The emergence of ASN.1 as an XML schema notation

John **Larmouth** <j.larmouth@salford.ac.uk>>

Abstract

This paper describes the emergence of ASN.1 as an XML schema notation. Use of ASN.1 as an XML schema notation provides the same functionality as use of W3C XML Schema (XSD), but makes compact binary representations of the data available as well as XML encodings.

ASN.1 also provides a clear separation of the specification of the information content of a document or message from the actual syntax used in its encoding or representation. (Examples of representation differences that do not affect the meaning - semantics - being communicated are the use of an attribute instead of an element in an XML encoding, or of space-separated lists instead of repeated elements.)

Examples are given of ASN.1 specification of an XML document, and some comparisons are made with XSD and RELAX NG

Table of Contents

1. Brief historical background

ASN.1 came out of the "how to define protocols for computer to computer interaction", and more than that, came out of the "binary-encodings are best" camp.

Note

Today, ASN.1 thoroughly embraces the use of XML - character-based - encodings, along-side binary encodings. The above remark is simply history!

The ASN.1 notation was first developed in the early 1980s to support the OSI X.400 (e-mail) protocols, and was then widely used in many other OSI protocols - not a parentage that would commend it to workers today! (See http://www.btinternet.com/~j.larmouth/tutorials/hist/lineage.ppt)

But ASN.1 broke away from that OSI background, and is today very heavily used in many telecommunications applications, and in such diverse fields as control of nuclear plants, tracking parcels, air traffic control, intelligent transportation systems, biometrics (and other smart card applications), and in multimedia protocols such as those used in Microsoft NetMeeting.

A "history of protocol definition" is provided at http://www.btinternet.com/~j.larmouth/tutorials/hist/shortHist.ppt. This goes back to the earliest developments, 1.5 billion years (whoops - seconds!) ago, and describes the civil war between the Montagues (believers in binary encodings) and the Capulets (believers in character-based encodings). It ends with the marriage of ASN.1 and XML - the marriage of Romeo and Juliet!

This is very different from the XML (and XSD) background in SGML (the publishing industry) and the later HTML (Web documents).

There are no rights and wrongs here - just different backgrounds. But today notations for protocol specification (or schema definition) are converging, to the advantage of all.

ASN.1 brings a particular perspective to the definition of the form of XML documents (an XML schema). It differs significantly from the perspective of XSD or of RELAX NG. (It also provides a notation for the definition of an XML schema that is noticeably different from these alternative XML schema notations, and, arguably, more readable and less verbose.)

ASN.1 tools also generally provide a different approach to the handling of an XML document from that provided by XSD and RELAX NG tools.

The focus of ASN.1 is very much on the **information content** of a message or document. A distinction is drawn between whether changes in the actual representation of a message or document affect its meaning (and hence its effect on a receiving system), or are just variations of encoding that carry the same information. Thus the use of an XML attribute rather than a child element does not affect the information content. Nor does the use of a space-separated list rather than a repetition of an element.

ASN.1 tools provide a **static** mapping of an XML schema definition to structures in commonly-used programming languages such as C, C++ and Java, with highly efficient encode/decode routines to convert between values of these structures and the informnation content of XML documents. By contrast, most tools based on XSD or RELAX NG are more interpretive in nature, providing details of the infoset defined by the XML document through enquiries by the application or by notifications to the application (a highly interactive - and CPU intensive procedure).

No value judgment is here intended. This is a difference of perspective and approach, and both approaches are of value for different applications.

ASN.1 started, essentially, as a notation for formally describing a TLV (Type, Length, Value) style of binary encoding at a high-level of abstraction - roughly, at the level of abstraction provided by data-structure definition in programming languages such as C, C++ and Java.

However, ASN.1 - **Abstract** Syntax Notation One - tried to provide a clear separation of the information content of messages or documents from the encoding or representation of those documents.

It is, perhaps, surprising, that twenty years after the first standardisation of ASN.1, this remains its major strength as a schema notation.

The extension of ASN.1 to provide XML encodings (use of it as an XML schema notation) retains this fundamental separation.

ASN.1 has all the power of XSD - and a standardised mapping from XSD to ASN.1 is available, with tools supporting the mapping.

However, that does not compromise the clear separation of the definition of the information content of an XML document (which can be mapped into C, C++, and Java datastructures) from aspects of encoding such as the use of attributes v elements, or of space-separated lists.

Finally, the background of ASN.1 in binary encodings (and the clear separation of abstract definition of information content from encoding representation) means that the use of ASN.1 as a schema notation automatically provides **both** an XML representation of data **and** an efficient binary representation of the same information. (The ASN.1 binary representation is much more efficient than current binary XSD proposals.)

ASN.1 tools can provide relays between incoming messages in compact binary and outgoing messages in strict XML format, and vice-versa, provided the basic definition is done using ASN.1 as the schema notation. The ASN.1 schema can be generated as a mapping from XSD, using the standardised mapping, or can be written directly using a text editor or a with ASN.1 GUI tool support).

2. Relevant documents

The bibliography provides links to the ASN.1 specifications[ASN.1]. These are common text between ITU-T and ISO, **and are available free from ITU-T**. The days when ITU-T Recommendations and ISO Standards cost an-arm-and-a-leg to acquire are long-since gone.

There is also an introduction to ASN.1 [INTRO]for those unfamiliar with it, and a description of some of the uses of ASN.1[USES]. Two books [LARMOUTH] [DUBUISSON] are available free on the Web, and are also available in hard-copy. There is also the original French version[DUB-FR] of[DUBUISSON] available in hard-copy.

Links are also available to the tool that maps XSD to ASN.1[FROM-XSD] and to a variety of other ASN.1 tools and resources[LINKS].

3. The concept and value of an abstract syntax notation

As was indicated above, ASN.1 provides a clear separation of the definition of information content from encoding issues.

It does not *a priori* claim to be defining the syntactic form of an XML document. The definition of the form of the XML document is determined by the application of defined ASN.1 XML Encoding Rules (XER), possibly modified by ASN.1 XER encoding instructions (see later).

Simply defining what is a "valid XML document" - that is, the syntax of the document, does not in general unambiguously identify the information content of that XML document. Where there are valid XML documents that are syntactically different, do they or do they not carry different application semantics? Do we want them to carry different semantics? This syntactic approach raises many unanswered questions, and is not the ASN.1 approach.

The ASN.1 approach says simply "Define first and primarily the information content, and then define the allowed syntaxes (encodings) for that content."

Providing a **canonical** encoding - an XML document specification where there is a one-to-one correspondence between the syntactic form of the XML document and its information content - becomes easily possible with this approach.

Canonical representations of information are essential for most applications where there are security concerns, and can also help to reduce implementation costs, particularly in relation to complete testing suites.

4. The need for XML Encoding Rules for ASN.1

Workers on ASN.1 Standards recognised some four or five years ago that there was a requirement (from users of ASN.1) to have an XML representation of the information structures defined by an ASN.1 specification.

The reasons for this need a little further explanation.

Of course there was "hype" - XML gained (and still gains) a lot of press attention. It is widely (and in many cases probably rightly) regarded as "the right format" for the transfer of information between computers today. New

protocols in Biometrics, E-business and E-Health just **have** to be encodable in an XML format. (But transmission over a readio link, or storage on a smart-card still generally require a compact binary format.)

But behind the "hype", there are real advantages (and disadvantages - did I say that - sorry NO, strike that remark!) in an XML format. XML formats provide a very easy means to identify and pick out parts of an information content. Delimitation is simple, and selecting an element can be done by the most elementary PERL script or text search tool. So partial processing of a document by different systems becomes easily possible. Transformation of such representations using, for example, XSLT, is easy and straight-forward, providing a whole raft of tools for manipulating documents in XML formats. Extracting information from an XML document for input to a database (or insertion from a database) is also in principle easy. (But see the later discussion on the handling of "mixed" by ASN.1.) Data mining is eased: It does not matter what document an XML element is embedded in, it can be identified and extracted easily. American-English is now a world-wide lingua-franca, so XML tags that are expressed in Anglish can carry important semantics to the four corners of the world! (Whoops - I am getting a bit OTT = Over The Top, now! And my French and Japanese colleagues are highly likely to disagree!) These are all very real gains over compact encodings.

Coming from the negative side: People will assert that the verbosity of XML encodings is no longer an issue in this age of high-bandwidth communications lines. Compression strategies based on an initial XML encoding are developing rapidly. How true is this? It is probably still an open issue. People concerned with radio links for mobile phones might feel that bandwidth considerations are still **very** important, and people working with smart-cards have similar concerns.)

An aspect of a verbose transfer syntax (representation, encoding) that is often neglected is the very negative impact it can have on transaction processing rates. Verbosity does not just affect communications bandwidth, it also affects the CPU-cycles needed to generate and to process messages.

5. The development of the ASN.1 XML Encoding Rules - BASIC-XER

Once the decision was taken that ASN.1 users needed to have an XML encoding for ASN.1-defined data, it was relatively easy to produce a set of Encoding Rules that would generate an XML document for the information content of an ASN.1 type definition. Here is an example of an ASN.1 type definition:

```
PersonnelRecord ::= SET {
                    name            Name,
                    title           UTF8String,
                    number          EmployeeNumber,
                    dateOfHire      Date,
                    nameOfSpouse    Name,
                    children        SEQUENCE OF
                                    ChildInformation DEFAULT {} }

ChildInformation ::= SET {
                    name            Name,
                    dateOfBirth     Date}

Name ::= SEQUENCE {
                    givenName       UTF8String,
                    initial UTF8String,
                    familyName      UTF8String}

EmployeeNumber ::= INTEGER

Date ::= UTF8String  -- YYYYMMDD
```

and here is an example of an XML encoding of a possible value of that abstract type:

```
<PersonnelRecord>
     <name>
        <givenName>John</givenName>
        <initial>P</initial>
        <familyName>Smith</familyName>
     </name>
     <title>Director</title>
        <number>51</number>
        <dateOfHire>19710917</dateOfHire>
        <nameOfSpouse>
           <givenName>Mary</givenName>
           <initial>T</initial>
           <familyName>Smith</familyName>
        </nameOfSpouse>
     <children>
        <ChildInformation>
           <name>
              <givenName>Ralph</givenName>
              <initial>T</initial>
              <familyName>Smith</familyName>
           </name>
           <dateOfBirth>19571111</dateOfBirth>
        </ChildInformation>
        <ChildInformation>
           <name>
              <givenName>Susan</givenName>
              <initial>B</initial>
              <familyName>Jones</familyName>
           </name>
           <dateOfBirth>19590717</dateOfBirth>
        </ChildInformation>
     </children>
</PersonnelRecord>
```

The first request for an XML encoding for ASN.1-defined data (for an ASN.1 schema) camr from a group that wanted an XML transfer format for messages that had already been defined (many years before) using ASN.1. The ASN.1 specification of the information content was in place. What was needed was Encoding Rules to determine the form of an XML document that would convey that information content.

This gave rise to the "BASIC-XER Encoding Rules". These provided a very safe and robust means of specifying an XML document. The specification of BASIC-XER used (and uses) only ASN.1 notation for type (information content) definition. Thus with BASIC-XER, the XML schema notation is "pure" ASN.1. But this provides no control over the XML representation that is generated (attributes and space-separated lists are never used).

But BASIC-XER provided the ability for an application written using C, C++ or Java to easily and efficiently produce both compact binary encodings and XML documents, carrying the same information.

The meaning of "very safe" needs to be clarified. The BASIC-XER Encoding Rules are probably the safest means of defining an XML document that exists today! (Now I really **am** out on a limb!)

The point here is that ASN.1 BASIC-XER does **not** say that "If you define a document that has ambiguous content, that is an illegal specification". It is actually impossible to define such a specification with BASIC-XER.

This issue of when an XML message carries unambiguous semantics is not a well-publicised area, although (in a slightly different form, it occurred and was recognized in the EDIFACT graphical syntax as early as 1984!). Suppose you have a repetition of a number of elements (with some XML element name), and with the repetition carrying some specific semantics. And now suppose you follow that with an XML element with the same name (but carrying different semantics), or even worse, with a further repetition of the same element (carrying different semantics). For example, the first repetition may be specifying allowed elements in a response, and the second repetition may

be specifying forbidden elements in a response. How does a receiver know where the repetition of one semantics ends and the start of the other begins?

When ASN.1 says it is totally **safe** in its basic encodings, it means that it **guarantees** that the two repetitions will be delimited within separate XML elemnts. Hence the different semantics can be identified. This, of course, restricts the form of XML document that can be provided. Done simply, it means requiring an XML element "wrapper" round any repetition of elements. BASIC-XER does this.

But suppose you do not want ASN.1 to be patronising? Suppose **you** have taken responsibility for ensuring that a repetition of elements is always followed by an element with a different XML tag-name. In this case there is no problem in the encoding, no ambiguity, and the verbosity of the XML document will be less than if there was a wrapper. This leads us to EXTENDED-XER.

6. Addition of XER Encoding Instructions - EXTENDED-XER

Input from users showed that enhancements to ASN.1 BASIC-XER were needed to allow the users to do several things:

- To let them control the form of the XML document that was being produced, for example, the use of XML attrinbutes rather than elements, and of XSD space-separated lists; (Why they wanted to do this is very much a separate discussion that can be had in the bar! Are these features really needed in an XML encoding? Why? Bar discussion!);
- To let them take control of whether the encoded document was (easily - or even at all, unambiguously!) decodable, rather than being forced to use a "safe" (but inflexible) defined encoding;
- To be able to vary the way in which basic ASN.1 types such as BOOLEAN or ENUMERATED (or INTEGER with named values) were encoded.

An example of the latter is the encoding of a BOOLEAN value TRUE as the XML empty-element tag "< /true >" or as the text "true" - both have advantages and disadvantages. (Compatibility with XSD means that "true" shold be used, but is this always best?)

This led to the development of XER encoding unstructions, to modify the form of the XML encoding for an ASN.1 schema definition. An example of the application of such encoding instructions is:

```
Company DEFINITIONS ::=
BEGIN
  Employee ::= [UNCAPITALIZED] SEQUENCE {
    id              [ATTRIBUTE] INTEGER(0..MAX),
    recruited       Date,
    salaries        [LIST] SEQUENCE OF
                           salary REAL }
END
```

This results in a BASIC-XER encoding of:

```
     <Employee>
        <id>239</id>
        <recruited>27-11-2002</recruited>
        <salaries>
          <salary>29876</salary>
          <salary>54375</salary>
          <salary>98435</salary>
        </salaries>
     </Employee>
```

being changed to:

```
<employee id = "239">
  <recruited>27-11-2002</recruited>
  <salaries>29876  54375  98435</salaries>
</employee>
```

It also became rapidly clear that application of XER Encoding Instructions tended to follow a pattern. Frequently (almost) all occurrences of "SEQUENCE OF" in an ASN.1 type definition were to be encoded as a space-separated list. Similarly for variations on BOOLEAN or INTEGER or ENUMERATED encodings. Specifying this on every occurrence of these types was verbose, so the concept of an "Encoding Control Section" was introduced. This enables XER encoding instructions to be applied globally (or within limited regions) to ASN.1 types with a single instruction, reducing the verbosity of the schema specification, and increasing the separation of the abstract syntax definition from encoding matters.

The above application of encoding instructions can now become:

```
Company DEFINITIONS ::= BEGIN
  Employee ::= SEQUENCE {
    id         INTEGER(0..MAX),
    recruited  Date,
    salaries   SEQUENCE OF
               salary REAL }
  ENCODING-CONTROL XER
    NAME               Employee AS UNCAPITALIZED
    ATTRIBUTE          Employee.id
    LIST               Employee.salaries
END
```

keeping the ASN.1 specification itself very pure!

An important part of EXTENDED-XER was the introduction of the UNTAGGED encoding instruction. This provides insanity, and total loss of parental control, into ASN.1 encodings (my colleagues would probably disagree with at least the former sentiment!). But it is unfortunately necessary in order to support the current style of use of XML document definition (with DTDs) and mappings from XSD.

As a totally personal view, it would be better if it were difficult or impossible to do unsafe things in the specification of an XML schema. Doing unsafe things either results in a bust specification (ambiguity in encodings), or in a major task for tools (often at run-time) to check for ambiguous documents.

However, current XML schema notations allow users to do unsafe things, and ASN.1 would not be credible as an XML schema notation if it did not also do so. So .... the [UNTAGGED] XER encoding instruction in ASN.1 allows the removal of any XML start-tag (and the corresponding end-tag) that BASIC-XER would provide. This includes XML element wrappers that provide "safety" round repetitions, as well as XML tags on character strings (removal of which produces mixed content.)

If two consecutive character strings have their tags removed, we have an ambiguous encoding. (Where does one abstract value finish, and the other begin?). This particular one is forbidden in EXTENDED-XER, but there are other more subtle cases where it is very much "caveat emptor".

7. Mapping from W3C XML schema (XSD)

It became clear that if ASN.1 was to be taken seriously as an XML schema notation, it had to provide pretty much the full power of XSD. The EXTENDED-XER provision of XER encoding instructions did much to pave the way for this.

However, a decision was taken to produce an ITU-T Recommendation and ISO Standard that would specify a mapping from any XSD specification into an ASN.1 specification, such that:

- Any XML document that was valid under the XSD definition would be a legal encoding under ASN.1 with EXTENDED-XER and the associated encoding instructions; and
- Any ASN.1 encoding that was legal under EXTENDED-XER (with the associated encoding instructions) would produce a document that was valid under the XSD definition.

That was a pretty tall order! And needless to say it was not completely met, but the ASN.1 specification of EXTENDED-XER, and the mapping from XSD have satisfied 99% of this requirement. For almost all purposes, you can easily and simply map XSD into ASN.1 (using a free tool), and be sure that the application of EXTENDED-XER encodings to the ASN.1 specification will result in the same XML document.

But why should you want to? This is a critical question.

If you provide an ASN.1 definition for your information, then all the other Encoding Rules of ASN.1 can be applied to that information. In particular, the mapping from XSD enables the extremely compact encodings produced by the ASN.1 Packed Encoding Rules (PER) (much shorter than an XML encoding) to be applied to the information. It also enables ASN.1 tools that decode (very efficiently and very fast) incoming XML documents to present those documents to application code in a readily digestible and machine-efficient form. Finally, it makes it possible to provide a "relay" or "mapping" application from binary-encoded data to XML-encoded data. This XML-encoded data can then be sent to XML tools, and in particular to browsers, providing a state-of-the-art line monitoring-system.

8. Legacy protocols and file formats

This topic is slightly outside the remit of this conference paper, but it is important. ASN.1 has a feature called "Encoding Control Notation" (ECN). ECN operates in a similar way to the XER encoding instruction control of the form of an XML document, but use of ECN enables an ASN.1 specification to generate almost arbitrary (with safety) binary encodings of the information content of an ASN.1 schema. This feature of ASN.1, linked to its XML encoding (BASIC-XER and EXTENDED-XER) capability is extremely powerful.

An ASN.1+ECN definition of a legacy protocol enables a "relay" or "transformation" system to operate in which an incoming "legacy" file format or protocol is converted into a newly-defined XML-format (possibly using EXTENDED-XER encoding instructions). All this can be done using existing ASN.1 tools.

9. Support for "MIXED"

This is a very techy and detailed area. However, it represents the marriage of "protocol" or "computer-to-computer" messaging with "document transfer", and is therefore important.

XML has its roots in SGML, which came out of the publishing industry. The focus there was on a text document for publication with mark-up of specific items (such as headings, or text in bold or italic) that required special treatment.

The XSD "MIXED" concept tried to reflect the need for defining a document in which most of it was pure text - perhaps provided statically (or rarely cahnged), but in which was embedded elements that would be inserted by a database. The inserted XML elements would conform to an XML schema.

In the ASN.1 approach, there is a text document (transferrable with binary encoding if required), in which there are references to embedded values (at given points) of an XML schema (XML Elements). This can be equated with the MAIL-MERGE functionality of a well-known text-handling systems!

Thus ASN.1 support of "MIXED" is different from other XML schema notations. It allows the specification of text, with an embedded "template" saying where database information is to be included. The details of that database information are provided in the following ASN.1 schema definition.

For example, the following ASN.1 definition:

```
Request-for-payment ::= {
```

```
       text-of-letter       [EMDED-VALUES] UTF8String,
       account              INTEGER,
       final-date           Date }
```

can produce the following instance of an XML document:

```
<Request-for-payment>
Dear Sir/Madam,
     We note that you have not yet paid your account for last month.
     Please remit funds for account <account>283923</account> no later
     than <final-date>25-08-2003</final-date>
Yours sincerely,
     xyz Manager
</Request-for-payment>
```

10. The XML infoset

Work is in progress to define the XML infoset as an ASN.1 type. (So any possible value of an infoset would be a value of that type.) This will give two things:

- It will provide a concrete representation of an infoset, with the ability to serialize it into a compact interchange format using the ASN.1 Packed Encoding Rules, (in wide use since the early 1990s, in many applications.)
- It will enable ASN.1 tools to provide mappings of an infoset to C, C++ and Java code, as an alternative to an interpretive approach for access to an infoset.

There is a growing need for direct interchange of infosets between systems without use of XML serialization, although it is recognised that many applications will still prefer to work with XML documents rather than directly with infosets. These facilities become available from the definition of an infoset as an ASN.1 type, using existing tools. This is probably an advantage over other proposals for binary encodings of interchange formats for the infoset.

11. Comparison of ASN.1 and XSD

This paper and presentation is in no-way attempting to make negative remarks about XSD. Any XSD specification can be mapped to an ASN.1 specification.

Pure use of an ASN.1 schema definition, or mapping to ASN.1 from XSD, or use of XSD alone without ASN.1, all have their merits. XSD has the merit that it uses an XML format for its specification. How important that is for an XML schema definition (or whether it is just "politically-correct") has to be determnined. (RELAX NG provides both an XML syntax and a "compact" syntax). Here is an example of a hypothetical "Invoice" defined in ASN.1:

```
Invoice ::= SEQUENCE {
                    number  INTEGER,
                    name            UTF8String,
                    details         SEQUENCE OF
                                      LineItemPair,
                    charge          REAL,
                    authenticator   BIT STRING}

LineItemPair ::= SEQUENCE {
                    part-no         INTEGER,
                    quantity        INTEGER }
```

And here is the definition of the three-line (in ASN.1!) "LineItemPair" in XSD:

```
<xsd:complexType  name="LineItemPair">
    <xsd:sequence>
         <xsd:element
                    name="part-no"  type="xsd:number"/>
         <xsd:element
                    name="quantity" type="xsd:number"/>
     </xsd:sequence>
</xsd:complexType>
```

However, it is important to note that the ASN.1 definition supports many encoding rules (including use of ECN for "legacy" encodings), whereas XSD supports only XML encodings.

12. Comparison of ASN.1 and RELAX NG

This comparison is much more interesting! The RELAX NG compact syntax is extremely close to an ASN.1 type definition. It just uses a few special characters rather than keywords. For example, here is an earlier ASN.1 specification, usng an in-line defnition of LineItemPair for simplicity:

```
Invoice ::= SEQUENCE {
               number          INTEGER,
               name            UTF8String,
               details         SEQUENCE OF
      SEQUENCE {
                              part-no         INTEGER,
                              quantity        INTEGER},
               charge           REAL,
               authenticator   BIT STRING}
```

This would be the equivalent in RELAX NG:

```
element Invoice {
  element  number     {integer},
  element  name       {text},
  element  details    {
         element part-no   {integer},
         element quantity  {integer} } * ,
  element  charge      {float},
  element  authenticator {bits} ,
```

In both cases there is a well-defined semantic underpinning of the notation, and the RELAX NG "pattern" concept maps simply and straight-forwardly into the ASN.1 concepts of (untagged) "repetitions", "concatenations", and "alternatives". There are also clear similarities in the syntactic structure, but even though the RELAX NG syntax is less verbose than XSD, there is still the repeated "element" keyword (this is the default in ASN.1: if you want an attribute, you add "[ATTRIBUTE]".

There are also differences in the handling of repetitions (use of a "*" in the case of RELAX NG, and use of "SEQUENCE OF" in the case of ASN.1. Similarly optionality is handled with "?" in RELAX NG and with "OPTIONAL" in ASN.1. Similar differences occur for alternatives and for "in any order". These differences are largely cosmetic, but can affect both readability and the ease of producing a schema definition, particularly for a beginner.

A more in-depth comparison of the functionality and syntax of the two notations is outside the scope of this paper, but a big difference is that (as with XSD) a RELAX NG specification does not support the wide range of binary encodings available from an ASN.1 specification (nor the clear separation of encoding matters from content definition).

13. Conclusions

This paper has presented the emergence of ASN.1 as an XML schema notation, and its advantages in that role.

ASN.1 remains a (the?) major notation for specification of binary transfers, with strong tool support giving very rapid time-to-market for implementations, but it now provides an opportunity for XML schema definition which has the functionality of XSD or RELAX NG, with the added advantage of specification of extremely compact binary encodings in addition to the XML encodings.

Bibliography

[ASN.1] http://www.itu.int/ITU-T/studygroups/com17/languages/

[INTRO] http://asn1.elibel.tm.fr/introduction

[USES] http://asn1.elibel.tm.fr/uses/

[LARMOUTH] http://www.oss.com/asn1/larmouth.html

[DUBUISSON] http://www.oss.com/asn1/dubuisson.html

[DUB-FR] http://asn1.elibel.tm.fr/fr/livre

[FROM-XSD] http://asn1.elibel.tm.fr/xsd2asn1

[LINKS] http://asn1.elibel.tm.fr/links/

Glossary

XSD	W3C XML Schema

Biography

John **Larmouth**
Larmouth T & PDS Ltd
Bowdon
United Kingdom
j.larmouth@salford.ac.uk>

John Larmouth has been involved with standards activity in ISO and ITU-T, particularly ASN.1, for over twenty years. He is the ISO and ITU-T Rapporteur for ASN.1 and is now retired and running his own consultancy company, Larmouth Training & Protocol Design Services Ltd. He was previously on the academic staff at Cambridge University England, and latterly was a Professor and Head of Department at Salford University England. He continues to be active in standards development, particularly in ASN.1 and its support for XML. The use of ASN.1 as the notation for protocol specification brings him into contact with standardization in a number of fields, including biometrics, smart cards, e-business and intelligent transportation systems.