John Larmouth
ITU-T ASN.1 Rapporteur
ITU-T SG17

j.larmouth@salford.ac.uk

ASN.1 and its use for e-health standards

Workshop on Standardization in E-health
Geneva, 23-25 May 2003

With (no!) apologies! Light relief?

- This presentation is not about what exchanges you need for e-health
- It is about the notation to use for such definitions, and the form of encodings (bits on the line) for your messages
- Not important? Please think again.
- My focus is on ASN.1 and its value as such a notation
- But alternatives will be discussed

What is ASN.1? (1)

- It is a notation for defining the content – the abstract syntax of documents
  - supported by binary encoding rules
  - supported by XML encoding rules
- Hence Abstract Syntax Notation One
- It can also be called an XML Schema Notation

What is XML?

- eXtensible Mark-up Language
- It is a human-readable form of encoding for messages with a number of advantages, and an immense amount of hype!
- XML usually talks about documents not messages
- An XML schema notation (there are several, XSD is the best known) defines the form (structure, content, syntax, of XML documents

A quick look at an XML document

```
<Patient>
        <name>George Bush</name>
        <occupation>Politician</occupation>
        <age>40</age>
        <previous-history>
                <admission>……</admission>
                ……
                </surgery>……</surgery>
        </previous-history>
        <current-illness>
                <encrypted>AFEC9D49</encrypted>
        </current-illness>
</Patient>
```

What is ASN.1? (2)

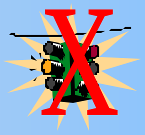
- Quite old – early 1980s.
- Mature! But still developing.
- ISO Standards and ITU-T Recommendations
- Heavily used to define messages in many industrial and commercial sectors
- Until recently, all ASN.1-defined messages were encoded in binary

ASN.1 - lineage

- ASN.1 was borne around 1982-ish
- First ASN.1 Standard (CCITT X.409) in 1984
- Borne from X.400 (mother with an early child, and the e-mail standard the world should have had!)
- Fathered by X.500 (certified insane at birth, but totally secure)
- Grand-parents (OSI) died prematurely and are not discussed in polite conversation today

Waving a magic wand

- Without ASN.1-defined messages:
  - The lights go out!
  - Portable phones don’t work!
  - Parcels get lost!
  - Traffic lights fail!
  - Aircraft fall from the sky!
  - Your impending marriage suffers as Net Meeting fails!



- On second thoughts – it might be a better world?

A sample of ASN.1 notation

```
BBCard ::= SEQUENCE {
    name          IA5String,
    team          IA5String,
    age           INTEGER,
    position      IA5String,
    handedness ENUMERATED {
              left-handed,
              right-handed,
              ambidextrous },
    batting-average REAL }
```

A sample of ASN.1 notation with constraints (encouraged)

```
BBCard ::= SEQUENCE {
      name          IA5String (SIZE (1..60)),
      team          IA5String (SIZE (1..60)),
      age           INTEGER (1..100),
      position      IA5String (SIZE (1..60)),
      handedness ENUMERATED {
                  left-handed,
                  right-handed,
                  ambidextrous },
      batting-average REAL }
```

The C data-structure for the base-ball card

```
typedef struct BBCard {
        char name [61] ;
        char team [61] ;
        short age ;
        char position [61] ;
        enum {
                left_handed = 0,
                right_handed = 1,
                ambidextrous = 2,
                } handedness ;
        float batting_average ;
} BBCard ;
```

A base-ball card value in XML syntax

```
<BBCard>
        <name>Jorge Posada</name>
        <team>New York Yankees</team>
        <age>29</age>
        <position>C</position>
        <handedness>right-handed</handedness>
        <batting-average>0.277</batting-average>
</BBCard>
```

The Montagues and the Capulets

(A shorter history of contending philosophies)

(With apologies to William Shakespeare and to those from a non-UK culture!)

The Montagues and Capulets

- A long and on-going civil dispute
- Montagues =>
  Binary-based specification
- Capulets =>
  Character-based specification

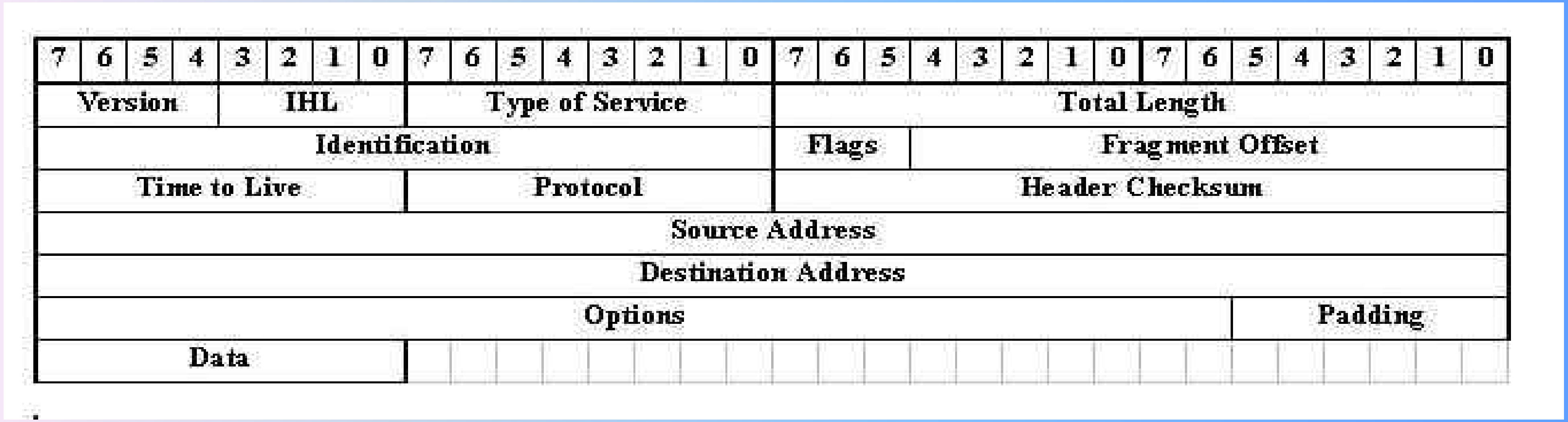
Understanding of protocol specification techniques

- 1.5 billion seconds ago …..
  Computers started to communicate
- Major advances every 150 million seconds
- There was a need for
  - A means of syntax (data structure) specification
  - Procedure (sequence) specification
  - Test suite specification
  - Validation
- And tools to support rapid and (largely) error-free implementation!

The stone-age Montagues

Diagrams of bits and bytes - e.g. IPv4

(The earliest approach, simple and clear, but focusing totally on the bits-on-the-line.)



Tool support not possible

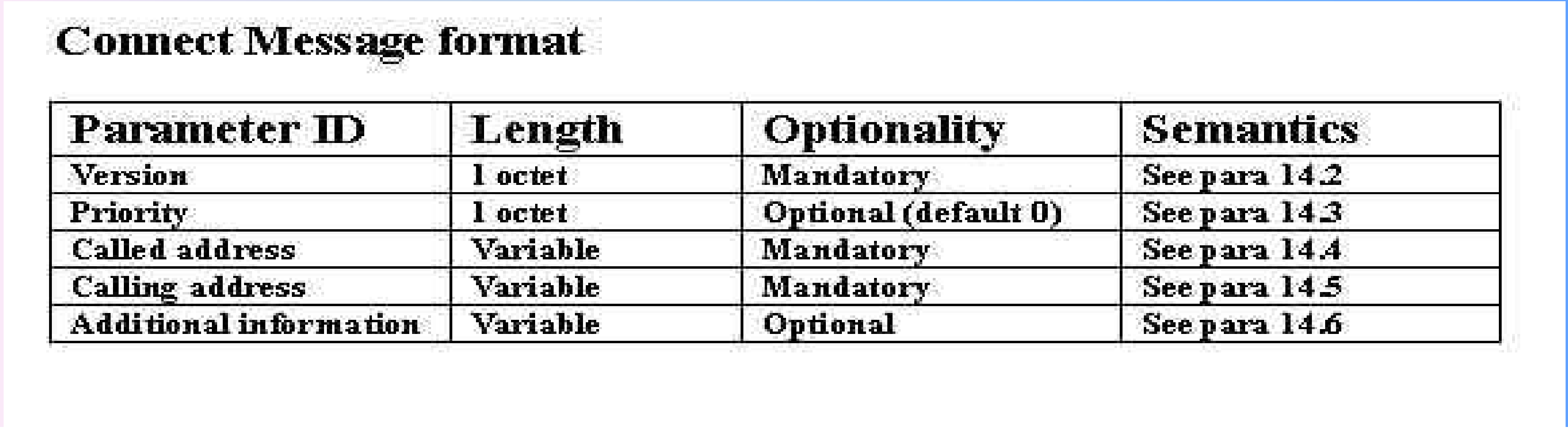
Extensibility support crude - based on reserved fields.

The stone-age Capulets

- Simple “command lines” – in ASCII!
- Simple character mnemonics and error codes (eg “200 OK”)
- Simple comma-separated parameters
- Good for simple dialogues
- Extensibility by adding commands in V2, with unknown commands ignored by V1 systems

The Bronze Age Montagues invent TLV and Tabular Notation

- Each PDU and each parameter has an ID (or Tag), a Length, and a Value
- Nested TLVs in TLVs – sounds familiar to XML cognoscentiae?
- (Should have been patented? No XML!)
- Tables list each parameter: Tabular Notation



Connect Message format

Parameter ID	Length	Optionality	Semantics
Version	1 octet	Mandatory	See para 14.2
Priority	1 octet	Optional (default 0)	See para 14.3
Called address	Variable	Mandatory	See para 14.4
Calling address	Variable	Mandatory	See para 14.5
Additional information	Variable	Optional	See para 14.6

Tabular Notation and TLV was a break-through

- Extensibility was EXCELLENT.
- Version 1 systems just skipped (using TLV) anything they did not know.
- Tool-support, however, not possible.
- **But it was verbose!**

But not as verbose as the character-based encodings used by the Capulets!

The Bronze Age Capulets invent BNF

- The Capulets’ main concern was with precise specification of correct syntax
- This was the dawning of Backus Naur Form (BNF)
- This potentially allowed more complex information to be specified in a “command”
- But it never really made it to the modern era of **automatic** mapping to Java, C++, C etc.

150 Million seconds after the Bronze Age

- Recognition of:
  - Separation of "abstract syntax" (content) from encoding
  - Encoding rules
- ASN.1 specifications define a de facto, platform and language independent API
- Tools emerge to support the transformation of ASN.1 to a platform-specific API, and the encoding of data across that API
- Supported on a wide range of platforms and languages – typically C, C++, Java

Profits for all!

300 Million seconds later, the Capulets develop XML

- Don't really need to describe it to this audience!
- But focus still more on what is a syntactically correct document, rather than on its content
- Can be seen as the TLV approach stolen from the Montagues (!) but with an end-tag instead of a length field to delimit elements
- Rapidly gained popularity!
- The encoding-of-choice for many applications today

And finally, after another Million seconds

ASN.1 develops XML Encoding Rules

Romeo and Juliet marry!

Back to the serious business!

- The separation of the specification of document content from representation issues has been retained in the marriage
- XML is (just) another way of encoding data defined using ASN.1
- The same ASN.1 definition (schema) defines **both** XML **and** (very efficient) binary encodings
- This includes **canonical** encodings for security sensitive work and signatures

Current work using ASN.1 as an XMLschema

- There are two pieces of OASIS work that use ASN.1 as an XML schema
- The XCBF work (Extensible Common Biometrics Formats) uses ASN.1 alone to define the XML document
- The UBL (Universal Business Language) work uses XSD as the master schema, but this is algorithmically mapped into an ASN.1 schema by a mapping tool in order to provide binary protocols

Mapping between binary and XML

- All message formats defined using ASN.1 have a both an XML format and a binary format (no additional work needed)
- Tools are available to map message formats between compact binary and XML formats for such messages (in both directions)
- For some applications you may want only binary or only XML, but you use only one notation

ASN.1 support for XML - basic

- The first Recommendation developed for XML support in ASN.1 was called "basic XML encoding rules", or BASIC-XER
- This provided no control over details of XML representation such as use of attributes instead of elements, space-separated lists, xsi:type, or namespaces (Do you care?)
- This was very much basic XML!
- A canonical representation was also defined

Extended ASN.1 support for XML

- Encoding Instructions were introduced into the ASN.1 notation
- These control things such as encoding as an attribute rather than as an element, or as a space-separated list
- Also support for XSD concepts such as union, default-for-empty, nillable, and so on
- Encoding instructions can be prefixed or can be included separately in an Encoding Control Section (see later)

“Coloring” the BaseBall card schema

```
BBCard ::= SEQUENCE {
    name        [ATTRIBUTE] IA5String,
    team        [ATTRIBUTE] IA5String,
    age         INTEGER,
    position    IA5String,
    handedness ENUMERATED {
            left-handed,
            right-handed,
            ambidextrous },
    batting-average REAL }
```

Unchanged API

The new XML syntax

```
<BBCard
        name = “Jorge Posada”
        team = “New York Yankees” >
        <age>29</age>
        <position>C</position>
        <handedness>right-handed</handedness>
        <batting-average>0.277</batting-average>
</BBCard>
```

Another example

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

Without the coloring, we have basic XML:

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

With the coloring, we get:

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

Much less verbose, assuming you care!

But the same information content

Equivalently we could write:

```
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
```

At first sight this is more verbose and less clear! But …..

Advantages of the separate Encoding Control Section

- It keeps a much clearer separation of encoding issues from content
- In most cases it is much less verbose, as you are able to apply encoding instructions globally, or to selected parts of the spec, and also to use NOT on selected parts
- The embedded form is usually good for examples
- The separate section is best for real specifications

Part of an invoice in XSD and the ASN.1 equivalent

```
<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>
```

maps to:

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

The full invoice in ASN.1 (couldn't get the XSD onto a single slide!)

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

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

The XML encoding of an Invoice (1)

```
<Invoice>
        <number>32950</number>
        <name>funny-name with &lt;</name>
        <details>
                <Line-item>
                        <part-no>296</part-no>
                        <quantity>2</quantity>
                </Line-item>
```

Cont

The XML encoding of an Invoice (2)

Continuation

```
                <Line-item>
                        <part-no>4793</part-no>
                        <quantity>74</quantity>
                </Line-item>
        </details>
            <charge>397.65</charge>
            <authenticator>
                EFF8 E976 5403 629F
            </authenticator>
        </invoice>
```

The importance of binary encodings available

- Re-opening the Montague and Capulet war!
- Romeo and Juliet, happily married for a few years, get divorced! Maybe not!
- There are real merits in using the same schema definition for both character and binary encodings
- The next example can be taken quickly

A personnel-record defined (1)

```
PersonnelRecord ::= SEQUENCE {
   name              Name,
   title             VisibleString,
   number            EmployeeNumber,
   dateOfHire        DATE-TIME (Date),
   nameOfSpouse      Name,
   children          SEQUENCE OF
                             child    ChildInformation
                             DEFAULT {} }
```

A personnel-record defined (2)

```
ChildInformation ::= SEQUENCE {
          name              Name,
          dateOfBirth DATE-TIME
  (Date)}
Name ::= SEQUENCE {
          givenName         VisibleString,
          initial           VisibleString,
          familyName        VisibleString}
EmployeeNumber ::= INTEGER
```

An example personnel-record (1)

```
<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
```

Cont

An example personnel-record (2)

```
<children>
    <child>
        <name>
            <givenName>Ralph</givenName>
            <initial>T</initial>
            <familyName>Smith</familyName>
        </name>
        <dateOfBirth>19571111</dateOfBirth>
    </child>
    <child>
        <name>
            <givenName>Susan</givenName>
            <initial>B</initial>
            <familyName>Jones</familyName>
        </name>
        <dateOfBirth>19590717</dateOfBirth>
    </child>
</children>
</PersonnelRecord>
```

A count of octets for the personnel-record value

- With white-space omitted, 653 octets
- Fully human-readable with white-space, can be double that!
- Binary TLV-style encoding 136 octets
- Compact binary encoding 94 octets (other examples compress better)
- ZIP compression ….. of XML or binary?
- But does size (or transaction processing speed) matter anyway?

Tail-end remarks

- Everyone is bored by now!
- One example is much like another.
- Refer to the paper if you wish.
- Messages from these examples:
  - Simplicity and clarity of specification
  - Content clearly separated from syntax
  - Ability to apply coloring for XML
  - The same schema definition provides all forms of encoding as well as C, C++ and Java APIs

Future developments

- Discussions are ongoing on
  - The application of ASN.1 to SOAP
  - Use of ASN.1 in web services
  - An ASN.1 definition of the infoset
  - Mapping UML to ASN.1
- Much is in place already, but work continues on the full integration of ASN.1 with XML
- ASN.1 earlier shed its dependence on OSI and is now shedding its dependence on binary encodings
- But it still fully supports efficient binary encodings

In conclusion

Notations for e-health message content definition, and the encoding of those messages may seem boring.

But it is an important aspect of any e-health standardisation.

Conscious and informed decisions are needed.

Consider the ITU-T Recommendation ASN.1

The End – clapping is allowed!