ASN.1 and Estelle Implementation support tools

Gregor v. Bochmann, Daniel Ouimet
Université de Montréal
and
Gerald Neufeld
University of British Columbia

Abstract

Formal specifications are a well-known technique for improving software devel
the context of OSI communication protocol standards, Formal Description T
(FDT's) have been developed for the description of communication protocols a
In addition, a notation called ASN.1 is used for the descriptions of the data st
protocol data units exchanges between communicating entities at the applic
Existing FDT's, such as Estelle, LOTOS and SDL, do not include facilities to d
manipulate data structures defined in ASN.1. This makes using FDT's for d
applications difficult. This paper deals with the integration of ASN.1 with Este
issues involved with the integration of corresponding implementation tools.
how the encoding and decoding routines automatically generated from t
definitions can be combined with implementation code semi-automatically gen
the Estelle specification of the protocol. An application for a simple protocol is g

1. Introduction

Standards for communication protocols and services are being developed for O
Interconnection (OSI) [OSI 83] which are intended to allow the interwo
heterogeneous computer systems and applications. In this context, the
specifications are of particular importance, because they represent the standa
the basis for the implementation and testing of compatible OSI systems. Th
specification defines the behavior of a protocol entity, in terms of its interacti
local service user, called service primitives, and its interactions exchanged with
entity, called Protocol Data Units (PDU's). The following aspects of the behavi
specified (for more details, see [Boch 89d]):

(a) temporal ordering of interactions,
(b) range of possible interaction parameters (data types of parameters),
(c) rules for interpreting and selecting values of interaction parameters for ea
instance of communication, and
(d) coding of the PDU's.

In the case of OSI application layer protocols, the aspects (b) and (d) are usuall using a notation, called ASN.1 [ASN.1], which provides facilities for defining data types in terms of primitive data types and certain number of compositior data types in programming languages, such as Pascal. This notation also p definition of encoding rules [ASN.1 C] that are used to represent values of an data type for transmission.

ISO and CCITT have also developed standardized specification languages respectively called Estelle [Este 89], LOTOS [Loto 89], and SDL [SDL 87]. The can be used to describe the aspects (a), (b) and (c) above, but are less suita coding aspect (d). A formal specification of a protocol given in one of the FD used for the partial automation of the protocol implementation process [Boch text of the specification can be translated to obtain automatically part of the im code. In the case of the OSI protocols below the application layer, the coding defined in an ad hoc manner; reduction of implementation effort can only l through code sharing for encoding routines. In the case of application layer however, because of the regular structure of the ASN.1 encoding scheme, it is automate the production of encoding routines. It is therefore advantageous existing FDT implementation tools with the ASN.1 support tools. This is the to paper.

Unfortunately, such a combination is not straightforward. The main difficulty the fact that the FDT's have been developed separately from ASN.1 and als language constructs for defining data types independent of ASN.1. It is necessary to define a mapping between the ASN.1 data types and the corresp types of the FDT.

Section 2 of this paper describes an ASN.1 support tool which converts the PI OSI application layer protocol into C data types as well as generating the enc decoding routines. It also gives an overview of Estelle implementation sup discusses strategies for combining ASN.1 and FDT languages. Section 3 discu strategy for the case of the FDT Estelle [Este 89]. Section 4 present a coordina implementation support tools corresponding to the strategy described in Sectio 5 concludes this paper.

2. Implementation support tools for ASN.1 and Estelle

2.1. ASN.1 implementation tools

There are two aspects to ASN.1: (1) the so-called abstract syntax notation whicl define the data types of the PDU's and (2) the transfer syntax that is the representation of the instances or values of the data types.

ASN.1's abstract syntax notation is defined in an analogous way to most programming languages. A considerable amount of work has been done tc approaches for the implementation of the ASN.1 encoding rules and support t ASN.1 notation. Certain approaches only support the encoding rules, by routines that translate between the encoded form of PDU's used for transmiss internal data structures. Other approaches provide in addition that receiv verified to satisfy the structure defined by the ASN.1 definition of the given specification.

One can distinguish the interpretive and the compiling approaches. In the approach [Boch 86b], the ASN.1 definition is read by the ASN.1 tool during time and an internal representation, called the type tree, is constructed which by a set of fixed, generic coding and decoding routines included in the softwa This approach is useful for systems that must adapt to a variety of differe definitions, such as conformance testing tools. In the case of the compiling [Yang 88, Hase 88, ISOD 89], the ASN.1 definition is read and processed du ASN.1 compilation phase which reads the definition and produces programmir specific (i.e. C or Pascal) data type definitions for representing the internal valu set of coding and decoding routines to be incorporated in the protocol process It is important to note that the generated code is protocol specific and lends it natural data structures for the internal representation of the PDU's (i.e. value approach is therefore particularly interesting for protocol implementation proje

In the case of the CASN.1 tool [Yang 88], which adopts the compilation appr compiler generates programmer-intuitive data types which can be used directly parts of the protocol software. In order to encode a PDU, the programmer can the encoding routine for the "top-level" data element of the value tree represent This encoding routine calls other encoding routines for all the other data types When the top-level routine returns, the PDU is returned encoded in the tran

ready for transmission. Decoding an incoming PDU is very similar. On inpu level decoding routine is called, which calls all other descendant decoding rou decoding routine is responsible for allocating the associated C data structure an input value. On return,the top-level decoding routine points to the top-le structure.

The application PDUs defined in ASN.1 are provided as input to the compil produces four different files; one containing the corresponding data structures containing the encoding and decoding procedures (encode.c and decode.c containing initialization code to set up the environment.

2.2. Strategies for integrating ASN.1 with Estelle

In Estelle [Este 89, Budk 87], a specification module is modelled by an extended machine. The extensions concerning the range of possible values for in parameters, and rules for interpreting and selecting values of these parameters and (c) mentioned in the Introduction) are covered by type definitions, expr statements of the Pascal programming language. In addition, certain "Estelle s cover aspects related to the creation of the overall system structure consisting in hierarchy of module instances. Communication between modules takes place interaction points of the modules which have been interconnected by the par Communication is asynchronous, that is, an output message is stored in an inp the receiving module before it is processed.

In order to take advantage of the implementation tools available for Estelle [Boc 90] in conjunction with the ASN.1 notation used for OSI application layer p together with the automatic generation of coding and decoding routines, it is clearly define a relationship between the ASN.1 data structure definitions corresponding concepts of the respective Estelle. The relationship must be def levels. First, the correspondence between the respective language constructs f data types (aspect (b) of the Introduction) must be defined. Secondly, issues r combination of the ASN.1 and Estelle tools must be addressed. The followi approaches can be distinguished [Boch 90f] for the integration of ASN.1 with a substitution, (2) combination, and (3) translation.

In the translation approach, as described in this paper, the ASN.1 notation is tr equivalent constructs of the FDT. The formal specification of an OSI applicat protocol will therefore include the definition of the PDU structure as obtained b from the given ASN.1 definition contained in the protocol standard. For the a individual elements of this structure, the available constructs of the FDT can be

The substitution and combination approaches require a redesign of the FDT, major undertaking. In the case of the translation approach, the FDT remains In order to make this approach successful, however, it is necessary to define th in such a manner that the resulting PDU data type definitions are simple to rea for the designer of the FDT protocol specification. The details of the translatic in the case of the FDT Estelle is further persued in the subsequent sections. The similar for SDL. For the case of LOTOS, the issues are discussed in [Boch 89h].

Various tools have been developed independently for ASN.1 abstract syntax n the encoding rules, on the one hand, and for the Estelle on the other hand. Th of these languages should therefore also lead to an integration of the associat the case of the translation approach described above, the main issue for tool i the compatibility of the respective implementation data structures used in t representing the PDU's. As shown in Figure 1, the ASN.1 definition of the PD from the protocol standard document, is used for generating the ASN.1 encod which translate between the encoded transmission format and an internal da format used by the ASN.1 tool. The same ASN.1 definition is also translated data type definitions which become part of the formal protocol specificat specification is then translated by the FDT tool. For the integration of these therefore necessary that the implementation data structures representing the P by the translation from the FDT tool are the same as those used by the encod generated by the ASN.1 tool.

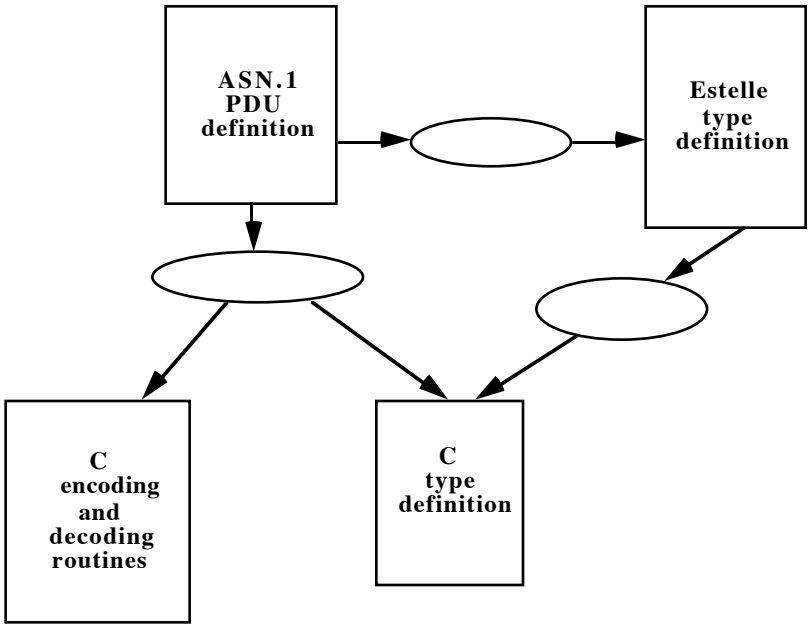

Figure 1 - General Overview

3. Translation from ASN.1 into Estelle/Pascal and C

In this section we discuss in more detail the translation from ASN.1 into Este
Since ASN.1 only covers aspects related to the range of possible PDU paramet
and data structures of PDU parameters, the result of the translation will be
definitions in the target language. In the case of Estelle, data types are defined
notation as in the Pascal programming language. The data typing facilities in
modern programming languages are similar. This is also the case for C. In the
we therefore mention the translation into C only in certain cases where its f
evident from the corresponding discussion for Estelle.

Most constructs of ASN.1 are similar to constructs existing in programming l
such as Pascal. For these constructs, a natural translation into Estelle and C i
straightforward, as discussed in Section 3.1 to 3.4. However, certain particular
often difficult to match, for instance the arbitrary precision of integers in AS
compatible with the fixed size integers in Pascal or C. Certain other constructs
such as SEQUENCE OF, have no equivalent in Estelle. It is therefore necessary
data types in Estelle (and C) which correspond to these ASN.1 structures, as d
Sections 3.5 throuth 3.7.

3.1. Predefined data types

The predefined data types of INTEGER and BOOLEAN have a correspor representation in Estelle. As mentioned above, the ASN.1 INTEGER has no upp This is the same in Estelle, however, most Estelle compilers impose resti corresponding to the target language, on the size of integers that can be hanc "integer" and "short" in C.

The ANY type of ASN.1 is a choice of an arbitrary number of types which is d runtime. This type can be represented in Estelle by the notation " ... " which m type is not yet defined and more information would be provided for an implen the implementation in C, it would correspond to a pointer to any kind of value.

The ASN.1 types representing time (Generalized Time and Universal Time represented in Estelle and C by a predefined record data structure, containir year, month, day, hour, minute, second, time-difference, and zone.

3.2. SEQUENCE and SET

An ASN.1 SEQUENCE or SET contains a certain number of elements of differen The only difference between the two constructs is the order of transmissio elements, which is sequential in the case of a SEQUENCE, and not defined in the SET. This difference has only an impact on the encoding, not the meanir structures. Therefore they can be translated into identical structures in Estell natural translation in Estelle is a "record", or a "struct" in C. For example, t definition

```
Type1 ::=       SEQUENCE { a INTEGER,  b BOOLEAN }
```

would be translated into Estelle as:

```
Type1  =        record (*@SEQUENCE*)
                a: INTEGER;
                b: BOOLEAN; end;
```

The translation of type structures from ASN.1 into Estelle also determines elements of the data types can be accessed by the body of the Estelle specifica Estelle includes a notation for accessing Estelle data structures, nothing has to

by the translation process. In the case of the above example, for instance additional type definition

```
Type2 ::=          SET { x Type,  y BOOLEAN }
```

and a variable K2 of type Type2, one would use the Pascal "dot" notation to SEQUENCE x by the expression "K2.x", and the integer of x by "K2.x.a".

3.3. CHOICE

The ASN.1 type CHOICE indicates that a data item must be of one of several types. For instance, a value of the following type Type3 may be either an boolean or an octet string.

```
Type3 ::=   CHOICE {
        id1     INTEGER,
        id2     [UNIVERSAL 1] IMPLICIT BOOLEAN,
```

The natural translation in Estelle (and Pascal) is a record with variants, su following:

```
Type3 = record (*@CHOICE *)
          case choice: integer of
          Type3_id1_tag:(* = 0x2 *)
                (id1:       integer);
          Type3_id2_tag:(* = 0x1 *)
                (id2:       (*@T [UNIVERSAL 1] IMPLICIT *) boolean);
          end;  (* of type Type3 *)
```

3.4. Tags

Tags are introduced into ASN.1 definitions in order to specify the identifier val inserted into the encoded form of the data values. They have no influence on of the data structures and their values can therefore be ignored in the tra structures (possibly represented as comments in the Estelle or C translation). they are clearly important for the encoding and decoding routines, but a distinction between the different cases of a CHOICE data structure. The latter is why identifiers have been introduced in the above translation representing t possible tag values for the given ASN.1 CHOICE.[1] These identifiers can be use

[1] This is not necessary if the target language has types as first-class valu Certain languages have statements for testing the type of a value.

body of the Estelle specification (or the C-code implementation) to determine th
received CHOICE value.

3.5. Strings

ASN.1 distinguishes between OCTET STRING and BIT STRING. BIT STRING a
packed to 8 bits per octet. There are several specializations of OCTET STRING,
in the kinds of octets values that are allowed. For instance, an IA5String allo
usual standard characters, NumericString only for numerical digits. These spe
can all be represented by the same manner in Estelle or C.

Estelle has no data type directly suitable for strings. Annex B1 of the Estelle
[Este 89] proposes a set of procedures for manipulating values of a so-called "
which is a kind of octet string. It would therefore be natural to consider
"data_type" as the translation of the ASN.1 type OCTET STRING.

But in the case of translation into C, it is important to obtain an efficient implen
string storage and manipulation. Whereas the "data_type" structure of Estell
entire string in an array, such an array structure is not very effective since fi
length of a string is often not known in advance, and second, concatenation w
copying at least one of the two parts. Therefore we have adopted a data st
strings which consists of one or more so-called "chunks", each containing a
string. An OCTET STRING in ASN.1 is therefore translated into a pointer to a
(called OCTS) containing the following three fields: an octet string representing
of the OCTET STRING, an integer specifying the length of that chunk, and a poi
next chunk. In order to simplify the translation from Estelle to C, we have adopt
data structure to represent OCTET STRING in Estelle. The correspondir
manipulation routines are listed in the Annex-1, and are similar to those c
"data_type" in the Annex B1 of [Este 89].

In order to store and manipulate bit strings, we have adopted an approach si
case of octet strings. The manipulation routines are similar to those for octet s
names have the prefix "b_" to distinguish them from the manipulation routir
strings, which have the prefix "o_".

3.6. SEQUENCE OF and SET OF

A value of type SEQUENCE OF <element type> is a sequence of values, each <element type>. A value of type SET OF <element type> is similar, except that of the elements has no significance. Because of this slight difference, t representation could be used in Estelle or C.

There are several ways to represent such a sequence in Estelle. For instance, a l with pointers could be used as follows. Assuming the ASN.1 type

Type4 ::= SEQUENCE OF Integer

the Estelle equivalent could be

```
Type4 =     ^Type4_list;
Type4_list = record
                 item: integer;
                 next: ^Type4_list
             end;
```

In order to simplify the manipulation of the list structure, we have adopted a structure including additional information, and a set of standard list manipula which can be used for all lists, independently of the type of the element corresponding structure can be used in C.

The generic list structure in Estelle has the following form:

```
UNIV     = ...;       (* universal type, can represent any type *)
LIST_ITEM =           record
    item:             UNIV;
    next:             ^LIST_ITEM; (* next item in the list *)
    end;
LIST = record
    count:            integer; (* number of items in the list *)
    top:              ^LIST_ITEM;(* first item in the list *)
    current:          ^LIST_ITEM; (* next item to be accessed *)
    end;
LIST_OF  =            ^LIST;       (* SET/SEQUENCE OF *)
```

The fields "count" and "current" are useful for the list manipulation. In order consistent, however, a set of manipulation routines have been defined (see primitives prefixed by C_List) which should be used in the body of the Estelle sp for accessing or updating the list structures.

Using the data structures above and the predefined manipulation routines, we variable x of type SEQUENCE OF Integer (see Type4 above) as follows. The nu items in the list is given by "C_ListCount (X, result)" and the first element of obtained by the expression "C_ListFirst (X, result_item)". However, the generi data structure poses some problems in Estelle and Pascal because of strong t fact, the items in the list structure are of type UNIV, which is "implementation They can be converted into elements of the appropriate type by using type cas generated by the ASN.1-Estelle compiler. For each type definition of tl SEQUENCE OF <element type>, the routines PutItem_<element type> GetItem_<element type> are defined which provide type casting from <eleme UNIV and UNIV to <element type> , respectively. For example, to obtain th which is the first element of the variable X considered above, we cou C_ListFirst (X, result_item); GetItem_Integer (result_integer, result_item). T these routines permits the programmer to write application code without k internal implementation of these list structures.

3.7. OPTIONAL

A element within a SEQUENCE or SET may be declared OPTIONAL. In this ca SEQUENCE value may contain this element, or may not. The presence of the within the data structure may be represented in different manners in Estelle following two approaches are possible:

(1) There is an additional boolean field included in the data structure for each e may be optional. The boolean value indicates whether the element is present.

(2) If the element is represented by a pointer to the element value, the value pointer means that the element is not present.

The first approach seems more natural and has also been used for translation [Boch 89h]. For example, the ASN.1 definition

Type5 ::= SEQUENCE { a INTEGER OPTIONAL; b BOOLEAN }

would be translated into:

```
Type5 =     record (*@SEQUENCE*)
    a_IsPresent:    BOOLEAN;
    a:              INTEGER;
    b:              BOOLEAN;
    end;
```

3.8. Discussion

The above list of ASN.1 constructs covers the basic part of ASN.1. The so-calle and certain recent extensions are not addressed, however. It is interesting to n features of ASN.1 can be translated in a natural manner into specification and p languages, as shown above. The ASN.1 construct that encountered most diff clearly the SEQUENCE/SET OF construct for which no corresponding concept Estelle and most programming languages. The data structure adopted was partl by considerations for implementation efficiency, since a natural translati implementation data structures was a concern.

The implementation considerations for the C data structures lead to the use of various list structures. The access and update routines discussed in the subse have the property of completely hiding the pointer structures from the user. the user wants to make a complete copy of a given PDU, which is represented pointer data structure, he/she has to know its detailed structure. In order t difficulty, the ASN.1-Estelle compiler provides suitable copying routines for ea type definition of the form SEQUENCE, SETand CHOICE. These routines have th "s_copyXXX" were XXX is the name of the defined type. These routines mak of a variable of type XXX into another variable of that type by allocating the memory for storing the data structures to be copied. For the list structures c to SEQUENCE OF/SET OF , the predefined routines C_ListCopy listed in Anne be used.

4. A coordinated set of support tools

In the following, we describe a set of support tools which have the purpose of p implementation environment for OSI Application layer protocols in conjuction v of the Estelle specification language. It is assumed that an ASN.1 description of exist, and that an Estelle specification of the protocol is used in the implementa The support tools provide for the following steps:

(1) the automatic generation of encoding and decoding routines for the PDU's, definitions.
(2) the automatic translation of the ASN.1 description of the PDU's into corr Estelle data structures, and the generation of declarations for the corre manipulation primitives, and
(3) the automatic generation of implementation code from the Estelle specifica generation of the algorithmic bodies for the corresponding primit

As indicated in Figure 1, this approach requires the compatibility betwee structures used in the encoding and decoding routines and the corresponding d obtained from the Estelle translation. A more detailed configuration of th environment is shown in Figure 2. It is important to note that the Estelle dat obtained by the translation step (2) above must be integrated into the Estelle sp the protocol. As discussed in Section 3, these structures have an impact on th which the processing of the PDU's can be described in the protocol specific indicated in Figure 2, the complete Estelle protocol specification is not automatically. In order to simplify the implementation process, it would be use specifications of the OSI standard protocols would be available.

The ASN.1-Estelle implementation environment comprises three tools correspor three steps identified above, which provide translation into the implementation The first step is provided by the tool CASN.1 developed at the University o Columbia [Yang 88]. The third step is provided by the NBS Estelle compiler [E The ASN.1 to Estelle translation of the second step was specially developed purpose.

We believe that an integrated set of tools as described here will provide an imp environment which reduces the number of programming errors in the impl because the PDU encoding and decoding is automatically generated, and the c with the PDU implementation data structures is automatically enforced.

Figure 2 goes here; it comes from a PowerPoint document

4.1. Handling the incompatibilities between the ASN.1 and Estelle tools

The integration of two independently developed tools for steps (1) and (3) a single environment was the major challenge in the development of the ASN.1 translation tool which must satisfy the compatibility requirement shown ir Among the different ASN.1 tools that we investigated, CASN.1 was the only or showed sufficient compatibility with the C data structures generated by the Este to make the integrated approach feasable.

A number of incompatibilities had to be bridged and a number of imple restrictions imposed by the two existing tools had to be accomodated. The adjustments were made for each of these cases by using the following approach

(a) Adjusting the ASN.1 to Estelle translation (step (2)) accordingly. The ob realizing a "natural" translation between the two languages limits the range changes.

(b) The automatic introduction of changes into the data structures generated by tool. This approach was adopted for certain incompatibilities, and was realized automatic generation of text editing commands during step (2) above, and th execution of these commands on the source code generated in step (1). T (standard Unix stream editor) is used for this purpose.

4.2. Some practical experience

Some simple applications of the tools described above were made duri development. We describe in the following the application of a version of the al protocol (ABP). A complete Estelle specification of the ABP is given in Annex B2 89]. We adapted this specification by introducing the following PDU definition ASN.1.

```
AB-PDU DEFINITIONS ::=BEGIN
Npdu-type ::=     CHOICE
{data-pdu         [APPLICATION 1] Data-pdu-type,
ack-pdu           [APPLICATION 2] Ack-pdu-type }
```

```
Data-pdu-type ::=SEQUENCE {
ndata              U-Data-type,
seq                Seq-type }

Ack-pdu-type ::=  SEQUENCE {seq    Seq-type }

U-Data-type ::=    SET OF IA5String

Seq-type ::=       INTEGER { seq0 (0), seq1 (1) }
END --             of AB_PDU definitions --
```

The declaration part of the specification includes two source files generat ASN1ESTL compiler (step(2)). The file "const.e" shown in Annex-1 conta constants for tags, named integers and named bits in the ASN.1 definitions "ab.asn1.e" shown in Annex-1 contains corresponding type and procedure defi

The latter file consists of five major parts, respectively in this order: (1) the types for the CASN.1 tool, such as OCTS, BITS, LIST, etc., (2) the predefined ty ASN.1 such as IA5String, GeneralString, UTCTime, etc., (3) the translated types PDU definition, (4) the predefined primitives for manipulating predefined stru (5) useful primitives corresponding to the PDU definitions, such as coding an primitives, mapping primitives for List items, primitives for duplicating or structures like SEQUENCE, SET, CHOICE, etc.

A complete discussion of this example can be found in [Boch 90f]. An applicat tools for the implementation of the ACSE protocol was also performed.

5. Conclusions

We have shown how ASN.1 and Estelle may be combined to form a complete s application layer protocol specification and implementation. This approa following characteristics:

(a) Given an Estelle specification of the protocol, including the PDU description automatically by the translation of the ASN.1 definitions in the OSI stand implementation process is largely automated.

(b) This automation provides a faster and more reliable implementation pro
direct use of the standard ASN.1 definitions, and possibly of an Estelle specifica
has been validated in respect to the protocol standard.

(c) It is possible to take advantage of already existing implementation tools for
Estelle, and combine them by developing a translation tool from ASN.1 to Estell

(d) The Estelle protocol specification used in this context uses the PDU data
obtained by the automatic translation from the ASN.1 protocol definition. It is
that such a specification would be given as an annex of the protocol standard.

(e) It is also possible to use ASN.1 type definition for other purpose than PDU
such as service parameters or internal types in a specification; list manipulati
provided by the ASN.1 support could be used for processing these data structu

Our preliminary experience with the here described semi-automatic imple
approach seems to indicate that it is very promising to integrate the ASN.1 nc
other formal languages that can be used to formally describe the other asp
protocol definitions. We are presently working on the application of this impl
approach to the OSI Association Control (ACSE) protocol. It would be inter
provide extensions to the tools for handling the ROSE ("Remote Operations", IS
macros used by many OSI application layer protocols. A similar approach can a
with the language SDL. The translation from ASN.1 to Lotos is discussed in [Boc

Acknowledgements

This work would not have been possible without the dedication and competer
Yang who designed and implemented the CASN.1 tool. Financial support f
National Science and Engineering Research Council of Canada is acknowledged.

References

[ASN.1 C] ISO IS 8825, "Information Processing - Open systems Interconnectio
Encoding Rules for Abstract Syntax Notation One (ASN.1).

[ASN.1] ISO IS 8824, "Information Processing - Open systems Interconne
Specification of Abstract Syntax Notation One (ASN.1).

[Boch 86b] G.v.Bochmann, M.Deslauriers and S.Bessette, "Application layer p testing and ASN1 support tools", Proc. IEEE GLOBECOM Conf., Houst Dec. 1986, pp. 767-771.

[Boch 87h] G.v.Bochmann, G.Gerber, and J.M.Serre, "Semiautomatic implemen communication protocols", IEEE Tr. on SE, Vol. SE-13, No. 9, Septen 1987, pp. 989-1000 (reprinted in "Automatic Implementation and Con Testing of OSI Protocols", IEEE, edited by D.P.Sidhu, 1989).

[Boch 89d] G.v.Bochmann, "Protocol specification for OSI", to be published in C Networks and ISDN Systems.

[Boch 89h] G.v.Bochmann and M.Deslauriers, "Combining ASN1 support w LOTOS language", Proc. IFIP Symp. on Protocol Specification, Testing Verification XI, June 1989, North Holland Publ.

[Boch 90f] G. v. Bochmann, D. Ouimet and G. Neufeld, Implementation suppor OSI application layer protocols, submitted for publication.

[Budk 87] S. Budkowski and P. Dembinski, An introduction to Estelle: a spec language for distributed systems, Computer Networks and ISDN Syste 14, no. 1, pp.3-23, 1987.

[Este 87b] NBS, "User guide for the NBS prototype compiler for Estelle", Final Report no. ICST/SNA - 87/3, Octobre 1987.

[Este 89] ISO IS9074 (1989) "Estelle: A formal description technique based on a state transition model".

[Hase 88] T.Hasegawa, et al., "Automatic ADA program generation from p specifications based on Estelle and ASN.1", Proceedings of Interna Conference on Computer Communications (ICCC), J.Raviv editor, I Israel, 1988.

[ISOD 89] "ISODE-5.0, ISO Development Environment", Software Package b Wollongong Group, Palo Alto CA, USA, 1989.

[OSI 83] Special issue on Open Systems Interworking, Proc. of the IEEE, Dec. 19

[Sidh 90] D. P. Sidhu, Semi-automatic Implementation of OSI Protocols, publ Computer Networks and ISDN Systems 18 (1989/90) 221-238.

[Yang 88] Y.Yang, "ASN.1-C Compiler for Automatic Protocol Implementation", Degree Thesis, Department of Computer Science, University of B Columbia, October 1988, 111pp.

Annex-1: Extract from generated files of ASN1-Estelle tool.

This is the generated file from ASN1-Estelle compiler for
Alternating-Bit ASN.1 PDU definition given in Section 4.2.

```
-----FILE const.e
const
(*  ..... some constants deleted ..... *)
       (* list of named integers for Seq_type: *)
seq0   = 0;
seq1   = 1;

       (* tags constants for CHOICE type Npdu_type *)
Npdu_type_data_pdu_tag       = ANY INTEGER; (* defined in c_include.h as 0x60000001 *)
Npdu_type_ack_pdu_tag        = ANY INTEGER; (* defined in c_include.h as 0x60000002 *)

-----FILE ab.asn1.e
(*  ..... some types deleted ..... *)
U_Data_type    = (*@SetOf *) LIST_OF; (* of IA5String *)
       (* primitives to map/unmap Items of this type will be declared as:
          GetItem_IA5String and PutItem_IA5String procedures
          at the end of types *)

Seq_type      = integer;
       (* list of named integers for Seq_type:
       seq0   : 0
       seq1   : 1 *)

Data_pdu_type         = record (*@SEQUENCE *)
       ndata: U_Data_type;
       seq:   Seq_type;

Ack_pdu_type   = record (*@SEQUENCE *)
       seq:   Seq_type;

Npdu_type     = record (*@CHOICE *)
       case choice: integer of
       Npdu_type_data_pdu_tag:
         (data_pdu:   (*@T [APPLICATION 1] *) ^Data_pdu_type);
       Npdu_type_ack_pdu_tag:
         (ack_pdu:    (*@T [APPLICATION 2] *) ^Ack_pdu_type);
       end;

(* Predefined primitives for manipulating LIST structure *)
procedure C_ListCount(TheList: LIST_OF; var Count: integer); primitive;
       (* returns the list length (number of list items) of a list *)
procedure C_ListFirst(TheList: LIST_OF; var TheFirst: UNIV); primitive;
       (* returns the first item of the list *)
procedure C_ListNext(TheList: LIST_OF; var Next: UNIV); primitive;
       (* returns the  next item *)
procedure C_ListTrim(var TheList: LIST_OF; var LastRemoved: UNIV); primitive;
       (* remove the last list item from the list *)
procedure C_ListEOL(TheList: LIST_OF; var ENDOFLIST: boolean); primitive;
       (* returns TRUE if end of list has been reached.*)
procedure C_ListAdd(var TheList: LIST_OF; TheItem: UNIV); primitive;
       (* inserts a new list item into the list as the first list item *)
procedure C_ListAppend(var TheList: LIST_OF; TheItem: UNIV); primitive;
       (* inserts a new list item at the end of the list *)
procedure C_ListEmpty(TheList: LIST_OF; var Empty: boolean); primitive;
       (* returns TRUE if the list contains no items,*)
procedure C_ListRemove(var TheList: LIST_OF); primitive;
       (* removes the last list item accessed by C_ListNext or C_ListFirst *)
procedure C_ListMerge(var Dest: LIST_OF; var Source: LIST_OF); primitive;
       (* merges two lists into one *)
```

```
procedure C_ListFree(var TheList: LIST_OF); primitive;
       (* frees the space occupied by a list. *)
procedure C_ListCopy(var Destination: LIST_OF; Source: LIST_OF);
            primitive;
       (* make a copy of a list copying complete structures. *)

(* Predefined primitives for manipulating OCTSTRING structure *)
procedure o_length(OS: OCTSTRING; var LEN: integer); primitive;
       (* return the actual length of OS variable *)
procedure o_null(var OS: OCTSTRING); primitive;
       (* initialize the variable OS to a null OCTSTRING *)
procedure o_copy(FROM_OS: OCTSTRING; var TO_OS: OCTSTRING); primitive;
       (* copy FROM_OS into TO_OS *)
procedure o_create(var OS: OCTSTRING; InitVal: DATA_TYPE); primitive;
       (* create OS given a DATA_TYPE InitVal *)
procedure o_get(OS: OCTSTRING; offset: integer; var value: OCTET); primitive;
       (* return the octet at the specified offset (0 is first) *)
procedure o_put(var OS: OCTSTRING; offset: integer; value: OCTET); primitive;
       (* put the octet at the specified offset (filler is '\0') *)
procedure o_assemble(var BASE: OCTSTRING; var ADDITION: OCTSTRING); primitive;
       (* append ADDITION to BASE; set ADDITION pointer to NULL *)
procedure o_free(var ToScrap: OCTSTRING); primitive;
       (* releases the space of OCTSTRING ToScrap *)

(* Similar primitives for BITSTRING are omitted here *)
```