

# A Binary XML for Scientific Applications

Kenneth Chiu<sup>2</sup>, Tharaka Devadithya<sup>1</sup>, Wei Lu<sup>1</sup>, Aleksander Slominski<sup>1</sup>

1. Computer Science Department, Indiana University

2. Department of Computer Science, State University of New York (SUNY) at Binghamton

## Abstract

*XML provides flexible, extensible data models and type systems for structured data, and has found wide-acceptance in many domains. XML processing can be slow, however, especially for scientific data, thus leading to the conventional wisdom that XML is not appropriate for such data. Instead, data is stored in specialized binary formats, and is transmitted via work-arounds such as attachments and base64 encoding. Though these work-arounds can be useful, they nonetheless relegate scientific data to second-class status within the web services framework; and they generally require yet another API, data model, and type system. An alternative solution is to use more efficient encodings of XML, often known as “binary XML”. Using XML uniformly throughout an application simplifies and unifies design and development. In this paper we present a binary XML format and implementation for scientific data called Binary XML for Scientific Applications (BXSA). We show that performance is comparable to that of commonly used scientific data formats such as netCDF. These results challenge the prevailing practice of handling control and data separately in scientific applications, with web services for control and specialized binary formats for data.*

## 1. Introduction

With their focus on loose-coupling, extensibility, and interoperability, web services have become the dominant framework for building large-scale distributed systems. They have garnered much attention from both industry and academia, leading to wide-spread activity in research, specification, and implementation. This activity will culminate in a set of standards, which, if widely-adopted, may herald an unprecedented degree of wide-scale interoperability.

XML holds a unique, pervasive role within web services, and XML-based concepts and abstractions are inextricably intertwined with web services. Though not strictly required by web services standards, XML is also the de facto standard wire format for web services.

For large data sets, however, XML’s processing costs are usually considered prohibitive. Thus, the conventional wisdom is that web services are for *control*, while *data* must be stored, managed, and transmitted using binary, non-XML-based formats. To maintain this separation between control and data, distributed scientific applications such as LEAD [4] typically handle data in a non-web-services-

based, ancillary framework consisting of binary data formats, various mechanisms from transmitting files in these formats, and additional libraries and APIs for using these formats. The result is essentially two systems that parallel each other: a web services infrastructure for the control of scientific applications, and a bespoke infrastructure for scientific data.

Treating data in a completely different conceptual framework has several disadvantages. Beyond the cost of using and maintaining a separate code infrastructure for the data, users and developers must bear the additional mental cost of learning two different sets of concepts, terminology, and software. Furthermore, no universally-accepted standards exist for integrating the two frameworks. One technique is to use attachments of some kind to send the binary data. Another common technique is to send a URL referring to the data, which is then used to retrieve the data. The emerging SOAP Message Transmission Optimization Mechanism (MTOM) [28] standard promotes yet another technique.

We posit that the perceived performance problems of XML are not due to its logical structure (based on trees consisting of parent and child nodes with attributes), but rather to its lexical and syntactic format. This suggests addressing XML’s performance issues by developing alternate, efficient serializations of this logical structure. With these alternate serializations, scientific data could be directly transmitted as “binary XML”,<sup>1</sup> thus directly integrating data into the web services framework, and obviating any special treatment. The resulting unified framework will bring the benefits of web services to the data—as well as the control—of scientific applications. A common API representing the logical structure can be used for both textual XML and binary XML, thus allowing applications to transparently use either serialization without code modifications.

The World Wide Web Consortium (W3C) has recognized the importance of binary XML by forming the XML Binary Characterization (XBC) Working Group (WG) [23]. Unfortunately, most of the activity has focused on the needs of business and industry rather than the scientific computing community. Thus, many binary XML proposals are deficient for scientific computing.

Therefore, we have developed a specific binary XML format called Binary XML for Scientific Applications

---

<sup>1</sup>Technically this term is an oxymoron, but in practice the it seems to create no confusion. In this paper, we use the term “XML” broadly to include non-textual serializations of XML-based data models, usually relying on context to disambiguate our meaning where applicable.

(BXSA) (pronounced “bix-sah”). BXSA is intended to serve as a research vehicle through which the scientific computing community can express their needs and experiences, which can then be used to guide the development of a W3C binary XML standard. The active participation of the scientific computing community can ensure that the resulting standard will satisfy the requirements of scientific computing as well as business computing.

The adoption of a binary XML will not solve all data format problems. Element names and the structure can differ between producers and users of data. This is also true, however, even for textual XML, and we anticipate that different communities will create their own XML schemas and vocabularies. Furthermore, these communities will likely create convenience wrappers around the binary XML that will simplify their common usage patterns. Wrappers that clone existing APIs such as that of netCDF are also likely.

By basing the data formats on XML, however, each community avoids reinventing the wheel for aspects which the web services community has already addressed, and can instead focus on problems unique to that community. Using XML also provides a common basis for interoperability that can easily leverage techniques such as semantic mediation [12] and XSLT [24], and assist with automated provenance management. Finally, by building libraries for formats such as netCDF, HDF5, GRIB, and others on a common BXSA implementation, they can avoid duplicating code for low-level data handling such as laying out arrays on disk.

This paper makes several significant contributions. First, we present BXSA, a fast binary XML format, and show a research implementation that it is comparable in speed to netCDF and HDF5, thus warranting its use as an alternative to existing data formats. Second, we analyze most of the properties suggested by XBC WG in the context of scientific computing, characterizing their relevance and importance to scientific applications. Third, and perhaps more importantly, we also challenge the conventional position that scientific data must necessarily be divorced from web service concepts and principles. Although we understand that change must be gradual, we believe that eventual unification will lead to benefits, both foreseen and unforeseen.

In Section 2, we consider the motivation for BXSA in greater depth. Section 3 describes the notion of binary XML. BXSA itself is described in Section 4, along with an analysis of different binary XML properties in the context of scientific applications. We present in Section 5 performance results. Related work is discussed in Section 6.

## 2. Motivation

XML is a mature, nearly ubiquitous format for structured data, with powerful type languages such as XML Schema and RELAX NG. These languages define type systems for XML by enabling the precise specification of sets of XML documents, thus facilitating interoperability between producers and consumers of XML documents. Though many web service standards directly depend on XML, WSDL, the

standard for actually defining a web service, does not mandate the use of XML as the communication format for web services. XML is so deeply ingrained in web services, however, that the use of non-XML formats would probably create more issues than it would resolve. Thus, XML is the *de facto* standard for communications via web services.

XML’s poor performance, however, prevents it from being used to transmit and store scientific data. This poor performance arises from a number of causes, including its verbosity and the computational costs of converting floating-point numbers from ASCII to a native machine representation such as IEEE 754 [8]. Because of these performance issues, most developers of distributed scientific applications assume as a foregone conclusion that application data must be handled separately from application control, via the specialized transport of binary data files. Since this data system is not based on XML and XML-concepts, it cannot be seamlessly integrated into the web services framework; Handling data in this completely different conceptual framework has a number of disadvantages:

- Multiple type systems.
- Multiple concepts, terminology.
- Multiple data models.
- Multiple specifications, documentation.
- Multiple libraries.
- Non-standard, incompatible ways for linking data to control.

## 3. Binary XML

XML is specified as a textual format for structured data. Beyond its syntax, however, the specification also implicitly defines an abstract data model for tree structured data consisting of parent nodes and child nodes, with their associated attribute, content, and other information. This model does not inherently preclude an efficient encoding, and thus alternate serializations of an XML document’s logical structure can have much greater performance than textual XML, especially for scientific data consisting of large arrays of numbers. These alternative serializations are commonly known as “binary XML”, since they utilize the full 8-bit range of the constituent bytes.

The need for an abstract XML data model has been anticipated by the W3C, and has been explicitly defined in the XML Information Set [26] specification, which uses the term *infoset* to denote the information content of an XML document. The XML infoset data model closely follows the information contained within an XML document in isolation. Several other data models, such as the post-schema validation infoset [27] and the XQuery data model [29] exploit additional information obtained from other sources such as a schema.

A high-performance serialization for XML could obviate the need to handle data in a separate framework. As XML, a data set could simply be just another element within a SOAP message (also in binary XML), thus promoting data to first-class citizenship within the web services framework. Data

could then be described and validated using XML schema and associated standards and tools. All the benefits of web services provided to application control can now also be shared by application data. Furthermore, since the API is independent of the underlying serialization, one library and API can be used for both control operations and data operations. Since WSDL allows bindings to transport protocols other than HTTP, bringing data into the web services framework will still allow the use of mechanisms such as GridFTP, and will actually encourage research in alternative transport bindings for web services.

We do not see binary XML as a replacement for existing data formats, but rather as a standard representation for their common aspects. Communities already use textual XML in this manner, developing schemas and vocabularies for specifying specialized XML languages such as MathML [21] and CML [17]. With a fast serialization of XML available, we anticipate that communities will extend existing XML languages to include data that they might have previously considered impractical due to performance constraints. We also see benefit in wrapping a binary XML representation of netCDF within an netCDF API, for which the XML representation of netCDF (NeML) metadata [20] could serve as the basis.

### 3.1. Proposed Formats

The W3C convened a workshop in 2003 to discuss the need for binary XML. During this workshop, over 25 different binary formats were proposed. We briefly discuss here some of the more relevant proposals. Most of the proposals did not handle large arrays of doubles in binary form. Since this necessitates an expensive ASCII-to-double conversion, we do not consider most such formats here.

**bnux.** The bnux format is part of the Nux [3] project, an open-source extension of the XOM [13] and Saxon [16] XML libraries. It does not focus on large arrays of doubles, however, and is buffer-oriented. It is thus not suitable as an alternate XML serialization for large data, which may need to be streamed to avoid excessive memory costs.

**BXML.** BXML [6] is being developed by CubeWerx for an OpenGIS standard. It does support arrays of doubles in IEEE 754 format. Some features important for a scientific data format, such as accelerated sequential access, are not important for GIS, and they do not appear to be interested in pursuing these features [1].

**Fast Infoset.** The Fast Infoset [19] is an effort by Sun to develop a fast serialization for XML. Unfortunately, since it is being submitted as an ISO specification, it is currently not publicly available for review and comment. It is based on ASN.1 [15], which is commonly used in the telecommunications industry. ASN.1 is a large, complex standard, and is probably not ideally suited to the needs of scientists.

**Millau.** Millau [11] is an efficient format for XML. It does handle single-precision floating-point numbers in IEEE 754 representation, but does not have a facility for representing arrays of numbers. Millau is primarily targeted toward the business community.

## 4. BXSA

Although some of the binary XML proposals seem promising for scientific data, none of them explicitly focus on scientific use cases, which are dominated by large arrays of numbers, or data streams of sensor data. For example, many of them ignore the single greatest cause of poor XML performance for scientific data, which is the conversion of floating-point numbers from ASCII to the machine representation. Thus, to address the needs of scientific applications, we created BXSA as our own vehicle for investigating binary XML for scientific data.

Our design for BXSA incorporates a number of techniques driven by the requirements of scientific applications. BXSA is a layered format, and depends on XBS [7] to pack fundamental types into a sequence of bytes. The XBS format is a minimalistic format that supports 1-, 2-, 4-, and 8-byte integers; 4- and 8-byte floating-point numbers, and 1-dimensional arrays. All numbers are aligned to a multiple of each types size. Both little-endian and big-endian formats are supported. We currently do not support Unicode, but plan to in the future.

The BXSA data model is currently based on the XML infoset, but has been augmented with atomic, typed values. This allows our API to represent numbers in their native, machine form, rather than as character strings. Our API is DOM-like, but more closely follows the XML infoset. Future work will include developing SAX and pull APIs, and will also investigate the XQuery data model [29], which explicitly specifies typed, atomic values for element content, and can represent sets of XML nodes.

BXSA focuses on scientific computing, and the current implementation is not necessarily fast for all XML documents. As BXSA matures, we anticipate that additional research will improve performance for a wider set of XML documents.

### 4.1. Variable-Length Sizes

To avoid arbitrary limits without sacrificing compactness, BXSA uses variable-length sizes in a number of places. These sizes use the leading bits to code the number of bytes in the size. A leading 0 bit indicates that the size has only one byte, giving a total of 7 bits for the size. A leading 10 bit sequence indicates that the size has two bytes, giving a total of 14 bits. A leading 110 bit sequence indicates that the size has 4 bytes, giving a total of 29 bits. Finally, a leading 1110 bit sequence indicates that the size has 8 bytes, giving a total of 60 bits. This can easily be extended to 123 bits if necessary.

### 4.2. Arrays

Since the XML Schema specification does not define an array, we have followed the SOAP data model and encoding rules in defining BXSA arrays to be a parent element containing a sequence of child elements. We also impose the additional restriction that the child elements must all have the same name and type, and have no attributes other than a `xsi:type` attribute. This representation can be modeled

by the corresponding XML Schema `<sequence>`, and we believe that it will be compatible with the vast majority of array representations used in web services.

By using this definition of arrays, BXSA can then use a packed storage format which will be fast and efficient, both in memory and on disk, on most architectures. Since each child element does not have separate attributes, they can all be packed together. Our current API does not provide access to the individual array elements as separate XML elements (and our implementation does not store the child elements' name), but a future implementation may provide such an option for certain use cases. In such a scenario, the implementation would have enough information to create objects representing the individual XML elements from the BXSA format.

### 4.3. Frames

A BXSA document consists of a sequence of *frames*, each of which may be one of several kinds, as identified by its type code. All frames have a one-byte prefix (Figure 1) consisting of two byte-order bits and a 7-bit type code, followed by the size of the frame as a variable-length size.

The byte-order bits are associated with each frame rather than the entire BXSA document in order to make it simpler to embed BXSA documents within other documents. An embedded document can simply be directly wrapped by an outer document without regard for possibly differing byte-orders. This also simplifies transport, especially in multi-hop situations involving intermediaries, where BXSA documents may be split and then recombined in various ways.

The possible frame types are described further below.

**Leaf Element Frame.** This frame is used to code an element that has no child elements (Figure 1). The value is typed, which means that it is stored in a native form. Following the common frame prefix, a leaf frame contains the namespace declarations, the element namespace and name, the attributes, and finally the element value itself.

Each namespace declaration consists of the prefix and the URI. To save space, namespaces are referenced in elements via an index into a table. Each element defines a new namespace "scope", and thus creates a new table. As a namespace declaration is processed, an entry in a table is created. Since an element may use a namespace declaration created by a parent, a namespace reference also includes the namespace scope depth. This is a count backwards to indicate where the namespace was declared.

**Compound Element Frame.** This frame is used to encode an element that has child elements. Its format is similar to that of a leaf element, except that rather than a type code followed by the value, there is a count followed by a sequence of frames. Each frame defines either character data or a child element.

**Array Element Frame.** This frame is used to encode an element that contains an array. The format of the frame is identical to that of a leaf element, except that rather than being followed by a single number, it is followed by a count and a packed sequence of numbers. Our current design does

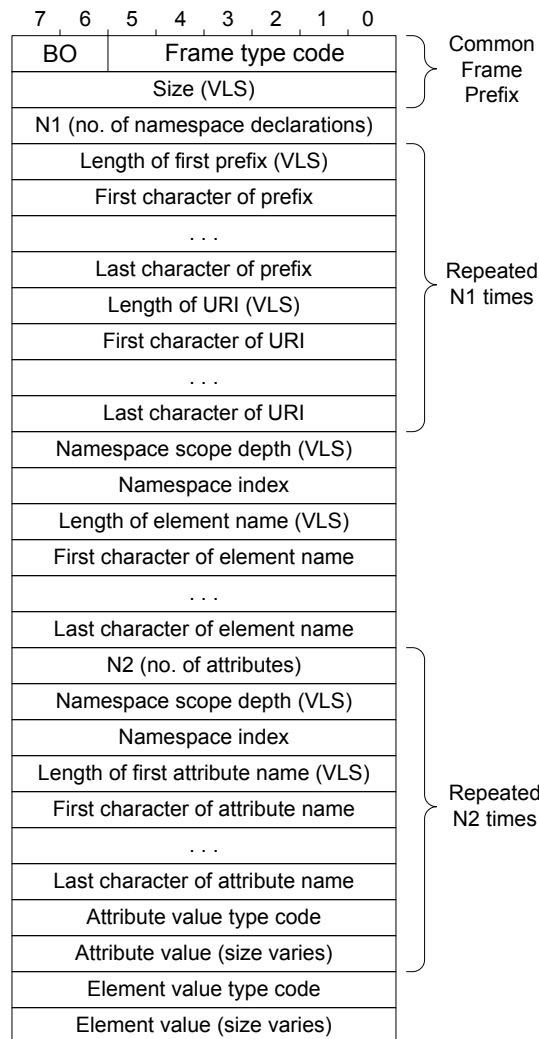


Figure 1: This shows the format of a leaf frame. BO is the byte-order bit. VLS indicates that the associated number is a variable length size as described in the text of the paper.

not support arrays of strings, though these could be represented simply as a sequence within a compound element.

**Character Data Frame.** This frame is used to encode character data in mixed content compound element frames. Such an element would have associated element frames and character data frames that would encode the child nodes. Its format consists of the common frame prefix followed by a count of characters, followed by the sequence of characters.

### 4.4. Properties

An XML data model can be serialized in many different ways. Each of these serializations may have different properties. The W3C working group on binary XML (XBC) produced a set of various properties that they consider important [22]. We discuss a subset of these properties in the context of scientific computing. In addition, we present here

a few properties not yet considered by the W3C.

We do not currently support all of the properties that we consider important for scientific computing. We believe that some of the properties require more real-world experience with BXSA to determine exactly to what degree and how some of the properties should be supported.

**Accelerated Sequential Access.** This is the ability to sequentially search through an XML document for particular items without fully parsing all parts of the document. This is useful in scientific applications that query data streams, or are needed to extract a particular element from an XML document. This would also support the use of footers for intermediaries that do not need to process the body of a message. BXSA supports this property by prefixing each frame with its size, thus allowing it to be skipped.

**Compactness.** A compact binary XML is achieved by eliminating redundancy. Because scientific data often consists of large arrays of numbers, any binary XML format for scientific computing must not incur any per number overheads for such arrays. That is, it must pack such numbers without any additional padding over their native machine representation.

BXSA encodes arrays of numbers as tightly packed machine representations. Arrays of records are not currently very compact, and must simply be represented as sequence of compound elements. This is highly inefficient, since metadata about each compound element is repeated many times. In the future, we plan to compress these arrays by using record descriptors. These descriptors will describe the structure of a record. An array of these records can then be packed without repeating the metadata.

**Encryptable.** Encryption can be handled at two levels. It can be handled within the binary XML encoding itself, or it can be handled at the XML data model level. We believe that in most cases it should be handled at the XML data model level, since this will leverage ongoing XML encryption efforts and standards.

**Fragmentable.** This refers to the ability to be able to represent fragments of an XML document, along with the necessary context. This requires the ability to serialize namespaces outside of any element, also possibly the values of some distinguished attributes such as `xml:lang`. We believe that this is useful for scientific computing, and is not too difficult to support. We have designed BXSA to support the serialization of namespace context, but have not yet addressed the issue of distinguished attributes.

**No Arbitrary Limits.** A serialization format, especially for scientific data, should be very wary about imposing any kind of size limits. We thus believe that this is important for a binary XML for scientific applications. BXSA uses variable length sizes to avoid arbitrary limits.

**Processing Speed.** This is a general property referring to the performance of parsing and related operations. This is important for scientific applications, since the data rate and size may be very high. Normally, for scientific applications the conversion of floating point numbers from ASCII repre-

sentation to IEEE 754 representation is by far the most time-consuming processing operation. Thus, any binary XML must avoid this conversion.

**Random Access.** Random access is the ability to access a particular element without any kind of sequential traversal. This property is important for scientific applications. A binary XML should be suitable as an on-disk storage format as well as a wire representation. As a file on disk, a scientific application may only need to access a subset of the data. If the file is very large, requiring that such access traverse the entirety of the bytes is prohibitively time-consuming. Such random access can be supported by the use of indices in the binary XML. BXSA does not currently support random access, though we note that netCDF does. Future work will address this issue.

**Robustness.** This refers to the ability to gradually degrade the content of a document when damaged. It requires the ability to isolate damaged sections. This might be important for transmission of data in noisy conditions. Though we consider robustness important, BXSA does not currently support this. Future work may address this issue.

**Self-Contained.** A self-contained binary XML document is one that can be completely interpreted without referring to any external metadata. This is important for simplicity when designing large data archives. For scientific applications, a binary XML format should support both self-containment in the usual case, and the ability to refer to external metadata. BXSA documents are self-describing and self-contained. They can represent XML without any schema information. If type information is available, either from the application or from other source, it is used to represent numerical data in machine-native format.

**Signable.** A format is signable to the extent to which it makes the creation and validation of digital signatures both straightforward and interoperable. Since the message digest must be computed on exactly the same sequence of bytes at both sender and receiver, this sequence must be unambiguously specified. Multiple byte representations of the same logical XML must be mapped to a single byte representation by defining a canonical form.

This presents some interoperability issues when considering alternate serializations of XML. If a message is signed as textual XML, but then transcoded to binary XML, some necessary information may be lost. For example, the textual number “1.2E10” may be reproduced as “12.0” on the receiving end, which would lead to an unverifiable signature.

Thus, properly computing the digest for XML transcoded from textual form requires that floating-point numbers be reproduced exactly as they were originally, even including lexical differences that are not numerically significant. This requires that such variations be stored somehow in the binary XML, so that they can be reproduced for computing the message digest. To avoid storing these variations for arrays of floating-point numbers, we will require that textual XML that is being transcoded to binary XML must have all numbers in

arrays in canonical XML Schema format [27]. Note that a Canonical XML [25] document does not necessarily contain canonical XML Schema numbers.

This property is important for scientific computing, and though we do not currently support this, we intend to address it in future work.

**Streamable.** A binary XML format is streamable if a partial XML document can be generated from partial input to the serializer, and vice versa for deserialization. This property is important for many scientific applications, especially for reducing memory usage during multiple processing stages. This property conflicts somewhat with the Accelerated Sequential Access property, however.

Accelerating sequential access typically involves inserting the sizes of elements before their contents, so that unwanted elements may be skipped in their entirety without processing. Computing the size of these elements during serialization may require lookahead, however. This would prevent the XML document from being output until a substantial portion of the input data structure has already been traversed, thus leading to a low degree of streamability.

We believe that a mixed approach is probably best. The size of many kinds of elements can be computed quickly for binary XML without actually serializing the element. For situations where even this quick traversal is too expensive, the size can be omitted, and in these cases, that particular element would not be skippable. BXSA is currently streams large arrays well, because size computation does not require traversal. It streams deeply nested, complex XML with many small elements less well.

**Transcodable to XML.** A binary format that is transcodable to XML can be converted to textual XML, and then back to binary XML without change. Such a binary format must also support the ability to convert a textual XML document to the binary XML, and then back to the textual format without change. This is important for scientific applications for interoperability purposes. Some required tools may only work with textual XML. The primary challenge for supporting this property is the handling of numerical data, as mentioned above. BXSA currently transcodes except for the handling of floating-point numbers, which are converted to full precision regardless of the original input.

**Memory-Mappable.** If the stored format of an array of numbers conforms to the format in-memory, then large arrays can be read by simply using memory-mapped file I/O. This will avoid an extra copy, making such I/O efficient. This property is not part of the XBC characterization document, but an additional property that we have deemed important for scientific applications. BXSA is memory-mappable.

**Embeddable.** Embeddability is the ability to quickly embed a BXSA document as a fragment within another BXSA document. This is important for sending a standalone data set (stored as a BXSA document) via a web service. This property is also not part of the XBC charac-

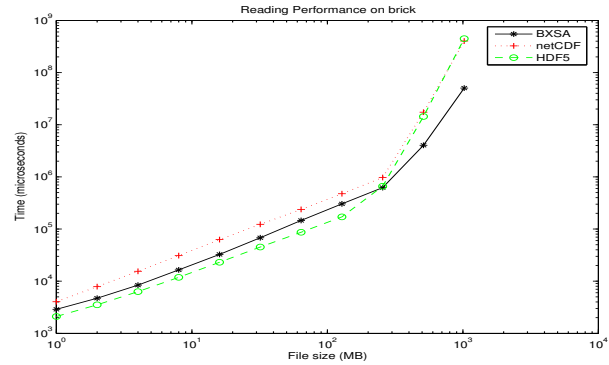


Figure 2: This graph compares the read performance of BXSA to netCDF and HDF5 on a 2.8 GHz Pentium with 1 GB of RAM. For the larger file sizes, the data cannot be entirely cached in RAM.

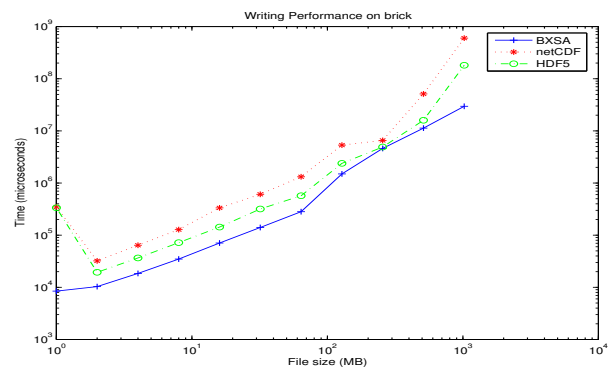


Figure 3: This graph compares the write performance of BXSA to netCDF and HDF5 on a 2.8 GHz Pentium with 1 GB of RAM. For the larger file sizes, the data cannot be entirely cached in RAM.

terization document, but is rather an additional property we consider important. BXSA is embeddable.

## 5. Results

We compared our BXSA implementation to two different scientific data formats, netCDF [18] and HDF5 [14], by testing the reading and writing a data file to local disk. Our data file was derived from a sample file used for the LEAD [4] project. The data file consisted of atmospheric information, which depended on four parameters, namely *time*, *y*, *x*, and *height*. These parameters were defined as dimensions. The file had two, one-dimensional short arrays and two, four-dimensional large arrays. All arrays contained double-precision, 8-byte floating-point numbers. The short arrays occupied little space and represented the values corresponding to time and height dimensions of the two large arrays. For BXSA, the multidimensional arrays were mapped to equivalently-sized 1-D arrays.

We tested file sizes from 1 MB to 1024 MB, increasing exponentially. Since two 4-dimensional arrays occupied most of the file, we varied only one dimension to get the required file sizes. The dimensions *y*, *x*, and *height* were fixed

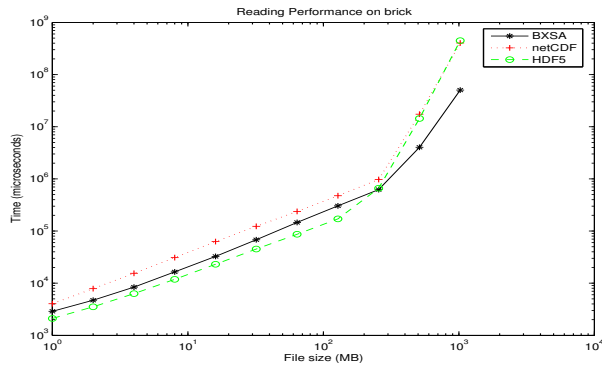


Figure 4: This graph compares the read performance of BXSA to netCDF and HDF5 on a 2.4 GHz dual-Xeon machine with 2 GB of RAM. Main memory is sufficient to cache the entire data set for all tested sizes.

at 128, 32, and 16, respectively. Since they contain double-precision numbers (8-bytes each), one record of these 3 dimensions required 1 MB. The record dimension was varied to get different file sizes. For each file size, we averaged five trials, with an initial run to warm the file cache. The times for writing include the overhead of populating the in-memory data structures with the initial data, as well as the time to actually write the data file.

The tests were performed on two machines: *thor2* and *brick*. The *thor2* machine is a Linux 2.4 box with 2, 2.4 GHz Pentium Xeon CPUs and 2 GB of RAM. The *brick* machine is a Linux 2.6 box with a 2.8 GHz Pentium CPU and 1 GB of RAM. The test on *brick* is meant to test conditions where there is not enough RAM to cache both the entire file and the in-memory representation of the data.

The results are shown in Figures 2 through 5 and show that BXSA is competitive with other data formats. In some cases it is even faster. For large arrays, we do not expect to see any significant change in this performance, even as BXSA evolves. BXSA treats a numerical arrays as solid, contiguous regions of memory. Serializing a large array consists of a small amount of overhead followed by a single, long, `write()` system call.

Our intent is not to demonstrate that our specific BXSA implementation is optimal in any sense, but rather to show a proof-of-concept that binary XML is a viable approach to unifying control and data.

## 6. Related Work

A number of other efforts seek to integrate foreign data into a web services framework. These efforts generally exploit the observation that foreign data formats can be mapped to XML-like abstractions. They can then be described with schema languages designed for XML.

While such efforts play an important role in that they can help integrate existing data into the web services framework, they do not fulfill the same role as binary XML. For example, "impedance" mismatches can result in more than

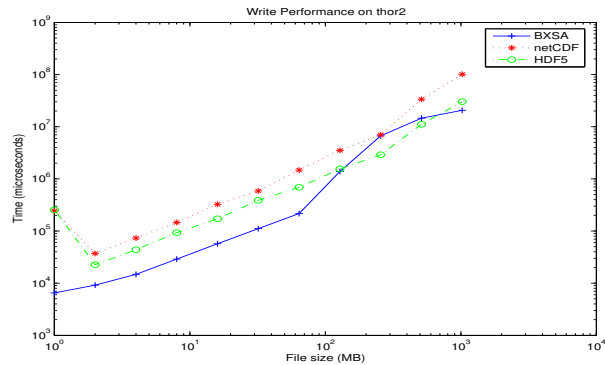


Figure 5: This graph compares the write performance of BXSA to netCDF and HDF5 on a 2.4 GHz dual-Xeon machine with 2 GB of RAM. Main memory is sufficient to cache the entire data set for all tested sizes.

one way to map a given foreign data model to XML. Also, they cannot easily inversely map from arbitrary XML to the foreign data model. Thus, they cannot serve as a complete alternate serialization for XML, and allow data to be seamlessly and efficiently sent within the SOAP messages typical of web services.

**DFDL and BinX.** The Data Format Description Language (DFDL) [9] is a product of the Global Grid Forum (GGF) [10], and, as its name suggests, is a language for describing data formats. DFDL describes all aspects of a data format, from byte-order to structure. DFDL is derived from BinX [5], which concentrates on binary data formats.

By mapping binary formats to XML Schema, DFDL can be used to parse binary data into an XML-based data model. Being descriptive, however, DFDL does not prescribe how the data is actually stored. That is, DFDL describes binary formats, while BXSA is a binary format. DFDL and BXSA may have some use cases which overlap, but are essentially complementary.

**XMIT.** XMIT [30] is a toolkit for describing PBIO data with XML schemas. Its approach is less ambitious than DFDL, in that it does not seek to describe arbitrary data formats. Rather XMIT assumes that separate mappings from the binary format to XML schemas will be developed for each binary data format. XMIT's primary benefit is that it improves the flexibility of existing binary communication formats. It does not provide any kind of alternative serialization of XML.

**ESML.** ESML [2] is language for describing binary used in the earth sciences. This allows applications to access these formats using a common interface, thereby simplifying and speeding application development.

## 7. Conclusion

The prevailing practice in distributed scientific applications is to separate data from control. Control mechanisms typically adopt the web services framework, while application data is maintained in multiple, incompatible formats.

The small amounts of data required for control communication is sent as XML within the web service messages, while the large amounts of application data are typically sent as data files using GridFTP or some other mechanism. The control and data frameworks are linked using ad hoc techniques such as including a URL in a control message.

This separation of data from control is often based on the conclusion that XML is simply too inefficient for scientific data, which typically consists of large arrays of floating-point numbers. This separation, however, results in duplication of type systems, data models, and code libraries. Thus, we believe that unifying data into the web services framework will simplify the development of scientific applications. Such a unification requires an efficient serialization for XML, however.

To that end, we have developed the BXSA binary XML format. We have shown that the BXSA binary XML format is comparable in performance to netCDF and HDF5 for a typical scientific data file, which were derived from data files used in the LEAD project. These data files consist primarily of two large arrays of single-precision floating point numbers.

This rough performance parity demonstrates that binary XML is suitable as an alternative to specialized scientific data formats. Storing and sending as XML would then also directly facilitate the use of various data techniques such as semantic mediation, ontologies for data, and provenance tracking.

We have also analyzed the properties of the W3C XBC Working Group in the context of scientific computing, to guide the development of BXSA. This analysis contributes significant scientific computing concerns to the results of the XBC Working Group, which is currently focused primarily on business and industrial requirements. Without this input, a resulting binary XML standard may not meet the needs of scientific computing.

Web services will not solve all format problems. They are a promising, widely-adopted approach, however, with an active research and development community. By unifying control and data into one framework with binary XML efforts such as BXSA, the promise of web services can be more fully realized for scientific computing.

## References

- [1] BXML-CWXML FAQ. <http://www.cubewerx.com/main/cwxml/faq.html>.
- [2] Earth Science Markup Language. <http://esml.itsc.uah.edu/index.jsp>.
- [3] Nux Overview. <http://dsd.lbl.gov/nux/>.
- [4] Linked Environments for Atmospheric Discovery project web page. <http://lead.ou.edu/>, 2005.
- [5] BinX project web page. <http://www.edikt.org/binx/index.htm>, 2005.
- [6] C. S. Bruce. Cubewerx position paper for binary XML encoding. [http://www.cubewerx.com/main/HTML/Binary\\_XML\\_Encoding.html](http://www.cubewerx.com/main/HTML/Binary_XML_Encoding.html).
- [7] K. Chiu. XBS: A streaming binary serializer for high performance computing. In *Proceedings of the High Performance Computing Symposium 2004*, 2004.
- [8] K. Chiu, M. Govindaraju, and R. Bramley. Investigating the limits of soap performance for scientific computing. In *HPDC '02: Proceedings of the 11 th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002 (HPDC'02)*, page 246. IEEE Computer Society, 2002.
- [9] Data Format Description Language project web page. <http://forge.gridforum.org/projects/dfdl-wg/>, 2004.
- [10] Global Grid Forum. <http://www.gridforum.org/>.
- [11] M. Girardot and N. Sundaresan. Millau: an encoding format for efficient representation and exchange of xml over the web. In *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications networking*, pages 747–765. North-Holland Publishing Co., 2000.
- [12] A. Gupta et al. Registering Scientific Information Sources for Semantic Mediation. In *21st International Conference on Conceptual Modeling*, 2002.
- [13] E. R. Harold. Xom. <http://www.cafeconleche.org/XOM/>.
- [14] NCSA - HDF5. <http://hdf.ncsa.uiuc.edu/HDF5/>, 2004.
- [15] ITU-T. Abstract syntax notation one (asn.1). <http://asn1.elibel.tm.fr/en/standards/index.htm#asn1>, 2002. ITU-T Rec. X.680 (2002) — ISO/IEC 8824-1:2002.
- [16] M. H. Kay. Saxon the xslt and xquery processor. <http://saxon.sourceforge.net>, 2005.
- [17] P. Murray-Rust and H. S. Rzepa. Chemical markup language. <http://www.xml-cml.org/>.
- [18] Unidata - NetCDF. <http://my.unidata.ucar.edu/content/software/netcdf/index.html>, 2005.
- [19] Sun. Fast infoset. <http://asn1.elibel.tm.fr/xml/finf.htm>.
- [20] Unidata. The netCDF markup language (NcML). <http://my.unidata.ucar.edu/content/software/netcdf/ncml/index.html>.
- [21] W3C. Mathematical markup language. <http://www.w3.org/Math/>.
- [22] W3C. Xml binary characterization properties. <http://www.w3.org/TR/xbc-properties/>.
- [23] W3C. Xml binary characterization working group. <http://www.w3.org/XML/Binary/>.
- [24] W3C. XSL Transformations (XSLT) Version 1.0. <http://www.w3.org/TR/xslt>.
- [25] W3C. Canonical XML 1.0. <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>, 2001.
- [26] W3C. Xml information set (second edition). <http://www.w3.org/TR/xml-infoset/>, 2003.
- [27] W3C. Xml schema. <http://www.w3.org/XML/Schema>, 2004.
- [28] W3C. SOAP message transmission optimization mechanism. <http://www.w3.org/TR/soap12-mtom/>, 2005.
- [29] W3C. Xml query (xquery). <http://www.w3.org/XML/Query>, 2005.
- [30] P. Widener, G. Eisenhauer, K. Schwan, and F. E. Bustamante. Open metadata formats: Efficient xml-based communication for high performance computing. *Cluster Computing*, 5(3):315–324, 2002.