マルチメディア通信と分散処理 36 － 1
（１９８８．２．１９）

ＡＳＮ．１ハンドラの設計と実装

姉崎章博

日本電気(株) 基本ソフトウェア開発本部第三開発部

今年、１９８８年はＯＳＩ（開放型システム間相互接続）の開発が大きく前進する年と言える。昨年の９月に開催されたデータショウ '８７における当社を含む７社によるＦＴＡＭ（ファイル転送，アクセスと管理／ＤＩＳ版）のデモの成果を踏まえて、本年１１月にＩＮＴＡＰ（情報処理相互運用技術協会）によるＯＳＩネットワーク(FTAM/IS版,MHS,ODA/ODIF,RDA)のデモがさらに規模を拡大して行われる。また、多くの応用プロトコルが本年早々に制定される見込みである。これら上位層のプロトコル・データ単位（ＰＤＵ）は、ＡＳＮ．１（ＩＳＯ　８８２４／８８２５）を使用して定義されている。

本稿では、まず、ＡＳＮ．１の複雑さについて述べる。次に、その対策として開発した、ＰＤＵを組立て／分解する汎用Ｃ関数ＡＳＮ．１ハンドラの設計方法と実装時考慮した移植性の問題について述べる。最後にＩＳＯ　８８２４でＰＤＵを記述する場合、間違い易い記述について述べる。

Design and Implementation of ASN.1-Handler

Akihiro ANEZAKI

Basic Software Development Division, NEC Corporation
10, Nisshincho 1-Chome, Fuchu City, Tokyo, 183 Japan

In this year,1988, we'll make a great progress in the OSI development. We demonstrated FTAM(File Transfer, Access and Management / DIS version) in September, 1987 in Japan. We'll advance the results and hold a demonstration of the OSI network(FTAM/IS version, MHS, ODA/ODIF, RDA) in November by INTAP(INTEROPERABILITY TECHNOLOGY ASSOCIATION FOR INFORMATION PROCCESSING,JAPAN). Many application protocols will be also standardized early in this year. These upper layer PDUs(protocol data units) is represented by ASN.1(Abstract Syntax Notation One, ISO 8824/8825).

This paper describes the complexity of ASN.1, then presents the design and implementation of ASN.1-Handler whitch can encode and decode PDU of those upper layers. And this points out some PDU notations to be mistaken.

1．はじめに

昨年９月に開催されたデータショウ '８７において、日本で初めて、ＯＳＩ(Open Systems Interconnection, 開放型ｼｽﾃﾑ間相互接続)　ＦＴＡＭ(File Transfer, Access and Management)による７社の異機種間相互接続デモが行われ、さらに、本年１１月に、『第２回　情報処理相互運用国際シンポジウム』に合わせてＩＮＴＡＰ（(財) 情報処理相互運用技術協会）によるＯＳＩネットワークのデモンストレーションが行われる。

当社ではデータショウデモで使ったＯＳＩ応用層プロトコルの製品を開発するに当たって、ＩＳＯ　８８２４「ＡＳＮ．１(Abstract Syntax Notation One,抽象構文記法1)仕様」[1] で記述されたＰＤＵ（プロトコルデータ単位）をＩＳＯ　８８２５「ＡＳＮ．１基本符号化規則」[2] に従って組立て／分解する汎用関数【ＡＳＮ．１ハンドラ】をＣ言語で開発したのでその報告を行う。第２章でＡＳＮ．１仕様の概要とその複雑さについて、第３章でＡＳＮ．１ハンドラの処理概要について、第４章で組立て／分解方法について、第５章で移植性保証について、第６章で間違い易いＡＳＮ．１記述ついて述べる。

2．ＡＳＮ．１仕様の概要とその複雑さ

ＡＳＮ．１仕様が規定された目的は、上位層のデータの構造が下位層に比べ極端に複雑であり、下位層のようにＰＤＵをオクット列で規定することが困難であることに起因する。このためＡＳＮ．１仕様では、ＰＤＵのタイプを記述するデータ型の記法とＰＤＵのインスタンスを記述する値の記法を対応させて規定し、オクテット・イメージを使うことなくＰＤＵの構造を定義可能としている。ＡＳＮ．１仕様で規定するデータ型には、“単純形”と“構造形”の２種類があり、単純形としては具体的には以下の型が

論理項目であることを表す	BOOLEAN 型
整数項目であることを表す	INTEGER 型
ビット列項目であることを表す	BIT STRING型
ｵｸﾃｯﾄ 列項目であることを表す	OCTET STRING型
空列項目であることを表す	NULL型
文字列項目であることを表す	IA5String 型等
など	

これらを組み合わせて新しいデータ型を構成するための構造形として

要素が順序項目であることを表す	SEQUENCE型
要素が集合項目であることを表す	SET 型
要素が選択項目であることを表す	CHOICE型

が規定されており、これらの型を組み合わせて各プロトコル仕様でそのＰＤＵの形式を定義するのである。

ＡＳＮ．１仕様の型の記述で定義されたＰＤＵに対応した値の記述を実際のオクテット列に符号化する手段としてＡＳＮ．１符号化規則が規定されている。

このように、ＡＳＮ．１仕様では複雑なオクテット・イメージを使うことなしに各プロトコルを定義可能にしているが、その複雑さが雲散霧消したわけではない。ＡＳＮ．１仕様で規定したＰＤＵを実際にプログラムで扱うためには、ＡＳＮ．１基本符号化規則で定める複雑なビットイメージに直面しなければならない。その打開策として、ＡＳＮ．１ハンドラを開発したのである。主な複雑さは以下の６点である。

［複雑さ①］プレゼンテーション層以上のＰＤＵは

identifier octets	length octets	contents octets

の３項目を基本とし、構造形ならばこのcontents octets にはさらに、データ型に従って、０個以上の上記３項目の並びで構成される。トランスポートＰＤＵでは２レベルの入れ子であり、セションＰＤＵは３レベルの入れ子であるのに対して、ＡＳＮ．１では入れ子の深さに制限がない。

［複雑さ②］自分のcontents octets が上記構造を持っていると、子のlength octets により自分のcontents octets 長が分かるので、自分のlength octets を '80'H として、end-of-contents octetsを加え、

identifier octets	length octets	contents octets	end of contents octets

という構造にすることもできる（不定長形式）。例えば図１のように１つの値に対して２つのＰＤＵ形式が作成可能である。

PDU ::= SEQUENCE { INTEGER, IA5String }

{ -1, "ABC" }

(a)ASN.1型の記述と値の記述

30	08	02	01	FF	16	03	41	42	43

(c) ＰＤＵ形式その１

30	80	02	01	FF	16	03	41	42	43	00	00

(d) ＰＤＵ形式その２（不定長形式）

図１．２つのＰＤＵ形式

この、どちらの形式にするかは、ＰＤＵの作成者の任

意であるので、受信側は両方の形式のＰＤＵを分解できなければならない。

［複雑さ③］このＡＳＮ．１エンコードは、length octets があることから分かるようにcontents octets は可変長であるのに加え、length octets およびidentifier octets 自身も可変長である。

［複雑さ④］しかも、identifier octets は列を表し、length octets は正の整数値を表すという性質の違いのためか可変長の表し方も図２．のように異なる。

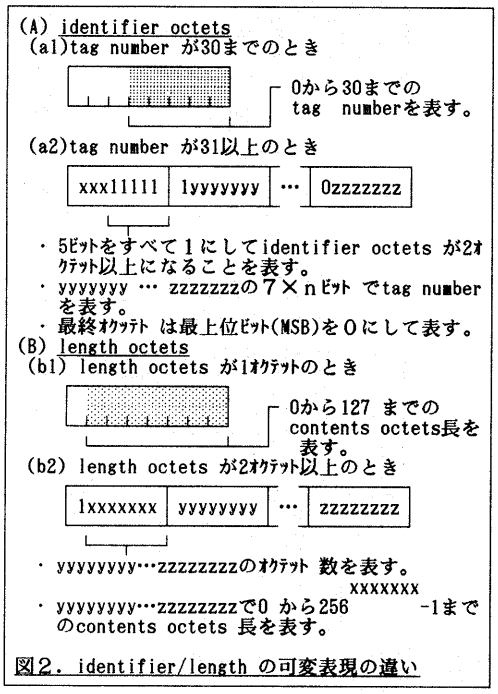


図２．identifier/length の可変表現の違い

［複雑さ⑤］また、整数値型の値は、整数の値が表現できる最小のオクテット列でなければならない。例えば、－１はＦＦの１オクテット、１２８は０１００の２オクテット、１２７は７Ｆの１オクテットのオクテット列でなければならない。

［複雑さ⑥］その他、今までの通信メッセージ形式ではあまり馴染みのなかった以下の３つの概念が導入されている。

①ＣＨＯＩＣＥ型

```
A ::= CHOICE {
        b B,
        c C,
        d D }
```

型A のPDU は、A::=BまたはA::=C またはA::=D と全く区別がなくなる。

②ＳＥＴ型

```
A ::= SET {
        b B,
        c C,
        d D }
```

型A 中の項目B,C,D に順序性はなく、どのように入れ替わったPDU でも同一に処理しなければならない。

③省略属性(SET型およびSEQUENCE型)

```
A ::= SEQUENCE {
        b B OPTIONAL,
        c C DEFAULT xx,
        d D }
```

型A のPDU 中に項目B,C が現れなくてもよく、そのPDU は、項目B,C が省略されると

```
A ::= SEQUENCE {
        d D }
```

のPDU と全く等しく、項目B,C が存在していたことを示す痕跡すらPDU 上に現さない。

３．ＡＳＮ．１ハンドラの処理概要

上記で述べたようにＡＳＮ．１仕様は、様々なＰＤＵ形式を表現するために必要な規則ではあるが、応用層を実現する様々なプログラムがこの規則に従って個々別々にＰＤＵの組立て／分解を行うのは効率的ではない。例えば、プレゼンテーション層で全応用ＰＤＵを組立て／分解してしようとすると、

- 抽象構文名のみならず、他プロトコル仕様で定義しているその構文自体も意識しなければならない。
- 分解してみたものの、上位へ引き渡すために再び何らかの形式にまとめてインターフェースを取る必要がある。

という問題が出てくるため、上位のＰＤＵ部分の抽象構文としてＰＤＵの形式のまま上位へ引き渡すインターフェースが考えられる。例えば、図３．に示すデータショウ '８７　ＦＴＡＭデモ時に実現したような上位層のソフトウェア構成が考えられる。

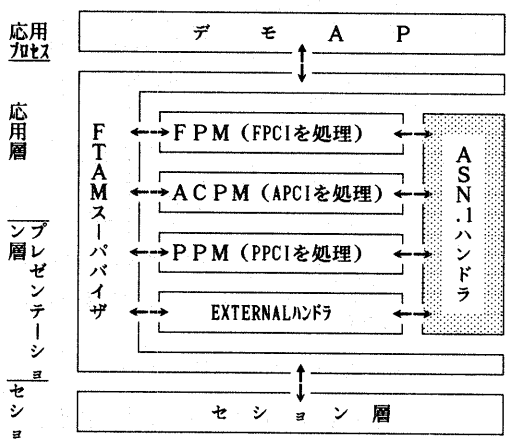
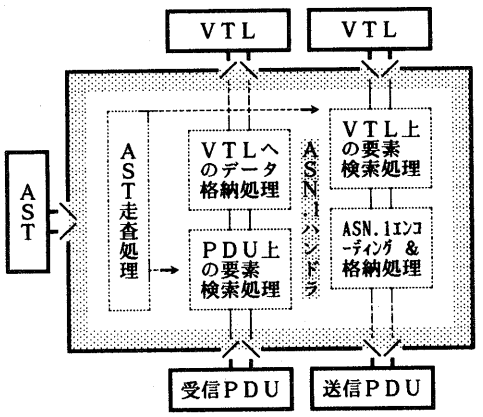


図３．ＦＴＡＭデモ時のソフトウェア構成

これをインプリメントするためには、各応用サービス

要素を実現するプログラムでＡＳＮ．１符号化規則に従った組立て／分解処理を行わなければならない。つまり、各プログラムで第２章で述べた複雑さを克服しなければならないのであるが、これでは多くの労力が重複浪費されることになる。そこで、図３のようにＡＳＮ．１ハンドラを各プログラムで共通に利用可能な形式にし、ＡＳＮ．１基本符号化規則に従った組立て／分解処理を一箇所で処理するようにした。

ＡＳＮ．１ハンドラの機能概略を図４．に示す。図は、ＰＤＵの分解時／組立時に係わらず各プログラムはＡＳＮ．１ハンドラに対して、処理対象とするＡＳＮ．１の型の記述に対応したＡＳＴ(ASN.1 Syntax Tree) を与えることを表している。そのＡＳＴに従って、分解処理時は指定された受信ＰＤＵの構造を調べＶＴＬ(Value Tree List) 上にそのＰＤＵの構造に対応するＡＳＮ．１の値の記述を出力し、組立処理時は入力のＶＴＬ上に表現されたＡＳＮ．１の値の記述に基づき、指定されたバッファ上にＰＤＵを生成することを示している。



AST:ASN.1 Syntax Tree,ASN.1 の型の記述を表す木構造体
VTL:Value Tree List,ASN.1 の値の記述を表す構造体の配列

図４．ＡＳＮ．１ハンドラ処理概要

これにより、応用層の各プログラムはＰＤＵを直接参照することなく、ＡＳＮ．１の値の記述に対応したＶＴＬ上の値を参照するだけで各プロトコルの処理を行うことができる。その処理イメージを図５．に示す。

４．組立て／分解方法

図５のようにＡＳＮ．１ハンドラでは、

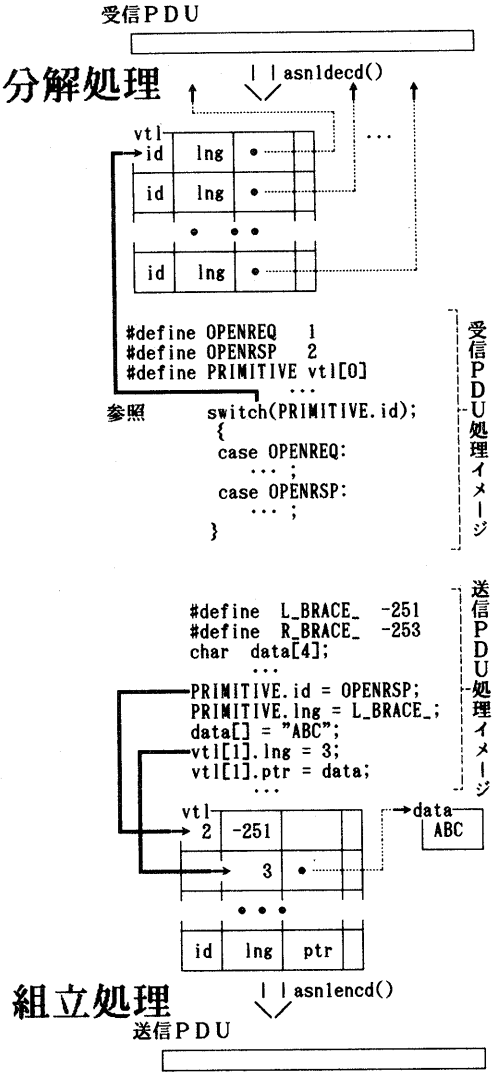


図５．応用層プログラムの処理イメージ

```
ａｓｎｌｅｎｃｄ（）；
    指定されたﾃﾞｰﾀ からPDU を生成する。
ａｓｎｌｄｅｃｄ（）；
    目的のﾃﾞｰﾀ を取り出せるようPDU を分解する。
```

の２つの関数を提供する。

前述したようにＡＳＮ．１ハンドラの利用者は、組立て／分解するデータのアドレス／長さ等の情報を、

ＶＴＬ(Value Tree Lsit)

と呼ぶ表形式のデータ構造にＡＳＮ．１の値を格納してasnlencd関数を呼び出してＰＤＵを組み立てたり、asnldecd関数を呼び出してＰＤＵを分解しその結果のＶＴＬ

の情報から目的の情報を参照したりできる。

組立て／分解時には処理対象のデータの他にＰＤＵの構造を定義しているＡＳＮ．１の型の記述を

ＡＳＴ(ASN.1 Syntax Tree)

と呼ぶ木構造でＡＳＮ．１ハンドラに与えて、特定プロトコルのＡＳＮ．１記述に依存しないＰＤＵ組立て／分解関数としている。ＡＳＴ／ＶＴＬの構造を以下に詳細に述べる。

４．１　ＡＳＴの構造

ＡＳＮ．１記述をＡＳＮ．１ハンドラで扱うに当たって、もしＡＳＮ．１記述を文字列のままで扱おうとすると、組立て／分解の度にパターンマッチングや名標の解決など行わなければならず実用的な効率を実現するのは困難である。

そこで、ＡＳＮ．１記述が木構造で表現できることに注目し、各データ型を節（ノード）または葉（リーフ）としＡＳＮ．１記述全体を木構造に変換して取り扱う。これをＡＳＮ．１ハンドラの制御用データとして組立て／分解時に使用する。葉以外に成り得る型は、

- ＳＥＴ型
- ＳＥＱＵＥＮＣＥ型
- ＣＨＯＩＣＥ型
- UNIVERSALｸﾗｽ以外のＴａｇｇｅｄ型

の４つの型で、Ｔａｇｇｅｄ型のみ要素（子）が唯一つで、他の３つの型は複数の要素を持ち得る。葉には、UNIVERSALｸﾗｽのＴａｇｇｅｄ型がなる。

例えば、図６．(a) のようなＡＳＮ．１記述を(b) のような木構造で表す。ＡＳＮ．１の型の記述をそのまま木構造で表すと図７のように多分木になるが、このままでは子へのポインタを不定個持つ節の構造になってしまうため、

ＡＳＮ．１記述＝多分木構造
↓
二分木化
↓
節（ノード）の大きさが固定
↓
ＡＳＴジェネレータが開発しやすい

の理由からＡＳＴとしては、各節が持つポインタは、

長男へのポインタ

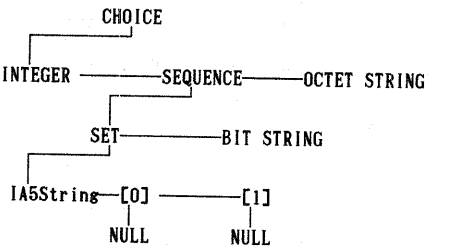
弟へのポインタ

のみとし、図６(b) に示すような二分木構造とした。

このような構造にすることにより、節の構造を固定にしＡＳＴを作り出すジェネレータ（ＡＳＴジェネレータ）も開発しやすいようになった。

```
PDU ::= CHOICE {
          a INTEGER,
          b SEQUENCE {
                SET {
                    a IA5String,
                    b [0] NULL,
                    c [1] IMPLICIT NULL },
                a BIT STRING OPTIONAL },
          c OCTET STRING  }
```

(a) ＡＳＮ．１の型の記述



(b) (a) の記述を表す木構造

図６．ＡＳＮ．１記述を表す二分木構造

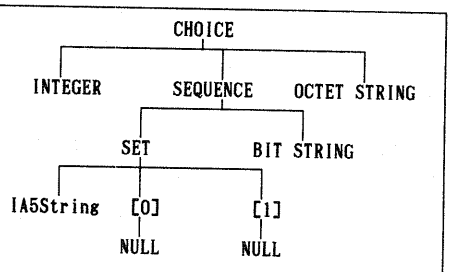
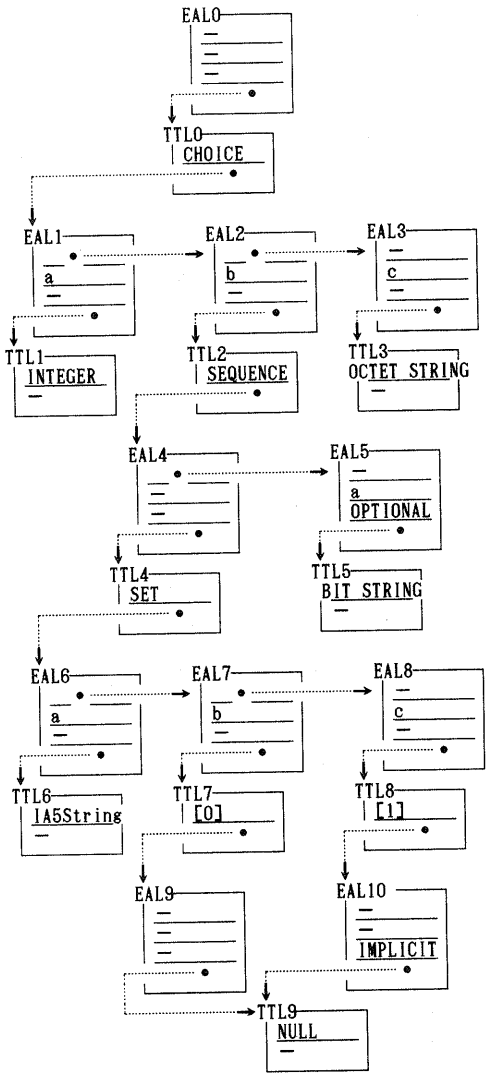


図７．ＡＳＮ．１の型の記述を表す多分木構造

また図６(a) のNULL型のように同じ型ではあるが、一方のNULLだけがTagged型[1] のIMPLICIT属性の要素として存在している場合が考えられる。このような同じ型でも参照のされ方でかかる属性が異なる使い方を、ＡＳＮ．１仕様では型参照名(Type Reference, ::=の左辺）で積極的に使用する仕様である。そのため、節（ノード）を、型に対応する構造体ＴＴＬ(Type Tag List) と要素に対応するＥＡＬ(Element Attribute List)とに分けて図８．に示すようなＡＳＴの構造とした。

この構造を持つＡＳＴを各応用サービス要素を実現するプログラムで定数として定義し、ＡＳＮ．１ハンドラの使用時に指定してもらうことにより、ＡＳＮ．１ハンドラをプロトコルに依存しない汎用のＰＤＵ組立て／分解Ｃ関数とすることができた。



TTL：データ型に対応
id: データ型に対応するidentifier octets
ea: 要素がある場合長男のＥＡＬへのポインタ

ＥＡＬ：それぞれのデータ型の要素に対応
ba: 次の弟ＥＡＬへのポインタ
ic: 付加された識別子（に対応する整数値）
at: OPTIONAL/DEFAULT/IMPLICIT を表す属性
ta: 対応するデータ型のＴＴＬへのポインタ

図８．図6.(a) のASN.1 記述を表すＡＳＴ

４．２　ＶＴＬの構造

ＶＴＬは、ＡＳＮ．１ハンドラのユーザが、組み立てるデータをａｓｎ１ｅｎｃｄ関数へ与えたり、分解された値をａｓｎ１ｄｅｃｄ関数から受け取るためのテーブルである。ＡＳＴで表されるＡＳＮ．１の型の記述に対応した、値の記述が表現される表構造である。

ＶＴＬの構造は、１要素が項目ｉｄ，ｌｎｇ，ｐｔｒ，ｗｋの４項目からなる構造体の配列である。

	id	lng	ptr	wk
vtl[0]				
vtl[1]				
vtl[2]				
	・・・			
vtl[n]				

id:ASN.1記述上の以下の識別子に対応したEAL.ic
SEQUENCE型の省略可の要素の識別子
SET 型の集合要素の識別子
CHOICE型の選択肢を指定する識別子
lng:組立てるﾃﾞｰﾀ 長又は分解したｵｸﾃｯﾄ/ﾋﾞｯﾄ 長
ptr:組立てるﾃﾞｰﾀ へのﾎﾟｲﾝﾀ、または分解したPDU 上のデータへのポインタ
wk:ASN.1ﾊﾝﾄﾞﾗ用の作業領域かつｴﾗｰｺｰﾄﾞ格納領域

図９．ＶＴＬの構造

ＶＴＬ．ｌｎｇの値としては、データの長さの他に

```
L_BRACE_     (=-251)
R_BRACE_     (=-253)
SKIP_CELL_   (=-255)
```

L_BRACE_，R_BRACE_はそれぞれＡＳＮ．１記述の値の記述における“{”，“}”に対応し、値の記述の入れ子関係をＶＴＬ上に表現するために使用する。ＳＫＩＰ_ＣＥＬＬ_は、ＯＰＴＩＯＮＡＬ／ＤＥＦＡＵＬＴ属性の要素がＶＴＬ上値が記入されていても省略したことを利用者が指定できるように用意した機能である。

という値を取る。

このようにＶＴＬという構造体上には、利用者がＡＳＮ．１ハンドラに渡すデータを、またはＡＳＮ．１ハンドラが利用者に渡すデータを、ＡＳＮ．１記述の値の記述に対応させて表現する。例えば、図６．(a) のＡＳＮ．１記述に対する値の一つとして図１０．(a) のような値の記述が有り得る、これをＶＴＬ上図１０(b) のように表現する。

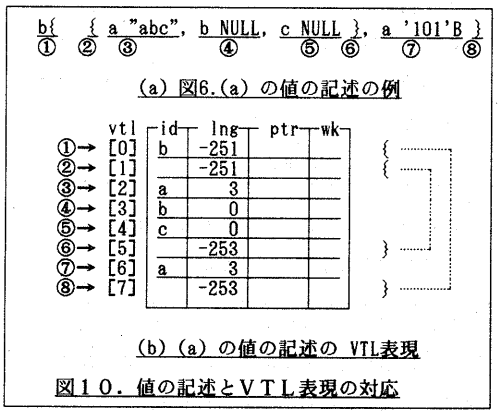
４．３　組立て／分解処理方式

ＡＳＮ．１の型の記述／値の記述をＡＳＴ／ＶＴＬにマッピングさせることによりＰＤＵの組立て／分解処理が可能となった。そのＡＳＴは、

左部分木：子供

右部分木：弟

であり、ＰＤＵ上は親から（もしくは兄から）要素がエンコードされなければならないので、ＡＳＴの木構造の



(a) 図6.(a) の値の記述の例

(b) (a) の値の記述の VTL表現

図１０．値の記述とＶＴＬ表現の対応

各ノードを“行きがけ(preorder)方式”に走査して、親から（もしくは兄から）組立時および分解時に以下の処理を行う。

a) ＡＳＮ．１ハンドラはＶＴＬの値からＰＤＵを組立てる場合、基本的には次の処理を各ノードに対し行う。

① EAL.ic に一致するVTL.idを検索する。
② TTL.id から Identifier Octetsをｴﾝｺｰﾄﾞ する。
③ VTL.lngから Length Octetsをｴﾝｺｰﾄﾞ する。
④ VTL.ptr先の値から Contents Octetsをｴﾝｺｰﾄﾞ する

ただし、不定長形式ではないLength Octets を求める場合、構造型の値の Length Octets値を求めなければならないので、予めＡＳＴの木構造を“通りがけ(inorder)方式”に走査して、子の Length Octets値をすべて求め、親の Length Octets値を求めてる。

b) ＡＳＮ．１ハンドラはＰＤＵをＶＴＬ上に分解するために、基本的には次の処理を各ノードに対して行う。

① TTL.id に一致する Identifier Octetsを検索する
② EAL.ic をVTL.idに格納する。
③ Length OctetsからVTL.lng をﾃﾞｺｰﾄﾞする。
④ Contents OctetsのｱﾄﾞﾚｽをVTL.ptr に格納する。

図１１．は、値の記述図１０(a) のＰＤＵの一つを分解処理した結果の様子を示している。

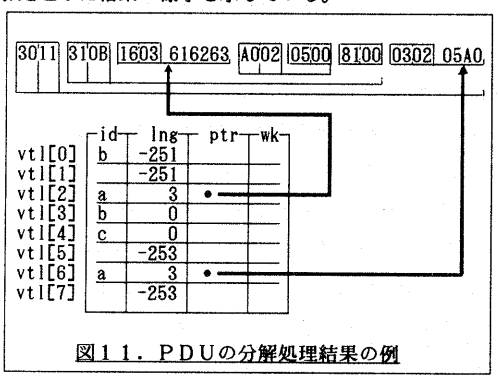


図１１．ＰＤＵの分解処理結果の例

５．移植性

ＯＳＩは今後、パソコンからメインフレームさらにデジタル交換機に至るまで、広範囲に渡って実装されることは必至の状況である。このためＡＳＮ．１ハンドラを移植性のあると言われている言語Ｃで開発した。Ｃであっても機種によっては互換性のない構文が存在し、この互換性のない構文を使うとＡＳＮ．１ハンドラの移植性が悪くなる。このため、互換性上問題のある構文の洗い出しを行った。主なものを以下に列挙する。

①名標は英小文字を英大文字に変換しても重複しないこと。
②内部名は８文字以内、外部名は６文字以内。
③再入可能プログラムとするため外部変数、静的変数は使用しないこと。
④再入可能ﾌﾟﾛｸﾞﾗﾑ とするためﾌｧｲﾙ制御ﾌﾞﾛｯｸ等を作成してしまうﾍｯﾀﾞﾌｧｲﾙ は#includeしないこと。
⑤JISﾏｼﾝとEBCDICﾏｼﾝ とがあるため、文字の算術演算・大小比較およびｺｰﾄﾞ による参照はしないこと。
⑥引数や式の評価順序に依存したｺｰﾃﾞｨﾝｸﾞはしないこと。
⑦全ﾃﾞｰﾀ 型のｻｲｽﾞ は、定数で与えるのではなくsizeof演算を使用すること。
⑧ ﾏｼﾝの語長で文字数が異なる複数文字の文字定数は使用しないこと。
⑨文字型の ｼﾌﾄは unsigned charでｷｬｽﾄしてからｼﾌﾄ するか、有効ﾋﾞｯﾄ のみ論理積を取って使用すること。
⑩ﾎﾟｲﾝﾀ値にｾｸﾞﾒﾝﾄﾌｨｰﾙﾄﾞが有り得るのでﾎﾟｲﾝﾀの ｼﾌﾄ演算はしないこと。
⑪「1ﾊﾞｲﾄ=9ﾋﾞｯﾄ」ﾏｼﾝがあるため、型変換を伴う文字変数への代入は、0xFFで論理積を取ってから代入すること。

これらをコーディング規則に反映させて開発した結果、何の修正もなく再コンパイルするだけで正しく動作することを、ＡＣＯＳ－２／４／６(ﾒｲﾝﾌﾚｰﾑ),ＮＣＯＳ１(ﾐﾆｺﾝ)，ＥＷＳ４８００(ﾜｰｸｽﾃｰｼｮﾝ),Ｎ５２００＋ＭＳ－ＤＯＳ(ﾊﾟｿｺﾝ)について確認済みである。

６．間違い易いＡＳＮ．１記述

前述したようにＡＳＮ．１ハンドラは「型の記述」／「値の記述」の構造情報をそのまま制御構造体ＡＳＴ／ＶＴＬとして使用しており、ＩＳＯ　８８２４で正しく記述されたＰＤＵはすべてＡＳＮ．１ハンドラの処理対象とすることが可能である。しかし、ＩＳＯ　８８２４で記述されているようにＡＳＮ．１で記述しさえすれば正しくＰＤＵを定義できるわけではない。

```
ChoiceType ::= CHOICE{AlternativeTypeList}
AlternativeTypeList ::=
                NamedType                          |
                AlternativeTypeList,NamedType
NamedType ::= identifier Type | Type |
              SelectionType
ChoiceValue ::= NamedValue
NamedValue ::= identifier Value | Value
```

図１２．ＣＨＯＩＣＥ型の定義

例えば、ＩＳＯ　８８２４では図１２のようなＣＨＯＩＣＥ型の型の記述と値の記述の構文の定義に対して、

図１３のような規約を定めている。

22.2 AlternativeTypeListで定義される型は、すべて異なったタグを持つ。
22.4 この国際標準で異なるタグを持つ型を必要としているところでこの型が用いられる場合(18.3,20.3及び22.2参照)、AlternativeTypeListの中で定義された全ての型のタグはその他の型のタグと異なったものとする。

図１３．ＣＨＯＩＣＥ型の規約（その１）
注:18.3,20.3はそれぞれSEQUENCE型の省略可の要素,SET型の要素に関する22.2のような規定である

この規約に基づき図１４ような正誤の例がＩＳＯ　８８２４に記載されている。

正

```
A ::= CHOICE
      { b B,
        c C }
B ::= CHOICE
      { d [0] NULL,
        e [1] NULL }
C ::= CHOICE
      { f [2] NULL,
        g [3] NULL }
```

誤

```
A ::= CHOICE
      { b B,
        c C }
B ::= CHOICE
      { d [0] NULL,
        e [1] NULL }
C ::= CHOICE
      { f [0] NULL,
        g [1] NULL }
```

図１４．CHOICE型におけるﾀｸﾞの用法

さらに図１５の規約が規定されている。

22.5 AlternativeTypeListのNamedType 並びの全てのidentifierは（もし、あれば）、異なっていなければならない。
22.6 この国際標準が異なるidentifierを持つ型を必要としているところでこの型が用いられる場合、AlternativeTypeListの中の全てのNamedTypesのidentifierは（もし、あれば）、他のNamedTypesのidentifier（もし、あれば）とそれぞれ異なっている。

図１５．ＣＨＯＩＣＥ型の規約（その２）

この規約に対しては例が記載されていないが、図１３と図１５を突き合わせれば容易に図１６のような正誤の例を描くことができる。

正

```
A ::= CHOICE
      { b B,
        c [2] NULL }
B ::= CHOICE
      { d [0] NULL,
        e [1] NULL }
```

誤

```
A ::= CHOICE
      { b B,
        e [2] NULL }
B ::= CHOICE
      { d [0] NULL,
        e [1] NULL }
```

図１６．CHOICE型におけるidentifierの用法

型Ａの値としてはタグ番号０から２までの３種類が有り得る。そのエンコードと値の記述の対応を正誤それぞれの場合で図１７に示す。

ﾀｸﾞ	Ｐ Ｄ Ｕ	値の記述（正）	値の記述（誤）
0	A0020500	d NULL	d NULL
1	A1020500	e NULL	e NULL
2	A2020500	c NULL	e NULL

図１７．図１６正誤ごとの値の記述比較

これは、22.5および22.6で「誤」の値の記述"e NULL"のような型の定義を禁止していることを表している。つまり、ＣＨＯＩＣＥ型の型Ｂに付くidentifier"b"は無効となり、値の記述におけるidentifierはＣＨＯＩＣＥ型により最終的に選択される要素に付加されたidentifierのみただ１つ指定し、逆にその

identifier１つで最終的な要素を選択できる構造で型を定義しなければならない。

このことは、選択的な要素に対して空(empty)のidentifierを１要素に対してだけ認めていることからも導くことができる。つまり、複数のidentifierを認めてしまうと空のidentifierが存在しているのかいないのか判別できないためである。

また図１６の「正」の場合でも要素ｄまたはｅの型がＮＵＬＬ型ではなく型Ａであるときも、やはりidentifierが識別できない構造の型となる（図１８）。

```
Filter ::= CHOICE {
        item [0] Item,
        and  [1] IMPLICIT SET OF Filter,
        or   [2] IMPLICIT SET OF Filter,
        not  [3] IMPLICIT SEQUENCE{Filter} }
```

図１８．注意すべきＡＳＮ．１記述
下線部分が必要。SEQUENCEで識別子のｺﾝﾃｸｽﾄを切替

ＡＳＮ．１でＰＤＵを定義する場合、このような規約に違反しないよう気を付けなければならない。

７．おわりに

本稿では、ＡＳＮ．１基本符号化規則の複雑な部分を紹介し、ＡＳＮ．１仕様で記述された上位層ＰＤＵを組立て／分解するＣ関数ＡＳＮ．１ハンドラについて報告した。今後、プレゼンテーション層／応用層のプログラムを実現する方の参考になれば幸いである。また、ＡＳＮ．１記述でＰＤＵ形式を定義する場合、間違い易い構文についても指摘したので、各プロトコルのＡＳＮ．１記述の検証に役立てていただきたい。

本関数は開発後既に約一年が立ち、ＡＳＴジェネレータを利用して、FTAM、ACSE、ﾌﾟﾚｾﾞﾝﾃｰｼｮﾝの他、様々な応用サービス要素および社内プロトコルの分散処理製品(ﾘﾓｰﾄﾌｧｲﾙｱｸｾｽ、集中管理機能）に使用中である。今後より使い易い汎用関数とする改善を図っていく予定である。

最後に、日頃御指導、支援していただいた基本ソフトウェア開発本部の皆様に感謝いたします。

[参考文献] [1]ISO 8824 "Specification of Abstract Syntax Notation One(ASN.1)"
[2]ISO 8825 "Specification of Basic Encoding Rules for Abstract Syntax Notation One(ASN.1)"