Lecture 6:
Abstract Syntax Notation One
ASN.1

Prof. Shervin Shirmohammadi
SITE, University of Ottawa

Abstract Syntax Notation One

- Both the information and communications models need to be specified **syntactically** and **semantically**.
- This requires a language that specifies the management protocol in the application layer.
- This is where Abstract Syntax Notation One (ASN.1) plays a role.
- ASN.1 is actually more than a syntax; it’s a **language**.
- Addresses both syntax and semantics
- Two type of syntax
  - Abstract syntax: set of rules that specify data type and structure for information storage
  - Transfer syntax: set of rules for communicating information between systems
- Makes application layer protocols independent of lower layer protocols.
- Can generate machine-readable code: Basic Encoding Rules (BER) is used in management modules.
- It is based on the Backus-Nauer Form (BNF)
Backus-Nauer Form (BNF)

- BNF constructs are developed from primitives.
  - `<name>` ::= `<definition>`
  - `<name>` is “entity”
  - ::= “defined as”
  - `<definition>` is “primitive”
- Example: Simple Arithmetic Expression entity (<SAE>) is constructed from the primitives <digit> and <op>
  - `<digit>` ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  - `<number>` ::= `<digit>` | `<digit>`<number>
  - `<op>` ::= + | - | x | /
  - `<SAE>` ::= `<number>`|<SAE>|<SAE><op><SAE>
- Example:
  - 9 is primitive 9
  - 19 is construct of 1 and 9
  - 619 is construct of 6 and 19

Type and Value

- The format of each line is defined as an assignment
  - `<BooleanType>` ::= BOOLEAN
  - `<BooleanValue>` ::= TRUE | FALSE
- ASN.1 module is a group of assignments; for example:

```plaintext
person-name Person-Name::=
{  
   first    "John",
   middle   "I",
   last     "Smith"
}
```

Example: 13 x 2
Data Type: Example 1

- Module name starts with capital letters
- Data types:
  - Primitives: NULL, GraphicString
  - Constructs
    - Alternatives: CHOICE
    - List maker: SET, SEQUENCE
    - Repetition: SET OF, SEQUENCE OF:

```plaintext
PersonnelRecord ::= SET
{  Name,
   title GraphicString,
   division CHOICE {  marketing [0] SEQUENCE
                        {Sector,
                         Country},
                        research [1] CHOICE
                        {product-based [0] NULL,
                         basic [1] NULL},
                        production [2] SEQUENCE
                        {Product-line,
                         Country} }  

etc.
```

Figure 3.13 ASN.1 Data Type Definition Example 1

Data Type: Example 2

A list of invoices.

```plaintext
Trade-message ::= SEQUENCE
{invoice-no INTEGER
 name     GraphicString,
 details  SEQUENCE OF
  SEQUENCE
  {part-no INTEGER,  
   quantity INTEGER),
   charge REAL,  
   authenticator Security-Type}

Security-Type ::= SET
{  ... 
  ...  
   ...  }
```

Figure 3.14 ASN.1 Data Type Definition Example 2
ASN.1 Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>::=</td>
<td>Defined as</td>
</tr>
<tr>
<td></td>
<td>or, alternative, options of a list</td>
</tr>
<tr>
<td>-</td>
<td>Signed number</td>
</tr>
<tr>
<td>--</td>
<td>Following the symbol are comments</td>
</tr>
<tr>
<td>{}</td>
<td>Start and end of a list</td>
</tr>
<tr>
<td>[]</td>
<td>Start and end of a tag</td>
</tr>
<tr>
<td>()</td>
<td>Start and end of subtype</td>
</tr>
<tr>
<td>..</td>
<td>Range</td>
</tr>
</tbody>
</table>

MIB Definition Example

The terms DEFINITIONS, BEGIN, and END, are keywords. This statement means that the RFC1213-MIB Module is being defined

```
RFC1213-MIB DEFINITIONS ::= BEGIN

...  
...  
...  

END
```
Data Type: Structure & Tag

- We will now use the ASN.1 notation to define the various data types and apply them to describe objects of SMI and MIB.
- Data types are defined based on **Structure** and **Tags**
  - **Structure** defines how data type is built
  - **Tag** uniquely identifies the data type

![ASN.1 Data Type Structure and Tag Diagram](image-url)
Structure

- **Simple** (values are specified directly)
  - PageNumber ::= INTEGER
  - ChapterNumber ::= INTEGER

- **Structure / Construct** (Contains other types)
  - BookPageNumber ::= SEQUENCE
    {ChapterNumber, Separator, PageNumber}
    Example: {1-1}
  - BookPages ::= SEQUENCE OF {BookPageNumbers}
    Example: {1-1, 2-3, 6-25}

- **Tagged** (Used primarily for efficiency)
  - Derived from another type; given a new ID

- **Other types**: (Data type that is not pre-defined)
  - values chosen from CHOICE and ANY types

---

Other Types

- **CHOICE (No Data Types)**
  
  research Research ::= CHOICE
  {product-based [0] NULL,
   basic [1] NULL}

- **CHOICE (Data Types)**

  research Research ::= CHOICE
  {product-based ProductType,
    basic VisibleString}
  ProductType ::= VisibleString

- **ANY (From the previous definitions)**

  Research ::= CHOICE
  {product-based ANY,
   basic ANY}
Example of Sequence Structure

- Two ways to define all the pages of a book as a sequence of page numbers making certain that they are in order.

\[ \text{BookPages ::= SEQUENCE OF \{ BookPageNumber\}} \]

\[ \text{or} \]

\[ \text{BookPages ::= SEQUENCE OF} \]

\[ \{ \text{SEQUENCE} \]

\[ \{ \text{ChapterNumber, Separator, PageNumber} \} \]

Tag

- Tag uniquely identifies a data type. It is required for encoding the data types for communication.
- Every data type except CHOICE and ANY have data tags associated with them.
- Comprised of a class and tag number.
- Four Class types:
  - Universal – is the most common, like global variables in a software program.
  - Application - only in the application used, override universal.
  - Context-specific - specific context in application.
  - Private - used extensively by commercial vendors.
Context Specific Tag Example

PersonnelRecord ::= SET
{   Name,
    title   GraphicString,
    division    CHOICE
      marketing   [0] SEQUENCE
        {Sector,
        Country},
      research   [1] CHOICE
        {product-based   [0] NULL,
        basic   [1] NULL},
      production   [2] SEQUENCE
        {Product-line,
        Country} } }
ASN.1 Module SNMP MIB Example

Address translation table.

IpNetMediaEntry ::= SEQUENCE {
    ipNetToMediaIfIndex INTEGER,
    ipNetToMediaPhysAddress PhysAddress,
    ipNetToMediaNetAddress IpAddress,
    ipNetToMediaType INTEGER
}

Object Names and SMI Object Tree

- In a MIB there is an identifier for each occurrence of an object.
  internet OBJECT IDENTIFIER ::= 
  {ISO(1) ORG(3) DOD(6) INTERNET(1)}

Private type identifier for IBM

1.3.6.1.4.1.2
Walkthrough Example

Description of a Personal Record

| Name:       | John P Smith          |
| Title:      | Director              |
| Employee Number | 51                   |
| Date of Hire: | 17 September 1971    |
| Name of Spouse: | Mary T Smith         |
| Number of Children | 2                   |
| Child Information |
| Name:       | Ralph T Smith         |
| Date of Birth | 11 November 1957     |
| Child Information |
| Name:       | Susan B Jones         |
| Date of Birth | 17 July 1959         |

Walkthrough Example (…)

ASN.1 description of the record structure

PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
  Name,
  title [0] VisibleString,
  number EmployeeNumber,
  dateOfHire [1] Date,
  nameOfSpouse [2] Name,
  children [3] IMPLICIT SEQUENCE OF ChildInformation DEFAULT { } }

ChildInformation ::= SET {
  Name,
  dateOfBirth [0] Date }

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
  givenName VisibleString,
  initial VisibleString,
  familyName VisibleString }

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString -- YYYYMMDD
Walkthrough Example (...)

ASN.1 Description of a Record Value

{{ givenName "John", initial "T", familyName "Smith"},
  title "Director"
  number 51
  dateOfHire "19710917"
  nameOfSpouse { givenName "Mary", initial "T", familyName "Smith"},
  children
  {{ givenName "Ralph", initial "T", familyName "Smith"},
   dateOfBirth "19571111"},
  {{ givenName "Susan", initial "B", familyName "Jones"},
   dateOfBirth "19590717"}}

TLV Encoding

• In ASN.1 ASCII text data is encoded into a bit-oriented data representation called TLV (Type, Length, and Value).
• TLV type, length, and value are components of the structure.

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class (7-8th bits)</td>
<td>P/C (6th bit)</td>
<td>Tag Number (1-5th bits)</td>
</tr>
</tbody>
</table>

specifies primitive, or construct

<table>
<thead>
<tr>
<th>Class</th>
<th>8th bit</th>
<th>7th bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Application</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Context-specific</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Private</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
ASN.1 Macro

- ASN.1 allows extensions to define new data types and values through a Macro definition.
- They also facilitate grouping of instances of an object.

\[
<\text{macroname}> \text{MACRO} ::= \\
\text{BEGIN} \\
\text{TYPE NOTATION} ::= <\text{syntaxOfNewType}> \\
\text{VALUE NOTATION} ::= <\text{syntaxOfNewValue}> \\
<\text{auxiliaryAssignments}> \\
\text{END}
\]

Macro Example

From ITU-T X.219:
ERROR MACRO ::= 
BEGIN 
TYPE NOTATION ::= Parameter 
VALUE NOTATION ::= value (VALUE CHOICE 
\{ localValue INTEGER, 
\quad globalValue OBJECT IDENTIFIER 
\}) 
Parameter ::= "PARAMETER" NamedType | empty 
NamedType ::= identifier type | type 
END

Usage:

BadQueueName ERROR 
PARAMETER QueueName 
::= 0