since XML Schema is standardized by the W3C, tools for editing and validating schemas exist; also, query languages such as XPath and XQuery are specifically conceived to operate on XML documents; moreover, the Document Object Model [9] offers a standard programmatic interface to such documents. We use these benefits in the rest of the paper, and we assume that the reader is familiar with the XPath notation [4]. We now turn to the problem of declaring the physical representation of datasets whose abstract structure has been specified by XML Schema

5 Mapping to the Physical Representation of Datasets

We next present our approach to specifying the physical representation of datasets. First, since the structure of a dataset is specified by an XML Schema and since a given dataset may have different physical representations, it is appropriate to characterize the physical representation in a document distinct from the schema, so that the schema can be shared by multiple representations. We expect the physical representation specification to refer to the Schema (using XML namespaces and conventions), but we do not want to annotate the schema directly with physical representation information (as does DFDL [2]), since this would prevent the schema from being reused for multiple representations of a same dataset.

Second, we recognize, like DFDL, the need to *contextualize* physical representations, so that we can allow a given type that occurs multiple times in a dataset to be encoded into different physical representations depending on its position in the dataset. Whereas annotation can directly be scoped by XML Schema's block structure, we now need to provide an alternative mechanism, since we have just precluded the use of annotations and therefore cannot take advantage of their natural scoping.

Third, our focus is on representing datasets, that is, aggregation of data, rather than on specifying the format of arbitrary (binary) data files (for example, for legacy reasons). Hence, we do not expect a single formalism for representing physical formats to be able to characterize arbitrary files in a descriptive manner. Instead, we support a mix of declarative and procedural representations; concretely, we adopt notations from which we can derive methods to read the physical representation of a dataset into an abstract one, and vice versa to write the abstract representation into a physical one. We anticipate that a library of such converters can be made available to the user and that new converters are permitted to be defined, hereby providing for extensibility. Such a pragmatic operational view of conversion to and from physical representation allows for direct reuse of legacy converters for pre-existing formats.

Requirements have identified that it is not desirable to add annotations to an XML Schema directly. Instead, we need a mechanism by which we can refer uniquely to an element in a dataset and indicate how it should be converted (remembering that XML Schema does not impose element names to be unique in a schema). Such a notation already exists: given that datasets structures are specified by XML Schema, we can use the XPath notation to identify the path that leads to a given element in a dataset.

We now introduce an XML notation for the mapping between abstract and concrete representation. Such a mapping is itself contained in a file so that it can be made explicit with datasets and published with their XML Schemas. XDTM (XML Dataset Type and Mapping) is the system that uses the schema and mapping documents in order to help users manipulate arbitrary datasets *as if* they were XML documents.

Figure 3 presents an excerpt of the mappings between abstract and physical representations for DR2. Each individual mapping identifies an element in DR2 using an XPath expression, and specifies its physical representation by one of the XML elements `directory`, `file`, `line`, `url`, and so forth. Specifically, Figure 3 states that the whole DR2 dataset is accessible from a URL; `imaging` and `run` datasets are each represented by a directory; and `fpobjc` datasets are represented by files. Other mappings could refer to tar talls or zipped files.

```
<xdtm:mappings>
    <xdtm:mapping path="/DR2">
        <xdtm:URL namingMethod="Self"  location="http://das.sdss.org/DR2/data"/>
    </xdtm:mapping>
    <xdtm:mapping path="/DR2/imaging">
        <xdtm:directory namingMethod="Self"/>
    </xdtm:mapping>
    <xdtm:mapping path="/DR2/imaging/run">
        <xdtm:directory namingMethod="RepresentRun"/>
    </xdtm:mapping>
    <xdtm:mapping path="//fpObjc">
        <xdtm:file namingMethod="RepresentFpObjc" type="opaque"/>
    </xdtm:mapping>
</xdtm:mappings>
```

Fig. 3. Excerpt of Mapping to/from Physical Representation

For each kind of physical representation (directory, file, etc.), we use the attribute `namingMethod` to specify the name it is expected to have. The value of this attribute denotes a procedure that produces, from the abstract representation, a string that is the physical dataset's name (and vice versa). For example, `Self` returns the name of the element in the abstract representation; `RepresentRun` returns the value of the `run` attribute; `RepresentFpObjc` is used to create the fpObjc filename from field, run, and rerun attributes. In the case of fpObjc files, the mapping also specifies that the file type is "opaque"; that is, we do not describe the internal physical representation of this dataset.

6 Implementation

The aim of an XDTM implementation is to present, as DOM objects, datasets encoded in their physical formats so that they can be navigated by an XPath library like any other DOM object; symmetrically, it allows one to construct DOM objects that can be serialized to their physical representation. The implementation of XDTM consists of several components: *(i)* parsers for mappings and types, *(ii)* a library that reads datasets as DOM objects and vice versa, *(iii)* an XPath library that allows navigation of such DOM objects.

Our Java implementation uses the Jaxen XPath engine (jaxen.org), which processes XPath queries over DOM objects. Hence, the aim of our implementation is to materialize physical datasets as DOM objects so that they can be navigated by Jaxen, even though they refer to the actual physical representation of datasets. To this end, our implementation provides a library of classes implementing the standardized DOM interface [9] for all the physical representations found in DR2.

For each kind of physical representation, we have defined an associated class: directory in a file system (`VDSDirectoryElement`), reference to a http server through a URL (`VDSURLElement`), file containing an opaque or a fully described dataset (`VDSFileElement`), line within a file (`VDSLineElement`), and conjunction of multiple representations (`VDSGroupElement`). In addition, we need to allow for immediate values such as primitive types, which we abstract by the class `VDSImmediateElement`. All these classes implement the DOM interface, which specifies standard methods to access and create node children, attributes, parent, and siblings. For example, the dataset `imaging` is mapped to a directory in Figure 3; in this case, XDTM internally uses the class `VDSDirectoryElement`, which implements accessor and creator methods for a directory in a file system.

In more detail, when the dataset `imaging` is read from the filesystem, it is represented by an instance of `VDSDirectoryElement` in the abstract representation. Whenever the dataset's children are accessed in the abstract representation, they are read from the contents of the directory in the filesystem and are made available as DOM elements themselves, all chained as siblings. The `namingMethod` attribute is also used at that point because the procedure it denotes reads the physical name and returns an element name in the abstract representation and potentially sets some attributes. Symmetrically, at the abstract level, one can instantiate the class `VDSDirectoryElement`, create new children for it, and insert it in other datasets. When saved into the physical representation, it is represented as a directory that contains its children encoded as files or directories. The `namingMethod` attribute value is again used to obtain the name of the directory from its abstract representation.

7 Related Work

The Data Format Description Language (DFDL) [2] is a descriptive approach by which one chooses an appropriate data representation for an application and then describes its format using DFDL. DFDL's ambitious aim is to describe *all* legacy data files, including complex binary formats and compression formats such as zip. This highlights a crucial difference with our approach: we seek not to describe binary formats but to express how data is aggregated into datasets. DFDL uses a subset of XML Schema to describe abstract data models, which are annotated by information specifying how the data model is represented physically. Specifically, mappings are declared as restrictions of primitive data types annotated by the concrete types they must be mapped to; the XML Schema is then decorated by scoped annotations specifying which mapping applies for the different types. Since the actual annotations are embedded inside an XML Schema, mappings and abstract types are not separable in practice. We see DFDL as complementary to our approach, however, since it can be used to convert XDTM atomic types.

The METS schema [10] is a standard for encoding descriptive, administrative and structural metadata regarding objects within a digital library. In particular, it contains a *structural map*, which outlines the hierarchical structure of digital library objects and links the elements of that structure to content files and metadata that pertains to each element. The METS standard represents a hierarchy of nested elements (e.g., a book, composed of chapters, composed of subchapters, themselves composed of text). Every node in the structural map hierarchy may be connected to content files that represent that node's portion of the whole document. As opposed to XDTM, METS does not separate an abstract data structure from its mapping to physical representation, nor do physical aggregations such as complex directory structures or tar balls.

XSIL (XSIL: Java/XML for Scientific Data) comprises an XML format for scientific data objects and a corresponding Java object model, so that XML files can be read, transformed, and visualized [14]. Binary Format Description (BFD) [11] extends XSIL with conditionals; it can convert binary files into other binary formats; to this end, datasets are successively converted into XML by a parser that uses the BFD description of the input set, translated into another XML document using XSLT transformations, and finally converted by an "inverse parser" into another binary format, also specified in BFD.

Hierarchical Data Format 5, HDF5 [5], is a file format (and associated software library) for storing large and complex scientific and engineering data. HDF5 relies on a custom type system allowing arbitrary nestings of multidimensional arrays, compound structures similar to records and primitive data types. HDF5 introduces a *virtual file layer* that allows applications to specify particular file storage media such as network, memory, or remote file systems or to specify special-purpose I/O mechanisms such as parallel I/Os. The virtual file layer bears some similarity with our mapping, but focuses on run-time access to data rather than physical encoding. While HDF5 is not capable of describing legacy datasets that are not encoded as HDF5, some tools allow conversion to and from XML [8]. Specifically, the XML "Document Type Definition" (DTD) can be used to describe HDF5 files and their contents.

Microsoft ADO DataSet [1] is a "memory-resident representation of data that provides a consistent relational programming model regardless of the data source." Such DataSets can be read from XML documents or saved into XML documents; likewise, DataSet schemas can be converted into XML Schemas and vice versa. The approach bears some similarity with ours because it offers a uniform way of interacting with (in-memory copies of) datasets, with a specific focus on relational tables: however, the approach does not support interactions with arbitrary legacy formats. To some extent, the product XMLSpy (www.xmlspy.com) provides a symmetric facility, by allowing the conversion of relational data base to XML (and vice versa), but using XML Schema as the common model for interacting with data stored in different databases. It does not either provide support for datasets in non-relational formats.

8 Conclusion

We have presented the XML Dataset Typing and Mapping system, a new approach to characterizing legacy datasets such as those used in scientific applications. The fundamental idea of XDTM is to separate the description of a dataset structure from the description of its physical representation. We adopt the XML Schema type system to characterize a dataset's structure, and introduce a context-aware mapping of dataset types to physical representation. The separation of concerns between structure and physical representation allows one to define workflows and programs that are specified in terms of dataset structures, but can operate on multiple representations of them. This makes such computational procedures more adapted to deal with the evolution of data formats, typical of open environments. Beyond the SDSS application described in this paper, we have experimented with applications in in high energy physics and medical imaging, by which we demonstrate that the XDTM approach can help users to operate on concrete datasets used in many disciplines.

We are working on a robust implementation of XDTM capable of operating over large datasets, with checkpointing and restart capabilities. We are also designing the second generation of the GriPhyN Virtual Data Language, which will use XDTM to make workflows independent of the physical format of datasets they are intended to manipulate.

Acknowledgments

This research is funded in part by the Overseas Travel Scheme of the Royal Academy of Engineering, EPSRC project (reference GR/S75758/01), and by the National Science Foundation grant 86044 (GriPhyN).

References

1. Ado .net dataset. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpcontheadonetdataset.asp.
2. M. Beckerle and M. Westhead. GGF DFDL Primer. Technical report, Global Grid Forum, 2004.
3. P. V. Biron and A. Malhotra. XML Schema Part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/, May 2001.
4. J. Clark and S. DeRose. Xml path language (xpath) version 1.0. http://www.w3.org/TR/xpath, November 1999.
5. National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC). HDF5 Home Page. http://hdf.ncsa.uiuc.edu/HDF5/.
6. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. Chimera: A virtual data system for representing, querying and automating data derivation. In *Proceedings of the 14th Conference on Scientific and Statistical Database Management*, Edinburgh, Scotland, July 2002.
7. I. Foster, J. Voeckler, M. Wilde, and Y. Zhao. The virtual data grid: a new model and architecture for data-intensive collaboration. In *Proceedings of the Conference on Innovative Data Systems Research*, Asilomar, CA, January 2003.
8. The HDF5 XML Information Page, jan 2004.
9. A. Le Hors, P. Le Hgaret, L. Wood, G. Nicol, J. Robie, Mike Champion, and Steve Byrne. Document object model (dom) level 3 core specification version 1.0. http://www.w3.org/TR/DOM-Level-3-Core/, April 2004.
10. Mets: An overview and tutorial. http://www.loc.gov/standards/mets/METSOverview.v2.html, 2003.
11. J. Myers and A. Chappell. Binary format description (bfd) language. http://collaboratory.emsl.pnl.gov/sam/bfd/, 2000.
12. Sloan digital sky survey. `www.sdss.org/dr2`.
13. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. http://www.w3.org/TR/xmlschema-1/, May 2001.
14. R. Williams. Xsil: Java/xml for scientific data. Technical report, California Institute of Technology, 2000.