マルチメディア通信と分散処理 39 － 4
（１９８８．９．１６）

ASN.1支援ツールの開発
- コンパイラおよびエディタ -

長谷川亨† 野村真吾† 堀内浩規††
†国際電信電話株式会社 上福岡研究所
††(株) オーエスアイ・プラス

開放型システム間相互接続(OSI)のアプリケーションプロトコル(AP)では、プロトコル要素(PDU)のデータ構造が標準記法ASN.1により定義され、そのデータ構造から転送するバイト列への符号化規則も同様にASN.1で定められている。このデータ構造および符号化規則は複雑なため、APプログラムを実装およびテスト(製品検証)する場合、ASN.1で定義したPDUの符号化/復号(エンコード/デコード)プログラムの作成およびデコードしたPDUの解析等の処理を全て人手により行うことは現実的でない。そこで、筆者らはこれらの処理を自動化するために、APプログラムの実装時のツールとして、ASN.1による記述からCのデータ型および専用的なC言語のコーダプログラムを自動的に生成するコンパイラを作成した。さらに、APプログラムのテスト時のツールとして、PDUを作成/解析するエディタおよびPDUとバイト列の間を変換する汎用的なコーダプログラムを作成した。本稿では、これらのASN.1支援ツールの設計概要について報告する。

Development of Software Tools for ASN.1
- Compiler and Editor -

Toru HASEGAWA† Shingo NOMURA† Hiroki HORIUCHI††
† KDD Kamifukuoka R & D Labs. 2-1-15, Ohara, Kamifukuoka-shi, Saitama, 356
†† OSI Plus Corp. 2-1-23, Nakameguro, Meguro-ku, Tokyo, 153

Data structures of protocol data units (PDUs) in OSI Application layer protocols are defined in Abstract Syntax Notation One (ASN.1), and conversion rules between these data structures and transferred octet strings, called encoding rules, are also defined in ASN.1. Since the data structures and encoding rules are complicated, implementation costs of programs for handling PDUs defined in ASN.1, such as coder programs, are increasing. In order to reduce these costs, we have developed software tools for ASN.1, such as a compiler, editor and coder. The compiler is used for implementations, and generates data types and coder programs in C language for each data type in ASN.1 specifications. The editor and coder are used for debugging of Application protocol programs. The editor is used to make or analyze PDUs, and the coder converts the PDUs made by the editor to octet strings, and vice versa.

1. はじめに

近年標準化が進捗しているOSIのアプリケーションプロトコル(AP)およびプレゼンテーションプロトコル(PP)では、プロトコル要素(PDU)のデータ構造が標準記法ASN.1[1,2]により定義され、そのデータ構造から転送するバイト列への符号化規則も同様にASN.1で定められている。このデータ構造および符号化規則は複雑なため、APおよびPPプログラムの実装では、ASN.1で定義したPDUを符号化/復号するコーダ(エンコーダ/デコーダ)プログラムの作成コストが問題となる。また、実装したプログラムの製品検証を行う場合、ASN.1で定義したPDUの作成や受信したPDUの解析は下位層のプロトコルと異なり非常に煩雑な処理となり、全て人手により行うことは非効率的である。

そこで、筆者らはこれらの作業を自動化するために、ASN.1支援ツールとして、ASN.1コンパイラ、エディタおよび汎用コーダを作成した。本稿では、これらの支援ツールの機能および設計の概要について報告する。

2. ASN.1支援ツールの機能

2.1 ASN.1の概要

ASN.1はOSIの上位2層(アプリケーションおよびプレゼンテーション)プロトコルのPDUのデータ構造を定義するための標準記法である。個々のプロトコルで使用されるPDUのデータ型は、ASN.1の提供する組み込み型を用いて定義され、定義されたデータ型の組は抽象構文と呼ばれる。組み込み型は表1に示すように、単純型および構造型に分類される。単純型は整数やブーリアンのように構造を持たず、直接に値を示すデータ型である。構造型はSEQUENCE, SETのように構造を持ち、構成要素の値を持つデータ型であり、構造型を用いることにより新しいデータ型を定義することが可能になる。また、ASN.1では定義されたPDUと通信し合うシステム間で転送するバイト列の間の変換規則が符号化規則として規定されており、転送構文と呼ばれる。

表1 ASN.1の組み込み型

単純型	値	構造型	値
BOOLEAN	ブーリアン	SEQUENCE	固定長の順序を持った構成要素の値のリスト
INTEGER	整数	SET	固定長の順序を持たない構成要素の値のリスト
BIT STRING	ビット列	SEQUENCE OF	0個以上の順序を持った構成要素の値のリスト
OCTET STRING	オクテット列	SET OF	0個以上の順序を持たない構成要素の値のリスト
IA5STRING	キャラクタ列	CHOICE	構成要素の値のリストからの選択値

2.2 ツールの機能

APプログラムの実装および製品検証を支援するツールには、以下の点が要求される。

① 実装時のツール：コンパイラ
- ・実装者がPDUのコーダプログラムを作成する手間を省くために、コーダを自動生成する機能。
- ・実装者が行うPDUのフォーマッティングや解析等のプログラム作成を支援する機能。

② 製品検証時のツール：エディタおよび汎用コーダ
- ・テスト入力となるPDUの作成や受信PDUの解析を支援する機能。
- ・PDUをエンコード/デコードする機能。
- ・様々な抽象構文を扱える汎用性。

そこで、実装時のツールとしてASN.1コンパイラを、製品検証時のツールとしてエディタ、汎用コーダおよびASN.1パーザーを提案する(図1)。

これらのツールはC言語により実装し、C言語のプログラムを生成するが、それぞれのツールの実装に際して以下の基本方針を立てた。

(1) ASN.1の言語仕様

実数型や列挙型を含む最新の仕様[1,2]をサポートする。ただし、部分型をデータ型の概念の弱いC言語で扱うことは複雑であるためサポートしない。

(2) コンパイラ

実装者がCの変数としてPDUを扱うことができるように、コンパイラはPDUの定義から、コーダだけでなくPDUのデータ型およびPDUを処理するために必要な関数も生成する。汎用のコーダをあらかじめ用意する方法[3,4]も考えられるが、APプログラムの実装に使用することを想定し、実行時の速度を考慮して専用のコーダを生成する。

(3) エディタおよび汎用コーダ

様々な抽象構文により定義されたPDUを扱えるように汎用的なプログラムにする。そのため抽象構文で定義された個々のPDUを汎用的なテーブルにより表現し、それぞれのツールはこのテーブルを用いて処理を行うようにする。また、抽象構文を

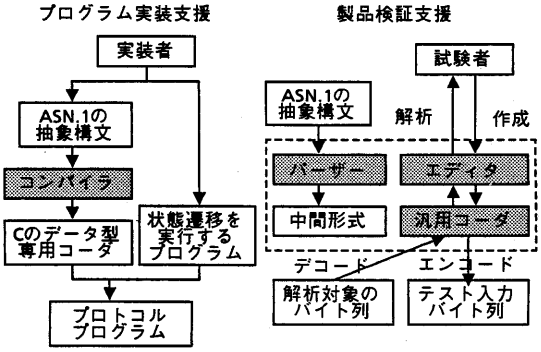

図1 ASN.1支援ツールの機能

PDUのテーブルに変換するパーザーも併せて用意する。

これらのツールはAPおよびPPプログラムの実装時および製品検証時に、以下の通り使用される。

① 実装時

実装者は、まずコンパイラに抽象構文を入力して、PDUのデータ型および専用のコーダプログラムを生成する。次に、生成したプログラムを用いてPDUのフォーマッティング等の動作を記述して、状態遷移を実行するプログラムを作成する。最後に、これらのプログラムをコンパイル/リンクすることにより実行プログラムを得る。

② 製品検証時

試験者は、まずASN.1パーザーにより抽象構文をPDUのテーブルに変換する。その後、エディタを起動してPDUの作成をエディタにより行う。また、受信PDUは汎用コーダによりデコードした後、エディタにより解析を行う。一般にASN.1で定義されるPDUのデータ構造は下位層のプロトコルのPDUに比較して複雑なため、エディタおよび汎用コーダを用いることにより上記の作業を効率的に行うことができる。

以下では、それぞれのツールに関して、機能の詳細および設計の概要について述べる。

3. ASN.1コンパイラ

3.1 方針

コンパイラはASN.1で定義されたデータ構造をデータ型とみなし、定義されたデータ型毎に、

① 対応するCのデータ型

② コーダ(エンコーダ/デコーダ)としてのCの関数

③ 生成したCのデータ型を持つ変数を操作するのに必要なCの関数

を生成する(図2)。

単純型および構造型の一部(SEQUENCE OF, SET OF)は、常に同一のCのデータ型に対応するため、あらかじめデータ型および関数をそれぞれ共通のヘッダファイル、ソースファイルとしてプログラムを作成しておく。このプログラムは、コンパイラが生成するプログラムが使用する。

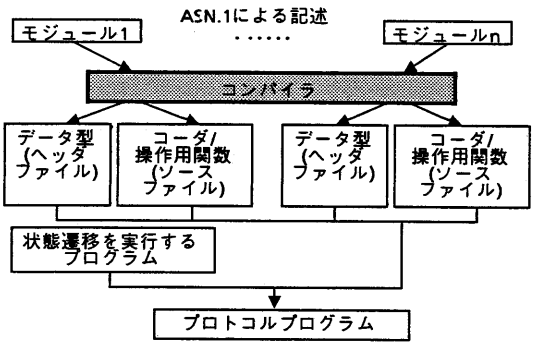

図2 ASN.1コンパイラの機能

ASN.1では一まとまりのデータ型の集合をモジュールとして定義するため、本コンパイラは複数のモジュールを入力とし、モジュール毎に、①をヘッダファイル、②、③をまとめてソースファイルとして出力する。また、他のモジュールのデータ型を参照することがあり、必要なモジュールのヘッダファイル、ソースファイルをC言語のinclude文、extern宣言により参照する。

3.2 生成するプログラム

3.2.1 データ型

ASN.1により定義されたデータ型をコンパイラがCのデータ型に変換し、PDUのフォーマッティングおよび解析は、このCのデータ型を持つ変数に対する値の設定および参照により行う。

以下の方針によりASN.1のデータ型をCのデータ型に対応させるが、新しく定義された全てのデータ型に対して、その型を識別するための識別子を与え、コンパイラが整数の定数として生成する。この識別子はコンパイラが生成するコーダの関数が使用する。

(1) 単純型

整数、ブーリアンはCの整数に対応させ、それ以外はCの構造体に対応させる。例えば、文字列、オクテット列等のストリングは全てストリングの長さとストリングへのポインタからなる構造体に対応させる。また、ストリングはキャラクタ列として扱う。

ASN.1では名前付き整数およびビットストリングのビット位置に名前を与えることができるため、コンパイラはこれらの定義された値をdefine文により整数の定数としてヘッダファイルに生成する。これにより、実装者はこの名前により名前付き整数を扱うことができ、ビットストリングのビット位置の指定にその名前を利用できる。

(2) 構造型

SEQUENCE, SETにより定義されるデータ型は、要素のデータ型をメンバとする構造体をコンパイラが生成する。OPTIONALと指定することにより値の存在が任意な構成要素に対しては、値の存在を示すフラグを設ける。

複数の要素の内の1つを選択するCHOICEにより定義されるデータ型は全て同一の構造体に対応させ、複数の要素のデータ型を定義するCの共用体へのポインタを持たせる。また、選択された要素を指定する識別子を構造体に持たせる。

要素として同一のデータ型の値を複数持つSEQUENCE OF, SET OFにより定義されるデータ型は全て同一の構造体に対応させ、その要素をリンクにより繋いでリストとして持つ。また、これにより定義された異なるデータ型を識別するための整数のフィールドを設け、コンパイラが生成した識別子を設定する。

図3にASN.1の抽象構文から生成されるCのデータ型の例を示す。

```
(1) ASN.1による定義
Age ::= INTEGER
Name ::= IA5STRING
IdNumber ::= INTEGER
PerRec ::= SEQUENCE {
 name Name, age Age OPTIONAL, num IdNumber }
PerRecList ::= SEQUENCE OF PerRec
AgeList ::= SEQUENCE OF INTEGER
(2) SEQUENCE OF のデータ型
typedef struct lst { /*要素のリストのためのリンク*/
    struct lst *next ; /*次のリンクへのポインタ*/
    char *elem ; } LST ; /*要素へのポインタ*/
typedef struct {
    int memid ; /*要素のデータ型の識別子*/
    LST *list ; /*要素のリストへのポインタ*/
    int size ; /*要素の個数*/
    int esize ; /*コーダが使用*/ } seqof ;
(3)生成されるデータ型の識別子
#define IDAge 0 /*Age*/
#define IDIdNumber 1 /*IdNumber*/
#define IDName 2 /*Name*/
#define IDPerRec 3 /*PerRec*/
#define IDPerRecList 4 /*PerRecList*/
#define IDAgeList 5 /*AgeList*/
(4)生成されるデータ型
typedef struct PerRec_seq {
    IA5STRING *name;
    int age;
    BOOLEAN age_flag;
    IdNumber num; } PerRec;
typedef seqof PerRecList ; typedef seqof AgeList ;
```

図3 生成されるCのデータ型

(3) ANY

ANYに関しては、APプログラムの作成を容易にするため、以下のような拡張を行っている。

アプリケーション層はさらに階層構造を持ち、上位のAPのPDUは下位のAPのPDUのユーザデータとなる。ASN.1による下位のPDUの定義ではユーザデータをANYとして明確には定義しない方法、またはANYに対応するデータ型をdefined byにより指定する方法があるが、複数の上位PDUがANYに対応する場合の定義はできない。そこで、ANYのdefined byを拡張して、ANYに対応する可能性のある複数個のデータ型を指定できるようにしている。明確にデータ型を定義しないANYはオクテット列として扱うことにし、defined byによる指定を用いた場合は、オクテット列と指定したデータ型のCHOICE(選択)として扱う。

3.2.2 操作用の関数

本コンパイラを用いて作成するAPおよびPPプログラムでは、PDUのフォーマッティングは、3.1のASN.1のデータ型を持つ変数に値を設定しエンコーダの関数を呼ぶことにより、解析は受信バイト列をデコード関数によりASN.1の変数に変換し、その変数を解析することにより行う。そこで、この変数の作成/解析を容易にプログラムできるように、ASN.1のデータ型を操作するのに必要な関数を準備しており、主な関数を表2に示す。

ASN.1のデータ型を持つ変数ではSEQUENCE OFのリンクと要素等のメモリ領域を動的に割り付け/解放する必要があり、そのための関数もあわせて準備している。これらの処理は通信時に頻繁に行われるため、領域を最初にまとめて割り付け(バッファエリアと呼ぶ)、その領域から必要に応じてメモリを取ってゆく構成とする。そこで、作成したPDUの変数のエンコード、送信または受信PDUのデコード、解析が終了して、変数が不要になった時点でバッファエリアを解放する必要がある。

メモリ割り付けは通常のmallocを用いることも可能であり、このどちらかをユーザが選択することができる。表2のSEQUENCE OFに対して用意した関数は、リンクのためのメモリ割り付けに、バッファエリアから割り付ける関数およびmallocを使用する関数の2つを準備している。

ASN.1の型	操作
SEQUENCE OF	要素を先頭または最後に挿入 要素を先頭または最後から取り出す (リンクをバッファエリアに取るまたはmallocで取るものの2種類)
SET OF	要素の挿入、取り出し(上と同様)
BIT STRING (ビット列)	任意の位置のビットのオン/オフおよび検査 ビット列の比較

表2 操作用の関数

3.2.3 コーダプログラム

生成するコーダプログラムは以下の特徴を持つ。

(1) プログラム構成

コーダプログラムは関数により実現し、処理速度向上の観点から、汎用的な関数ではなくデータ型毎に専用の関数を生成する[5]。SEQUENCE, SEQUENCE OF等の構造型に対する関数は、要素のデータ型を処理する関数を呼び出す形式である。

(2) エンコード/デコード方式

エンコード/デコードの方式に関しては、先に作成したASN.1からAdaへのトランスレータ[5]とほぼ同様であり、以下の特徴を持つ。

① ASN.1のデータ型ではバイト列への変換のしかたを指定するために、複数個のタグを付加すること

ができる。付加されたタグのエンコードはタグの出現のしかたに応じて異なり、ASN.1のデータ型のエンコードも付加されたタグにより異なる。そこで、付加された複数のタグからタグの配列(idアレイ)を生成し、コーダはこのidアレイを用いてタグ列のエンコード/デコードを行う。idアレイは、定義されたASN.1のデータ型および構造型の要素毎に生成され、全体もまた配列の定数として生成される。図4にidアレイのデータ型を示す。

```
タグのデータ型
typedef struct {
    int idsize; /*符号化されたタグのオクテット長*/
    unsigned char idelem[4]; /*符号化されたタグ*/
    } id_cell;
idアレイのデータ型
typedef struct {
    int idlevel; /*タグの個数*/
    id_cell id[maxtag]; /*タグの配列*/} id_array;
```

図4 idアレイのデータ型

② ASN.1のデータ型は、識別子、長さ、値の組からなるバイト列にエンコードされる。構造型を用いて入れ子になったデータ型では、1つの型を最後まで処理しないと長さが決定できないため、エンコードの関数はASN.1の変数がエンコードされた時の長さの決定とバイト列への変換の2パスで処理を行い、コンパイラはそれぞれに対して関数を生成する。ここで、バイト列はASN.1のOCTETSTRINGとして扱う。

(3)呼び出し関数

プロトコルプログラムが個々のPDUをエンコード/デコードする場合に、同一の関数名で呼び出せるように、全てのデータ型に識別子をコンパイラが付与し(3.2.1参照)、呼び出し時に引数として関数に与える。この関数は識別子に対応するエンコード/デコード関数を呼び出す。

(4)デコード関数

デコード関数はPDUのバイト列を解析し、その結果をASN.1のデータ型を持つ変数として、ユーザが指定したバッファエリア(3.2.2参照)上に作成する。従って、ユーザは解析したPDUが不要になった時点で、このエリアを解放する必要がある。また、デコード関数はデコード時に発生したエラー情報を構造体により返す。エラー情報は正しくデコードできたかどうか、エラーが発生した場合はエラーの種別および発生したバイト位置等である。

図5にコンパイラが生成するSEQUENCEのデコード関数の例を示す。

3.3 プロトコルプログラムの記述例

ASN.1コンパイラを用いたプログラムの作成例としてプレゼンテーションプロトコル(PP)プログラムの記述の一部を示す。コネクション確立要求時に

```
decodePerRec(var,edata,buf,index,error,ida)
 PerRec **var; /*デコード結果へのポインタ*/
 string *edata; /*デコードするバイト列*/
 BUFFER *buf; /*デコード結果を生成するバッファ*/
 int index; /*デコードを開始するバイト位置*/
 ERROR error; /*エラー情報*/
 id_array ida; /*idアレイ*/
{ int nextobj, blength;
if ((*error).errlevel<2 /*エラーが無かった?*/)
{ decodeIdLen(edata,index,&blenth,ida,error);
    /*タグ列の復号化*/
 nextobj = index + blength;
    if((*var = asnalloc(buf,sizeof(PerRec))) = = 0) {
    /*PerRecのメモリ割り付け*/
      ERRSER(FATAL,6,ida,index); /*エラー情報の設定*/
      return; }
 (**var).age_flag = false;
 if ((*error).errlevel<2 /*エラーが無かった?*/)
 { decodeIA5strng(&(**var).name,edata,buf,index,
        idinfo[nameのIdアレイの番号]);
    /*Nameのデコード関数呼び出し*/
 if ((*error).errlevel<2 /*エラーが無かった?*/)
  if(idcheck(edata,index,idinfo[ageのIdアレイの番号])
    /*OPTIONALな要素Ageが存在する?*/
  { decodeInteger(&(**var).name,edata,buf,index,
        idinfo[ageのIdアレイの番号]);
      /*Ageのデコード関数呼び出し*/
     (**var).age_flag = true; } }
 if ((*error).errlevel<2 /*エラーが無かった?*/)
 { decodeInteger(&(**var).name,edata,buf,index,
        idinfo[numのIdアレイの番号]); }
    /*EmployeeNumberのデコード関数呼び出し*/
 if(index! = nextobj) /*全体の長さチェック*/
 { ERRSER(FATAL,2,ida,index); }
} } }
```

図5 デコード関数

応用層から通知される抽象構文(AbstractSyntaxName)から、転送されるデータ構造を定める転送構文(TransferSyntaxName)を選択し、Connect Presentation プロトコル要素(CP-PPDU)を作成し、送信する部分を説明する。

```
ISO8823PRESENTATION DEFINITIONS ::=
BEGIN
- CP-PPDU
CpPpduType ::= SET { [2] IMPLICIT NonX410mode
    OPTIONAL }
NonX410mode ::= SEQUENCE
   { [3] IMPLICIT PresentationContext OPTIONAL }
PresentationContext ::= SEQUENCE
   { AbstractSyntaxName,TransferSyntaxNameList }
TransferSyntaxNameList ::=
   SEQUENCE OF TransferSyntaxName
- PCON req
PConReqParamType ::= SEQUENCE
   {PContextItem PContextItemType OPTIONAL }
PContextItemType ::= SEQUENCE { AbstractSyntaxName }
END
```

図6 モジュールの記述

まずCP-PPDUのデータ型はCpPpduTypeとしてモジュールISO8823PRESENTATIONに定義される(図6)。ASN.1はプリミティブの任意のパラメータの表示などに有効であり、またプリミティブとプロトコル要素の対応するパラメータの型を一致させることにより記述が見易くなるため、プリミティブのパラメータの宣言にもASN.1記述を使用する。

PCONreqのパラメータの型も同様にモジュール内に記述する。

次に、このモジュールをASN.1コンパイラに入力し、ヘッダファイルとソースファイルを得る。PPプログラムは、ASN.1の共通ヘッダファイルおよびこのヘッダファイルをincludeすることにより作成する。図7にそのメイン関数を示す。ここではPCONreqを受信した後、関数FormatCpによりCP-PPDUをフォーマッティングし、それをエンコードしてSCONreqのパラメータとして出力する。SEQUENCE OFにより定義されたTransferSyntaxNameList等の要素はリストとして扱われ、このリストのリンクはバッファエリアに割り付け、CP-PPDUをエンコードした後、不要になった時点で解放している。

```
#include <stdio.h>
#include ISO8823PRESENTATION.h
#include ASN1.h
main ()    {
    OCTETSTRING *encdata; /*エンコードデータ*/
    BUFFER buf; /*バッファエリア*/
    CpPpduType *CP; /*CP-PPDU*/
    PconReqParamType *PCON; /*PCONreq*/
/*PCON req受信時の動作*/
    buf.size = 1024; /*バッファエリアの大きさ指定*/
    asn_alloc(&buf); /*バッファエリアの割り付け*/
    FormatCP(CP,PCON,&buf); /*CP-PPDUのフォーマット*/
    encode(CP, encdata, IDCpPpduType); /*エンコード関数
    の呼び出し*/
    /*encdataをユーザデータとしてSCONreqを送出*/
    asn_free(&buf); /*バッファエリアの解放*/
}
```

図7 PPプログラム

最後に、CP-PPDUをフォーマッティングする関数FormatCpの記述例を示す(図8)。ここではPCONreqのパラメータPCONを受け取り、CpPpduType型の変数の各要素をセットする。

```
FormatCp ( CP, PCON, buf )
    CpPpduType *CP; PconReqParamType *PCON;
    BUFFER *buf;
{ PresentationContext *PC;
 AbstractSyntaxName *AbsSyn;
 if (PCON->pContextItem_flag = TRUE) {
    CP->NonX410mode_flag = TRUE;
    for(i = 0; i<PCON->pContextItem.size; i + +){
     remh( PCON->pContextItem, AbsSyn);
     /*SEQUENCE OFの要素(AbstrctSyntaxName)の取り出
     し*/
     PC = asn_malloc(sizeof(PresentationContext));
     /*バッファエリアからPCの領域の割り付け*/
     DefineTranferSyntaxList( PC, AbsSyn);
     /*抽象構文から転送構文を決定する関数の呼び出し*/
     CP->NonX410mode->PresentationContext->
       AbstractSyntaxName = AbsSyn;
     addh( CP->NonX410mode->PresentationContext, PC,
    *buf); /*PCをSEQUENCE OFの要素として挿入*/
    }
} }
```

図8 関数FormatCp

4. エディタおよび汎用コーダ

4.1 機能

APプログラムを製品検証するためのツールは、PDUのエディタ、汎用コーダおよびASN.1パーザーから構成される(図9)。まず、ASN.1パーザーは、ASN.1の抽象構文により定義されたPDUのデータ型を表現する中間形式(テーブル)をCの構造体の形式でメモリ上に生成する。一般にPDUはASN.1の構造型を用いて定義され、この中間形式はPDUの構造を示す木構造になる(PDUのデータ型の木)。エディタおよび汎用コーダはこのテーブルを用いて、個々のPDUに対する処理を行う汎用的なプログラムである。エディタにおいて対象とするデータ型の木を指定する必要があるため、ASN.1のデータ型と対応するデータ型の木へのポインタを持つ名前表もあわせてパーザーが生成する。

個々のPDUもまた、データ型の木と同様なCの構造体を用いた木構造(PDUの木)により表現され、汎用コーダはこの木構造とバイト列の間の変換を行う。ただし、エディタおよび汎用コーダが使用する木のノードの形式は、必要な情報が異なるため、異なる形式としている。

これらのツールを使用する場合、まずASN.1パーザーにより抽象構文をデータ型の木に変換した後、エディタを起動してPDUの木を作成し、汎用コーダを用いて送信PDUをバイト列にエンコードして外部ファイルに蓄える。

外部ファイルに蓄えられた受信PDUのバイト列は、エディタから汎用コーダを起動してPDUの木にデコードして、エディタによりその中身を解析

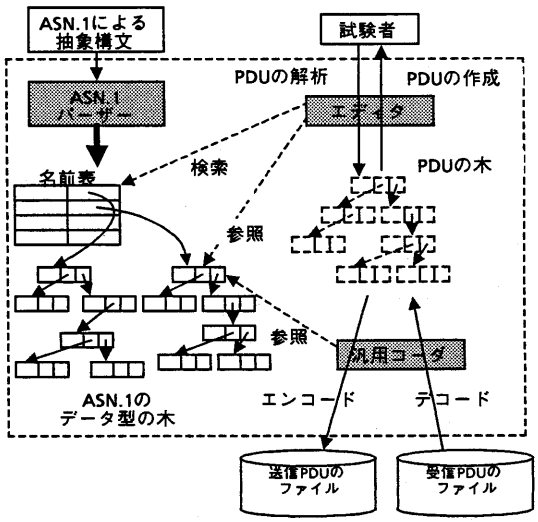

図9 エディタおよび汎用コーダの機能

する。また、一度編集したPDUを汎用コーダによりデコードして、再編集することも可能である。

エディタおよび汎用コーダに関しては、先に作成したMHSの製品検証テスタ[6]のエディタとコーダを基にして、その機能拡張により実現している。

4.2 木構造

(1) データ型の木構造

抽象構文で定義されたPDUはCの構造体を用いた木構造に変換されるが、各ノードはASN.1で定義されたデータ型、データ型に付加されるタグ毎に生成される。また、定義されたデータ型は他のデータ型の定義に使用されるため、データ型名とそのノードへのポインタを名前表に登録する(4.1参照)。

各ノードに対応するCの構造体はASN.1のデータ型に関する情報として、

- データ型の識別子(整数、SEQUENCE、タグ等)
- 名前
- 識別子オクテットを規定するクラス、コンストラクタ、識別子番号等の情報
- 構造型の要素のノードへのポインタ列
- 構造型の各要素のノードに関する情報
- ノードに対応するデータ型の値が設定されたかどうかを示すフラグ
- 親ノードへのポインタ

等のメンバを要素として持っている。

構造型の要素の情報を持つノードは、OPTIONALを示すフラグ、リファレンス、デフォルト値、部分型により定義されたデータ型への制約等の情報を持ち、エディタを使用するユーザに対するガイドラインとして、この情報を表示できるようにしている。ただし、部分型に関しては、エンコード/デコード時に部分型の制約のチェックは行っていない。

(2) PDUの木構造

編集およびエンコード/デコード対象となるPDUの木構造の各ノードは、(1)と異なり、汎用コーダがエンコード/デコードするのに必要な情報だけを保持している。例えば、識別子オクテットに関する情報、親および子へのポインタ、エンコード時の長さおよび実際の値等の情報である。

4.3 エディタ

エディタはユーザの指定したデータ型の木構造(4.2(1)参照)に従って各ノードの情報を画面に表示し、ユーザがノードの追加/削除およびノードの値の中身を書き込みまたは変更できるようにする。ここで、一画面が1つのノードに対応している。また、編集を支援するためのコマンドとして、表3に示すものを用意している。

コマンド	操作
number	構造型の要素の個数表示
returnキー	親ノードの編集画面へ戻る
delete	指定する構造型の要素の削除
save	編集したPDUをファイルに保存
help	コマンドの表示
quit	終了

表3 編集用のコマンド

以下では図10に示す抽象構文のPDUを例にしてエディタによる編集方法およびエディタの動作について述べる(図11)。

```
PerRec ::=  SEQUENCE {
    familyname [0] IMPLICIT Name,
    firstname [1] IMPLICIT Name OPTIONAL,
    age [2] IMPLICIT Age OPTIONAL }
Age ::=  INTEGER (22..60)
Name ::=  IA5STRING size(1..20)
IdNumber ::=  INTEGER
```

図10 抽象構文の例

PDUを作成するためには、エディタを起動する前に対象とする抽象構文をASN.1パーザによりデータ型の木(中間形式)に変換する。その後エディタを起動して、対象とするPDUのデータ型名を指定する。また、先に作成して外部ファイルに蓄えたPDUおよび受信PDUを再編集および解析する場合は、さらに対象とするPDUのファイルを指定する(図11の(a))。この時、エディタはデータ型の木へのポインタを引数として、汎用コーダを呼び出して外部ファイルに蓄えたPDUのバイト列を値の木にデコードする。

エディタはユーザの指示に従って、対象とするデータ型の木をたどりノードの情報を表示し、ユーザはその表示に従って値の編集を行う。新しく値が設定された場合は値の木にそのノードが追加され、エディタは値の木を作り上げてゆく。ノードが構造型の場合、編集画面には要素のデータ型名およびその値、長さが表示され、データ型名に付与された番号を入力することによりその要素の編集画面に移る(図11の(b))。ここで、OMITTEDと記された要素は値が無いことを示している。

値を編集する必要があるノードでは、その旨が表示され、プロンプトの後に新しい値をキー入力できる(図11の(c))。この編集画面には部分型により定義されたデータ型への制約等の情報も表示される。また、キー入力を用いず外部ファイルからの入力も可能であり、ASN.1のデータ型を単位として入出力が可能である。編集が終了するとsaveコマンドにより、汎用コーダが呼び出され値の木はバイト列にエンコードされて外部ファイルに保存される。

```
*** Editor start ***
# INPUT Data Type Name
> PerRec
# INPUT Data File Name(Return at making a new PDU)
> Hasegawa.dat
```

(a) エディタの起動

```
PerRec              ID : U-C-16 (30)h "Sequence"
1. familyname       L[8] V[HASEGAWA]
2. firstname        OMITTED
3. age              L[1 ] V[30]
CMD> 2
```

(b) PerRecからFirstNameを選択

```
      Name
1. Name                          OMITTED
<VALUE INPUT>
    Name        ID : U-P-22 (16)h "IA5String"
    SINGLE VALUE
    RANGE 1..20
    VALUE LENGTH = 0
    VALUE = []
INPUT DATA (From File: '*' + filename)
> HASEGAWA
```

(c) firstnameに値を入力

図11 エディタのユーザインタフェース

4.4 汎用コーダ

(1) エンコーダ

エンコーダはエディタにより作成した値の木を以下の手順でバイト列にエンコードする。

① 値の木を辿り構造型の値の長さを木のリーフから計算し、その値を対応するノードに記録する。

② 値の木をルートから辿り、ノードに記された情報に従って、先頭からバイト列に変換する。

エンコーダはデータ型の木は参照せず、値の木だけを参照するため、値の木がデータ型の木の示す抽象構文に従っていなくても、バイト列にエンコードできる。これにより製品検証時のテスト入力として構造型の要素が抜けている等の誤ったPDUを作成することが可能になる。

(2) デコーダ

デコーダはバイト列が正しく抽象構文に従っているかどうか調べる必要があるため、データ型の木を参照し、バイト列を検査しながら値の木を作成してゆく。バイト列に誤りがあった場合は、デコードエラーをエディタに表示する。

5. 考察

提案したASN.1のツールに関して以下の考察を得た。

(1) ASN.1 コンパイラ

現在、コンパイラは実装中であるが、実装終了後、生成プログラムの規模および処理速度を評価する予定である。コーダは専用のプログラムであるため規模は大きくなるが、高速な処理ができると考えられる。

コンパイラが生成するデータ型および関数を用いてPPプログラムの一部を作成し、提案した手法により容易にプロトコルプログラムを作成できることを確認した。

遠隔手続き呼び出し(RPC)プロトコルを実現するROSE上に構築されるMHSやディレクトリ等のプロトコルにおいては、遠隔手続き(オペレーション)の実現が重要であり、現在、ASN.1コンパイラの機能を拡張して、抽象サービス定義を入力としてC言語のスタブを生成するスタブジェネレータの検討を行っている[7]。

(2) エディタおよび汎用コーダ

PDUの木構造をOSにおけるのディレクトリと同様にとらえて、各ノードの表示、挿入および削除をUNIXのls, mkdirコマンドと同様なコマンドを用いて行う方法[3]も考えられるが、本エディタではノード毎に編集画面単位で編集する方法を取っている。編集時に編集画面に抽象構文で定義されたリファレンス等の情報を表示することにより、使用者に対して有効なガイドラインを与えることができるため、効率的な編集が可能になると考えられる。

6. おわりに

本稿ではASN.1支援ツールの設計概要について報告した。現在、この設計に基づいて実装中であり、実装後その評価を行うとともに機能の拡張を進める予定である。最後に日頃御指導頂くKDD上福岡研究所小野所長、湯口次長、小西通信ソフトウェア研究室長、鈴木コンピュータ通信研究室長、若原通信ソフトウェア研究室主任研究員に感謝する。

参考文献

[1]:ISO,"ASN.1", ISO / IS 8824 / 8825

[2]:CCITT, Rec. X.208 / 209 , Nov. 1987.

[3]:中川路,勝山,水野, "ASN.1ツールAPRICOTの設計", 情報処理学会第35回全国大会, 5U-9, Oct. 1987.

[4]:安部,中村,中川, "X.409支援システムの考察", 情報処理学会設計自動化研究会資料, 40-15, Dec. 1987.

[5]:長谷川, 野村,加藤,堀内,"EstelleとASN.1に基づくプロトコル仕様からAdaへのトランスレータ",電子情報通信学会 情報ネットワーク研究会, IN87-74, Nov. 1987.

[6]:加藤,長谷川,堀内,鈴木, "MHSテスタの作成", 情報処理学会第35回全国大会, 5U-10, Oct. 1987.

[7]:長谷川,西山, 野村, "OSIアプリケーションプロトコルのためのスタブジェネレータへの抽象サービス定義の適用", 情報処理学会第37回全国大会, 2E-4, Sept. 1988.