Proposal to use ASN.1 & ECN in ISO/IEC 7816-4

Olivier DUBUISSON

France Télécom R&D

ITU-T ASN.1 Project leader

Olivier.Dubuisson@francetelecom.com

John LARMOUTH

Larmouth Training & Protocol Design Services Ltd.

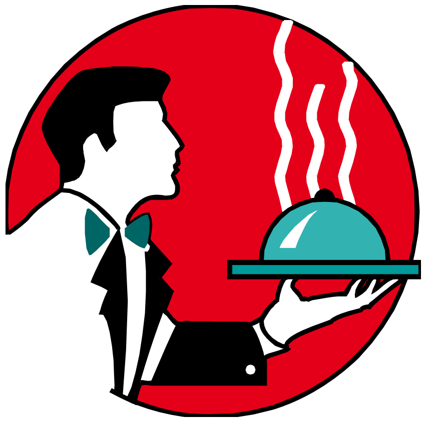
ASN.1 Rapporteur

J.Larmouth@salford.ac.uk

http://asn1.elibel.tm.fr/ecn

ASN.1: What is it?

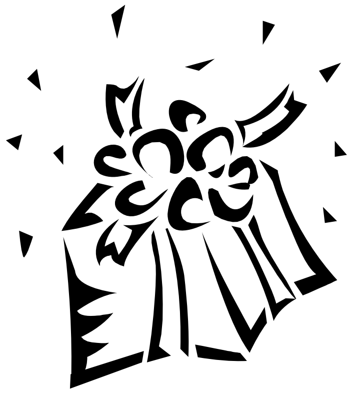
- *Abstract Syntax Notation number One*
- International standard : ITU-T X.680 to X.683 | ISO/IEC 8824-1 to 4
- Diversity of operating systems and programming languages
- Describes data exchanged between two communicating applications
- Formal notation supported by tools:
  - no ambiguity;
  - makes validation easier, at low cost;
  - reduces the *time-to-market;*
  - readable by experts of the application domains;
  - easily understandable by implementors;
  - translates easily into any programming language (C, C++, Java, Cobol, etc, and over 150 platforms)



- ASN.1 is a critical part of our daily lives; it's everywhere, but it works so well it's invisible!
- More information: http://asn1.elibel.tm.fr/introduction

Other advantages of ASN.1

- Several associated standardized encodings, such as:
  - efficient (binary) encoding: *Packed Encoding Rules* (PER)
  - canonical encoding for digital signatures: *Distinguished Encoding Rules* (DER)
  - XML encoding rules (XER)
- Mature, long record of reliability and interoperability
- Sends information in any form (audio, video, data…) anywhere it needs to be communicated digitally
- Offers extensibility: interworking between previously deployed systems and newer, updated versions designed years apart
- Full and direct support of international alphabets (Unicode)
- Has evolved over time to meet industry needs



Current uses of ASN.1

- Audio & Video over the Internet
  AT&T, Intel, IBM, Microsoft, 3COM
- Electronic Commerce
  American Express, GTE, MasterCard, VISA
- Telephony
  AT&T, MCI, Motorola, Nokia, Sprint
- Aviation
  FAA, ICAO
- Manufacturing
  Ford, Mercedes Benz, Mitsubishi
- Network Management
  Bull, Compaq, Hewlett-Packard, Sun
- Routers
  Bay Networks, Cisco, Racal, Xyplex
- For other uses,
  see http://asn1.elibel.tm.fr/uses/

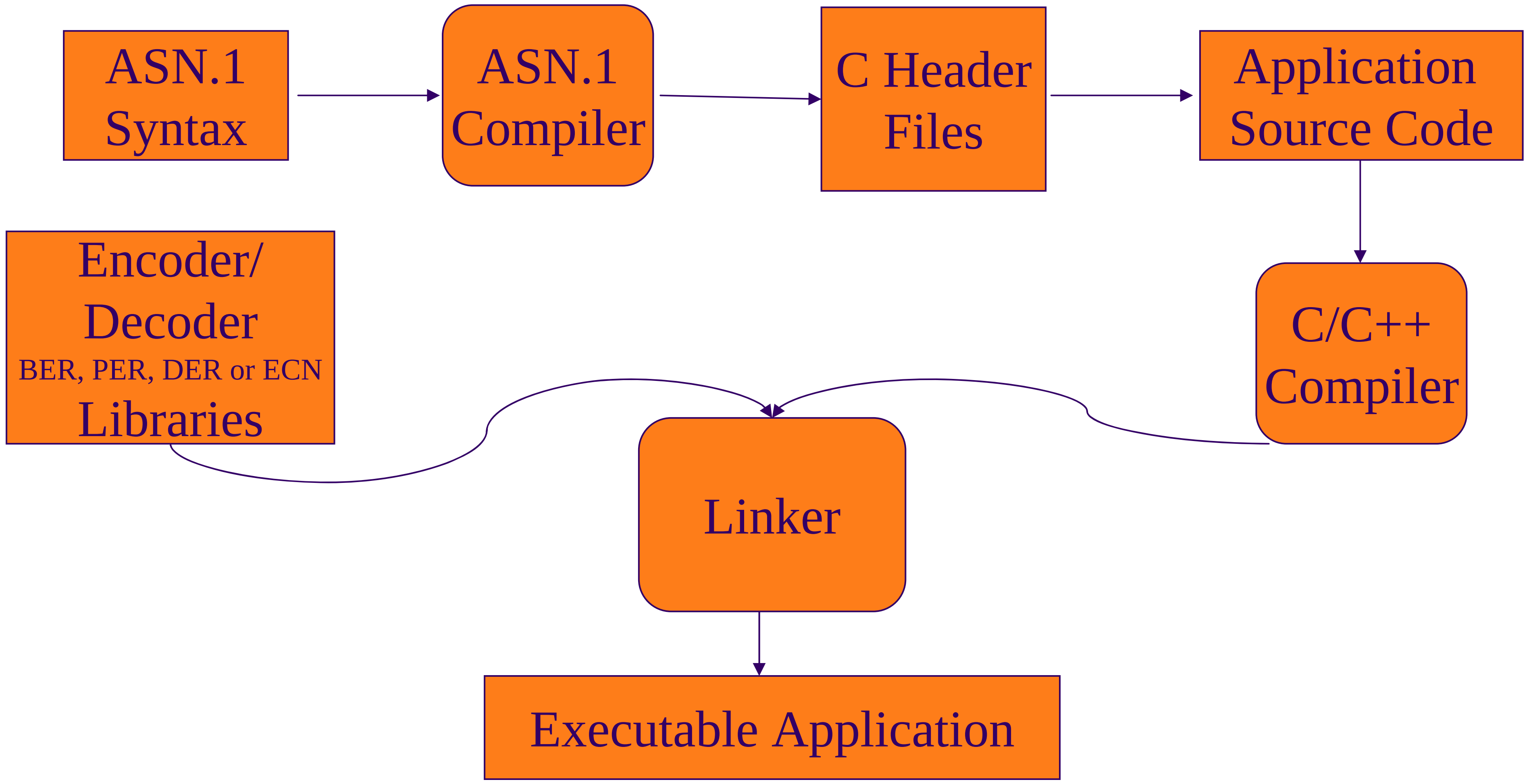
ASN.1 and Encoding Rules standards

- ITU-T Rec. X.680 | ISO/IEC 8824-1 - Basic ASN.1 Notation
- ITU-T Rec. X.681 | ISO/IEC 8824-2 - Information Object Classes
- ITU-T Rec. X.682 | ISO/IEC 8824-3 - Constraints
- ITU-T Rec. X.683 | ISO/IEC 8824-4 - Parameterization

- ITU-T Rec. X.690 | ISO/IEC 8825-1
  Basic Encoding Rules (BER)
  Canonical Encoding Rules (CER)
  Distinguished Encoding Rules (DER)
- ITU-T Rec. X.691 | ISO/IEC 8825-2
  Packed Encoding Rules (PER)
- ITU-T Rec. X.692 | ISO/IEC 8825-3
  Encoding Control Notation (ECN)
- ITU-T Rec. X.693 | ISO/IEC 8825-4
  XML Encoding Rules (XER)
- ITU-T Rec. X.694 | ISO/IEC 8825-5
  Encoding XML-Defined Data Using ASN.1

All **freely downloadable** from http://www.itu.int/ITU-T/studygroups/com17/languages/

Development work flow



Using ASN.1

ASN.1 can be used with most modern programming languages, including Java and C++, as well as older ones such as C and COBOL.

ASN.1

```
PersonalInfo ::= SEQUENCE {
  married BOOLEAN,
  age     INTEGER (123456..124000),
  name    PrintableString}
```

Generated C header file:

```
typedef struct PersonalInfo {
   ossBoolean married;
   int        age;
   char       *name;
 } PersonalInfo;
```

Encoding/decoding:

```
ossEncode(world, PersonalInfo_PDU,
        &unEncodedData, &EncodedData);
ossDecode(world, &pdunum, &EncodedData,
        &outBufPtr);
```

Some free tools

- OSS Nokalva syntax and semantics checker (ASN.1, ECN):
  - http://www.oss.com/products/checksyntax.html
- France Telecom R&D ASN.1 syntax checker and pretty-printer:
  - http://asn1.elibel.tm.fr/tools/asnp
- OSS Nokalva Visual ASN.1:
  - http://www.oss.com/products/visual.html
- Translator of XML Schemas into ASN.1 modules:
  - http://asn1.elibel.tm.fr/xsd2asn1
- List of other tools:
  - http://asn1.elibel.tm.fr/links/#tools
- ITU-T database of ASN.1 modules:
  - http://www.itu.int/itu-t/asn1/database
- Object identifiers (OID) database:
  - http://asn1.elibel.tm.fr/oid

To find more information

ASN.1 Information site

http://asn1.elibel.tm.fr

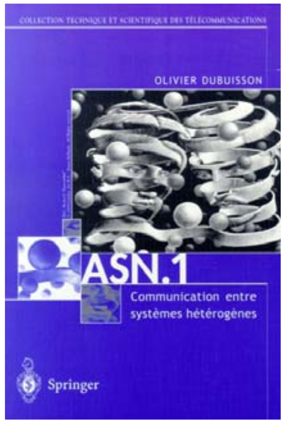
The answer for communication between systems

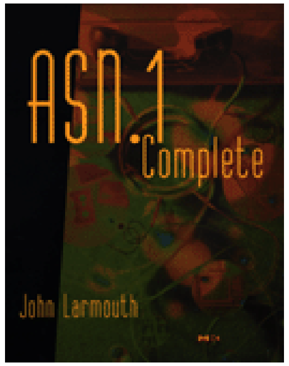
Now available!

Abstract Syntax Notation number One (ASN.1) is an international standard that aims at specifying data used in communication protocols. It is a powerful and complex language: its features are designed to describe accurately and efficiently communications between homogeneous or heterogeneous systems.

Webmaster 5 / 5 / 2000 This site is designed by Yoann Regardin and Olivier Dubuisson.

- ASN.1 website: http://asn1.elibel.tm.fr

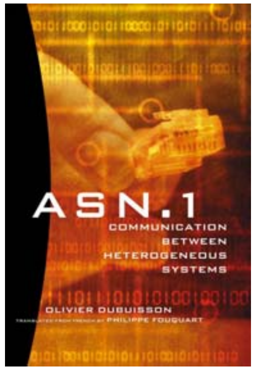

COLLECTION TECHNIQUE ET SCIENTIFIQUE DES TÉLÉCOMMUNICATIONS
OLIVIER DUBUISSON
ASN.1
Communication entre systèmes hétérogènes
Springer



– *ASN.1 - Communication entre systèmes hétérogènes*
(Olivier Dubuisson, Springer Verlag, 1999)
http://asn1.elibel.tm.fr/fr/livre

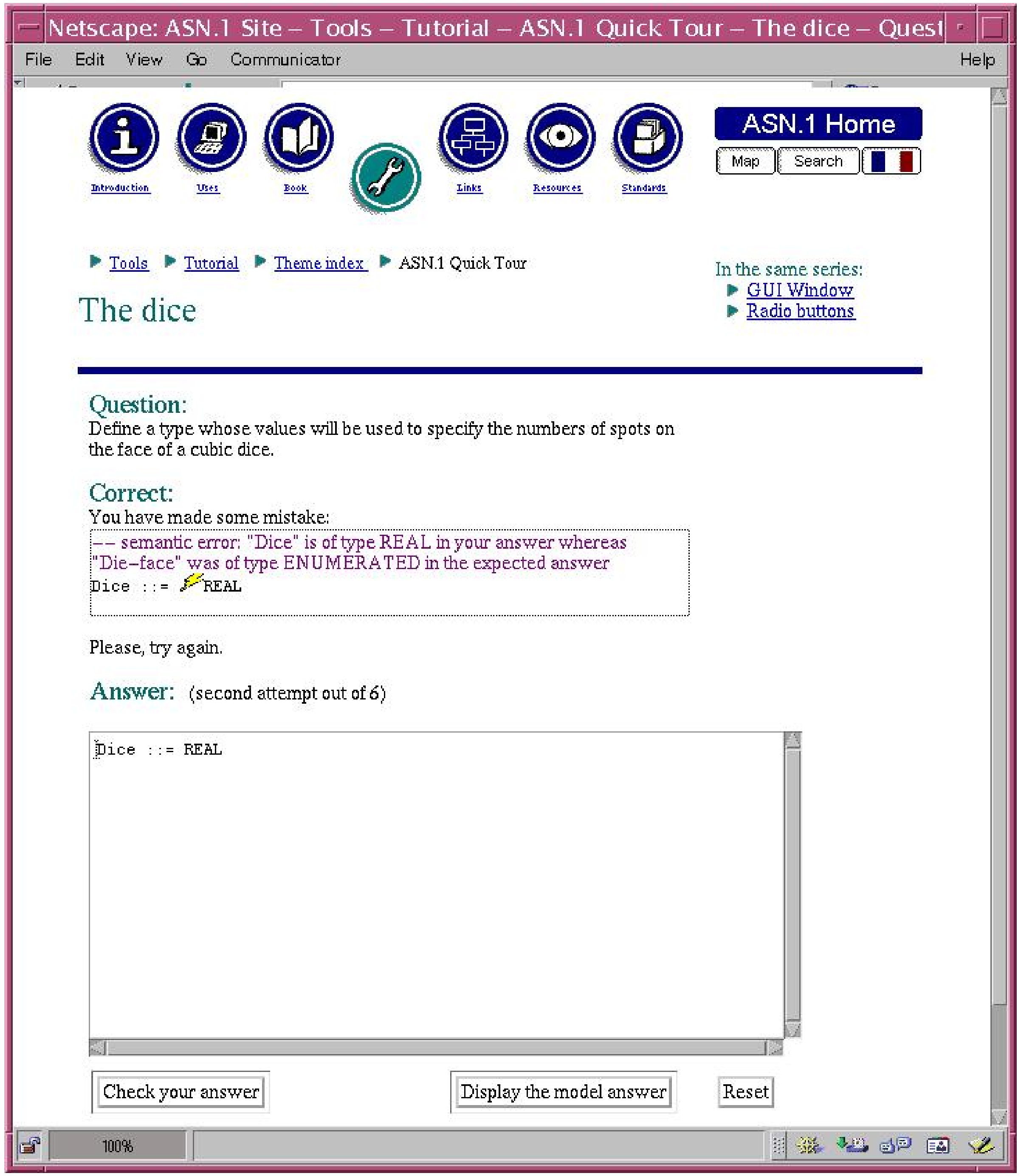
– *ASN.1 - Communication Between Heterogeneous Systems*
(Olivier Dubuisson, Morgan Kaufmann, 2000)
free download: http://www.oss.com/asn1/dubuisson.html

– *ASN.1 Complete*
(John Larmouth, Morgan Kaufmann, 1999)
free download: http://www.oss.com/asn1/larmouth.html



e-tutorial on ASN.1

http://asn1.elibel.tm.fr/tutorial


Netscape: ASN.1 Site – Tools – Tutorial – ASN.1 Quick Tour – The dice – Quest
File Edit View Go Communicator Help
Introduction
Uses
Book
Links
Resources
Standards
ASN.1 Home
Map
Search
100%

Tools ▶ Tutorial ▶ Theme index ▶ ASN.1 Quick Tour

In the same series:
- GUI Window
- Radio buttons

The dice

Question:
Define a type whose values will be used to specify the numbers of spots on the face of a cubic dice.

Correct:
You have made some mistake:

```
-- semantic error: "Dice" is of type REAL in your answer whereas
"Die-face" was of type ENUMERATED in the expected answer
Dice ::= REAL
```

Please, try again.

Answer: (second attempt out of 6)

```
Dice ::= REAL
```

Check your answer | Display the model answer | Reset

France Télécom R&D

ITU-T ASN.1 Project

- Promotes ASN.1 to all ITU-T study groups and also to other standardization organizations (ISO, ETSI, ECMA, W3C, IETF…)
- Helps users understand and write ASN.1 specifications
- Provides tools and ensures quality of the specifications to be published
- First actions:
  - Free ASN.1 module database
  - Object identifier (OID) repository: http://asn1.elibel.tm.fr/oid
- Website: http://www.itu.int/ITU-T/com7/asn1

ASN.1 Consortium

- Group of ASN.1 users (people and companies), specifiers and tool-vendors
- Shares resources and information
- Promotes ASN.1 (towards newspaper and journals, companies, universities…)
- No standardization work
- Three forums:
  - Industry forum
  - Standardization forum
  - Academic forum
- Website: http://www.asn1.org

So ... What are we selling?

- An ASN.1 specification for 7816-4 will add clarity to the specification.

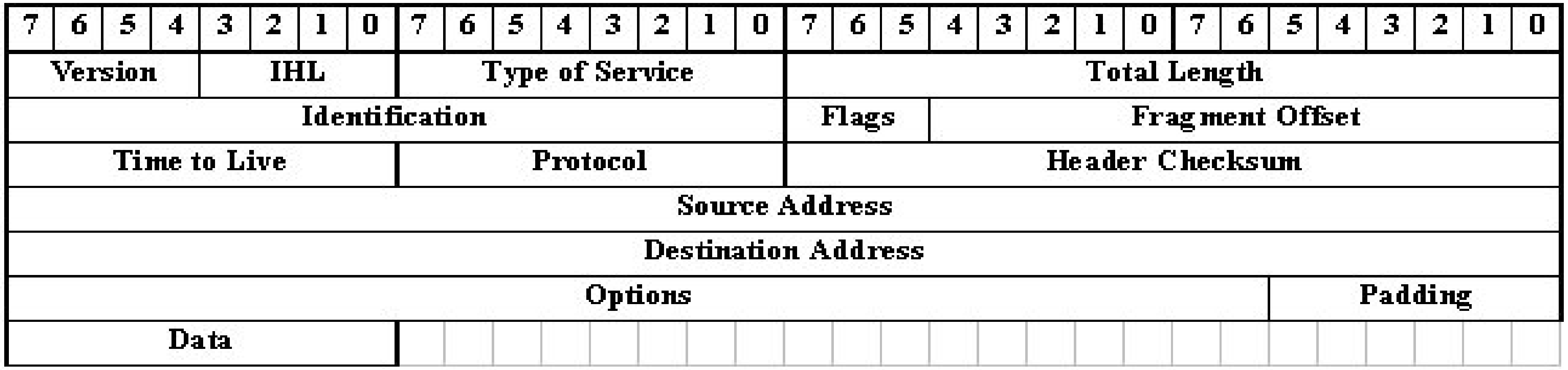
- An ASN.1 specification for 7816-4 will allow rapid implementation using ASN.1 tools.

- Use of ASN.1+ECN will provide a specification of exactly the same encoding as you have now.

Development of encoding notation concepts (1)

Diagrams of bits and bytes - e.g. IPv4

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

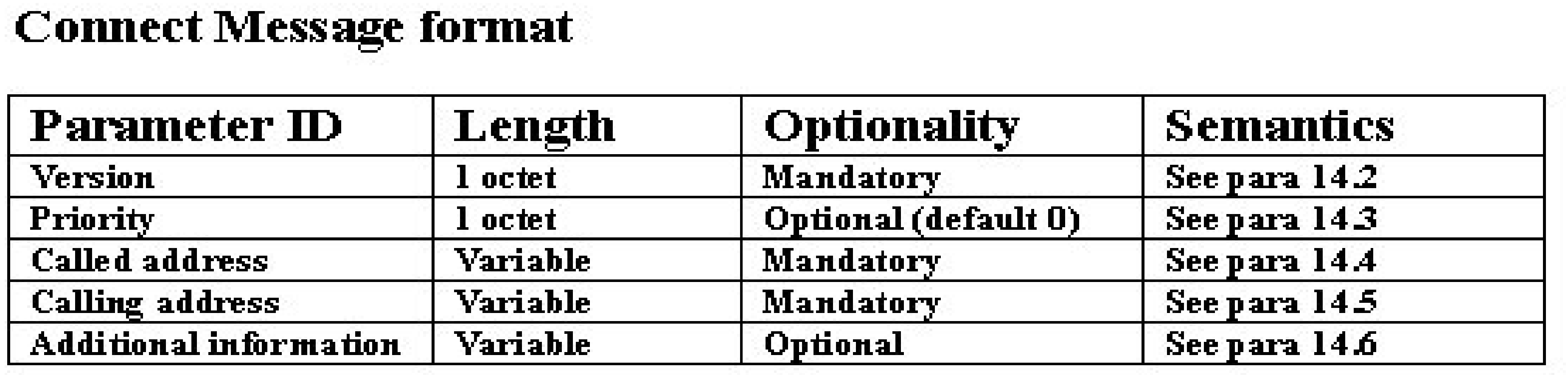


Tool support not possible.
Does not work well for optional or variable-length fields.

France Télécom R&D

Development of encoding notation concepts (2)

- Each parameter has
  Parameter ID (or type), length, value
- 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

Tool support not possible.
Handles optionality and variable length and variable repetitions.
Handles extensibility. Very powerful. But verbose.

Problems solved by TLV encodings

- Variable length fields (Length determinants.)
- Optionality (Optionality determinants)
- Permits random order. Once – no longer – considered A GOOD THING! (Testing and security problems)
- Supports alternatives (Choice determinants)
- Generalizes to arbitrary depth
- Provides “extensibility” easily

The emergence of ASN.1

- **Clear separation of information content from encoding**
- Originally supported only by TLV-style Basic Encoding Rules, but now by other encoding rules such as DER, PER and XER - all very important.
- Tools available to map ASN.1 specification into C, C++, Java classes / data structures, so can implement on many many platforms. Library routines provide many different encodings.
- Used for embedded systems with little memory.
- Minimum requirement is a C cross-compiler.
- Provides rapid development of applications, with minimum de-bugging and testing.

The need for ECN (1)

- Existence of legacy encodings:
  - Encodings are informally described as diagrams of bits and bytes, or tables and tables and more tables!
- **Legacy protocols won't die:**
  - Tools (and staff training) investments lead to new protocols being defined in the same way, using the same encoding techniques, as old ones
  - Need to re-define with new notations that can provide tool support, but **retain the bits-on-the-line**
- Need to (formally) reverse-engineer:
  - Clearer specification of the information content of messages;
  - Easier evolution and addition of new messages;
  - Automatic implementation with the help of tools;
  - Reduces time and cost of test and validation
  - Allows easier use of new encodings with relays mapping to (for example) XML
- Many varied approaches to encodings (for example, use of hex 00 to represent 256!)
- Need to have a formal notation to describe the encoding of these legacy protocols – **a challenging but rewarding task**.

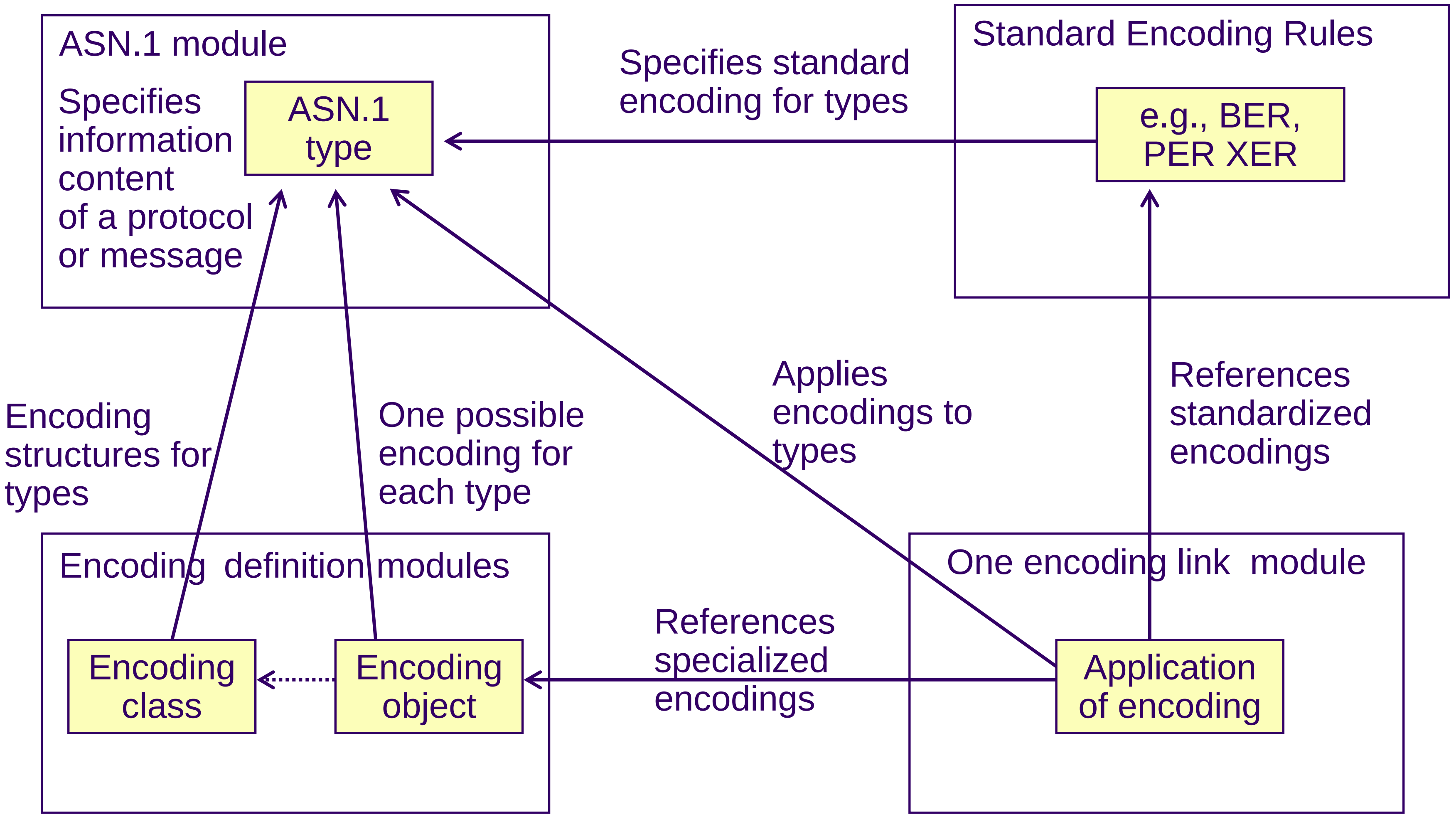
The need for ECN (2)

- When is ECN needed?
  - **Not as often as you might think!**
  - Use of unaligned PER (or, for TLV-based protocols, of BER or DER) can often do 90% of the job.
  - **But the last 10% matters!**
- **The process of applying ECN:**
  - Examine carefully the information content of messages, and define that in ASN.1
  - See if a combination of existing encoding rules will produce the required encoding (if YES, you need only minimal ECN to switch between BER-encoded parts and PER-encoded parts)
  - **Otherwise identify the specialised encodings that are needed.**
- Examples from ISO 7816-4:
  - The length field for Lc is an integer, but (if it is small enough) it can be encoded as either a short form or a long form, as an encoder’s option (subject to constraints based on previous messages), with the choice determined by whether the first octet is “00” or not.
  - The Le field is an integer in the range 1 to 256, with the encoding “00” representing 256.
  - These “curiosities” in an encoding cannot be handled by PER or BER, and need ECN to provide specialized encodings for these fields.
- **There are some, but not many, places in ISO 7816-4 that need specialized encodings**

The ECN notation

- *Encoding Control Notation* (ITU-T Rec. X.692 | ISO/IEC 8825-3)
- Formal notation supported by tools
- Helps integration: a common notation, common tools, for all protocols
- Classification of techniques used in legacy protocols:
  - length, choice, optionality determination,
  - padding bits, alignment, self-delimited encodings…
- ECN modules applied to ASN.1 modules:
  - an **encoding object** defines the encoding of an ASN.1 type;
  - the ASN.1 specification is “pure”: it only deals with data that has semantics for the communicating applications;
  - the ECN specification inserts auxiliary fields into the **encoding structure** that are specific to the encoding
- The addition of ECN to ASN.1 makes it very powerful in this area
- Information, download: http://asn1.elibel.tm.fr/ecn

The Big Picture - ECN concepts


ASN.1 module
Specifies information content of a protocol or message
ASN.1 type
Standard Encoding Rules
e.g., BER, PER XER
Specifies standard encoding for types
Encoding structures for types
One possible encoding for each type
Applies encodings to types
References standardized encodings
Encoding definition modules
Encoding class
Encoding object
References specialized encodings
One encoding link module
Application of encoding

France Télécom R&D

In summary

- All ASN.1 type definitions implicitly define (generate, in ECN terms) a basic “encoding structure” which contains only the fields needed to carry the information content of the ASN.1 type.
- If, for example, BER is applied to this encoding structure, additional T and L fields will be added. If PER is applied to this structure, different fields will be added for determinants.
- Means of encoding length, choice, and optionality determination are very varied (this is one of the main focuses of ECN) and we need to control the addition of such fields, and identification of their use as determinants.
- Approaches to extensibility are very varied, all involve either differential encode/decode or trailing bits in encodings. Use of “tail-end optionality” – if there are any bits left, use them – is common.

Encodable item – an early concept

- Part of an ASN.1 definition whose encoding can be independently specified.
- Originally just the definition of encodings of bit-fields (boolean, integer, etc). This is a straight-forward part of ECN!
- Extended to the definition of procedures for length, repetition, choice, and optionality determination, and for transforming values, for example, by adding a constant value, or …. (a number of different transformations are available).
- All encodable items can have an encoding defined for them that covers all possible values of that item.
- For a bit-field such as boolean or integer, the specification would cover the encoding of all possible values.
- For a procedure such as optionality determination, the specification covers how the presence or absence is to be determined (often using some other field).

Detailed ECN - Encoding classes

- The set of all possible encodings for an encodable item.
- Includes encoding definitions for bit-fields (boolean and integer classes).
- Includes definitions for handling sequence-of, optionality, etc.
- The definitions of the encoding are encoding objects for the encoding class.
- Encoding classes start with a “#” to make a clear distinction from ASN.1 types.
- Examples: **`#INTEGER`**, **`#SEQUENCE`**, `#My-defined-type`.
- These are synonyms for primitive classes **`#INT`**, **`#CONCATENATION`**.
- These form the building blocks of encoding structure definitions.

Detailed ECN - Encoding structure definitions

- Specify the structure of fields (and construction mechanisms) in an encoding.
- Similar to ASN.1 type definition, but underlying concepts very different.
- Fundamental to ECN work.
- The basic concept is that we can define the structure of an encoding (including determinant fields) and then map the ASN.1 abstract information content into some of those fields, defining the others as providing determinants.

Detailed ECN - Syntax to define encoding objects

- Encoding objects are to encoding classes as ASN.1 values are to ASN.1 types:

```
my-int **INTEGER ::= ......**

my-boolean-encoding **#BOOLEAN ::= ......**
```

- The syntax on the right for defining an encoding object of class `#BOOLEAN` is specific to the **`#BOOLEAN`** class, and differs from the `#INT` class syntax.

- Some examples of the syntax for defining encoding objects is given in the supplementary slides.

Detailed ECN - Encoding structures

- Describe the structure of an encoding.
- Implicitly generated structures are defined as being produced by all ASN.1 type definitions.
- Explicitly generated structures are similar to an ASN.1 type definition, and are freely written by the ECN specifier. Typically, the values carried by the fields of an implicitly generated structure will be mapped into the fields of an explicitly generated structure, and the values of other fields will be defined as determinants.
- Encoding objects both encode fields and perform transformations and identify procedures for determinants.

Detailed ECN – A (fairly!) simple example (1)

```
My-type ::= INTEGER (1..65536)
```

We want an encoding in which integer values 1 to 255 encode in a single octet, but values above that have a zero octet then two octets giving the value, with all zeros in the last two octets meaning 65536. (This is almost – but not quite! – the Lc field of ISO/IEC 7816-4. It is “not quite” because values 1 to 255 can also be encoded in the 3 octet form. That adds an encoder’s option that is not covered here.)

Detailed ECN – A (fairly!) simple example (2)

The implicitly generated structure is just:

```
#My-type ::= #INTEGER (1..65536)
```

We define the explicit encoding structure as:

```
#My-type-encoding ::= #ALTERNATIVE {
       one-octet       #INTEGER (1..255),
       three-octet     #CONCATENATION {
              determinant     #PAD,
              value           #INTEGER (256..65536) }
```

There are three further things to do: We need to specify that the values of `#My-type` map into the “one-octet” and “value” fields of `#My-type-encoding`; We need to specify that `#PAD` is all zeros; We need to specify that the alternative determination depends on the value of the first octet of the chosen alternative (zero or non-zero). All this is possible in ECN!

Detailed ECN - Coloring

- It is frequently the case that some integers in a protocol need one sort of encoding, and others a different one. Thus Lc is an `INTEGER`, but so are some fields that are encoded in a more “normal” fashion!
- It is necessary to distinguish between which integers are encoded one way and which are encoded another way. (That is, what encoding object to apply to one and what encoding object to apply to the other.) Another example is differences in optionality determination.
- This is normally handled by “coloring” – one is a “red” integer, and the other is a “blue” integer. This has to be identified. (ECN does not use the word “coloring”.)
- This is usually done by a `RENAMES` clause, changing the class name of one or more fields in a generated structure.
- Coloring is generally only needed for “generic” specification – see later.

Detailed ECN – Coloring example

```
Coloring-module {joint-iso-itu-t(2) -- etc. --}
ENCODING-DEFINITIONS ::=
 BEGIN
 IMPORTS ......
 EXPORTS ......
 RENAMES
   #OPTIONAL AS #My-Optional IN ALL;
   #SEQUENCE-OF AS #My-Sequence-of
      IN ALL EXCEPT #Group-Identity-Uplink, #Proprietary;
   #INTEGER AS #Mod256Int IN #Command-Lc-Length;
 -- etc
 END
```

Generic definitions

- It is (fairly) easy to define special encodings (adding determinants, specifying encodings of fields and constructions) for any single given ASN.1 type.
- It is generally more difficult to define a complete set of new encoding rules that can apply to any (perhaps not-yet-defined ASN.1 type).
- ECN addresses this, but for a specific protocol, it is often simpler to encode the given definitions, not to produce generic encoding rules.
- Generic definitions (and coloring) are probably going to be needed for SIMPLE-BER encoding in ISO/IEC 7816-4.

Parameterization is key to generic definitions

- It is possible to specify that the first action of an encoding object (particularly an encoding object for the #OPTIONAL class of procedures) is to do a replacement of the optional element. But the element can be anything. So we use parameterization.

- For example, adding a preceding presence bit for optionality:

We replace

```
        #Any-class #OPTIONAL
```

with

```
        #New-component {< #Any-class >} ::=
            #SEQUENCE { presence-bit    #BOOL,
                        component       #Any-class #My-Inner-Optional }
```

- The `#My-Inner-Optional` identifies the "presence-bit" as the optionality determinant for the "component" in the structure.
- This is advanced stuff and unfortunately requires a good understanding of ASN.1 parameterization!

How to go about using ASN.1 plus ECN (1)?

- The first task is to try to understand the protocol being worked on, and to determine its information content.
- That is never easy! Experience has shown that a domain expert working with an ECN expert is essential in producing an ASN.1+ECN specification.
- For ISO/IEC 7816-4, there are many parts of it that are already close to ASN.1 (use of the BER-TLV encodings). However, SIMPLE-TLV is ASN.1 BER-like, but will need ECN to define the encodings.
- For BER-TLV parts, the only real issue is to check carefully that no changes or restrictions have been placed on BER. However, the ability to include “00” and “FF” bytes at random before, between, and after TLV encodings is likely to be a problem. Do these contribute to the “L” part of an enclosing “TLV”? This may need a pre-filter.

How to go about using ASN.1 plus ECN (2)?

- As with most work on ECN, putting brain-cycles in can often produce simple and elegant (but not obvious) solutions. So, for example, defining the “T” part of a TLV encoding as having arbitrary padding octets of “00” or “FF” could be possible to solve the problem of those octets, but it would not address those that are inserted after the last TLV – more thought needed!

- There are also questions to be resolved on whether absence of an Lc field can be determined by the CLA or INS fields, or whether it has to be based on how many octets remain in the message. Only a domain expert can answer such questions, and the need for complicated ECN (or not!) depends very much on the answer.

- In general, understanding a specification is greatly eased (and speeded) by interaction with people involved in its development.

How to go about using ASN.1 plus ECN (3)?

- Once the protocol is fully understood, bog-standard ASN.1 can be produced for the information content of its messages.
- That is in principle easy! But there are always choices. Use of a `BOOLEAN` or an `INTEGER` or an `ENUMERATED` can often make a big difference to the readability of the ASN.1, and to the ease of producing ECN for it. This is an issue of the quality of the result.
- For ISO/IEC 7816-4, there are many parts of it that are already close to ASN.1 (use of the ASN.1 BER TLV-style encodings), and can be cast into ASN.1 notation very easily.
- For these parts, the only real issue is to check carefully that no changes or restrictions have been placed on BER

How to go about using ASN.1 plus ECN (4)?

- *To be pure or not to be pure, that is the question!*
- It is often the case that inclusion of “determinants”, or of additional “reserved values” (encoding aspects) in the ASN.1 specification itself can reduce or eliminate the need for an ECN specification.
- This is generally considered “dirty”, but is often pragmatically the best answer.
- An example of “dirt”: PER normally encodes an integer with a range of -128 to 127 as the hex values 00 to FF. But a protocol wants to encode it as a 2’s complement integer. ECN can be used to do this. But a “dirty” pragmatic solution is to define it as on the following slide …

How to go about using ASN.1 plus ECN (5)?

- We define:

```
MyInt ::= CHOICE {
          positive      INTEGER (0..127),
          negative      INTEGER (-128..-1)}
```

and encode with the standard PER encoding rules.

- You need to understand PER to know why this works!

- But it gives a two’s complement encoding without using ECN.

- It is VERY VERY dirty! But it allows ASN.1 tools to be used. At least one International Standard has chosen to take this approach.

How to go about using ASN.1 plus ECN (6)?

- The above illustrates that to efficiently deploy ASN.1 with ECN, you need:
  - A domain expert that fully understands the protocol being defined
  - An ASN.1 expert that both understands the ASN.1 notation AND its encoding rules
  - An ECN expert that is conversant with the functionality and (less important) syntax of ECN.
- Access to an ECN tool to check for incorrect syntax is also desirable!

Reams and reams of ECN?

NO!

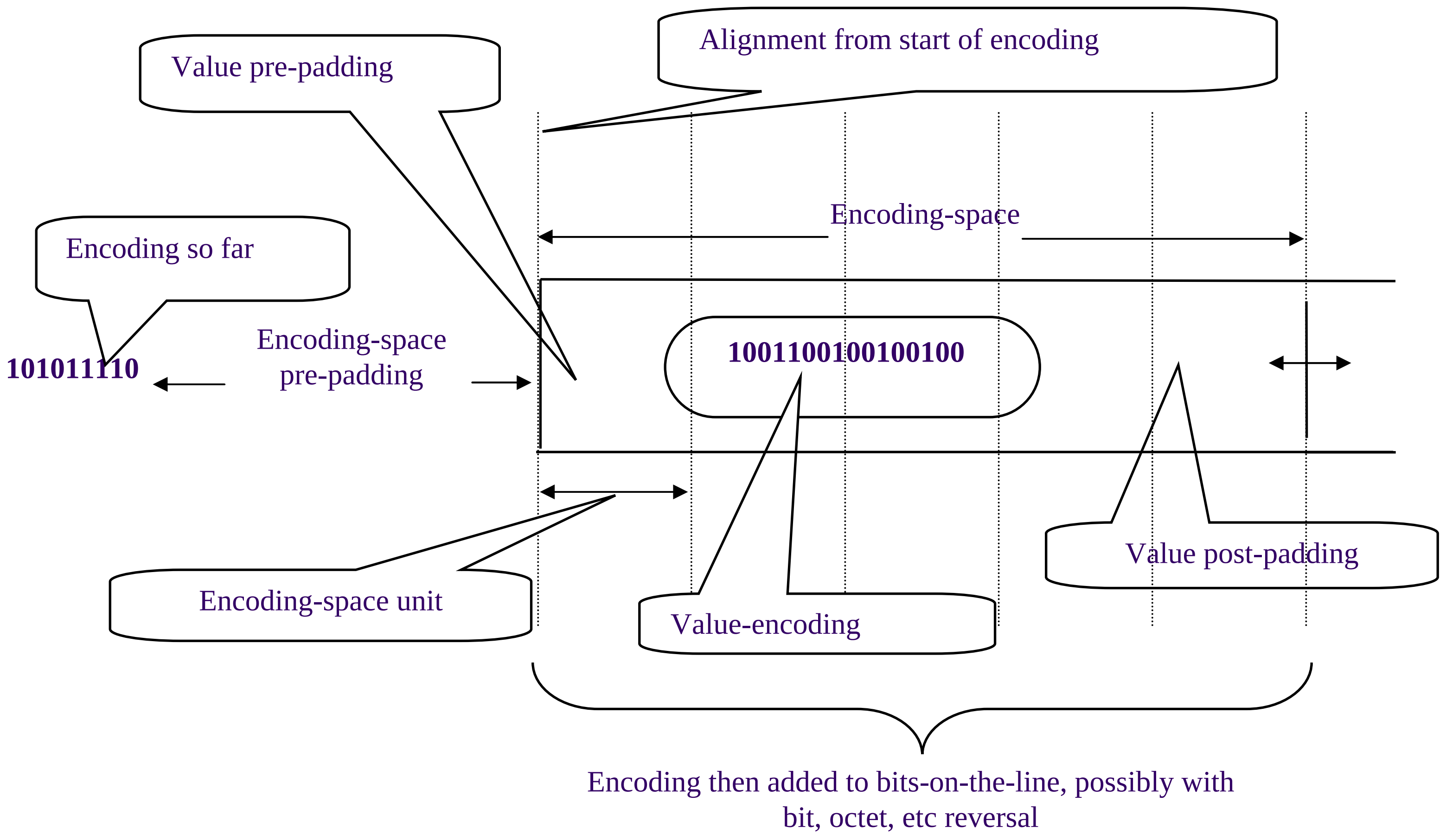
For most protocols only a small amount of ECN is needed – the bulk is almost always standard ASN.1.

There are often only a few special encodings needed, and they tend to be repeatedly used.

Using ASN.1 for a legacy protocol

- Write the ASN.1 - no auxiliary fields, no encoding influence! Just clearly identify the information content.
- Then get dirty – that is, pragmatic!
- *To be pure or not to be pure, that is the question - again!*
- You can often get the encoding you want by slightly contorting the ASN.1 rather than using ECN. It is important to get the balance right.
- Apply "coloring" if needed.
- Define mappings of structures
- Define any needed encoding objects.
- Applying the encodings (this is easy boiler-plate stuff, but ECN can be used to combine BER and PER encodings, alongside specially-defined encodings).

Model for defining simple bit-field encoding objects


Value pre-padding
Alignment from start of encoding
Encoding-space
Encoding so far
101011110
Encoding-space pre-padding
10011001100100100
Encoding-space unit
Value-encoding
Value post-padding
Encoding then added to bits-on-the-line, possibly with bit, octet, etc reversal

Likely not needed for ISO/IEC 7816-4!

Example #BOOL encoding object

```
my-bool-encoding #BOOL ::=
                {ENCODING-SPACE
                        SIZE 1
                        MULTIPLE OF bit
                 TRUE-PATTERN  bits:'0'B
                 FALSE-PATTERN bits:'1'B }

#BOOL2 ::= #BOOL
my-bool2-encoding #BOOL2 ::=
                {ENCODING-SPACE
                        ALIGNED TO NEXT octet
                        SIZE 1
                        MULTIPLE OF octet
                 VALUE
                        TRUE-PATTERN octets:'00'H
                        FALSE-PATTERN other }
```

Defining encoding objects (1)

- Use “defined syntax” (see ITU-T Rec. X.692 | ISO/IEC 8825-3, Annex A and previous slide) – this is pretty simple for specialised encodings of bit-fields.

- Use an existing encoding object set – utterly simple! When it works!:

```
per-int-encoding #INTEGER ::= {ENCODE WITH PER-BASIC-UNALIGNED}
per-structure1-encoding #Structure1 ::= {ENCODE WITH DER}
```

- Use value mappings – the advanced stuff! True ECN!:

```
 encoding-for-old-class #Old-class ::=
               { USE #Replacement-structure
                 MAPPING
                 ......
                 WITH {ENCODED WITH PER-BASIC-UNALIGNED}
                   -- The above line is defining encoding
                   -- for the class #Replacement-structure.
                   -- The whole definition defines the encoding
                   -- for #Old-class --}
```

Defining encoding objects (2)

- Encoding a structure component by component:

```
#My-Sequence ::= #SEQUENCE {
                a                 #Structure1,
                inserted          #BOOLEAN,
                b                 #INTEGER #OPTIONAL,
                c                 #OCTET-STRING }
```

- An encoding object could be:

```
My-sequence-encoding #My-Sequence ::= {
             ENCODE STRUCTURE {
               b OPTIONAL-ENCODING
                     { PRESENCE DETERMINED BY field-to-be-set
                       USING inserted }}
             WITH PER-BASIC-UNALIGNED }
```

Mapping values (1)

- **These slides illustrate some of the functionality of ECN. Most functionality will probably NOT be needed for ISO/IEC 7816-4!**
- **Mapping single values:**

```
encoding-for-old-class #UTF8String ::=
             { USE #INTEGER
               MAPPING VALUES {
                  "0" TO 0, "1" TO 1, "2" TO 2, "3" TO 3,
                  "4" TO 4, "5" TO 5, "6" TO 6, "7" TO 7,
                  "8" TO 8, "9" TO 9, "*" TO 10, "#" TO 11 }
               WITH encoding-of-integer }
```

- **Mapping by matching fields (usually when determinants are added):**

```
encoding-for-old-class #Old-class ::=
             { USE #Replacement-class
               MAPPING FIELDS
               WITH encoding-of-replacement-class }
```

- **Mapping using #TRANSFORM objects:**

```
encoding-for-old-class #INTEGER1 ::=
             { USE #INTEGER2
               MAPPING TRANSFORMS { {INT-TO-INT divide:2} }
               WITH encoding-of-integer2 }
```

Mapping values (2)

- **Mapping by abstract value ordering:**

```
encoding-for-old-class #Old-class ::=
                { USE #Replacement-class
                  MAPPING ORDERED VALUES
                  WITH encoding-of-replacement-class }
```

- **Mapping by distributing values:**

```
encoding-for-old-class #Old-class ::=
                { USE #Replacement-class
                  MAPPING DISTRIBUTION {
                          0 .. 9 TO field1,
                          10 .. 99 TO field2,
                          REMAINDER TO field3}
                  WITH encoding-of-replacement }
```

- **Mapping integer values into bits and Huffman encodings**

Example of a complete ECN module

```
EncodingLinkModule LINK-DEFINITIONS ::= BEGIN
IMPORTS
   Specialized-encodings FROM EncodingDefinitionModule
   #TopLevelType FROM ASN1Module;
ENCODE #TopLevelType WITH Specialized-encodings COMPLETED BY PER-BASIC-UNALIGNED
END
EncodingDefinitionModule ENCODING-DEFINITIONS ::= BEGIN
IMPORTS #BooleanType, #PositiveInteger, #NegativeInteger, #BitStringType,
   #OctetStringType, #CharacterStringType, #NormallySmallValues,
   #SparseEvenlyDistributedValueSet, #IntegerRightAligned, #ChoiceBetweenIntegers
   FROM Example-ASN1-Module;
Specialized-encodings #ENCODINGS ::= { -- encoding object set
   bool-encodingobject | positiveInteger-encodingobject |
   negativeInteger-encodingobject | bitString-encodingobject -- ... -- }
sparseEvenlyDistributedValueSet-encodingobject
   #SparseEvenlyDistributedValueSet ::= {
       USE #INT (0..7) MAPPING ORDERED VALUES WITH integer-Encoding }
-- ...
END
```

ECN and legacy protocols

- Context:
  - Protocol messages have originally been specified without ASN.1, e.g., as bits diagrams or tables of parameters or in simple English.
- Problem:
  - Need for ASN.1 to express logical message contents, e.g., for test purposes, for clarity, for API specification, or to enable ASN.1 tools to be used for implementation.
- Solution:
  - ECN can be used to fill the gap between message content definitions and message encoding
- Use of ASN.1 plus ECN produces:
  - Separation of abstract message content and auxiliary information
  - Clear specification of presence and length determinants, and avoidance of ambiguity
  - But …. A complex message encoding => complex ECN! Unavoidable!

ECN and specialization

- Context:
  - Protocol messages can easily be defined using ASN.1
  - Standard ASN.1 encoding rules (e.g., UNALIGNED PER, or BER) can be used to provide encoding for messages in almost all cases.
- Problem:
  - Standard encoding rules do not provide all the needed encodings, or a mix of PER and BER is needed.
- Solution:
  - Use standard encoding rules for the majority of encodings
  - Use ECN for specialized encodings, and to identify when BER is to be used and when PER is to be used

Application domains

- UMTS (RAN, GERAN)
- GSM, GPRS
- Bluetooth
- HIPERLAN
- TETRA (Terrestrial Trunked Radio)
- SS7 ISUP (ISDN User Part)
- INAP (Intelligent Network Application Protocol)
- SCCP (Signalling Connection Control Part)
- Can we add ISO/IEC 7816-4?

Bluetooth™

Some free ECN tools

- OSS Nokalva syntax and semantics checker (ASN.1, ECN):
  - http://www.oss.com/products/checksyntax.html
- France Telecom R&D ECN syntax checker and pretty-printer:
  - http://asn1.elibel.tm.fr/ecnp
- List of other tools:
  - http://asn1.elibel.tm.fr/links/#ecn

ISO/IEC 7816-4 First example (1)

- Let us take the “simple” case of the encoding of an integer for a length of 1 to 256 (or 1 to 65536) with zero representing 256 (or 65536).
- How to do that? Brain-CPU-cycles whir!
- Maybe a `CHOICE`, with field-mappings? That would probably work.
- Maybe mapping to bits? That looks simpler:

```
special-integer-encoding  #Lc  ::= {
        USE #BITS (SIZE (8))
          MAPPING TO BITS {
              256 TO '00'H,
              1 .. 255 TO '01'H .. 'FF'H }
        WITH PER-BASIC-UNALIGNED }
```

- But there is another option!

ISO/IEC 7816-4 First example (2)

- ISO/IEC 7816-4 is actually specifying a well-known mathematical transform (although it does not say so!)

- Try this:

```
special-integer-encoding  #Lc ::= {
        USE #INT (0 .. 255)
        TRANSFORM {INT-TO-INT modulo:256} }
```

  This says that the Lc value is to be encoded in an integer field capable of carrying values in the range 0 to 255, with the value encoded being the original value taken modulo 256.

- Which do you prefer? Mapping to bits or the modulo transformation? This is purely a matter of style! They both work.

ISO/IEC 7816-4 Second example

- If extended encodings are agreed, then 7816-4 allows an Lc value (which is in the range 1 to 65536) to be encoded as EITHER a single octet OR as three octets if it is in the range 1-255 (as an encoder's option).

- This needs to be specified as an encoder's option in ECN. We get an ECN specification for an encoding object for Lc of:

```
encoding-of-Lc  #Lc ::= {
          OPTIONS {one-octet-encoding, three-octet-encoding}
          WITH {DETERMINED BY use-handle
                  HANDLE "first-octet" }
  -- The last line tells a decoder how to distinguish the two cases
```

- The one-octet-encoding and three-octet-encoding need to be defined as outlined earlier. They need to declare a handle of "first-octet", which in one case will be any non-zero value, and in the other case a zero value.

ISO/IEC 7816-4 Third example (1)

- This example is pure ASN.1 – no ECN is involved. The majority of new text (if it is agreed to use ASN.1) will be like this. (Forget the esoteric stuff!)

- Tables 2 and 3 in the current draft (see 5.1.1) would be replaced or supplemented (probably with added comments) by:

```
CLA ::= CHOICE {
        inter-industry-class   Inter-Industry-CLA,
        other-class            UndefinedClass }

UndefinedClass ::= BIT STRING (SIZE (7))
                   -- Values and meaning not standardized

Inter-Industry-CLA ::= CHOICE {
       type-four  Type-Four-CLA,
       type-sixteen     Type-Sixteen-CLA }
```

(cont)

ISO/IEC 7816-4 Third example (2)

```
Type-Four-CLA ::= CHOICE {
        current       Type-Four-2002,
        reserved      BIT STRING (SIZE (5)) }

Type-Four-2002 ::= SEQUENCE {
        last-or-only-of-chain       BOOLEAN,
        secure-messaging            ENUMERATED {
                      no-SM-or-no-indication,
                      proprietary-SM-format,
                      SM-not-authenticated,
                      SM-authenticated },
        logical-channel-number      INTEGER (0..3) }

Type-Sixteen-CLA ::= SEQUENCE {
        no-SM-or-no-indication      BOOLEAN,
        last-or-only-of-chain       BOOLEAN,
        logical-channel-number      INTEGER (0..15) }
```

Bottom line stuff

- Does SC17 want to proceed with the inclusion of an ASN.1 (with ECN where necessary) specification?
- As a replacement of current text, or as a separate normative annex?
- Is there a "domain expert" prepared to work with ASN.1/ECN experts on this?
- What is the time-scale for production of the ASN.1+ECN specification?
- Decision time! (Hope I manage to get to this slide!)

ASN.1 and XML

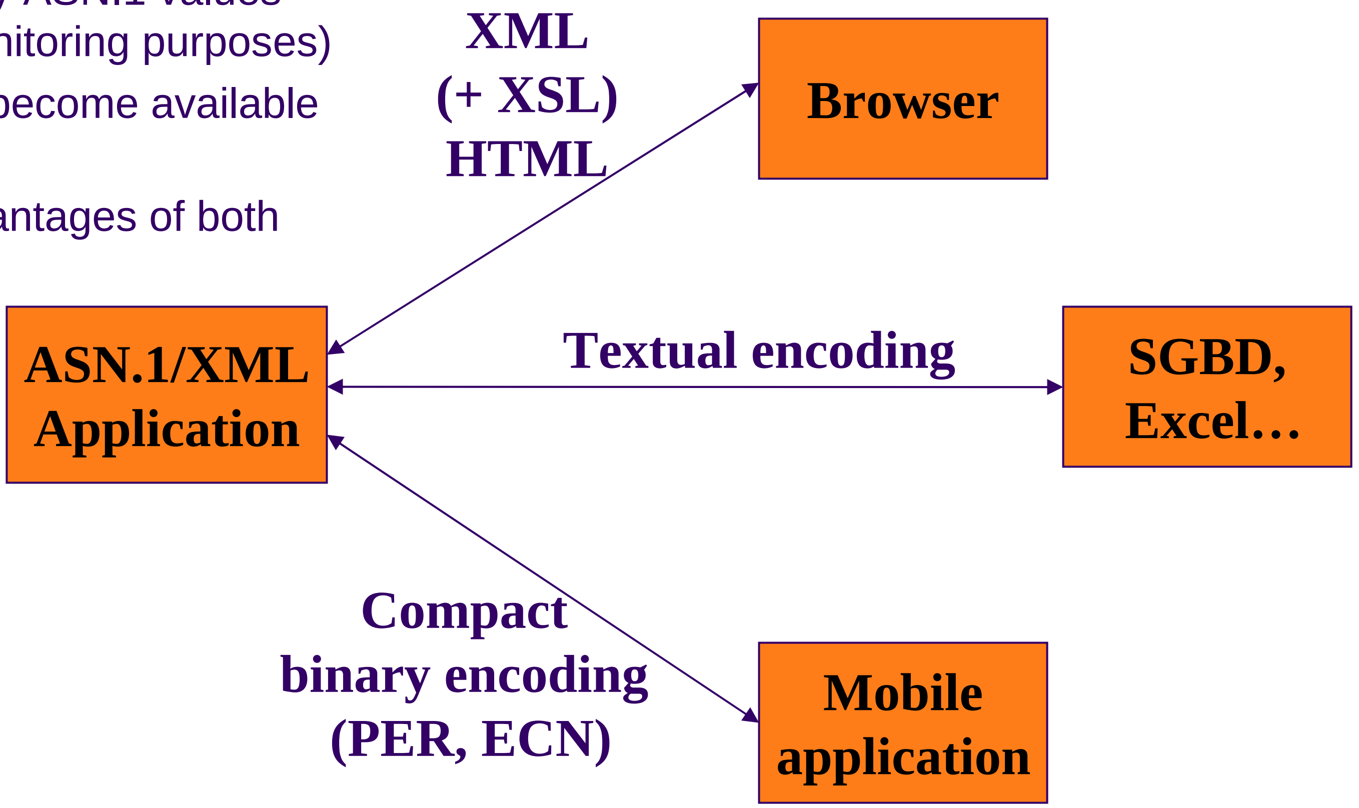
How each can benefit the other

This may not be directly relevant to ISO/IEC 7816-4, but is likely to be of interest.

(Time permitting, the following slides will be presented!)

Benefits of a linkage between ASN.1 and XML

- Allows a closer integration of XML schema specification languages and traditional tools for protocol implementation
- Browser support to display ASN.1 values (useful for testing and monitoring purposes)
- Very compact encodings become available for XML
- Exploits the very real advantages of both



XER encoding rules

- *XML Encoding Rules*, ITU-T Rec. X.693 | ISO/IEC 8825-4
- Defines an XML syntax (with a canonical variant) for ASN.1 values
- Offers the advantages of both techniques (win-win) :
  - Display in a browser data encoded in BER, DER or PER
  - ASN.1 applications can now generate XML data
  - Compact (PER, ECN) and/or canonical (DER) transmission of data:
    - XML has no efficient and mature binary encoding (Binary XML is not a good answer)
    - PER reduces the size by a factor 3 to 4 over a Binary XML encoding
    - XML has no easy-to-implement canonical encoding for digital signatures
- Encodings associated with ASN.1 are now binary and textual
- The ASN.1 module is the unique schema to validate data
- Used by OASIS XCBF (XML Common Biometric Format),
  OASIS UBL (Universal Business Language),
  ITU-T Rec. F.515 (Unified Directory Specification)…
- More information: `http://asn1.elibel.tm.fr/xml`

XER-encoding example (1)

```
ChildInformation ::= SEQUENCE {
  name          AnyName,
  dateOfBirth   INTEGER (1..MAX) } -- yyyymmdd
AnyName ::= SEQUENCE {
   givenName    VisibleString,
   initial      VisibleString (SIZE (1))  OPTIONAL,
   familyName   VisibleString }

<ChildInformation>
   <name>
      <givenName> Lee </givenName>
      <familyName> Owen </familyName>
   </name>
   <dateOfBirth> 19501003 </dateOfBirth>
</ChildInformation>
```

XER-encoding example (2) – Yup, it’s ISO/IEC 7816-4!

```
<Type-Four-CLA>
      <current>
        <last-or-only-of-chain>
            <true/>
         </last-or-only-of-chain>
         <secure-messaging>
              <no-SM-or-no-indication/>
         </secure-messaging>
         <logical-channel-number>
              5
          </logical-channel-number>
       </current>
   </Type-Four-CLA>
```

➡ Would this be interesting or not???

France Télécom R&D

XER encoding instructions

- Ability to tune the generated XML encoding (ITU-T Rec. X.693/Amd. 1):

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

or

```
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
```

```
<employee id="51">
  <recruited>19710917</recruited>
  <salaries>121.96 76</salaries>
</employee>
```

So... ASN.1 today offers....

- A simple, mature, well-known (among the best people!) notation for protocol (abstract syntax == API) definition.
- V1 with V2, etc, interworking, with support for vendor-specific additions in a controlled way (the ASN.1 Information Object Class and related concepts). *
- Support for "legacy" protocols
- Ability to generate XML marked-up data.
- Very strong links with SDL and TTCN. *
- Mappings to C, C++, Java, etc. *
- Good support via mature canonical encodings, for security concepts. *
- Wide use in a large range of industries: *
  - Keeping the lights burning!
  - Portable phones – we need them!
  - Keeps aircraft from crashing!
  - Your impending marriage could suffer!

* = Not really discussed

Questions? And perhaps some answers? Are you interested?

Example of an EDM

```
Example-EDM {joint-iso-itu-t(2) -- to be supplied --} ENCODING-DEFINITIONS ::=
BEGIN

RENAMES ......;
IMPORTS
  #MyType                                -- Implicit encoding class for an ASN.1
FROM Example-ASN1;                       -- type, #MyType ::= #INT (0..7)

MyEncodings #ENCODINGS ::= { myType-encoding }
      -- Encoding object set with one object.

myType-encoding #MyType ::= {            -- Encoding object which produces
  ENCODING {                             -- the same encoding as PER would.
      ENCODING-SPACE
            SIZE          3
                  MULTIPLE OF  bit
            ENCODING positive-int } }
END
```

France Télécom R&D

The Encoding link module (ELM)

- A (unique) link module specifies how encoding objects are applied to ASN.1 types

```
Example-ELM LINK-DEFINITIONS ::=
BEGIN

IMPORTS
  #MyType             -- Implicit encoding class for an ASN.1 type
FROM Example-ASN1
  MyEncodings         -- Encoding object set containing
                      -- an encoding object for #MyType
FROM Example-EDM;

ENCODE #MyType         -- MyEncodings is used for #MyType
  WITH                MyEncodings
  COMPLETED BY        PER-BASIC-UNALIGNED
                      -- The rest is encoded using PER

END
```

Applying encodings

- Define encoding objects for encoding classes.
- Form an encoding object set (only one object for each class).
- “Apply” (in the ELM) an encoding object set to an ASN.1 type.
- Not quite true - to a structure generated from the ASN.1 type.

Syntax to define encoding objects

- Encoding objects are to encoding classes as ASN.1 values are to ASN.1 types:

```
my-int INTEGER ::= ......

my-boolean-encoding #BOOLEAN ::= ....…
```

- Governor concept carries over.

France Télécom R&D

Encoding structures

- Describe the structure of an encoding.
- Implicitly generated structures derived from ASN.1 type definitions.
- Explicitly generated structures transform these into the actual fields needed in the final encoding.
- Encoding objects both encode fields and perform transformations.

Encoding structure features

- Encoding structures are simpler than ASN.1 types, but still have primitive fields and constructors.
- Primitive fields could be just bit strings, but easy mapping from ASN.1 requires richer primitive fields.
- Constructors have different names from ASN.1 because they operate on bit-fields, not on abstract values.

Encoding structure definition (1)

```
#Structure1 ::=  #CONCATENATION {
                field1  #INT,
                -- Comment can be embedded, as in ASN.1
                field2  #BOOL,
                field3  #NUL,
                field4  #PAD,
                field5  #ALTERNATIVES {
                         alt1  #CHARS,
                         alt2  #BITS },
                field6  #REPETITION { #OCTETS },
                field7  #Structure2 }

#Structure2 ::= #CONCATENATION {
                field1  #OBJECT-IDENTIFIER,
                field2  #RELATIVE-OID,
                field3  #OPEN-TYPE,
                field5  #REAL,
                field6  #Structure3 }
```

Encoding structure definition (2)

```
#Structure3 ::= #TAG #CONCATENATION {
                field1  #BOOL,
                field2  #INT #OPTIONAL,
                field3  #TAG #Structure4,
                field4  #INT (0..7),
                field5  #OCTETS (SIZE (6)) }

#Structure4 ::= #CONCATENATION {
                field1 #INT,
                #EXTENSIONS {
                        #VERSION-BRACKET {
                            field1    #INT,
                            field2    #BOOL } ,
                        #VERSION-BRACKET {
                            field1    #CHARS,
                            field2    #OCTETS } } ,
                field2 #BOOL,
                field3 #Structure5 }
```

Encoding structure definition (3)

```
#Structure5 ::=  #SEQUENCE {
                field1  #INTEGER,
                -- Looks rather like #Structure1, yes?
                -- But the names are more like ASN.1
                field2  #BOOLEAN,
                field3  #NULL,
                field5  #CHOICE {
                         alt1  #UTF8String,
                         alt2  #BIT-STRING },
                field6  #SEQUENCE-OF {
                        #OCTET-STRING } }
```

Built-in synonyms for classes

```
#SEQUENCE          ::= #CONCATENATION
#INTEGER           ::= #INT
#BOOLEAN           ::= #BOOL
#NULL              ::= #NUL
#CHOICE            ::= #ALTERNATIVES
#UTF8String        ::= #CHARS
#BIT-STRING        ::= #BITS
#SEQUENCE-OF       ::= #REPETITION
#OCTET-STRING      ::= #OCTETS
```

Note the assignment of synonyms for constructor classes.

Implicitly generated encoding structures

- ASN.1 types => Implicit structure.
- Value references resolved.
- Named bits ignored.
- Names of enumerations lost.
- Parameterization resolved.
- Components of and Selection types expanded.
- etc.

ASN.1 example

```
Color ::= [1] INTEGER {red(10), orange(20), yellow(30),
                       green(40), blue(50), indigo(60), violet(70)}

Version ::= [2] BIT STRING {version1(0), version2(2)}

My-type ::= SEQUENCE {
           field1      Version,
           field2      Color,
           field3      CHOICE {
                alt1    [3] INTEGER (0..15),
                alt2    [4] INTEGER (16..MAX) },
           field4      SEQUENCE OF
                          OCTET STRING (SIZE (6)) DEFAULT ‘000000'B}
```

Corresponding implicitly generated structure

```
#Color ::= #TAG #INTEGER

#Version ::= #TAG #BIT-STRING

#My-type ::= #SEQUENCE {
            field1      #Version,
            field2      #Color,
            field3      #CHOICE {
                alt1            #TAG #INTEGER (0..15),
                alt2            #TAG #INTEGER (16..MAX) },
            field4      #SEQUENCE-OF {
                                #OCTET-STRING (SIZE (6)) #OPTIONAL }
```

France Télécom R&D

Explicitly generated encoding structures

- Sometimes called “coloring” an implicitly generated structure.
- Replacing some classes with synonyms to identify places needing different encodings.

```
EDM-coloring-module {joint-iso-itu-t(2) -- to be supplied --}
   ENCODING-DEFINITIONS ::=
BEGIN
EXPORTS My-encodings; -- An object set containing all specialized encodings
RENAMES
   #OPTIONAL AS #My-Optional IN ALL
   #SEQUENCE-OF AS #My-Sequence-of
      IN ALL EXCEPT #Group-Identity-Uplink, #Proprietary;
 -- etc
 END
```

Defining encoding objects – another example

- Differential encode/decode:

```
object-for-tail-end-additions  #PAD ::=
                { ENCODE-DECODE
                      -- Specifies an empty encoding for ENCODE
                     {ENCODING-SPACE
                            SIZE fixed-to-max
                            PAD-PATTERN bits:''B}
                 DECODE AS IF
                     {SIZE variable-with-determinant
                      DETERMINED BY container}}
```

- User-defined functions:

```
encoding-of-funny-type #Funny-type ::=
NON-ECN-BEGIN {joint-iso-itu-t -- etc --} -- specifies the notation used
      -- Pseudo-code for encoding values of #Funny-type
      ......
      -- Pseudo-code for decoding values of #Funny-type
      ......
NON-ECN-END
```

ECN & Bluetooth

- ECN has been used to (re-)define the Bluetooth “Service Discovery Protocol”
- Provides greater precision, and a formal definition
- Rules for extensibility (version-1 actions on receipt of version-2 data) are more explicit
- Can use ASN.1 (and hence SDL, etc) tools for Bluetooth implementations
- There was some difficulty in determining what "extensions" were envisaged for Bluetooth version 2, and what extensions equipment suppliers could make in their version 1 specifications.
- More at `http://asn1.elibel.tm.fr/en/ecn/bluetooth/`

Bluetooth Service Discovery Protocol (ASN.1 with ECN) – a snippet

```
bluetooth-tag-encoding #TAG ::=
        {ENCODING SPACE SIZE 8
         EXHIBITS HANDLE “Bluetooth tag” AT {0..7} }

more-bit-delimited-repetition{< REFERENCE:more-bit > #REPETITION ::=
        {REPETITION-ENCODING
                        {REPETITION-SPACE
                               SIZE variable-with-determinant
                               MULTIPLE OF octet
                               DETERMINED BY flag-to-be-set
                               USING more-bit } }
```

France Télécom R&D

Additional slides

(HIPERLAN example)

ECN and legacy protocols

- Context:
  - Protocol messages have originally been specified without ASN.1, e.g., as octet tables
- Problem:
  - Need for ASN.1 to express logical message contents, e.g., for test purposes
- Solution:
  - ECN can be used to fill the gap between message content definitions and message encoding
- Forces:
  - Separation of abstract message contents and auxiliary information
  - Specification of presence and length determinants
  - Complex message encoding => complex ECN

Hiperlan example

- Purpose of the example:
  - Show how messages that were originally specified using tables can be specified using ASN.1 and ECN
- Real-life Hiperlan protocol:
  - Existing ASN.1 definitions
  - Existing tables for message encoding
  - RLC-RADIO-HANDOVER-COMPLETE-ARG used as an example message

- **URL**

Hiperlan - Message table form

Padding bits

Octet-aligned fields

Length determinant field for several octet fields

Presence determinants

	8	7	6	5	4	3	2	1
Octet 1	Defined in DLC TS		MSB Sequence number					
Octet 2	Sequence number				MSB EXTENSION-TYPE			Future use
Octet 3	MSB RLC LCH PDU type							
Octet 4	Future use				mac-id-old			
Octet 5	mac-id-old				ap-id-old			
Octet 6	ap-id-old						net-id-old	
Octet 7	net-id-old							
Octet 8	mac-id-new							
Octet 9	cl-id							
Octet 10	Duc-ext-ind	cl-conn-attr-length(L)					# of DUC:s	
Octet 11	# of DUC:s(N)		Future use					
Octet12	(DUC1) direction		dlcc-id					
	cl-connn-attr							
Octet(12+L)								
Octet Y	(DUC1-FW) allocation-type			Future use	cyclic-prefix	fec-used	ec-mode	
Octet ...	(DUC1-FW) arq-nr-of-retr				Future use	(DUC1-FW) arq-window-size		
	(DUC1-FW) fec				Future Use			
Octet ...	sch-per-nb-frames				lch-per-nb-frames			
Octet ...	nb-of-sch	phy-mode-sch			phy-mode-lch			
Octet ...	nb-of-lch							
Octet ...	min-nb-of-lch							
Octet X	(DUC1-BW) allocation-type			Future use	cyclic-prefix	fec-used	ec-mode	
Octet ...	(DUC1-BW) arq-nr-of-retr				Future use	(DUC1-BW) arq-window-size		
	(DUC1-BW) fec				Future Use			
Octet ...	sch-per-nb-frames				lch-per-nb-frames			
Octet ...	nb-of-sch	phy-mode-sch			phy-mode-lch			
Octet ...	nb-of-lch							
Octet ...	min-nb-of-lch							
Octet ...	Not used							
Octet ...								
Octet 51								

Hiperlan - ASN.1

- The ASN.1 definition for the `RLC-RADIO-HANDOVER-COMPLETE-ARG` message is simple
- Some determinant fields are visible in ASN.1:
  - `cl-conn-attr-length`, common length for all `cl-conn-attr fields`
  - `fec-used` presence determinant
- Some reserved values:
  - `ALLOCATION-TYPE`
- Otherwise the definitions are plain old ASN.1

⇒ ASN.1 definitions can be used as is after determinant fields have been removed

Hiperlan - ASN.1

```
RLC-RADIO-HANDOVER-COMPLETE-ARG ::= SEQUENCE {
   mac-id-old                    MAC-ID,
   ap-id-old                     AP-ID,
   net-id-old                    NET-ID,
   mac-id-new                    MAC-ID,
   cl-id                         CL-ID,
   duc-ext-ind                   DUC-EXT-IND,
   duc-descr-list                DUC-DESCR-LIST}
```

- There are additional requirements:
  - Every `DUC-DESCR.cl-conn-attr` in `DUC-DESCR-LIST` is of the same length.
  - This cannot be simply expressed formally in ASN.1 but will be enforced in the ECN specification.

```
DUC-DESCR-LIST ::= SEQUENCE (SIZE(1..cMAX-DESCR-LIST)) OF DUC-DESCR
```

Hiperlan - ECN

- Encoding structure
  - Insertion of padding bits
    - `aux-future-use`
    - `aux-pad`
  - Binding of determinant and determined fields
    - `cl-conn-attr-length`
    - `fec-used`
  - Space for reserved values
    - `allocation-type`
    - `coder-type`
    - `interleaver-type`

Encoding structure for the message

- The encoding structure for `RLC-RADIO-HANDOVER-COMPLETE-ARG` has two additional fields:
  - `aux-future-use` for the reserved bits,
  - `aux-cl-conn-attr-length` for length determinant for `duc-desc-list.*.cl-conn-attr`

```
#RLC-RADIO-HANDOVER-COMPLETE-ARG-struct ::= #CONCATENATION {
  aux-future-use              #PAD,               -- ** Inserted
  mac-id-old                  #MAC-ID,
  ap-id-old                   #AP-ID,
  net-id-old                  #NET-ID,
  mac-id-new                  #MAC-ID,
  cl-id                       #CL-ID,
  duc-ext-ind                 #DUC-EXT-IND,
  aux-cl-conn-attr-length     #INT(0..31),        -- ** Inserted
  duc-descr-list              #DUC-DESCR-LIST }
```

Encoding object for the message

- The encoding object for `#RLC-RADIO-HANDOVER-COMPLETE-ARG`
  - maps fields to the fields of the encoding structure;
  - specifies how padding is encoded;
  - links the determinant field to the determined fields.

```
rlc-radio-handover-complete-arg-encoding #RLC-RADIO-HANDOVER-COMPLETE-ARG ::= {
  USE         #RLC-RADIO-HANDOVER-COMPLETE-ARG-struct
  MAPPING     FIELDS
  WITH { ENCODE STRUCTURE {
      -- Components
      aux-future-use      reserved-bits-encoding{< 4 >},
      duc-descr-list      duc-descr-list-encoding{< aux-cl-conn-attr-length >}
      -- Structure
      STRUCTURED WITH per-sequence-encoding }}}
```

France Télécom R&D

Encoding object for one message field

- Encoding of the `DUC-DESCR-LIST`

```
duc-descr-list-encoding{< REFERENCE : aux-cl-conn-attr-length >}
  #DUC-DESCR-LIST ::= {
  ENCODE STRUCTURE {
        -- Components
        duc-descr-encoding{< aux-cl-conn-attr-length >}

        -- Structure
        STRUCTURED WITH per-sequence-of-encoding}}
```

The length determinant for a sub-field is passed as an argument.

PER is used to construct the rest of the list encoding

Encoding object for list element

```
duc-descr-encoding{< REFERENCE : aux-cl-conn-attr-length >} #DUC-
   DESCR ::= {
   ENCODE STRUCTURE {
        -- Components
        cl-conn-attr
                cl-conn-attr-encoding{< aux-cl-conn-attr-length >},
        forward-descr      USE-SET OPTIONAL-ENCODING
                -- simplex-forward, duplex, duplex-symetric
                is-present-if{< direction, {0|1|3} >},
        backward-descr     USE-SET OPTIONAL-ENCODING
                -- simplex-backward, duplex
                is-present-if{< direction, {1|2} > }
        -- Structure
        STRUCTURED WITH    octet-aligned-sequence-encoding}
WITH-PER-BASIC-UNALIGNED}
```

Length determinant is passed further down

Conditional fields

List elements are octet-aligned

Encoding object for a field in the list

- Finally the length determinant is passed to the encoding object that uses it as a length determinant for an octet string:

```
cl-conn-attr-encoding{< REFERENCE : aux-cl-conn-attr-length >}
   #CL-CONN-ATTR ::= {
   REPETITION-ENCODING {
          REPETITION-SPACE
                    DETERMINED BY field-to-be-used
                    USING aux-cl-conn-attr-length }}
```

Spare values

- Spare values can be expressed by reserving more encoding space for fields:

```
allocation-type-encoding    #ALLOCATION-TYPE ::=
       fixed-length-int-encoding{< 3 >}
```

- Parameterized encoding object for fixed length integer fields
  - Two's complement, big-endian, size is `nbits`

```
fixed-length-int-encoding{< #CONDITIONAL-INT.&encoding-space-size:
  nbits >} #INT ::= {
  ENCODING { ENCODING-SPACE SIZE   nbits }}
```

Collection of encodings

- Encoding definitions are collected as an encoding object set:

```
Hiperlan-Encodings #ENCODINGS ::= {
  rlc-radio-handover-complete-arg-encoding   |
  duc-direction-descr-encoding               |
  allocation-type-encoding                   |
  arq-data-encoding                          |
  fec-encoding                               |
  fca-descr-encoding }
```

Hiperlan - ELM

- Encodings are applied to top-level types in the ASN.1 module:

```
Hiperlan-ELM LINK-DEFINITIONS ::=
BEGIN

IMPORTS
  #RLC-RADIO-HANDOVER-COMPLETE-ARG FROM Hiperlan-ASN1
  Hiperlan-Encodings FROM Hiperlan-EDM;

ENCODE #RLC-RADIO-HANDOVER-COMPLETE-ARG
  WITH            Hiperlan-Encodings
  COMPLETED BY    PER-BASIC-UNALIGNED

END
```

Hiperlan example summary

- Application of ASN.1 + ECN for Hiperlan is straightforward

- ASN.1 definitions shall contain only application-specific definitions
- Encoding structures contain also auxiliary fields like length and presence determinants
- Encoding objects
  - specify relations between determinant fields and determined fields
  - specify special encoding (octet-alignment, padding, spare bits)
- The encoding link module applies the encoding objects to the ASN.1 types

ECN and specialization

- Context:
  - Protocol messages are defined using ASN.1
  - Standard ASN.1 encoding rules (e.g., PER) are used to provide encoding for messages
- Problem:
  - Standard encoding rules do not provide all the needed properties for encoding
- Solution:
  - Use standard encoding rules for the majority of encodings
  - Use ECN to specialize encoding for wanted properties
- Forces:
  - A kind of specialization vs. a generic property

Specialization of `CHOICE` *index encoding (1)*

- Context:
  - There is a top-level message container type which encapsulates specific messages and provides identification for them

```
Messages ::= CHOICE {
      a       MessageA,
      b       MessageB,
      c       MessageC }
```

- Problem:
  - New messages are wanted to be added in the container.
  - Encoding for the new messages should be similar to the old messages, i.e., no extension container is needed.
  - The number of new messages is not limited
- Solution:
  - Encode `CHOICE` index using a Huffman-like encoding

Specialization of CHOICE *index encoding (2)*

- The following encoding object specifies that the encoding structure for the Messages type consists of
  - an `aux-messageId` field, which is used as a message determinant
  - a `message` field, which contains the selected message

```
messages-encoding #Messages ::= {
    REPLACE     STRUCTURE
          WITH          #Messages-struct
          ENCODED BY  messages-struct-encoding }

#Messages-struct{< #OriginalMessages >} ::= #SEQUENCE {
    aux-msgId           #MessageIdentifier,
    message             #OriginalMessages }

#MessageIdentifier ::= #INT
```

Specialization of CHOICE index encoding (3)

- The following encoding object specifies how the fields are encoded:
  - the `aux-msgId` field is encoded as an open-ended integer field;
  - the `aux-msgId` acts as a determinant for the `message` field.

```
messages-struct-encoding{<#OriginalMessages>}
       #Messages-struct{<#OriginalMessages>} ::= {
   ENCODE STRUCTURE {
       aux-msgId    msgId-encoding,
       message      {ALTERNATIVE DETERMINED BY field-to-be-set
                     USING aux-msgId }}
   WITH PER-BASIC-UNALIGNED }
msgId-encoding #MessageIdentifier ::= {
   USE #BITS MAPPING TO BITS {
       0 .. 2 TO  '000'B .. '010'B,
       -- 0 = MessageA, 1 = MessageB, 2 = MessageC
       3      TO  '1'B
       -- 3 = Extensions, like 10000, 10001, 10010 etc --}
   WITH self-delimiting-bits }
```

Length determinant for SEQUENCES *(1)*

- Context:
  - There is a group of SEQUENCE types which need to be extensible

```
MessageA ::= SEQUENCE {
     -- Whatever
     extensions     MessageA-Extensions   OPTIONAL }

MessageA-Extensions ::= SEQUENCE { -- Extensible -- }
```

- Problem:
  - The size of the encoding needs to be smaller than in case of normal PER extensibility

- Solution:
  - Introduce a length determinant for the selected SEQUENCE types
  - Length of encoding of extensions is delimited by the SEQUENCE length determinant

Length determinant for SEQUENCES *(2)*

- The following generic encoding structure is used as a replacement structure for extensible SEQUENCEs

```
#Sequence-with-length-determinant ::= #SEQUENCE

sequence-with-length-determinant-encoding
         #Sequence-with-length-determinant ::= {
    REPLACE STRUCTURE
         WITH        Seq-with-length-struct
         ENCODEB BY  seq-with-length-struct-encoding }

#Seq-with-length-struct{< #OrigSequence >} ::= #SEQUENCE {
    aux-length      #INT (0..512),
    seq             #OrigSequence }
```

Length determinant for SEQUENCES *(3)*

- The following parameterized encoding object specifies that:
  - the `aux-length` field is used as a length determinant for the `seq` field;
  - length of `seq` field is measured in bits;
  - otherwise the normal PER rules are used.

```
seq-with-length-struct-encoding{< #OrigSeq >}
         #Seq-with-length-struct{< #OrigSeq >} ::= {
    ENCODE STRUCTURE {
         -- aux-length as in PER
         seq   {ENCODING SPACE
                    SIZE  variable-with-determinant
                          MULTIPLE OF bit
                    DETERMINED BY field-to-be-set
                    USING aux-length }}
    WITH PER-BASIC-UNALIGNED }
```

Length determinant for `SEQUENCES` *(4)*

- The generic encoding structure and encoding object are applied for selected `SEQUENCE` types as follows:

```
RENAMES #SEQUENCE
        AS #Sequence-with-length-determinant
        IN #MessageA-Extensions, #MessageB-Extensions,
           #MessageC-Extensions
FROM Example-ASN1;
```

- As a result the property of length determined encoding is associated with the selected `SEQUENCE` types

Extension of value sets of `INTEGER` *types*

- Context:
  - There are integer types which need to have limited extensibility
  - The maximum number of extensions can be predicted
  - It is specified what to do when a spare value is received

```
-- Used range in version 1 is 1..224,
-- values 225-256 are spare values.
-- If a spare value is received, then the following error
-- procedure shall be initiated...
ExtensibleInteger ::= INTEGER (1..256)
```

- Problem:
  - Minimize the encoding size
  - Make sure that senders do not send spare values

Ignore spare values

- The following encoding object specifies that
  - it is not allowed to send spare values but it is allowed to receive them.

```
extensibleInteger-encoding #ExtensibleInteger::= {
  ENCODE-DECODE {
          USE #INT (1..224)      -- no padding bits needed
          MAPPING ORDERED VALUES
          WITH          per-int-encoding }
  DECODE-AS-IF   per-int-encoding }

per-int-encoding #INTEGER ::= {
  ENCODE WITH PER-BASIC-UNALIGNED }
```

Additional slides

(character-based vs. binary encoding)

The Montagues and Capulets

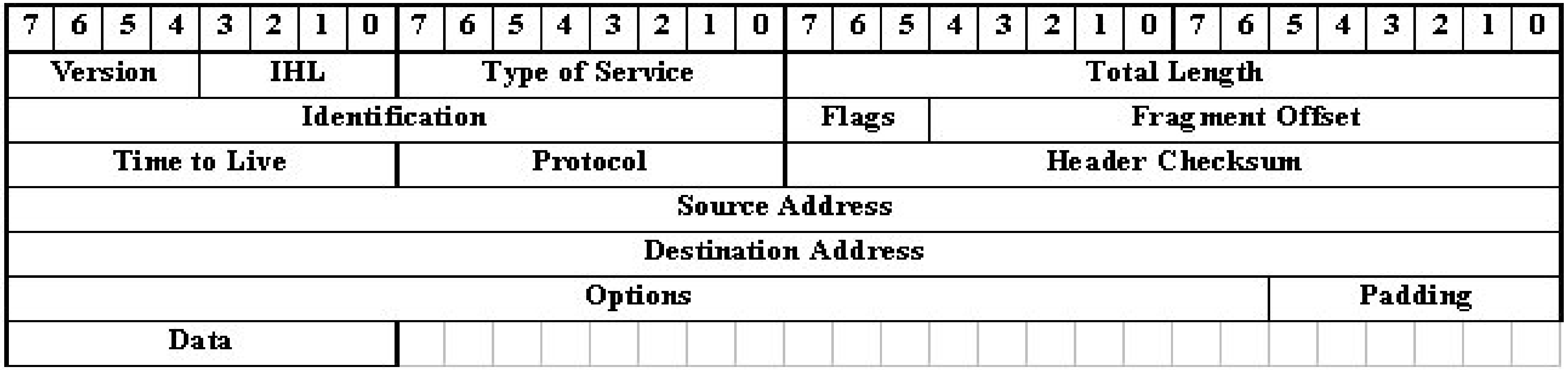
- A long and on-going civil dispute
- Montagues ⇒
  Binary-based specification
- Capulets ⇒
  Character-based specification

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

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.)



7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Version				IHL				Type of Service								Total Length															
Identification																Flags			Fragment Offset												
Time to Live								Protocol								Header Checksum															
Source Address																															
Destination Address																															
Options																										Padding					
Data																															

Tool support not possible - but see ECN discussion later.

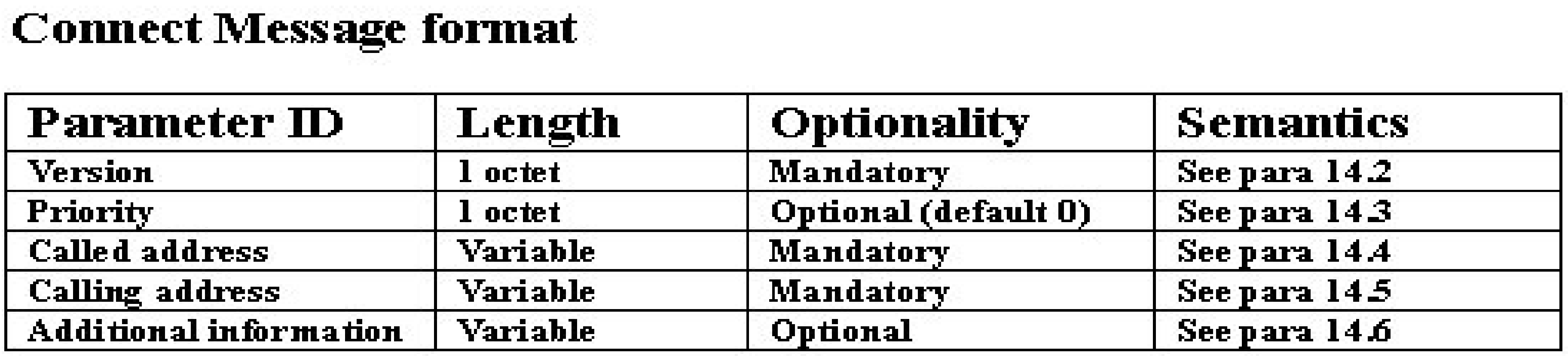
Extensibility support crude - based on reserved fields.

The stone-age Capulets

- Simple “command lines” – in ASCII!
- Three character mnemonics
- Simple comma-separated parameters
- Good for simple dialogues
- Extensibility by adding commands, with unknown commands ignored by V1 systems
- It worked, but the Capulets are still not much beyond the stone-age!

The Bronze Age Montagues invent TLV and Tabular Notation

- Each PDU and each parameter has an ID (or Type), a Length, and a Value
- 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 see the ECN discussion later).
- But it was utterly verbose!

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

The Bronze Age Capulets invent BNF

- The Capulets' main concern was with precise specification of the character strings used for communication.
- 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++, etc.

Last remarks on the Capulets – a biased view!

- Extensibility provision for protocols defined using BNF/ABNF was and remains weak.
- Have the Capulets gone down a (verbose) blind alley?
- But will XML (the Capulets fight back!) rescue them?
- All is still to play for. (To enter the Euro or not, that is the question!)

150 Million seconds after the Bronze Age

- Recognition of:
  - Separation of abstract and transfer syntax
  - Encoding rules
- ASN1 (later ASN.1 – see the difference?) emerges!
- ASN.1 specs define a de facto API.
- Tools emerge to support the transformation of ASN.1 to an API, and the encoding of data across that API.
- **Profits for all!**

300 Million seconds later

- Sophisticated approaches to extensibility (without a simple, verbose, TLV) are developed.
- Used in ASN.1 Packed Encoding Rules (PER)
- Allows very general ECN extensibility support, such as “tail-end additions”, and “end-of-container” optionals.

And another 300 Million seconds later, the Capulets develop XML

- Focus still on what is correct syntax, not content
- (This is still bad. What is syntax variation and what is a difference in the message? Covert channels.)
- Came out of SGML and HTML
- The “X” does not mean eXtensibility”
- Essentially a TLV style of encoding, but with human readable “<Start>….</End>” wrappers
- Rapidly gained popularity! Idiots can understand it! Oh dear!

And finally, after another Million seconds

- ASN.1 develops XML Encoding Rules
- “Coloring” added to allow control of (for example) attributes v elements

Romeo and Juliet marry!