A short introduction to ASN.1 Encoding Control Notation (ECN)

Markku Turunen
ETSI STF 169

http://asn1.elibel.tm.fr/ecn/
• Introduction
• ECN overview
• ECN application areas
• Legacy protocol example
• Specialization examples
Introduction

- Purpose of the presentation
  - To give an overview of ASN.1 Encoding Control Notation (ECN)
  - To show how ECN can be used in different application areas
The Big Picture - ECN concepts

- ASN.1 module
  - Specifies logical contents of a protocol message

- ASN.1 type
  - One possible encoding for each type
  - Encoding structures for types

- Encoding class
- Encoding object

- Encoding definition modules

- Encoding rules
  - e.g., PER
  - Specifies standard encoding for types

- One encoding link module
  - Application of encoding

- Applies encoding for types
• From ASN.1 point of view, ECN is transparent, no modifications in the ASN.1 specification.

Example-ASN1 DEFINITIONS AUTOMATIC TAGS ::= 
BEGIN

MyType ::= INTEGER (0..7)

END
Encoding definition modules

- Encoding definition module contains
  - encoding classes
  - encoding objects
  - encoding object sets

- Encoding classes specify
  - encoding structure, i.e., what are the bitfields that an encoding of a type is composed of

- Encoding objects specify
  - how abstract values of a type are mapped to the bitfields
  - what are the relations between different bitfields, like length and presence determinant fields and determined fields
  - how the bitfields are encoded

- Encoding object sets are
  - collections of encoding objects to be applied to ASN.1 types
Example-EDM ENCODING-DEFINITIONS ::= BEGIN

IMPORTS

#MyType  -- Implicit encoding class for an ASN.1
FROM Example-ASN1;

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

myType-encoding #MyType ::= {
    ENCODING {
        ENCODING-SPACE
        SIZE 3
        MULTIPLE OF bit
        ENCODING positive-int
    }
}

END
Encoding link modules

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

END
Application of ECN

- There are two main application areas for ECN:
  - Non-ASN.1 "legacy" protocols
  - Specialization of ASN.1 protocols
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 are 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
### Hiperlan - Message table form

<table>
<thead>
<tr>
<th>Octet 1</th>
<th>Octet 2</th>
<th>Octet 3</th>
<th>Octet 4</th>
<th>Octet 5</th>
<th>Octet 6</th>
<th>Octet 7</th>
<th>Octet 8</th>
<th>Octet 9</th>
<th>Octet 10</th>
<th>Octet 11</th>
<th>Octet 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defined in DLC TS</td>
<td>Sequence number</td>
<td>MSB</td>
<td>EXTENSION-TYPE</td>
<td>Future use</td>
<td>Sequence number</td>
<td>MSB</td>
<td>EXTENSION-TYPE</td>
<td>Future use</td>
<td>Frame ID</td>
<td>CL-ID</td>
<td>Future use</td>
</tr>
<tr>
<td>Future use</td>
<td>CL-ID</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
<td>Future use</td>
</tr>
</tbody>
</table>
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
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
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 structure for the message
Encoding object for the message

- Encoding object for #RLC-RADIO-HANOVER-COMPLETE-ARG
  - maps fields to the fields of the encoding structure
  - specifies how padding is encoded
  - links the determinant field to the determined fields

```plaintext
rlc-radio-handover-complete-arg-encoding #RLC-RADIO-HANOVER-COMPLETE-ARG ::= {
  USE #RLC-RADIO-HANOVER-COMPLETE-ARG-structMAPPING
  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
    }
  }
}
```
**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
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-WITH OPTIONAL-ENCODING
      -- simplex-forward, duplex, duplex-symetric
      is-present-if{< direction, {0|1|3} >},
    backward-descr USE-WITH OPTIONAL-ENCODING
      -- simplex-backward, duplex
      is-present-if{< direction, {1|2} > }
    -- Structure
    STRUCTURED WITH octet-aligned-sequence-encoding
  }
WITH-PER-BASIC-UNALIGNED
}
• Finally the length determinant is passed to the encoding object that uses it as a length determinant for an octet string

```plaintext
cl-conn-attr-encoding{< REFERENCE : aux-cl-conn-attr-length >} #CL-CONN-ATTR ::= {
  REPETITION-ENCODING {
    REPETITION-SPACE
      DETERMINED BY asn1-field
      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

• 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

• 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

- The following encoding object specifies how the fields are encoded
  - "aux-msgId" field is encoded as an open-ended integer field
  - "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 added-field 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

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

```plaintext
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**

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

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

- The following sequence-with-length-determinant-encoding

  ```sequence-with-length-determinant-encoding #Sequence-with-length-determinant ::= {```
  ```REPLACE STRUCTURE```
  ```WITH Seq-with-length-struct{< >}```
  ```ENCODER BY seq-with-length-struct-encoding{< >}```
  ```}```

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

- 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

```plaintext
def 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 added-field
      USING aux-length
    }
  }
}
```
Length determinant for SEQUENCEs

• 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

```plaintext
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
}
```
Collect encoding objects in one encoding object set

MyEncodings #ENCODINGS ::= {
    messages-encoding | sequence-with-length-determinant-encoding | extensibleInteger-encoding
}
Example-ELM LINK-DEFINITIONS ::= 
BEGIN 
IMPORTS
   #Messages, -- Implicit encoding classes
   #ExtensibleInteger
FROM Example-ASN1
   #MessageA-Extensions, -- Explicitly renamed encoding classes
   #MessageB-Extensions,
   #MessageC-Extensions,
   MyEncodings -- Encoding object set
FROM Example-EDM;
ENCODE #Messages, #ExtensibleInteger,
   #MessageA-Extensions, #MessageB-Extensions, #MessageC-Extensions
WITH MyEncodings
   COMPLETED BY PER-BASIC-UNALIGNED
END
ECN Presentation Summary

• The basic concepts of ECN are fairly simple

• Application area and wanted encoding features affect a lot of how ECN can be applied
  • Multiple ways to achieve the same goal
  • What is specified in ASN.1 and what in ECN
  • Generic ECN vs. specific ECN