

# **XML and Scientific File Formats**

Robert E. McGrath

National Center for Supercomputing Applications  
University of Illinois, Urbana-Champaign

August, 2003

---

This report is based upon work supported in part by a Cooperative Agreement with NASA under NASA grant NAG 5-2040 and NAG NCCS-599. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Additional support was provided by the Electronic Records Archive of the US National Archives and Records Administration under grant number NARA NSF 02-02GPG.

Other support provided by NCSA and other sponsors and agencies (See: <http://hdf.ncsa.uiuc.edu/acknowledge.html>).



## Contents

1. Introduction to XML And Scientific Data Formats.....	6
2. Definition of “Data Format”.....	6
3. Problems that data formats may address .....	7
3.1. Data description.....	7
3.2. Data storage and retrieval .....	7
3.3. Transfer across systems and situations .....	8
3.4. Data discovery .....	8
4. Brief Comparison of “Binary” and XML Data Formats .....	8
5. Using XML and Binary Data Formats .....	10
5.1. Translating Between Binary and XML .....	10
5.2. Mixed-Model: pointers to binary data .....	12
6. Summary: Best Uses of XML and Binary Files .....	15
6.1. Best Uses for Binary Data Formats .....	15
6.2. Best uses for XML.....	16
6.3. Poor Uses of XML.....	17
7. Summary and Discussion .....	17
7.1. Summary.....	17
7.2. Other Uses of XML .....	17
Acknowledgements .....	18
Appendix A. Tutorial Example: Storing Arrays of Numbers in XML.....	18
A.1 Features of XML .....	18
A.2. Tutorial Example: Markup for a Two-Dimensional Array.....	19
A.3. Discussion: why the markup strategies matter .....	21
A.4. XML With Pointers to External Data .....	22
References .....	24



### **Abstract**

For many years, scientific data has been stored and transferred using a variety of data formats. In recent years XML has become an important and popular for exchanging digital information. At the most abstract level, XML can be used for any purpose that a binary format might be used, and vice versa. This paper discusses the strengths and weaknesses of XML and binary formats, and speculates on the best uses of each in future scientific data systems.

## 1. Introduction to XML And Scientific Data Formats

Scientific data is complex and voluminous and the details are essential—every number matters. Managing and using scientific data presents many challenges, including:

- Handling huge volumes of data efficiently
- Managing provenance, quality, proper attribution and citation
- Integration of data from multiple sources (with different scientific meanings, spatial and temporal scales, etc.)
- Discovery of relevant data from diverse sources

In earlier work, we characterized one aspect of scientific data use as a “conversation with the data” [23, 24]. This notion emphasizes the point that scientists deal with data at many levels of detail, from broad and shallow searches for candidate data sources, through more and more detailed selections, to obtain numeric data from multiple sources, from which scientific findings may be derived and supported.

Scientific data users are also data producers, and science data is, almost by definition, intended to be shared, usually with a community of scientists with similar interests [7, 29]. Increasingly, there is demand for cross-discipline study and synthesis, which requires data sharing across multiple communities [26].

Our team at NCSA has been studying and providing tools to store, share, and manage scientific data for many years. One of our products is the Hierarchical Data Format (HDF), a format and library which is used by scientists from many communities around the world [18].

For many years, scientific data has been stored and transferred using a variety of data formats. In recent years XML has become an important and popular for exchanging digital information, including scientific data. At the most abstract level, XML can be used for any purpose that a binary format might be used, and vice versa. XML has many good features, but XML is not likely to replace binary data formats for scientific data.

At NCSA we have studied emerging Web and XML technologies for many years. In particular, we are interested to learn how XML and binary best be used by scientists, and how XML and binary formats should work together in systems. This paper summarizes some of the lessons we have learned.

## 2. Definition of “Data Format”

It is important to point out that all computer data is at least implicitly and informally “formatted”, although the format may be informal and undocumented. For trivial or evanescent data there is no little need to worry about how the data is represented or stored. However, for data that must persist, be transported, and/or be shared, it is important to carefully store the data in a known format, so that *the intended meaning can be accurately reconstructed*.

This paper discusses “data formats” for scientific data. A data format is a method for representing data in a digital store (which may be transient or persistent). Essentially a data format has the following aspects:

- A model of data representation
- A mechanism for describing data, i.e., for mapping concepts of the data model to digital objects such as files or memory

- Optionally, standard methods for manipulating data.
- When the data model is very general, a data format may also have means to specify profiles, i.e., more specific sub-cases of the data model that model particular application or community concepts

This paper considers two mechanisms for implementing formatted data: XML and “binary” data formats.

A “binary” format maps data concepts to the native entities of the computer system, to bytes, numbers, and perhaps pointers or other structures. A data format can also be defined in terms of composites or aggregations of ‘atomic’ binary elements. For example, many data formats define multidimensional array objects, specified as conventional arrangements of bytes. There are many examples of ‘binary’ formats, including HDF [18], netCDF [36], FITS [33], TIFF [34], GeoTIFF [10], and so on.

A “text” format maps the data concepts to character strings, which must be translated to and from computer entities. For example, to perform arithmetic, the string “-7.0” must be converted to an appropriate representation. One advantage of text representations is that they can (to a degree) be read by a human. XML has emerged as the most popular text format.

### 3. Problems that data formats may address

Storing data in a formatted form can meet several important goals. This section briefly lists some of the principle problems that data formats may address.

#### 3.1. Data description

By definition, formatted storage includes a description of the data. This description is very dependent on the data, the users, and the context. It may include:

- Basic specification of the layout (e.g., location and size of numbers, encoding scheme, order of rows, etc.)
- Specification of the model(s) and profile(s) that should be used to interpret the data, including units, geometry, and other vital concepts
- Specification of the measurements and algorithms that produced the data
- Pointers to publications, documentation, and other data required to interpret the data

This information is usually called ‘metadata’, and it is especially important for scientific data: the numbers are useless without the information necessary to form a valid scientific interpretation.

#### 3.2. Data storage and retrieval

A primary requirement is to store and retrieve data. At the most abstract level, this is a matter of transferring the information across time, and the overriding goal is that the *conceptual* information is successfully transferred. In non-trivial cases, it is always necessary to have a thorough description of the data, or else its meaning cannot be reconstructed at a later time.

Computer I/O is always slow, and fast storage is always at a premium, so it is important to use efficient algorithms and mechanisms for data transfer and storage. It is often necessary or desirable to create and access data piecemeal, i.e., writing and reading selected parts of large aggregates.

There is no one optimal storage or transfer method, different uses and access patterns place substantially different requirements on I/O systems. For this reason, it is important to provide options to control the details of data layout and transfer.

### **3.3. Transfer across systems and situations**

While some data is stored and retrieved within the same context, other data is shared or disbursed to many contexts. This may require reformatting, translation, or other modifications of the data in order to create or recreate the intended meaning at the time of use.

Again, a thorough description of the data is critical, or else the bits might not ever be interpreted at the destination. Computer systems are not identical, and users are heterogeneous as well. It is necessary to be able to decode the stored data into a form that makes sense to the user's system, and also for the user to understand it.

Some scientific questions require longitudinal study of data collected over years, decades, or longer. This requires transferring data across temporal contexts, and the pace of technical change assures that this will be challenging. In order to use data for many years requires that it be stored in a way that can be accessed and its meaning reconstructed.

### **3.4. Data discovery**

A third important use of data is for “discovery”: to somehow identify data that meets some criteria, e.g., it is relevant to some scientific question. This might be done in real time as data flashes by, or from stored representations of data. Either way, the problem is not trivial because the data must be selected by its scientific meaning, not according to trivial signatures.

The metadata in a file sometimes is explicitly designed to aid discovery. It may contain many facts of interest for investigators: when and how the data was generated, what the data is supposed to represent, who says so, and so on. There are as many possible types of metadata as there are potential things that might be said about the data: in other words, there is no limit to what might be useful metadata.

Data Mining and other automated learning techniques are increasingly important for the discovery of relevant data out of large volumes of data. These algorithms scan large volumes of data, which presents significant challenges for data storage and access. These issues will not be considered here.

## **4. Brief Comparison of “Binary” and XML Data Formats**

This section presents a summary of the characteristics of XML and “Binary” files for storing and transferring scientific data.

XML and binary formats such as HDF can be used for the essential requirements of scientific data. These are: storing objects (e.g., tables, arrays, lists, images, etc., and the values of numbers and strings), storing descriptions of data (including facts critical to interpret the scientific meaning of the numbers), and specifying the layout of bits.

In contrast, there are several important differences in the capabilities of XML and binary formats.

First, XML is, in theory, “human readable”, while binary data requires software. Of course, for any non-trivial use of data almost always requires software assist, even with XML.



Second, XML is a universal standard with widely available software. It is a good bet that anyone who needs to can get documentation and software to access XML. Because XML is a Web standard, it enjoys massive commercial use and support, and is built in ‘by default’ to most systems. As a result, there is a huge base of excellent commercial and free software that can be adopted for scientific use, at a huge cost savings. In the long run, the cost savings may be the overwhelming practical advantage of XML.

There are many binary formats, and none is as universal as XML. For any given binary format, there are readers and documentation, but it may be necessary to obtain and install extra software, and/or write code.

A third important difference is in the flexibility of the storage format. XML is, by design, organized as a stream of Unicode characters, with its objects organized as a tree. In contrast, binary formats store numbers in ‘native’ formats which are typically compact and require no translation in order to perform computation.

A binary format can implement many storage strategies, including a graph, mesh, or other multidimensional structure. Also, a binary file can be organized to optimize space and transfer speed. For instance, a binary data format may implement optimal numeric coding, data compression, or interleaving. Also, a binary file can access data in the middle of the file (“random access”).

These features are extremely critical for practical access to large complex data sets that cannot be stored in memory in their entirety. The ability to access sub-sets of large data (e.g., a region of a global dataset) requires careful organization of the data [21]. Compression and other strategies can have a very significant impact on cost and performance (e.g., [48]).

Fourth, XML and specific binary formats integrate with other software. XML is the ‘native format’ for the Web, and many standards are built on top of XML, including XSL stylesheets [38], Web Services [41], Grid services [35], and the Semantic Web [39]. On the other hand, binary formats can tightly coupled to software, for optimal integration to a specific environment.

Table 1 summarizes these points. Overall, there are advantages and disadvantages to different storage format. The follow sections discuss the best uses of binary, XML, and both together.

**Table 1**

<b>XML</b>	<b>‘Binary’</b>
‘Human readable’	Requires software interpretation
Single, Universal Standard <ul style="list-style-type: none"> <li>• Web-friendly</li> <li>• Massive commercial support</li> </ul>	Many binary formats, availability varies
Stored as a stream of Unicode, in a single file <ul style="list-style-type: none"> <li>• Organized as a tree</li> <li>• Cannot directly store ‘native’ numbers, must be encoded as unicode</li> </ul>	Many organizations, <ul style="list-style-type: none"> <li>• may use ‘native’ numbers and pointers</li> <li>• may support ‘random access’ to any part of the data</li> <li>• may support compression, etc.</li> </ul>
Integrated with Web/GRID standards, e.g., <ul style="list-style-type: none"> <li>• XSL [38]</li> <li>• Web Services [41]</li> <li>• Semantic Web [39]</li> </ul>	Tight integration in specific system, e.g., <ul style="list-style-type: none"> <li>• Optimized tiling and compression</li> <li>• Customize profile and library (e.g., HDF-EOS [32], CACTUS [1])</li> </ul>

## 5. Using XML and Binary Data Formats

Binary formats and XML each have advantages, and there are good reasons why scientists will want to use both XML and binary data in the same systems and applications. This section considers how to mix the two in the same application or environment.

Two principle approaches are considered: translation between XML and binary; and a mixed model of XML with pointers to binary data.

### 5.1. Translating Between Binary and XML

One method for using binary and XML together is to translate between the formats as needed. Many applications use XML as a *lingua franca* between systems, while using different formats internal to the system. Indeed, this is the core of the design of Web Services [41] and the Grid Services [35]. Also, most database systems can load and dump data as XML, while storing it in customized binary formats within the database (e.g., [19]).

Scientific data processing applications and systems usually deal with data in many formats, translating as needed, e.g., with “readers” for different binary formats. Ideally, the widespread use of well designed XML standards may reduce the effort required to combine data from different sources.

In the case of scientific data formats, it is possible to define translators that create a standard XML file from binary data, and creates a binary file from appropriate XML. For example, NCSA’s HDF5 has a standard XML schema, and tools to translate (as well as can be managed) binary HDF5 to XML, and XML to binary HDF5 [15, 47]. Other formats have or soon will have such capabilities (e.g., NCML [37]).

In essence, this translation is a program that reads all or selected objects in the binary, and creates an XML file which represents the same object according to some DTD or Schema. Figure 1 shows an example of some of the XML tags that are defined to describe and HDF5 Dataset. The XML is generated by a program that reads the binary HDF5 file and writes XML according to a published standard XML Schema. The XML document has objects to describe the logical components of the HDF5 file.

Going the other direction, a standard XML parser reads the XML, and creates the appropriate binary object. So, for the XML Figure 1, the translator would call the HDF5 library to create a Dataset named “/datasetF32”, with the dimensions, data type, and other attributes specified in the XML.

```
<hdf5:Dataset Name="datasetF32" OBJ-XID="xid_976-0"
  H5Path="/datasetF32" Parents="xid_928-0" H5ParentPaths="/">
  ...
  <hdf5:Dataspace>
    ...
  </hdf5:Dataspace>
  <hdf5:DataType>
    ...
  </hdf5:DataType>
  <hdf5:Data>
    ...
  </hdf5:Data>
</hdf5:Dataset>
```

Figure 1. Sketch of XML tags for an HDF5 Dataset

Scientific data usually includes large, multidimensional arrays of numbers. XML can model numeric concepts such as vectors and arrays, but there is no one right way to do so. In fact, XML is very flexible: it is very possible to “mark up” the same document using many different schemas. The resulting documents are legal XML, and they all accurately represent the same data, but they may be completely incompatible with each other. *Using XML enables, but does not ensure, interoperability.* Appendix A presents a tutorial example of different ways to “mark up” a two-dimensional array of numbers.

Of course, it is not necessary to literally transcribe from binary to XML. It may be desirable to create a description in some other form, selecting and reorganizing the information in the binary object. For example, a program could analyze the binary file, and generate an XML *description of the data* suitable for a catalog: the XML does not reproduce the data, it represents important facts (from some point of view) about the data. The XML might follow a standard such as the Federal Geospatial Data Committee (FGDC) Metadata Standard [3] or NASA’s Global Change Master Directory (GCMD) [25].

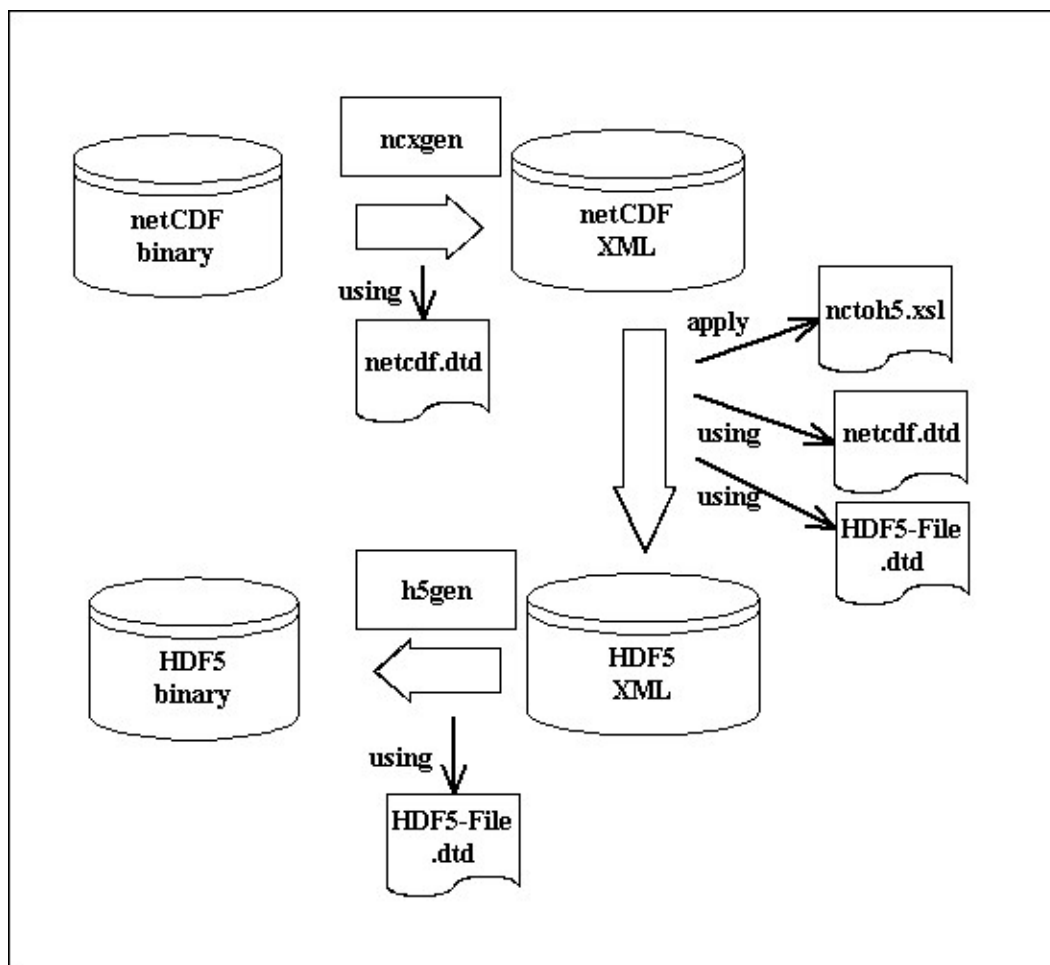


Figure 2. In principle, it is possible to convert binary to binary, via XML. (See [22])

XML stylesheets ([38]) offer an interesting mechanism for transforming from one XML “grammar” to another, which can be used for some kinds of data filtering and translation. For example, in one toy program we demonstrated conversion from binary netCDF to binary HDF5 via XML: the netCDF binary was written as a netCDF XML description, a stylesheet with JavaScript was applied to transform to an HDF5 XML description, and then the XML was processed to generate a binary HDF5 file [22]. Figure 2 shows a diagram of this process.

Thus, there are several ways to translate between binary and XML: convert binary to XML, XML to binary, and transforming XML to other XML. In general, once data has been translated to XML, it should be relatively easy to process it into other forms of XML. Furthermore, a given XML file can be filtered in many ways to generate other XML for various purposes.

The most general and powerful variation of the data translation approach would be to define an extensible framework in XML, and then map every binary format to the framework. This idea has been proposed many times, e.g., XDF [46], ESML [7], and now the “Data Format Description Language” of the Global Grid Forum [5].

It should be realized, of course, that perfect translation is seldom achievable. In many years of experience at NCSA (e.g., [13, 14, 16]) we have seen that, for any two data models, many concepts map easily, a few are very difficult, and there are usually some details that simply cannot be adequately translated [9]. As a result, information may be lost in the translation process.

It is even more difficult to determine if the intended meaning has been translated: can the translated object be used and interpreted correctly? This is especially sensitive and critical for scientific data, which usually can be interpreted only within a theoretical context which may be reflected in subtle properties of the data. The emerging “Semantic Web” [2, 11] may be useful to “mark up” data with additional information required to interpret it.

## 5.2. Mixed-Model: pointers to binary data

XML provides an additional option: the XML markup can contain pointers to external data including binary files or objects in a binary file. In this approach, the XML markup contains a description of *how to get the data*, rather than the data itself.

There may be good reasons to use a description of the data or data access method. The user may not require the data itself, or it may be infeasible to use the actual data, because:

- The data may be too large to transfer
- The data may be off-line
- The data may be unique, e.g., there can be only one true version of the “original” dataset, which cannot be replicated on a local system.
- There may be many, many data objects, the user seeks to find a relative few.

This mixed-model will certainly be a common use of XML and binary data, and standards are evolving to implement this concept.

The essence of the approach is for the XML to specify a pointer (an extended hyperlink) that can be correctly interpreted by software that reads the XML. The pointer can be a URL or similar tag, which can include information about how to access the data at that address. This approach is analogous to hypertext links that are used to embed graphics and scripts in HTML documents. In addition to a standard syntax for the link, there must be standard conventions for how to tell how to follow the link: how the data object can be accessed.

Of course, the XML could have much more information, including:

- Multiple addresses, e.g., for replicates
- Detailed parameters for the access method(s)
- Specification of the format of the retrieved data
- Specification of a sub-set or other selection from a large dataset

For example, instead of including the numbers in the XML, a matrix might be represented by a link, such as shown in Figure 3.

```
<matrix>
  <link URL="some.url" FileType="a mime type"
        ObjectName="id of matrix" />
</matrix>
```

**Figure 3. XML with a pointer to binary data.**

A stored data object is a representation of data (state) + algorithms. To interpret a stored object, it is necessary to implement the correct algorithms to interpret the stored data, e.g., to construct the object in memory based on the stored representation. This is not trivial because the stored representation of a data object need not be a literal copy of the representation in memory. In many cases, the data is stored following a non-trivial algorithm (e.g., compression), which the reader must reverse to reconstruct the data object.

So, the XML markup should be augmented to specify the algorithms that must be used, e.g., “this set of bits should be interpreted using this algorithm”. In general, there are three basic ways you can do this:

1. Include a standard name for the algorithm, e.g., MPEG.
2. Include a pointer to code that implements the algorithm, e.g., a library or service that can read the bits.
3. Include the algorithm in the XML, e.g., some kind of program expressed in XML

Note that in all of these approaches, the application using the XML must implement the algorithms to access the data, or else defer to other software that implements the algorithms.

SOAP [40] is an example of the first approach. A SOAP message is an XML document, i.e., XML is used to provide a verifiable and extensible framework for defining structured text messages. Of course, SOAP is not well suited for passing large amounts of numeric data, such as a JPEG image.

DIME [20] is one of several proposed standards for passing arbitrary chunks of binary data via SOAP. DIME is actually a very simple extension of SOAP, that allows the message to include blocks of non-XML data, identified by a conventional name (e.g., ‘image/jpeg’). DIME does not describe or interpret the binary data, the producer and consumer must use the data type to implement correct algorithms. So, when the message tags the data as ‘image/jpeg’, the receiver must, use a JPEG reader.

This approach could be augmented by adding a pointer to a service that implements the required access. For example, the message could include the URL of a Web Service that displays MPEG files, or the URL of MPEG software that can be downloaded. This would be an example of the second approach.

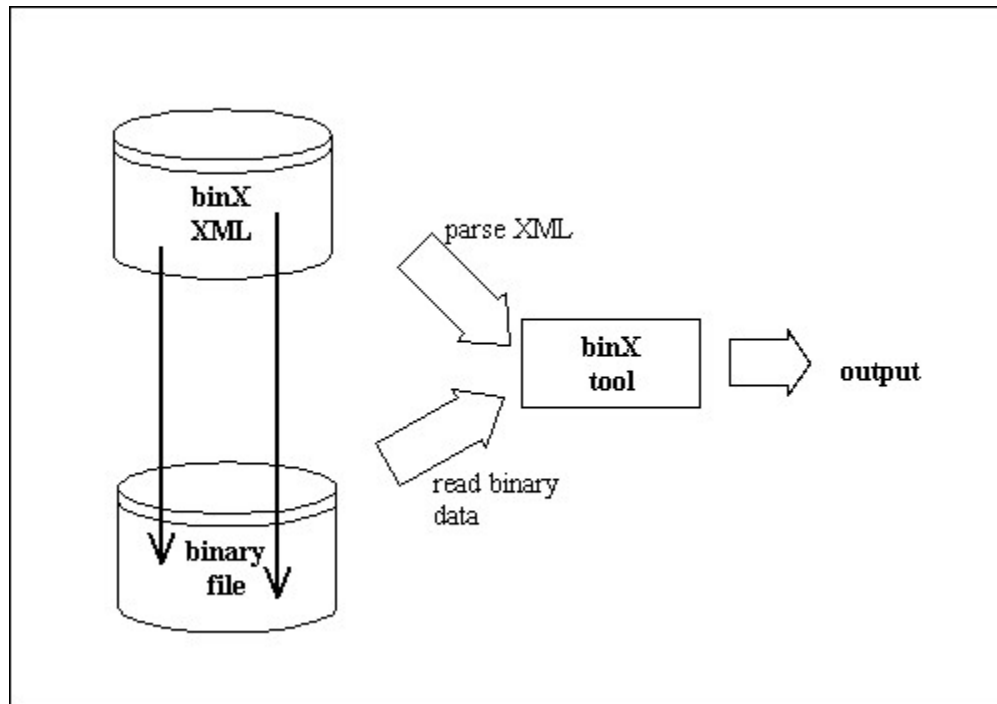
The third approach is illustrated by BinX. BinX is a proposed Grid standard which seeks to provide a generic data access service [44]. BinX intends to provide “the ability to describe the physical representation and the overall structure of arbitrary binary data” ([44], p. 4). To realize

this goal, BinX uses an XML language that gives a complete description of the binary data (Figure 4). BinX will likely evolve to use the “Data Format Description Language” (DFDL) [5] as its data description language.

The DFDL is an XML schema that attempts to provide a universal description language for data at several levels of detail:

1. The underlying physical representation (e.g. bit/byte ordering)
2. The primitive types used (e.g. IEEE float, integer)
3. The structure of the data itself (e.g. array, list of fields, table)

The physical representation may require description of the layout of the data in storage, i.e., the order of bits and bytes, and the interpretation of padding, and so on. The primitive types use XML schema’s rich data model, which must be mapped to and from the number models used by different formats. And lastly, the layout of the data elements must be described, including the dimensionality of arrays, and so on. For example, DFDL is designed to describe basic numeric objects, such as arrays.



**Figure 4. The notional usage for BinX (after [44], p. 6).**

The intention of DFDL XML is to describe the layout of the data in the file, effectively telling how to read the binary data. It is imagined that BinX or other tools will be able to analyze the XML, and use generic algorithms (e.g., seek to an offset, and read so many bytes) to retrieve the data.

However, as already discussed above, the description of the “file” must also describe both the data (where the bits are) and the algorithms necessary to interpret the bits. DFDL and similar approaches must assume that the algorithm required to retrieve the data is apparent from the XML description of the data, or else use some mechanism to discover what algorithms to use.

This is a difficult problem because the stored representation of a data object need not be a literal copy of the representation in memory. In many cases, the data is stored following a non-trivial algorithm, which the reader must reverse to reconstruct the data object.

For example, HDF5 stores arrays of data in several ways, including user-defined chunks (tiles), which are stored in arbitrary locations on the disk. When an array in memory is stored in HDF, the blocks of data are organized in a B-tree, and may optionally be compressed, packed, and might have computed fill values for unwritten data (see [17]). In short, the layout on disk is not trivially related to the conceptual layout of the array in memory. The stored state of a database or other complex object will have similar characteristics.

The principle here is that a description of the data (the bits) is not sufficient to retrieve the stored object. A description of the “file” must also describe the algorithms necessary to interpret the bits. It is not clear how a language such as DFDL should deal with this. While it is certainly possible to mark up *any* binary file, it doesn’t make much sense when the values must be decoded by an algorithm in memory. For example, it would be perfectly possible to describe the stored form of a compressed file as a sequence of so many bytes or bits. But this isn’t very useful because the numbers only have meaning when interpreted by the decoding algorithm.

A universal data format will likely be a least common denominator. This will be useful in some cases, but one may wonder if it will be useful for complex scientific data. Systems will probably use their own internal (binary) formats to fully represent their data. Therefore, systems may well need to provide different interfaces with different levels of detail, e.g., a very generic XML view, a more data format specific XML view, and a binary interface to the specific data.

## **6. Summary: Best Uses of XML and Binary Files**

This paper has made the case that a system or application can benefit from using a mix of XML and one or more binary formats. What is the best use for each?

### **6.1. Best Uses for Binary Data Formats**

Binary formats are usually very efficient (in both space and time), especially a format that is specifically designed for an application. For local use where sharing is not required or tightly controlled, binary formats may be the best choice, especially if performance is a critical issue.

Binary formats are particularly well suited to numeric computation, especially using dense matrices of numbers. A binary format is very close to native computer memory, often it is a direct copy of the memory image. For this reason, a binary format can load to memory (or write to disk) the data for a numerical computation efficiently. In particular, a dense matrix can be written or read as fast as the system can transfer data, using the minimum memory that could be required.

In contrast, data stored in XML must be converted to/from Unicode and/or XML tags into a memory image (e.g., an array of numbers). For a large matrix, this operation requires considerable memory and time. In general, XML is catastrophically slow for numerical application.

Binary formats can support highly customized data access algorithms, including selection, filtering, scatter-gather, compression, and parallel I/O (e.g., see [17]). These features may be essential for large scale computations, which must manage large amounts of complex data.

In contrast, XML can be written/read only as a continuous stream of objects encoded in Unicode. With stream I/O, it is very difficult to manage very large objects or to navigate a file with a large

number of objects. The XML standard Document Object Model (DOM) provides some forms of random access, with a corresponding memory and time overhead. It is worth pointing out that implementations of DOM achieve this by creating a binary representation of the information in the XML file, which is similar to the data structures used in binary data formats.

## **6.2. Best uses for XML**

While XML has disadvantages, it has many advantages. For many uses, the universality of XML is overwhelming, trumping all concerns of efficiency. For example, XML is clearly well suited for data catalogs and many kinds of data and process description.

XML is well suited for heterogeneous systems with multiple users and multiple purposes. The universality and portability of XML is essential for sharing data across space (geographic distribution), time (archiving), and conceptual domains (different users, communities, and uses). Furthermore, the availability of generic software makes it easier to share the complex details of scientific data, encapsulated in schemas and XML-enabled software.

An XML “document” can be used to deliver a description of the data without the data itself, as in a data catalog (e.g., [25]) or as a “virtual dataset” (e.g., [44, 45]). This enables users and communities to share data more easily, and to create customized views of data from many sources without replicating or reformatting.

There are many good reasons to use a description of the data instead of the data:

- The data may be too large to transfer, the user seeks to access only a small fraction
- There may be many, many data objects, the user seeks to find a relative few
- The data may be off-line or protected
- The data may be unique or tied to a specific source (e.g., a sensor array)

In these and similar cases, an XML description is preferable to the data itself. A common theme in these cases is that it is infeasible to copy all the data, and then ignore what is not needed. Instead, the user copies the XML, determines what data (if any) is needed, and then when the actual data values are needed, the user will perform the costly and difficult steps needed to retrieve exactly what is needed.

One of the weaknesses of computer storage is that data can be stored only once: in many cases, data might need to be used in more than one way, and not all uses can be served well by a given layout. For example, a time series with several variables can be grouped by time step or by variable. The former is very efficient for sampling by time, the latter is efficient for sampling by variable.

Using the XML linking as discussed above, XML is well suited for constructing multiple views of the same data. Furthermore, an XML file is not limited to the data in a single binary file, or files from a single source, or even to a single kind of data. A “virtual dataset” can be constructed to point to objects from several files (possibly from multiple on-line archives), as well as databases, programs, and documentation.

Different applications and communities can create “virtual datasets” to meet their needs, without the cost of copying and reformatting all the data.



### **6.3. Poor Uses of XML**

XML is very poorly suited for some important aspects of scientific data management, particularly the representation of large arrays of numbers and structures other than a tree. The challenges of representing numbers and arrays has been discussed above.

XML models data as a tree, which naturally represents one- or two-dimensional data. It is awkward and difficult to represent other graph structures using XML, and it is difficult to represent higher dimensional data.

XML has limited ability to represent a data definition that is a restriction on a set of objects, e.g., “all the data in this object are 32-bit integers”. The XML Schema language can express some kinds of restrictions (particularly on the type and range of data values), but must use tags to define structural relations (such as positions in a matrix).

Data streams are important to many scientific applications, e.g., data flowing from sensors or real-time computations. In this case, the data is transient and must be accessed as it arrives, it cannot necessarily be collected in a file and analyzed later. Neither XML nor binary data files are well suited to data which is not persistent and does not have a fixed length.

## **7. Summary and Discussion**

### **7.1. Summary**

Binary formats and XML each have advantages, and there are good reasons why scientists will want to use both XML and binary data in the same systems and applications. This section considers how to mix the two in the same application or environment. Two principle approaches are considered: translation between XML and binary; and a mixed model of XML with pointers to binary data.

For many uses, the universality of XML is overwhelming. For example, XML is clearly well suited for data catalogs and many kinds of data and process description. However, XML is very poorly suited for some important aspects of scientific data management, particularly the representation of large arrays of numbers or complex data structures.

### **7.2. Other Uses of XML**

This paper has considered issues addressed by a “data format”. These uses are only some of the requirements for creating, managing, and sharing data, and XML is used in many ways in information systems. For example, XML is useful for formatting documents, representing model parameters (e.g., [6]) and for controlling experiments and computations (e.g., [28, 30]). Furthermore, the use of XML for multiple purposes can improve interoperability, e.g., making it easier to cross reference published papers with the data that supports them [24].

XML is well suited for the definition and implementation of community standards. Scientific data is shared by communities of users, and sharing requires data standards. Standards do not need to be tied to a particular storage format, indeed, in many cases the best approach is to define abstract models. For example, the Geographic Markup Language defines an abstract standard for describing geospatial data [31]. This specification is expressed in XML, although much of the data will be stored in one or more binary formats.

The universal namespace of URLs may be the single most important technical advantage of XML. The ability to point to data, data descriptions, documentation, and other objects via a standard namespace is critical for open sharing of information. This feature is especially important for scientific publishing, which needs to cross-reference published articles, software, and data (e.g., [27]). It also enables easy reuse and sharing of community defined schemas for data and metadata.

XML is well designed for certain kinds of filtering and translation, e.g., through stylesheets. Stylesheets provide a declarative translation from XML to XML, i.e., in the form of a document rather than a executable program). For an example, see [22], discussed above. Stylesheets may be useful for interesting data and metadata translation, e.g., between related standards. While stylesheets are technically limited, a stylesheet is an open way to publish translation algorithms.

## **Acknowledgements**

This report is based upon work supported in part by a Cooperative Agreement with NASA under NASA grant NAG 5-2040 and NAG NCCS-599. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Aeronautics and Space Administration.

Additional support was provided by the Electronic Records Archive of the US National Archives and Records Administration under grant number NARA NSF 02-02GPG.

Other support provided by NCSA and other sponsors and agencies (See: <http://hdf.ncsa.uiuc.edu/acknowledge.html>).

## **Appendix A. Tutorial Example: Storing Arrays of Numbers in XML**

If anything characterizes scientific data it is the use of large, multidimensional arrays of numbers. Computers were designed from the beginning to directly implement matrix computations, and most computer languages directly represent numbers and arrays of numbers.

XML was designed from the beginning to describe “a document”, which is conceived to have relatively large structural units, with blocks of linear text. Furthermore, XML models the document as a tree, which is fundamentally a two-dimensional model.

XML can model numeric concepts such as vectors and arrays, indeed, there are many different ways to do so. But there is no one right way to implement this, and the analysis gives insight into the strengths and weaknesses of XML.

### **A.1 Features of XML**

XML is a standard language for “document mark up”: XML defines an open standard for embedding symbolic tags in a text document. Basically, standard XML assures that if two different programs read the same XML document, each will see the same tags and values, conforming to the same constraints. XML does not define what tags may or may not be used, nor does XML understand what tags “mean”. Furthermore, standard XML software does not interpret the values of attributes and text between tags, other than checking the constraints.

XML has three important concepts:

- Standard parser
- Standard schema
- Standard namespace

An XML file contains tags and attributes which follow the XML specification. An XML parser can read the file and faithfully discover all the tags, attributes, and values encoded. Any conforming parser will discover exactly the same tags, etc., in a conforming XML file.

Second, the structure can be constrained by a grammar defined by a DTD or XMLSchema, i.e., a “markup language”. A parser that implements XML schema (DTD) can interpret the grammar, and discover any violations of the constraints.

Third, XML defines a universal namespace for documents and schemas that makes it possible to reliably share XML documents anywhere on the Internet.

There is no one right way to use XML. In fact, XML is very flexible: it is very possible to “mark up” the same document using multiple schemas. The resulting documents are legal XML, but completely incompatible. Using XML enables, but does not ensure, interoperability.

## A.2. Tutorial Example: Markup for a Two-Dimensional Array

To illustrate the flexibility of XML, consider the case of a two-dimensional array of numbers. Conceptually, there is a single object, perhaps a matrix. Operations are defined for manipulating the object in terms of its constituent elements (numbers). In a computer, the conceptual object is represented in a natural analog, usually as a rectangle of elements. Each element is represented as a coded value, i.e., a bit pattern that represents an integer or floating point number.

When the object is stored on the computer, it is important to store sufficient descriptive information so that the object (matrix) can be reconstructed. The description can be done many ways. In XML, the data description is done with markup tags, and there are many ways to design the tags.

For instance, the XML might simply define a tag for ‘matrix’:

```
<matrix>
1 2 7 ...
8 9 55 ...
...
</matrix>
```

XML does not specify anything about the text between the tags. In this example, the contents inside the tags must be interpreted by the reader according to some convention; and much is left to the reader:

- The format of the element (how to interpret the text to derive the intended numbers), how they are separated.
- Which element is which (implicit in the order of the elements)

The XML markup can specify more of these details with attributes and tags. So, perhaps each row might be an XML element:

```
<matrix COLS=10>
  <row ROW="1">
    1 2 ... n
  </row>
```

```
<row ROW="2">
...
</matrix>
```

or each number in the array could be an XML element:

```
<matrix>
<row ROW="1">
<Item I="1">
3
</Position>
<Item I="2">
4
</Position>
...
</row>
...
</matrix>
```

Alternatively, the coordinates could be encoded as attributes the elements in several ways:

```
<item I = "1" J="7"> 8 </item>
```

Or

```
<item>
<x-coord>
1
</x-coord>
<y-coord>
7
</y-coord>
<value>
8
</value>
</item>
```

All of these approaches represent the intended layout of the elements, the relative merits depend on the intended use of the XML.

Beside the layout of the elements in the array, it may well be necessary to define the number type and format of the data elements as well, e.g., integer, floating point, decimal, hexadecimal, length, byte order, and so on. This can be done in many ways.

For instance, a number type an attribute of the matrix, or of each element. For instance,

```
<matrix NumberType=Int32>
...
```

```
</matrix>
```

or

```
<item x="3" y="8" NT="F4.2">  
5.6  
</item>
```

If a full description of the layout of the number (the digital encoding) is required, the description will be quite complex. See the XML Schema [42] and the HDF documentation [17] for two examples.

### A.3. Discussion: why the markup strategies matter

In the examples above, and other similar variations, each approach has advantages and disadvantages.

The critical distinction is that standard XML software can recognize, validate, and navigate among tags and attribute values, but not within the content of tags. So the more tags present in the XML, the more detailed the validation, navigation, and indexing that can be by *standard XML software*.

“Standard” XML software includes validating parsers, which assure the correctness of the tags and attributes, stylesheets which can transform the format, and indexing services that can analyze the tags and attributes for search engines. Usually, these services use tags and attributes, and cannot do much with the content between the tags.

In all cases, problem specific software must “take over” the interpretation at some point. One way to view it is that the standard XML software breaks up the text into pieces, and each piece is handed to the application to interpret. The markup determines what pieces will be passed to the application.

The distinction matters because the standard XML software is available everywhere, users do not need to download or install the software. The application specific software may not be installed everywhere, and must be distributed to all users and kept consistent. Thus, the tags and attributes will be accessible practically everywhere, the content of the tags will be accessible only where the right software is installed.

In the first example, standard XML based software can recognize the array, but then must pass the whole matrix to non-standard software. In the second example, XML can find the rows, but not a single element within a row. And so on, through more and more detailed markup: the more information that is in the XML tags, the more that standard services can understand about the structure of the information.

On the other hand, extremely detailed markup is annoying and makes the array many times larger than the binary representation would be, and far less human readable. Furthermore, this markup is unnecessary except for the XML parsers. In the example, the markup tags contribute nothing to the numeric meaning of the matrix, and in fact they are seriously detrimental to performing numeric algorithms: implementing the elegantly simple operation of matrix multiply using the XML tags would be preposterous. In fact, it is likely that the XML would be processed to build the matrix in memory (i.e., as binary data), and then process it.

#### A.4. XML With Pointers to External Data

XML provides an additional option: the XML markup can contain pointers to external data. The pointer can be a URL or similar tag, which can include information about how to access the data at that address. This approach is analogous to hypertext links that are used to embed graphics and scripts in HTML documents. Note that the pointer needs to have sufficient information so it is possible to access the numbers. For example, a conventional name (such as a MIME type) can state what software or conventions should be used, and some kind of identifier for the specific object may be needed, e.g., for a particular matrix in a complex file.

Continuing the example of a two-dimensional matrix, the array may be stored externally and pointed to with a URL or similar tag. So the matrix might be represented as:

```
<matrix>
<link URL="some.url" FileType="a mime type" ObjectName="id
of matrix" />
</matrix>
```

As in the examples above, the XML can have much more detailed information, including:

- Multiple addresses, e.g., for replicates
- Detailed parameters for the access method(s)
- Specification of the format of the retrieved data
- Specification of a sub-set or other selection from a large dataset

In this approach, the XML markup contains a description of *how to get the data*, rather than the data itself.

This concept can be extended to have the XML include description of algorithms that should be used to access the data. For example, the XDF language defines an XML language for simple data retrieval operations (readline, skip, etc.) [46]. XML Query languages define common database operations (select-where, and so on) [43]. The Semantic Web will provide other XML “programming languages”, to define logic programming statements [4, 8, 12].

There may be good reasons to use a description of the data or data access method. Fundamentally, the use may not require the data itself, or it may be infeasible to use the actual data, because:

- The data may be too large to transfer
- The data may be off-line
- The data may be unique, e.g., there can be only one true version of the “original” dataset.
- There may be many, many data objects, the user seeks to find a relative few

A distinctive advantage of this approach is that it is easy to construct a “virtual dataset” that refers to multiple data sources. For example, one might construct a multi-layer map with data from DEM, sensor data, satellite imagery, and other data from different sources. The XML could define characteristics of the layers, but point to the data “in situ” in the original sources.

```
<map>
<layer name="elev">
<pointer-to type="DEM" object="map1">
address of DEM file
</pointer>
</layer>
</layer name="satellite-product-id">
```

```
<pointer-to type="hdf-eos" object="datasetname">  
address of NASA data file  
</pointer>  
</layer>  
...
```

## References

1. Allen, Gabrielle, Goodale, Tom, Lanfermann, Gerd, Radke, Thomas, Seidel, Edward, Benger, Werner, Hege, Hans-Christian, Merzky, Andre, Masso, Joan, and Shalf, John, "Solving Einstein's Equations on Supercomputers," *IEEE Computer*, vol. 12, no. 12, pp. 52-58, 1999.
2. Berners-Lee, Tim, Hendler, James, and Lassila, Ora, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 35-43, 2001.  
<http://www.sciam.com/2001/0501issue/0501berners-lee.html>
3. Committee, Federal Geographic Data, "CSDGM Version 2 - FGDC-STD-001-1998,"  
<http://www.fgdc.gov/metadata/contstan.html>.
4. daml.org, "The DARPA Agent Markup Language Homepage," <http://www.daml.org>.
5. Data Format Description Language (DFDL) Working Group, "Data Format Description Language," <http://www.epcc.ed.ac.uk/dfdl/>.
6. Data Mining Group, "PMML Version 2.0," <http://www.dmg.org/pmml-v2-0.htm>.
7. ESML, "Earth Science Markup Language," <http://www.esmf.ucar.edu/>.
8. Fikes, Richard, Hayes, Pat, and Horrocks, Ian, "DAML Query Language (DQL): Abstract Specification (April 2003)," <http://www.daml.org/2003/dql/dql>.
9. Folk, Milke, "Personal Communication," , 2002.
10. geotiff, "GeoTIFF," <http://remotesensing.org/geotiff/geotiff.html>.
11. Goble, Carole and De Roure, David, "The Grid: An Application of the Semantic Web," *ACM SIGMOD Report*, vol. 31, no. 4, pp. 65-70, 2002.
12. Grosz, Benjamin N. and Poon, Terrence C., "Representing Agent Contracts with Exceptions using XML Rules, Ontologies, and Process Descriptions," *Workshop on Rule Markup Languages for Business Rules, at International Semantic Web Conference*, Sardinia, 2002.
13. HDF, "Converting between HDF (4.x) and HDF5," <http://hdf.ncsa.uiuc.edu/h4toh5/>.
14. HDF, "DODS and HDF5," <http://hdf.ncsa.uiuc.edu/apps/dods/>.
15. HDF, "The HDF5 XML Information Page," <http://hdf.ncsa.uiuc.edu/HDF5/XML/>.
16. HDF, "HDF-FITS Conversion Page," <http://hdf.ncsa.uiuc.edu/fits/>.
17. Hierarchical Data Format, *HDF5 User's Guide*, United States Patent
18. Hierarchical Data Format, "Hierarchical Data Format," <http://hdf.ncsa.uiuc.edu>.
19. IBM, "DB2 XML extender," <http://www-3.ibm.com/software/data/db2/extenders/xmlxt/>.
20. IBM, "Direct Internet Message Encapsulation (DIME)," <http://xml.coverpages.org/dime.html>.
21. Kapadia, Apu and Yeager, Nancy, "Performance Evaluation of HDF Version 4 Chunking Facility when used for Subsetting Pathfinder Data," , 1999.  
<http://hdf.ncsa.uiuc.edu/apps/dods/perfeval/report.html>
22. McGrath, Robert E., "Experiment with XSL: translating scientific data,"  
<http://hdf.ncsa.uiuc.edu/HDF5/XML/nctoh5/writeup.htm>.
23. McGrath, Robert E., "Integrating Scientific Datasets and Digital Libraries,"  
<http://www.ncsa.uiuc.edu/People/mcgrath/CESDIS/>.
24. McGrath, Robert E., Futrelle, Joe, Plante, Ray, and Guillaume, Damien, "Digital Library Technology for Locating and Accessing Scientific Data," *ACM Digital Libraries '99*, 1999.
25. NASA, "Global Change Master Directory," <http://gcmd.gsfc.nasa.gov/>.
26. NASA Astrobiology Institute, "<http://astrobiology.arc.nasa.gov/roadmap/roadmap.pdf>,"  
NASA, November 20, 2002. <http://astrobiology.arc.nasa.gov/roadmap/roadmap.pdf>
27. NCSA, "Astronomy Digital Image Library (ADIL)," <http://adil.ncsa.uiuc.edu/>.
28. NCSA, "Scientific Portal Alliance Expedition," , , 2003.  
<http://www.extreme.indiana.edu/alliance/>



29. NEES, *NEES Consortium, Inc.*, United States Patent
30. NEES, "NEESgrid," <http://www.neesgrid.org/about/index.html>.
31. OpenGIS Consortium, *OpenGIS(R) Abstract Specification*: OpenGIS Consortium, 1999.
32. Schmidt, Laurie J., "The Universal Language of HDF-EOS," NASA DAAC Alliance, December 22, 2000. <http://earthobservatory.nasa.gov/Study/HDFEOS/>
33. The FITS Support Office, "Flexible Image Transport System," <http://fits.gsfc.nasa.gov/>.
34. TIFF, "TIFF Software," <http://www.libtiff.org/>.
35. Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., and Vanderbilt, P., "Grid Service Specification," BWD-R, October 4, 2002.  
[http://www.ggf.org/meetings/ggf6\\_wg\\_papers/draft-ggf-gridservice-04\\_2002-10-04.pdf](http://www.ggf.org/meetings/ggf6_wg_papers/draft-ggf-gridservice-04_2002-10-04.pdf)
36. unidata, "NetCDF," <http://my.unidata.ucar.edu/content/software/netcdf/index.html>.
37. Unidata, "The NetCDF Markup Language (NcML)," <http://www.unidata.ucar.edu/packages/netcdf/ncml/>.
38. W3C, "Extensible Stylesheet Language (XSL): Version 1.0," W3C Recommendation, 16 November, 1999. <http://www.w3.org/TR/xslt>
39. W3C, "The Semantic Web," <http://www.w3.org/2001/sw>.
40. W3C, "SOAP Version 1.2 Part 1: Messaging Framework," W3C Candidate Recommendation, 19 December, 2002. <http://www.w3.org/TR/soap12-part1/>
41. W3C, "Web Services Architecture," W3C Working Draft, 14 November, 2002.  
<http://www.w3.org/TR/2002/WD-ws-arch-20021114/>
42. W3C, "XML Schema Part 2: Datatypes," W3C W3C Recommendation, W3C Recommendation 02 May, 2001. <http://www.w3.org/TR/xmlschema-2/>
43. W3C, "XQuery 1.0: An XML Query Language," W3C, 02 May, W3C Working Draft, 2003.  
<http://www.w3.org/TR/xquery/>
44. Westhead, Martin and Bull, Mark, "Representing Scientific Data on the Grid with BinX – Binary XML Description Language," EPCC, University of Edinburgh,, 2003.  
<http://www.epcc.ed.ac.uk/~gridserve/WP5/Binx/sci-data-with-binx.pdf>
45. Westhead, Martin and Chappell, Alan, "Data Format Description Language - Structural Description," Global Grid Forum, GGF Data Format Description Language Working Group, INFORMATIONAL, 4 June, 2003.
46. XDF, "eXtensible Data Format," [http://xml.gsfc.nasa.gov/XDF/XDF\\_home.html](http://xml.gsfc.nasa.gov/XDF/XDF_home.html).
47. Yan, Kun and McGrath, Robert E., "Experiments with JSP, XML, CORBA, and HDF," <http://hdf.ncsa.uiuc.edu/HDF5/XML/JSPEperiments/>.
48. Yeh, Pen-Shu, Xia-Serafino, Wei, Miles, Lowell, Kobler, Ben, and Menasce, Daniel, "Implementation of CCSDS Lossless Data Compression in HDF," *Earth Science Technology Conference*, Pasadena, 2002.