Introduction to ASN.1

Simple and structured types
Basic concepts

Version 1.1.2

Presentation

• Problem:
  – Heterogeneous systems
  – Multiple programming languages

• How to exchange information?

• Abstract Syntax Notation 1

ASN.1 is everywhere

ASN.1 is:
• A formal notation to describe data types
• A specification of encoding / decoding rules

Is it used in many fields, such as:
• Aeronautics: ATN
• Telecommunications: VoIP, MAP
• Network protocols: SNMP
• Security: cryptography, digital signature

ASN.1 processes

ASN.1 specification

Compilation (language specific projection)

ASN.1 objects source code

Encoding rules

30 80 02 02 A0 41 30 00 00 00

Decoding rules
ASN.1 components

• A notation syntax used to represent constrained data types: this syntax is abstract as it is not linked with a particular transfer syntax (encoding / decoding processes)

• Standard transfer syntaxes:
  – BER: Basic Encoding Rules
  – CER: Canonical Encoding Rules
  – DER: Distinguished Encoding Rules
  – PER: Packed Encoding Rules
  – XER (B-XER,C-XER, E-XER): XML Encoding rules

Simple types

• INTEGER
• REAL
• ENUMERATED
• BOOLEAN
• OCTET STRING, BIT STRING
• String types (NumericString, VisibleString, …)
• UTCTime
• GeneralizedTime
• RELATIVE IDENTIFIER
• OBJECT IDENTIFIER
• NULL (no associated value)

Identifiers

• RELATIVE IDENTIFIER
• OBJECT IDENTIFIER

Identifier ::= NumericValue
 ::= NumericValue Id
Id ::= "." Identifier

Example: 2.5.29.15 (certificate key usage)

Structured types

Placeholders for inner ASN.1 elements

• SEQUENCE → Ordered
• SET → Unordered

• SEQUENCE OF
• SET OF Collections

• EMBEDDED PDV, …
Structured types elements

Inner ASN.1 elements can be declared:
- Mandatory: must be initialized
- OPTIONAL: may not be initialized
- With a DEFAULT value: this value is used if not initialized

OPTIONAL and DEFAULT are mutually exclusive

These declarations along with a transfer syntax define the encoding (or decoding) process success or failure

Transparent types

- CHOICE → Alternative
  - Defined set of possible (tagged) elements
  - Each element must be uniquely identified
  - This identification is linked with encodings

- OpenType → Formerly ANY
  - “Blob” type

Do not appear in encodings

Types constraints

- Basic types (examples):
  - INTEGER: minimum / maximum values
  - REAL: minimum / maximum values
  - String types: characters restrictions, length, regular expressions match
  - ENUMERATED: list of accepted (significant) values
  - Time (UTC, Generalized): time validity
  - IDENTIFIER: positive integer values with restrictions

- Collections:
  - minimum / maximum size
  - Elements types (inner elements must belong to the same type)

Simple types derived types examples

Version ::= INTEGER

Name ::= UTF8String SIZE(1..50)

-- Not empty and cannot exceed 50 characters

Gender ::= ENUMERATED {
  male,
  female
}
### Values assignments

- **General format:**

  \[
  \text{instance} ::= \text{Type} \quad \text{Value} \\
  \quad | \langle\text{tag}\rangle \text{Value}\langle/\text{tag}\rangle \\
  \quad \text{-- XML format}
  \]

- **Examples:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>::= INTEGER</td>
</tr>
<tr>
<td>my-Value</td>
<td>::= Int 3</td>
</tr>
<tr>
<td>bool-Instance</td>
<td>::= BOOLEAN TRUE</td>
</tr>
</tbody>
</table>

### SEQUENCE derived type example

\[
\text{Individual} ::= \text{SEQUENCE} \{ \\
  \text{first} \quad \text{UTF8String}, \\
  \text{last} \quad \text{UTF8String}, \\
  \text{age} \quad \text{INTEGER (0..MAX)} \quad \text{OPTIONAL}, \\
  \text{gender} \quad \text{Gender} \quad \text{DEFAULT} \quad \text{female}
\}
\]

\[
\text{myself Individual} ::= \{ \quad \text{-- 'age' is not mandatory} \\
  \text{first} \quad "\text{foo}" , \text{last} \quad "\text{bar}" , \text{gender} \quad \text{male}
\}
\]

### SET OF derived type example

\[
\text{Individuals} ::= \text{SET OF} \quad \text{Individual}
\]

\[
\text{myIndividuals Individuals} ::= \{ \\
  \{\text{first} \quad "\text{Paul}" , \text{last} \quad "\text{Smith}" , \text{age} \quad 24 , \text{gender} \quad \text{male}\}, \\
  \{\text{first} \quad "\text{John}" , \text{last} \quad "\text{Smith}" , \text{age} \quad 30 , \text{gender} \quad \text{male}\}, \\
  \{\text{first} \quad "\text{Pamela}" , \text{last} \quad "\text{Smith}"\}
\quad \text{-- 'gender' DEFAULT value applies if not mentioned}
\}
\]

### CHOICE derived type example

\[
\text{Time} ::= \text{CHOICE} \{ \\
  \text{utc} \quad \text{UTCTime}, \\
  \text{general} \quad \text{GeneralizedTime}
\}
\]

\[
\text{myTime Time} ::= \{\text{utc} \quad "0612242359Z"\}
\]

- Only one inner ASN.1 element encoded
Generic linked list example 1

```
LinkedList1 ::= SEQUENCE {
    value       OpenType,
    next        NextElement
}

NextElement ::= CHOICE {
    other       LinkedList1,
    noElement   NULL
}
```

Example of 4 values

Generic linked list example 2

```
LinkedList2 ::= SEQUENCE {
    value       OpenType,
    next        LinkedList2 OPTIONAL
}
```

Example of 4 values

Generic linked list example 3

```
LinkedList3 ::= SEQUENCE OF OpenType
```

Specs evolution: types substitution

Replace any non OpenType element with CHOICE:

```
MyValue ::= INTEGER

MyValue ::= CHOICE {
    val1      INTEGER,
    val2      REAL
}
```
Specs evolution: extensibility

Use of ellipsis “…” within SEQUENCE (OF) and ENUMERATED types and types constraints (ASN.1 97 specifications)

Example:

```plaintext
Individual ::= SEQUENCE {
    first   UTF8String,
    last    UTF8String,
    age     INTEGER (0..MAX) OPTIONAL,
    gender  Gender DEFAULT female,
    …,
    [2: address   UTF8String,
      town      UTF8String],
    [3: email    GeneralString]
}
```

First specification extension

```plaintext
Individual ::= SEQUENCE {
    first   UTF8String,
    last    UTF8String,
    age     INTEGER (0..MAX) OPTIONAL,
    gender  Gender DEFAULT female,
    …,
    [2: address   UTF8String,
      town      UTF8String]]
}
```

Can be omitted: refers to version 2 extra components

Next specification extension

```plaintext
Individual ::= SEQUENCE {
    first   UTF8String,
    last    UTF8String,
    age     INTEGER (0..MAX) OPTIONAL,
    gender  Gender DEFAULT female,
    …,
    [2: address   UTF8String,
      town      UTF8String],
    [3: email    GeneralString]]
}
```

Comments on extensibility

• Opening and closing double square brackets used to group extensions together
• An optional version number can be mentioned

• Double squares are not mandatory:

```plaintext
[2: address   UTF8String,
  town      UTF8String],
email     GeneralString
```
Module description

ModuleName  DEFINITIONS
IMPLICIT TAGS
   -- Module header section
   ::=  
BEGIN
      -- IMPORTS section;
      -- EXPORTS section;

      -- Derived types declarations section
END

Introduction to ASN.1

CHOICE invalid definition 1

Invalid1 ::= CHOICE {
   str1  PrintableString,
   str2  PrintableString
}

Why is this definition incorrect?

CHOICE invalid definition 1 (ctn’d)

Invalid1 ::= CHOICE {
   str1  PrintableString,
   str2  PrintableString
}

2 elements of the same type: decoding ambiguity

CHOICE invalid definition 2

Invalid2 ::= CHOICE {
   blob  OpenType,
   octets  OCTET STRING
}

Why is this definition incorrect?
Invalid2 ::= CHOICE {
  blob    OpenType,
  octets  OCTET STRING
}

OpenType can hold an OCTET STRING: similar to previous case

Basic ASN.1 notation limits

TwoInt ::= SEQUENCE {
  val1    INTEGER OPTIONAL,
  val2    INTEGER OPTIONAL
}

Why is this description incorrect?

Basic ASN.1 notation limits (ctn’d)

TwoInt ::= SEQUENCE {
  val1    INTEGER OPTIONAL,
  val2    INTEGER OPTIONAL
}

What happens if only one INTEGER is initialized?

Decoding ambiguity: val1 or val2 received?

Basic ASN.1 notation limits (end)

OtherAmbiguous ::= SEQUENCE {
  val1    INTEGER DEFAULT v1(1),
  val2    INTEGER OPTIONAL
}

Same problem if the applied transfer syntax does not encode non-initialized DEFAULT-valued elements

Decoding ambiguity if a single INTEGER is encoded
Use tags!

Non-ambiguous declaration

Unambiguous ::= SEQUENCE {
  val1 [0] INTEGER OPTIONAL,
  val2 INTEGER OPTIONAL
}

Refer to ASN.1 tagging presentation…

Conclusion

- ASN.1 is used in various domains including security
- It is composed of:
  - A notation syntax
  - A set of standard transfer syntaxes, including XML-like
- ASN.1 is an improved working environment
- Allows backward compatible evolutions
- Transfer syntaxes ensure systems interoperability
- Many implementations available in many languages (Perl, PHP, C, C++, Java, .NET, …)
- Efficient octet-based and bit-based implementations for high performances and real-time communications