The Use of ASN.1 Encoding Rules for Binary XML

Ed Day (eday@obj-sys.com)
Objective Systems Inc., Exton, PA USA

Introduction

This paper discusses the use of Abstract Syntax Notation 1 (ASN.1) binary encoding rules for the efficient encoding and transfer of XML data. Its goal is not to provide exhaustive details on the intricacies of ASN.1 binary encodings, but rather to provide an overview of the different available encoding solutions and how they might apply to the requirements for a binary XML solution.

It is assumed that a schema is used to define the structure of the data. The schemas used throughout this paper are W3C XML Schema [XSD] to define the XML data and ASN.1 schema [ASN1] to define the format of the binary data encoding.

The following encoding rules standardized for use with ASN.1 will be discussed:

1. The Basic Encoding Rules (BER) [BER] and their derivatives, the Distinguished Encoding Rules (DER) and Canonical Rules (CER) as specified in ITU-T international standard X.690, and

2. The Packed Encoding Rules (PER) [PER] as defined ITU-T international standard X.691. This standard specifies multiple forms of PER including aligned and unaligned as well as basic and canonical. This paper will focus on the basic aligned form and makes the assumption that results achieved with the other forms would be similar to what is presented in this paper.

A standardization effort is currently underway within the ITU-T to define new standards that makes use of PER and other technologies that attempts to make further gains in reducing message sizes and processing times in this area. These new efforts are known as “Fast Web Services” [FWS] and “Fast InfoSet” [FIS]. They are mentioned briefly in this paper, but are still considered to be too preliminary at this time for detailed analysis. This paper instead will focus on the direct application of BER and PER technologies to XML data to determine message size and processing speed gains.

Requirements for Binary XML

The following are a set of requirements that would seem desirable for a binary XML protocol. This set of requirements was obtained mostly from the work of the *W3C Workshop on Binary Interchange of XML Information Item Sets* held in Santa Clara, CA in 2003. This set is by no means exhaustive. It represents what is believed to be the most recurrent themes of the items presented.

Reduced Message Sizes

A binary XML protocol should reduce message sizes by an order of magnitude over what is currently produced in XML form. The standard argument is that if this is not achievable, it is better to stay with the text-based alternative (XML), as it is easier to understand and work with.

Improved Performance

Reduced message sizes must lead to improved performance in terms of CPU processing time and message throughput. An argument against current compression technologies such as *gzip* [GZIP], is that they are very CPU intensive and gains in reducing messages size are offset by the amount of CPU processing cycles needed to achieve those gains.

Interoperability

An endpoint developed by a vendor using a binary XML solution must be able to interoperate with an endpoint developed by another vendor using different platforms, computer languages, compilers, etc. In short, the solution must be standardized. This is what makes XML attractive today and what made ASN.1 attractive to telecommunications and other companies many years ago.

Self-describing format

One of the reasons XML is successful is because everything you need to know about the message is built into the message itself. This combination of metadata and content is quite powerful, but it comes at the expense of large message sizes. A binary alternative will most likely not offer this same degree of self-description, but it should maintain some structure to allow someone to determine what the contents of the message are without the aid of specialized software (i.e. manual decoding).

Support streaming transfers

This refers to the ability to send data as a stream of bytes or characters without knowing up front the length of the data to be sent. It makes message processing easier when large amounts of data are to be sent because the process of looking ahead to count the number of items to be sent before actually sending the items can be expensive in terms of processing power. XML, with its encapsulation of items within start and end tags, provides this capability.

Random access to data (XPath)

It is sometimes desirable to select items from within a message or to search for a particular item within a message. XPath [XPATH] provides a way to do this with an XML document. A binary solution should provide a similar capability.

Standard XML API support: SAX and DOM

The Standard API for XML (SAX) is a de-facto programming standard [SAX] that defines a set of callbacks or events that are fired as an XML document is parsed. The Document Object Model (DOM) is a W3C standard [DOM] that defines a tree-structure that can be built from an XML document. Both of these items are used extensively in the XML world and many applications have been built around them. A binary XML solution should provide a means to work with these and other similar XML programming standards.

Validation: well-formed and schema compliant

A key component of XML message processing is the ability to quickly validate and reject messages that do not pass certain criteria checks. The simplest of these checks is well-formedness – is the overall structure of a document valid? Parsers are now able to validate a message against a schema (typically a DTD or XML schema) to verify that the data within the message is within the defined set of constraints. A binary XML solution should allow similar validation to be done.

Canonical Form

Security processing of messages requires that there be no variations in how a given message is constructed. A binary XML format must define a canonical form to allow security processing such as digital signatures to be applied to message parts.

Free tool support

Many free tools exist today for working with data in XML format. These work to make a technology gain widespread use. A binary XML solution should have free tool support – particularly in the form of viewer software that allows someone to examine the structure and contents of the binary message.

Schema Languages and Encoding Rules

Methodologies discussed in this paper for using ASN.1 Encoding Rules for Binary XML rely on the separation of schemas used to define message structures (sometimes referred to as "metadata") from the composition of the actual messages/documents themselves.

The following table shows schemas that are primarily related with XML documents and compares them with ASN.1. ASN.1 is a schema primarily designed for use with binary encoding rules:

Schema Language	***Encoding Rules***
XML Schema RelaxNG Schematron	XML
ASN.1	BER DER CER PER XER (which is XML!)

Table 1 – Schema Languages and Encoding Rules

The schema language describes the structure of messages; the encoding rules are used to describe instances of the messages themselves. Using an analogy from object-oriented programming: the schema is related to the concept of a class of an object (for example, a Java class definition) and the encoding rules would be used to create an instance of that class or an object.

XML Schema

XML Schema is a recommendation of the World-Wide Consortium (W3C) and would seem to be the predominant schema in use today to define XML message instances. It is used in many standards and is also used within Web Services Definition Language (WSDL) to define the structure of messages for web services communications. Although other schema languages are in use today for defining XML data, this paper will focus on XML Schema. It is assumed that the basic principles could be easily applied to other schema languages as well.

ASN.1

ASN.1 and its associated encoding rules are standards put forth by the ITU-T primarily for use within the telecommunications industry. These standards have been in place for nearly twenty years and have been constantly refined and updated over the years. The initial standards defined the ASN.1 schema language and the Basic Encoding Rules (BER). There has always been a clear separation between the schema language (ASN.1) and the encoding rules (BER, DER, PER, etc.).

The ASN.1 schema language has evolved to include such items as relational capabilities for defining multi-part messages, extensibility mechanisms, and parameterization. New encoding rules standards have been added over the years including PER and, the latest edition, XER, which provides an XML representation of ASN.1 data.

Transition Between XML Schema and ASN.1

XML Schema and ASN.1 have many things in common such as a common type system for defining message data and the concept of various structured containers such as sequence, choice, and repeating collections (similar to array types in computer languages). There are also differences as each schema language provides support for different things that the other lacks.

It is possible to translate between the two formats in either direction. In the case of XML Schema to ASN.1, a standard is in place (ITU-T X.694 [X694]) that defines a standardized way of doing this. This provides the migration path necessary for using ASN.1 encoding rules to produce standardized binary encodings of XML data.

Note that it is not necessary to support both an XML schema and ASN.1 version of a given specification in order to produce ASN.1 binary encodings. Tools are available that can do the automatic translation of XML schema to standardized ASN.1. This makes it possible to base everything off of an XML Schema document and not have to worry about what the ASN.1 version looks like. In fact, some tools can make the ASN.1 view of the data totally transparent when producing binary encoders and decoders that use the ASN.1 encoding rules.

ASN.1 Encoding Rules

It is assumed that anyone reading this paper is already familiar with the concepts of XML and what the structure of an XML document looks like. Readers may not be familiar with the structure of messages or documents created using the ASN.1 encoding rules. The following is a brief summary of the major encoding rule sets.

Basic Encoding Rules (BER)

The Basic Encoding Rules are defined in ITU-T standard X.690 [BER]. The basic structure of a BER-encoded message is based on “tag-length-value” or TLV for short. A block view of the simplest BER-encoded message would be as follows:

TAG	**LENGTH**	**VALUE**

All of these items are variable-length byte fields. Rules within the X.690 standard specify how each is constructed.

The TAG is one-or-more binary identifiers (usually one) that uniquely identify the VALUE. X.680 provides detailed rules on how tags are constructed and applied. These rules are beyond the scope of this paper, however, a few characteristics that are applicable to a binary-XML solution are as follows:

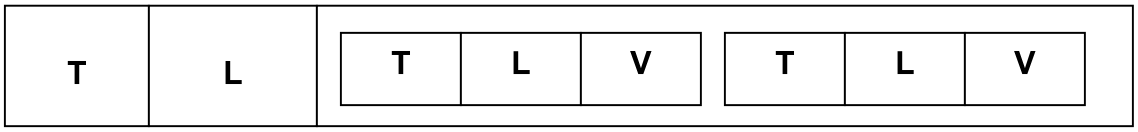
1. They are short. A tag length is typically one or two bytes. This would replace the markup needed in an XML document to identify a data item.

2. A universal set of identifiers is defined within the X.690 standard for all of the basic data types. This is important because it allows a message to be interpreted without the aid of a schema (more on this later).

The LENGTH is an integer value that identifies the length of the VALUE. This is optimized to fit in the smallest number of bytes. A length field may also be “indefinite length” (denoted by a 0x80 byte) that indicates an indefinite number of VALUE bytes follow. This stream of data is terminated by a special tag/length value combination (0x00 0x00) known as an end-of-context (EOC) marker.

The VALUE field is an encoding of the actual data content of the message. What makes this different from XML is that the value is a binary field rather than a textual field. Special rules are applied to integer and real values to produce encodings that are both compact and interoperable. Data that is essentially binary in nature (for example, digital signatures or images) is transmitted in native form as a stream of bytes. Conversion to and from textual format is not necessary.

If the simple form of the message described above was all there was to BER, then it would not be very useful. What makes it useful is the fact that the VALUE component can contain other TLV’s. This allows recursive structures to be built:



In this case, the ‘V’ in the outer encapsulating TLV is a set of other TLV’s. The basic contract of BER is that this recursive structure must be maintained throughout. This allows basic processing to be applied such as well-formedness checks and validation. As will be shown later, this is almost identical to the basic contract for XML encapsulation of data.

Distinguished Encoding Rules (DER)

The Distinguished Encoding Rules are also defined in ITU-T standard X.690 [BER]. Engineers and security architects may be familiar with DER since is used as the base encoding for Internet security standards such as X.509, PKIX, and PKCS (which at a higher level encompasses standards such as SSL and SMIME).

DER is almost identical to BER in encoding structure. The only difference is that DER does not allow any encoding options (i.e. it is canonical, there is only one correct form of an encoded DER message). In this respect, DER can be thought of as a subset of BER. A decoder that is capable of decoding BER messages would be able to decode DER as well. It is only on the encode side where special logic is needed to ensure that the message is correctly formed.

Canonical Encoding Rules (CER)

The Canonical Encoding Rules are also defined in ITU-T standard X.690 [BER]. Like DER, this is also a subset of BER that allows a single canonical form. CER is not as widely used as DER in applications in-use today. However, CER has a property that may make it more suitable for use in binary-XML applications. That property is the rule that all constructed component lengths use indefinite length encoding (DER requires definite lengths). This makes CER both canonical and streamable, two desirable properties on the list of binary-XML requirements.

Packed Encoding Rules (PER)

The Packed Encoding Rules are defined in ITU-T standard X.691 [PER]. This set of rules is a departure from the TLV format used in BER/DER/CER. Instead, PER makes extensive use of information defined in the ASN.1 schema to produce small, efficient encodings. Tags are eliminated and lengths are only used when they are absolutely necessary. Fixed length data includes no length identifiers. Variable length fields that are constrained to a small range use a length determinant that is packed into the minimum number of bits to define a delta-value from the known base. For example, if a string is constrained to be between 10 and 12 characters in length, only a 0, 1, or 2 value is encoded in two bits and sent over the wire. The receiver is expected to add 10 to this value to determine the actual length.

PER message content is optimized as well in some cases. A boolean value is encoded in a single bit. Integer value encoding makes uses of constraints on the values to use the minimal number of bits for a given number (this is much like the length example above). Permitted alphabet constraints can be used to define restricted character sets for character data in strings. PER will use these constraints to encode each character in the minimal number of bits required to represent the character. For example, an XML schema dateTime type is define in ASN.1 as follows:

```
DATE-TIME ::= VisibleString (FROM ("0".."9" | "TZ:.+-"))
```

PER will use the fact that there are only 15 possible characters that can be used in a date-time specification to produce an encoding that uses 4-bit index values to describe each possible character in the set.

This extreme efficiency comes with a price. For one thing, encoding is much more complicated then in the BER case. The concept of extensibility (i.e. producing an encoding that can be understood using different versions of a specification) requires special encoding to be done for extension items thus producing two distinct sets of rules. Also, a PER encoding is totally schema dependent. It is not possible to interpret the message in any way without knowledge of the schema used to construct it.

XML Encoding Rules (XER)

The XML encoding rules are defined in ITU-T standard X.693. They allow for the direct encoding of data specified using an ASN.1 schema using XML. Using XER with ASN.1 is the equivalent of using XML with XML schema. The schema language (ASN.1 or XML schema) defines the structure of the messages and XML is used as the format of the structured data.

Use of ASN.1 Encoding Rules for Binary XML

In this section, an analysis is done of the two main sets of ASN.1 binary encoding rules (BER and PER) in terms of how they meet the requirements presented earlier for binary XML Three common XML documents are used to show the gains in terms of message size and processing performance that are gained using the ASN.1 rules.

BER / DER / CER

This category of ASN.1 binary encoding rules is the tag-length-value rule set discussed in the earlier section on BER, DER, and CER. All of these rules are similar and will be lumped under the general term ‘BER’ for the remainder of this section.

When compared with XML, BER is found to have many similar properties. Both XML and BER messages have a hierarchal tree-like structure for representing the data. The following diagram shows a sample XML message and its BER counterpart side-by-side:

XML message (document)	BER-encoded message
<personnelRecord>	0x30 / 0x7B (personnelRecord tag/length)
<Name>	0xA0 / 0x10 (Name tag/length)
<givenName>John</givenName>	0x80 / 0x04 / “John”
<initial>P</initial>	0x81 / 0x01 / “P”
<familyName>Smith</familyName>	0x82 / 0x05 / “Smith”
</Name>	
<number>51</number>	0x81 / 0x01 / 51
<title>Director</title>	0x82 / 0x08 / “Director”
<dateOfHire>19710917</dateOfHire>	0x83 / 0x08 / “19710917”
<nameOfSpouse>	0xA4 / 0x10 (nameOfSpouse tag/length)
<givenName>Mary</givenName>	0x80 / 0x04 / “Mary”
<initial>T</initial>	0x81 / 0x01 / “T”
<familyName>Smith</familyName>	0x82 / 0x05 / “Smith”
</nameOfSpouse>	
	0xA5 / 0x3E (children tag / length)
<children>	0x30 / 0x1D (child record tag / length)
<name>	0xA0 / 0x11 (name tag / length)
<givenName>Ralph</givenName>	0x80 / 0x05 / “Ralph”
<initial>T</initial>	0x81 / 0x01 / “T”
<familyName>Smith</familyName>	0x82 / 0x05 / “Smith”
</name>	
<dateOfBirth>19571111</dateOfBirth>	0x81 / 0x08 / “19571111”
</children>	
<children>	0x30 / 0x1D (child record tag / length)
<name>	0xA0 / 0x11 (name tag / length)
<givenName>Susan</givenName>	0x80 / 0x05 / “Susan”
<initial>B</initial>	0x81 / 0x01 / “B”
<familyName>Jones</familyName>	0x82 / 0x05 / ”Jones”
</name>	
<dateOfBirth>19590717</dateOfBirth>	0x81 / 0x08 / “19590717”
</children>	
</personnelRecord>	

As can be seen, the XML and BER-encoded binary records line up rather nicely. As one would expect, the binary representation is terser and does not contain the clarity of data expressed in the textual view. But still, it has a structure and can be interpreted to some degree on its own without the aid of a schema.

Requirements Support

How does BER measure up in terms of the requirements that were presented earlier? Table 2 provides a summary as to which requirements each set of the encoding rules supports. BER would seem to cover the set best.

Requirement	**BER**	**PER**
Reduce message (document) size	x	x
Improved performance	x	x
Interoperability	x	x
Self-describing format	x	
Support streaming transfers	x	
Random access to data	x	
SAX / DOM support	x	
Validation of well-formedness	x	
Validation against schema	x	x
Canonical form	x	x
Free tool support	x	

Table 2 – BER / PER Support of Requirements

In terms of reduced message size (the first requirement), BER provides substantial message size reductions. This is mainly because of the following two size-reduction benefits over textual XML representations:

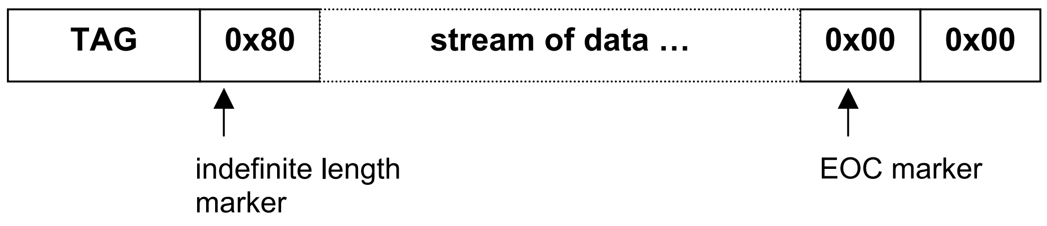
1. Markup is reduced considerably by replacing start/end tags with binary tag/length values. This is significant, especially in documents that contain more markup then content, and

2. Data that is inherently binary (integers, floating point numbers, raw binary data, etc.) is transmitted either as-is or in a platform independent manner. This is both smaller and faster then text-based approaches because the size is more compact and no transformations need to be done.

Probably the most important feature of BER is its self-describing nature. Most other binary implementations including gzip compression convert the textual data into an incomprehensible binary form that cannot be understood unless converted back to the original text. This makes it impossible to do any intermediate processing on a message without uncompressing and recompressing. It also make troubleshooting very difficult when things go wrong and an analyst is left with only parts of a message to try and determine the root cause of a problem.

The self-describing nature also makes compliance with many of the other requirements possible. Random access to elements within the structure is possible through the use of combinations of binary tags that can be used as a key to find specific items. It is in theory possible to create an expression that is similar to an XPath expression but which uses binary tags instead of element names. The requirements for SAX and DOM-like processing as well as validation of well-formedness are met in similar ways due the fact that BER is inherently a tree-like structure.

BER messages are streamable as well. A special feature known as “indefinite length encoding” makes this possible. This encoding replaces the standard length field (which is an encoded number) with a special start marker represented as a 0x80 byte. A decoder would then accept data until a special end-of-context (EOC) marker was reached. This is represented as a zero byte for both tag and length. A diagram showing this is as follows:



BER, as was shown earlier, has canonical forms, which is another requirement. The DER and CER forms both provide no encoding options. There is one, and only one, way to encode a message using these rules. As also mentioned earlier, the CER form uses indefinite lengths as described above making it both canonical and streamable.

Finally, many freely available software tools support BER. This is especially true of viewers/browsers. These tools are essential for working with data in this form. One of the reasons XML is popular is because it can be easily viewed using any text editor. There are always going to be times when someone is going to have the need to see what is in a message. The notion that these binary messages are only going to be exchanged at the lowest levels of highly optimized systems and that nobody outside of the people at the endpoints are going to have any need to see or examine them seems overly optimistic. Tools are required to allow people to view and/or edit these messages. If the technology is to be ubiquitous across the network, then free tools are necessary to allow all to participate.

Some examples of free tools that can be used to view BER-encoded data are as follows:

- *dumpasn1* by Peter Gutmann [DUMPASN1]
- *GUIdumpASN* by Gemini Security Solutions [GUIDUMP]
- *ASN.1 Editor* by Liping Dai [ASN1EDIT]
- *ASN.1 Viewer* by Objective Systems [ASN1V]

An example of the output generated by one of these tools can be seen in the following diagram:

```
0  123: SEQUENCE {
   2   16:   [0] {
   4    4:     [0] 'John'
  10    1:     [1] 50
  13    5:     [2] 'Smith'
        :     }
  20    1:   [1] 33
  23    8:   [2] 'Director'
  33    8:   [3] '19710917'
  43   16:   [4] {
  45    4:     [0] 'Mary'
  51    1:     [1] 54
  54    5:     [2] 'Smith'
        :     }
  61   62:   [5] {
  63   29:     SEQUENCE {
  65   17:       [0] {
  67    5:         [0] 'Ralph'
  74    1:         [1] 54
  77    5:         [2] 'Smith'
        :         }
  84    8:       [1] '19571111'
        :       }
  94   29:     SEQUENCE {
  96   17:       [0] {
  98    5:         [0] 'Susan'
 105    1:         [1] 42
 108    5:         [2] 'Jones'
        :         }
 115    8:       [1] '19590717'
        :       }
        :     }
        :   }
```

Figure 1 – Sample Output from *dumpasn1*

The numbers on the left are byte offset and length (in bytes) of each item respectively. The numbers in square brackets represent ASN.1 BER tag values. The keyword 'SEQUENCE' is inserted in places where the known UNIVERSAL tag for the ASN.1 SEQUENCE container type exists. As can be seen, this is very similar to XML in structure.

Possible Extensions to BER

As can be seen, BER does a good job of meeting many of the requirements for a binary XML solution to some degree. It is important to note that everything discussed above has already been standardized. This is not a new idea that will take years to develop and work all of the bugs out of. Systems have been deployed that have used BER for interoperability for the past twenty years. In fact, many of the base security technologies on the Internet at this time such as X.509 certificates and SMIME make use of this technology.

It would be possible to extend the standard to fill in the gaps where the requirements are not fully met. For example, BER tags are cryptic binary tags that carry little information as to what is actually in the fields they represent. At a simple level, this can be improved by simply stipulating that universal tags already defined within the X.690 standard be used in addition to context-level tags to identify all data items. This provides instant information on the type of binary data defined within each content field. For example, if an item were tagged with a UNIVERSAL 2 tag, a decoder would know that it contained an integer value without having to resort to a schema lookup to obtain this information. These universal tags are something that does not need to be invented – they exist in the standard for every primitive and constructed (wrapper) type of data element.

However, the way the X.694 standard is set up now causes this information to be discarded for the most part for the sake of efficiency. It would not be difficult to add an encoder's option to specify this additional information be included in a message.

It would also be possible to specify a special metadata record format that would equate binary tag information with field names and other properties. A possible definition of this type of record in ASN.1 would be as follows:

```
XMLMetaData ::= SEQUENCE OF XMLBinTag

XMLBinTag ::= SEQUENCE {
     berTag          SEQUENCE OF INTEGER,
     xmlQName        UTF8String,
     isAttribute     BOOLEAN,
     etc..
}
```

This could be the first record transmitted in a potentially long sequence of records, for example, in a database table dump. This would allow a reader to easily reconstruct the message in XML format without having any advanced knowledge of the schema. It would also allow all records that follow to be transmitted in an efficient BER-encoded binary format.

PER

The Packed Encoding Rules (PER) standard differs from BER in that it attempts to achieve improved compactness by making maximum use of information in the ASN.1 schema in order to use the smallest number of bits possible to represent a given data value. General characteristics were presented earlier in the overview section, but to summarize:

- PER does not encode tag values

- PER only encodes lengths when necessary, fixed length variables and variables that are constrained to be a fixed length (for example, X ::= OCTET STRING

(SIZE(10))) include no encoded length information.

- PER makes use of constraints on items (for example, I ::= INTEGER (5..10)) to produce further optimized encodings.

There are other properties as well, but these are believed to be the most significant.

Requirements Support

PER in most cases achieves smaller message sizes then BER. However, the cost of achieving these reductions is lost support for most of the identified requirements. Table 2 provides a summary as to which requirements each set of the encoding rules supports. As can be seen, PER does not support most of the listed requirements.

In terms of reduced message size (the first requirement), PER in most cases produces smaller messages then BER. The amount of reduction is very data dependent. Highly constrained numeric data will achieve significant size reductions over BER. However, UTF-8 character string data and raw binary octet (byte) data produces no savings. Character strings with restricted character sets such as those used in a date/time stamp can be reduced by PER, but in general, these savings are rare. Most character string content is encoded in native form with no compression.

Metadata is reduced even more in PER because tags are removed and many length values are reduced, if not eliminated. The cost of this is total dependence on the schema to determine where everything is. This dependence eliminates perhaps the most compelling trait of BER – its self-description capabilities.

Without self-description, compliance with many of the other requirements is not possible. Random access to elements within the structure, SAX and DOM-like processing and validation cannot be done using the data structure itself. A sophisticated run-time processor is needed that can link the schema elements to bit fields with the message.

In general, PER messages cannot be sent across a stream-oriented interface the way BER messages can. The length must be known up front. PER does contain an analog to BER indefinite length encoding for describing large data blocks, so in theory, it is possible to stream parts of a PER message. But this would most likely involve breaking the message into chunks where some chunks must be sent using the known length and other chunks could be streamed.

PER has a canonical form, so this requirement is met.

Freely available software tools for PER are hard to find. There is the Open H.323 project (http://www.openh323.org) that contains a free PER compiler, but this is geared towards H.323 applications and does not support most of the broader features of the syntax. Some tools have PER viewing capabilities built-in for specific protocols (H.323 being the prime example), but there is not much in terms of generic support. We have not been able to locate a quality, free viewer for PER-encoded messages that accepts a schema and provides a detailed dump of the bit fields within the message.

Beyond PER

Standardization efforts are now underway at the ITU-T to produce new standards to address the problem of the verbosity of XML when used in Web Services. The *Fast Web Services* [FWS] and *Fast Infoset* [FIS] standards address these issues. This paper does not provide an analysis of these new technologies as they are too new and little is known about them outside of the core working groups. From what can be discerned from the public postings, it would appear that these are primarily compression technologies that suffer from many of the same deficiencies described for PER above. In fact, PER is the encoding rules set used within the new standards. The bottom line is that these new standards can achieve impressive gains in reducing message sizes to even smaller levels, but they suffer from many of the disadvantages of PER such as lack of self-description, streamability, random access, and free tools support.

Benchmark comparisons

Some simple benchmark comparisons were done between messages (documents) encoded in XML form and in an equivalent binary form using the ASN.1 BER and PER encoding rules. Messages sizes were compared as well as elapsed times to serialize and deserialize 10,000 records of each message type.

Three well-known message types were chosen: a simple Employee data record as used within the ASN.1 standards for examples of the encoding rules, a Universal Business Library (UBL) Invoice XML document, and an XML digital signature (XMLDSIG) document.

Employee Data Record

The *Employee* example was chosen because it is a common data record type found in many of the ASN.1 standards. The sample instance of this record in XML format that was used for the benchmark tests is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<personnelRecord xmlns="http://www.obj-sys.com/Employee">
   <name>
      <givenName>John</givenName>
      <initial>P</initial>
      <familyName>Smith</familyName>
   </name>
   <number>51</number>
   <title>Director</title>
   <dateOfHire>19710917</dateOfHire>
   <nameOfSpouse>
      <givenName>Mary</givenName>
      <initial>T</initial>
      <familyName>Smith</familyName>
   </nameOfSpouse>
   <children>
      <name>
         <givenName>Ralph</givenName>
         <initial>T</initial>
         <familyName>Smith</familyName>
      </name>
      <dateOfBirth>19571111</dateOfBirth>
   </children>
   <children>
      <name>
         <givenName>Susan</givenName>
         <initial>B</initial>
         <familyName>Jones</familyName>
      </name>
      <dateOfBirth>19590717</dateOfBirth>
   </children>
</personnelRecord>
```

Figure 2 – Employee Sample XML Instance

A comparison of the message sizes in XML, BER, and PER formats is as follows:

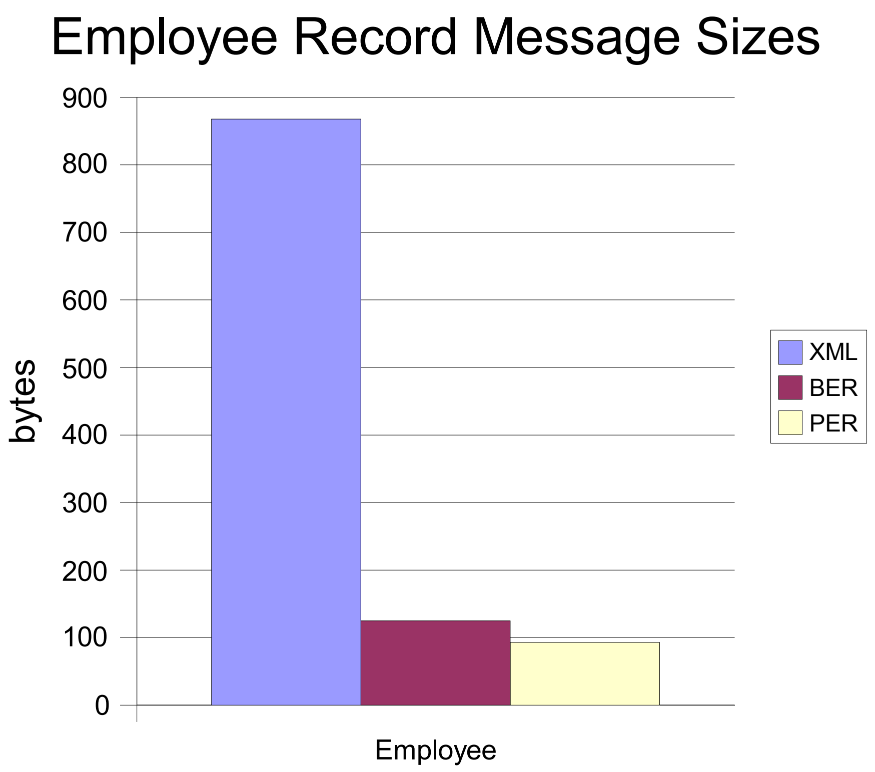


Figure 3 – Employee Record Message Sizes

As can be seen, the XML document size (868 bytes) is almost seven times larger then the comparable BER-encoded record (125 bytes) and more then eight times larger then the PER record (93 bytes).

In looking at timings, the time to serialize (encode) and deserialize (decode) a set of 10,000 records is shown below. Times are also shown for SAX and DOM – 2 popular XML API's. SAX refers to the time required to execute a set of empty content handlers (*startElement*, *endElement*, and *characters* callback functions) and can be considered to be the absolute best time that can be attained using this API. DOM refers to the time required to deserialize the document to a DOM tree structure. All measurements were done using test programs written in C and compiled with GNU gcc 3.3.1 on a Dell Dimension 8100 running SuSE Linux 9.1. The Dell processor speed is 1.6 GHz and it has 256 MB of memory. Objective Systems ASN.1 and XML tools were used to do the serialization/deserialization tests and LibXML2 [LIBXML2] was used as the XML parser for the SAX and DOM tests (and it is also the underlying parser for deserialization).

The timing results are as follows:

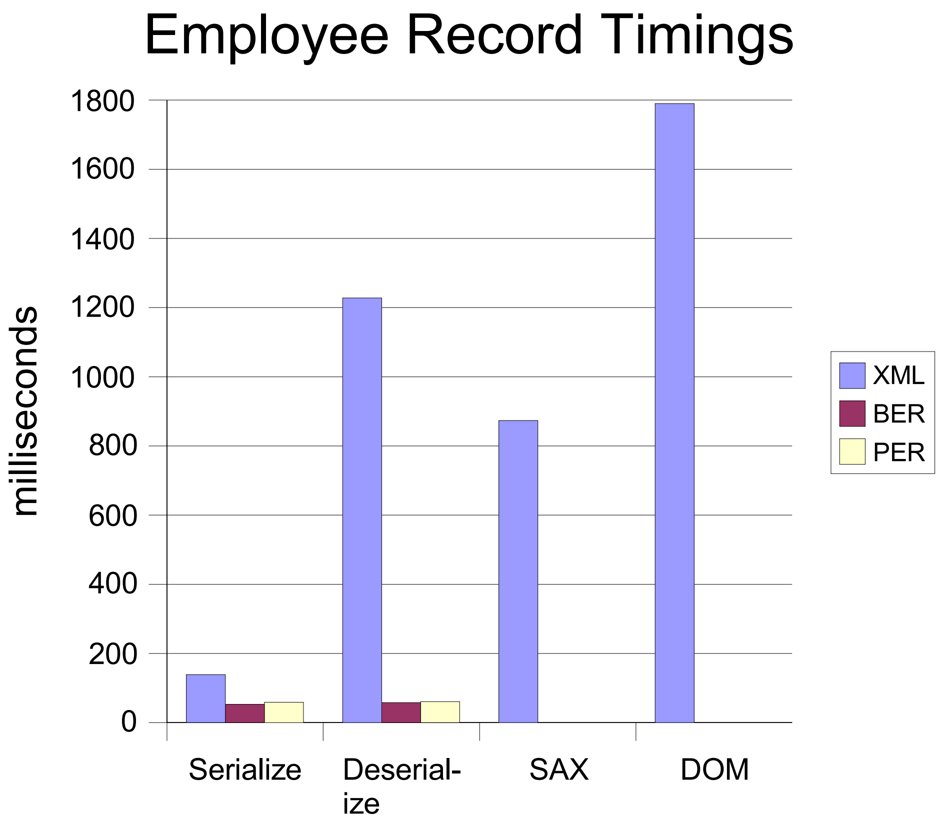


Figure 4 – Employee Record Timings

This shows that serialization times are comparable, although BER/PER does hold an advantage. Deserialization times are more striking. The time to deserialize using BER or PER is only a small fraction of the time required for XML. Even for SAX – considered the fastest XML API – with empty handlers, the relative time required was 10x more.

It is also significant that the LibXML2 XML parser was used for these tests. An independent study [XMLBENCH], found it to be one of the best performing XML parsers (open source or commercial) available at this time.

For the record, the times to (de)serialize between BER and PER were almost identical. BER was found to be slightly faster, but only on the order of a few milliseconds.

UBL Invoice

Similar tests were done on a UBL invoice XML document. The characteristics of this record that made it interesting are:

a. it is much larger (~8k bytes) as opposed to the employee record, and
b. it contains a large amount of heavily nested markup text (XML tags and attributes)

A chart showing the message sizes of text and binary forms is as follows:

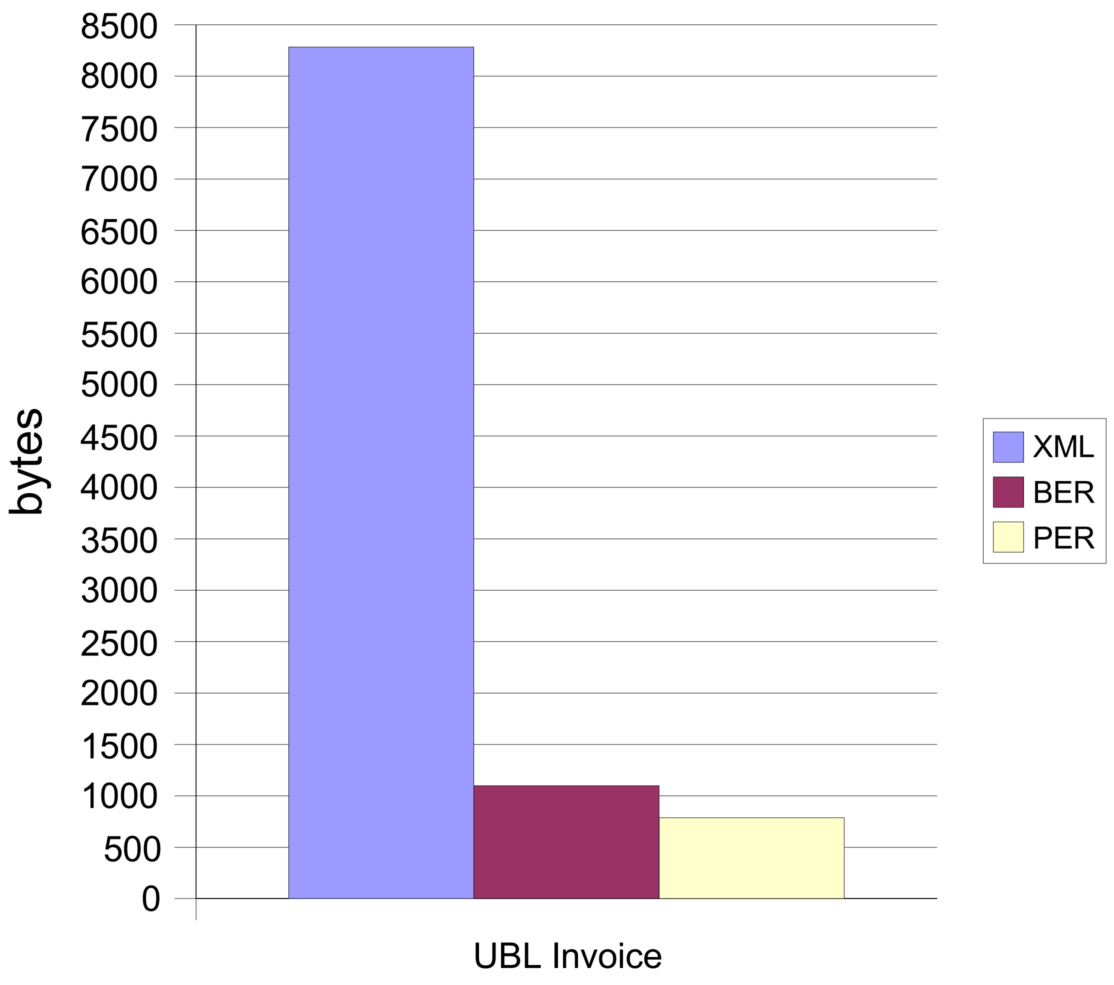


Figure 5 – UBL Invoice Message Sizes

Once again, there is an approximately 8x difference between XML and the binary encodings. PER is again slightly smaller then BER.

The timing chart is as follows:

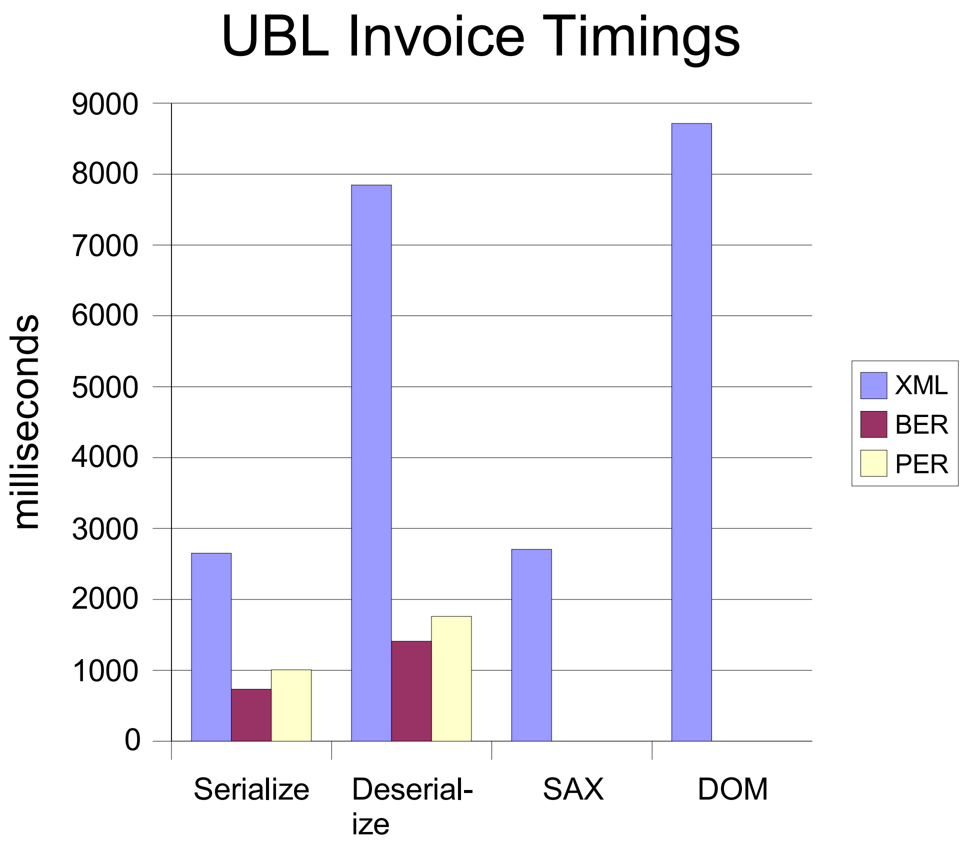


Figure 6 – UBL Invoice Timings

Results are similar to the Employee case except that SAX processing is closer in time binary deserialization than in the other case.

XML Digital Signature

The final document type tested was an XML digital signature document. This was found to be interesting because the record exists of inherently binary data that is turned into base64 for use in an XML document. The document that was tested is as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315" />
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
    <Reference URI="http://www.w3.org/TR/xml-stylesheet">
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>
    LaL1/t/XodYvDJDgSEbq47GX8ltnlx3FFURdi7o+UFVi+zLf0WyWaQ==
  </SignatureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>
          3eOeAvqnEyFpW+uTSgrdj7YLjaTkpyHecKFIoLu8QZNkGTQI1ciITBH0lqfIkdCH
          Si8fiUC3DTq3J9FsJef4YVtDF7JpUvHTOQqtq7Zgx6KC8Wxkz6rQCxOr7F0ApOYi
          89zLRoe4MkDGe6ux0+WtyOTQoVIGNTDDUFXrUQNbLrE=
        </P>
        <Q>
          hDLcFK0GO/Hz1arxOOvsgM/VLyU=
        </Q>
        <G>
          nnx7hbdWozGbtnFgnbFnopfRl7XRacpkPJRGf5P2IUgVspEUSUoN6i1fDBfBg43z
          Kt7dlEaQL7b5+JTZt3MhZNPosxsgxVuT7Ts/g5k7EnpdYv0a5hw5Bw29fjbGHfgM
          8d2rhd2Ui0xHbk0D451nhLxVWulviOSPhzKKvXrbySA=
        </G>
        <Y>
          cfYpihpAQeepbNFS4MAbQRhdXpDi5wLrwxE5hIvoYqo1L8BQVu8fY1TFAPtoae1i
          Bg/GIJyP3iLfyuBJaDvJJLP30wBH9i/s5J3656PevpOVdTfi777Fi9Gj6y/ib2Vv
          +OZfJkkp4L50+p5TUhPmQLJtREsgtl+tnIOyJT++G9U=
        </Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
```

Figure 7 – XML Digital Signature Document

In this document all of the element content is base64-encoded data. It also contains some textual attribute variables to specify various URI's. Overall, the amount of markup is quite a bit less then the other records.

The record size and timing results using the same methodologies as above are as follows:

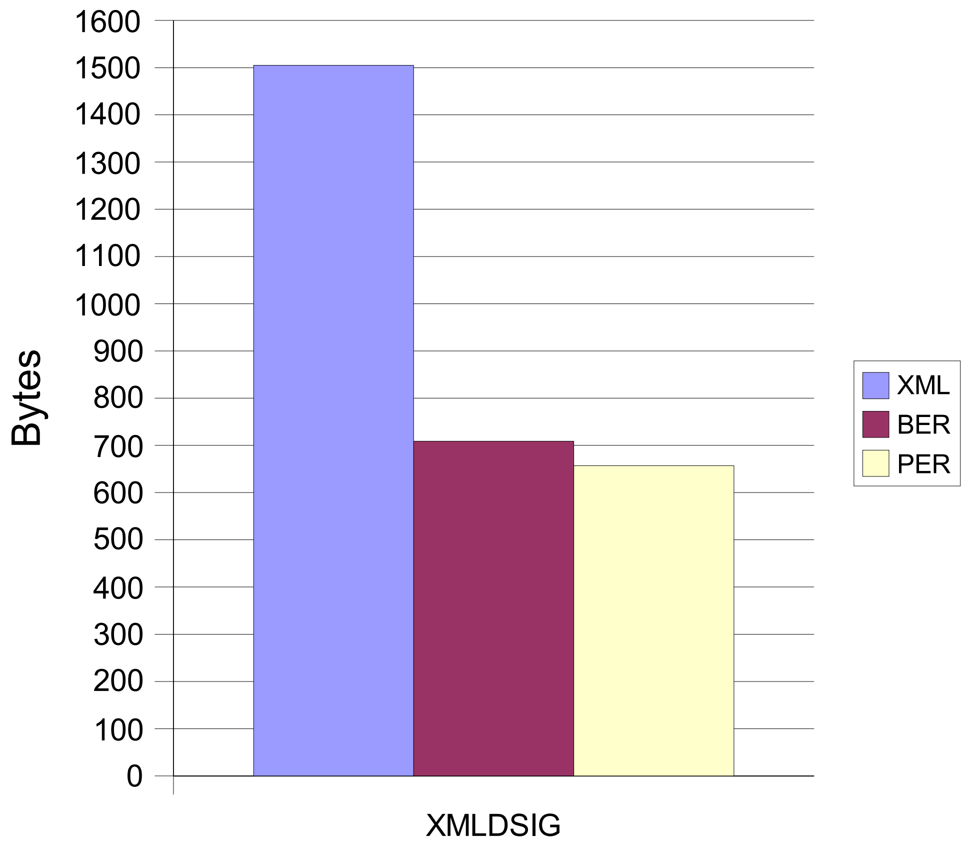


Figure 8 – XML Digital Signature Message Sizes

In this case, the message size differences are not as dramatic. The size of a binary message is roughly half that of its XML counterpart. This can be attributed to the fact that the XMLDSIG XML document contained relatively little markup. Most of the savings were because of direct use of binary data for the base64 encoded fields.

The timing results were as follows:

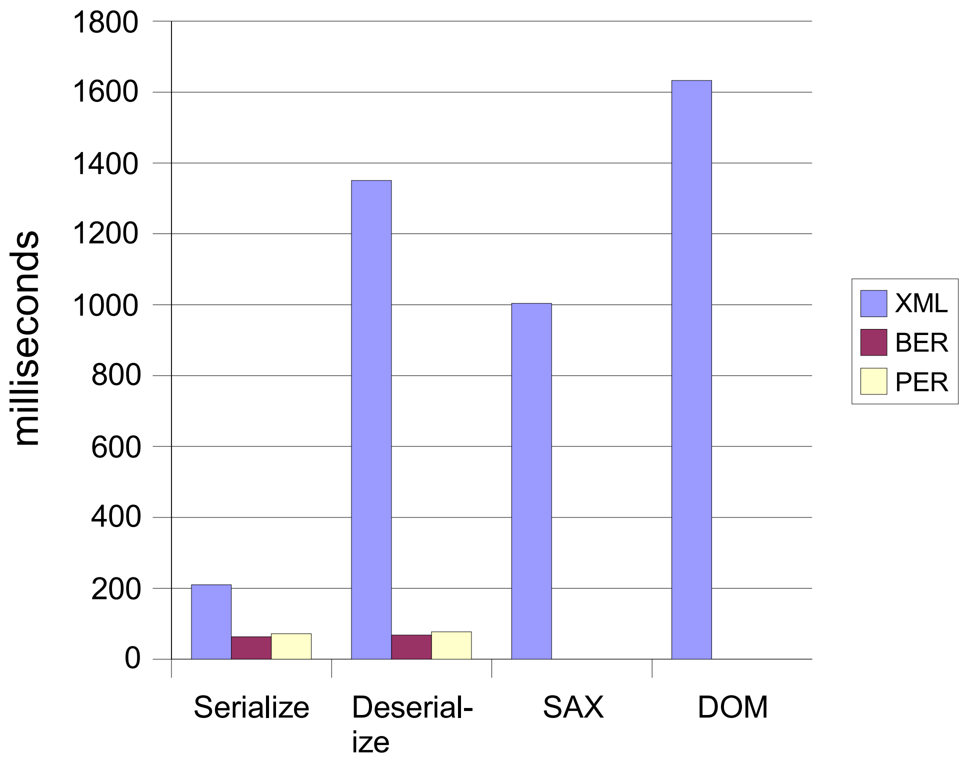


Figure 9 – XML Digital Signature Timings

As in the other cases, deserialization gains are most impressive, in this case achieving almost a 20x reduction. This is due to the combination of not having to deal with the XML markup as well as not having to do the binary data transformation to and from Base64 encoding.

Conclusion

XML has become a widely adopted data format standard mainly because it is simple and easy to understand and because of its self-descriptive qualities. A standardized binary format for XML should strive to achieve these same qualities while at the same time reducing the verbosity of XML. This paper has provided arguments that the Basic Encoding Rules of ASN.1 do just that. They provide a structured message format that is similar in basic qualities to XML but that provides compactness in two basic ways:

1. Reducing metadata by replacing bulky textual start/end tags with binary tag/length pairs, and

2. Allowing inherently binary content to be transmitted in binary form rather then as text

It was shown that these two items provide reasonable compactness and increased processing performance while at the same time preserving many of the basic structural features that make XML attractive. Free software tools that allow easy viewing of the contents of BER-encoded documents were also introduced.

Lessons learned from past standardization efforts such as CORBA have shown that the idea that what is on the wire does not matter and can be hidden using sophisticated API's does not work. HTML through its "view source" principal has laid the groundwork for how things are done today. XML is building on this success. An attempt to standardize a binary version of XML should not ignore these lessons. Technologies that turn messages into incomprehensible masses of bits through compression or other means are not likely to achieve widespread use unless ubiquitous free tool support can be attained (for example, gzip). Otherwise, the vast majority of potential users will turn away and tolerate slower, text-based solutions instead of a more efficient – but more complicated – binary solution.

References

[ASN1] ITU-T Recommendation X.680 (07/2002) –
Abstract Syntax Notation One (ASN.1): Specification of basic notation

[ASN1EDIT] ASN.1 Editor by Liping Di
http://www.codeproject.com/csharp/Asn1Editor.asp

[ASN1V] ASN.1 Viewer
http://www.obj-sys.com/asn1viewer.shtml

[BER] ITU-T Recommendation X.690 (07/2002) –
ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Rules (DER)

[DOM] World Wide Web Consortium – Document Object Model (DOM)
http://www.w3.org/DOM/

[DUMPASN1] ASN.1 Object Dump Utility by Peter Gutmann
http://www.cs.auckland.ac.nz/~pgut001

[FIS] Fast Infoset
http://java.sun.com/developer/technicalArticles/xml/fastinfoset/

[FWS] Fast Web Services
http://java.sun.com/developer/technicalArticles/WebServices/fastWS/

[GUIDUMP] GUI version of ASN.1 Object Dump Utility
http://www.geminisecurity.com/guidumpasn.html

[GZIP] GNU zip compression utility
http://www.gzip.org/

[LIBXML2] The XML C Parser and Toolkit of Gnome
http://xmlsoft.org

[PER] ITU-T Recommendation X.691 (07/2002) –
ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)

[SAX] Simple API for XML
http://www.saxproject.org/

[X694] ITU-T Recommendation X.694 (01/2004) –
ASN.1 encoding rules: mapping W3C XML schema definitions into ASN.1

[XPATH] World Wide Web Consortium – XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[XSD] World Wide Web Consortium – XML Schema 1.0

http://www.w3.org/XML/Schema#dev

[XMLBENCH] XML Benchmark Results 08.02.2004
http://xmlbench.sourceforge.net/results/benchmark200402/index.html