OSI Specifications using ASN.1 and other formal description techniques

Gregor v. Bochmann and Daniel Ouimet
Département I.R.O, Université de Montréal
CP 6128, Succursale A, Montréal, Québec, Canada, H3C 3J7

Abstract:

Service and protocol specifications are the basis for the design and implementation of distributed computer systems. In the area of OSI communication standards, specifications are written in natural language and various description techniques, including ASN.1 and other formal techniques. Since the scope of ASN.1 is essentially limited to the definition of data structures and coding conventions, these aspects must be combined with other aspects in order to obtain complete protocol and service specifications. The issues related to the integration of ASN.1 with specifications written in formal description techniques (FDT's), such as Estelle, LOTOS, SDL, or in the object-oriented specification language Mondel, are discussed in this paper. In particular, the translation of the concepts of ASN.1, ROSE, and entity-relationship modelling into the FDT's and Mondel are considered in some detail. The paper also describes the problems that had to be solved for the integration of an ASN.1 encoding and decoding tool with an Estelle compiler, which together provide support for the semi-automatic implementation of OSI Application layer protocols.

1. Introduction

Standards for communication protocols and services are being developed for Open Systems Interconnection (OSI) [OSI 83] which are intended to allow the interworking of heterogeneous computer systems and applications. In this context, the protocol specifications are of particular importance, because they represent the standards which are the basis for the implementation and testing of compatible OSI systems [Boch 90g]. The standard specifications are usually written in natural language, augmented with certain formalisms such as state tables. In addition, formal description techniques (FDT's) have also been developed and used. The formal nature of these specifications make it possible to apply automated tools during the protocol development life cycle [Boch 87c].

The protocol specification defines the behavior of a protocol entity in terms of interactions with the local service user, called service primitives, and its interactions exchanged with the remote peer entity, called Protocol Data Units (PDU's). In the case of OSI Application layer protocols, the data structures of the PDU's are usually described in ASN.1 (Abstract Syntax Notation One [ASN1]). This notation also includes information about the coding of the exchanged information between the communicating computers, according to associated encoding rules [ASN1 C]. These rules follow a fixed coding scheme, which permits the construction of ASN.1 support tools which generate automatically the required encoding and decoding software for a given protocol specification [Boch 90f, Neuf 90b].

The ASN.1 notation covers only aspects of data structuring. As indicated in Figure 1, this represents only a part of all those behavior aspects that must be covered by a protocol specification:

(1) Data structures
(2) Declarations of operations and their parameters
(3) Semantics of operations and their possible order of execution

All these aspects are covered by the three formal description techniques Estelle, LOTOS and SDL discussed below. However, these description languages also include their own language elements for data structures, corresponding largely to the ASN.1 notation. The relation between ASN.1 and the corresponding notations of the FDT's is discussed in more detail below in Sections 3 and 5.

data structures	declarations of operations and their parameters	behavior of object / operations semantics	object-orientation, inheritance, entity relationship model
ASN.1	ROSE		notation of [OSI mb]
FDT's			
MONDEL			

Figure 1

While ASN.1 has been originally developed in relation with the CCITT standard on message handling systems and has subsequently been used for most OSI Application layer protocols, certain extensions to this notation (using so-called macros) have been developed in relation with other standards, such as a notation for defining operations (corresponding to point (2) above), called ROSE [OSI RO] and a notation for writing specifications of managed objects, including the notion of object classes and inheritance [OSI mb]. These notations, together with the basic notation of ASN.1, are used for describing many OSI application layer protocols presently under development, as for instance in the area of distributed systems management.

The object-oriented concepts, indicated in the last column of Figure 1, are not essential for the description of communication protocols and distributed systems, however, they are useful for writing descriptions of complex systems in a more structured manner. They also fit easily with the data modelling concepts of the entity-relationship model [Chen 76, OSI ER]. In a joint research project with the Computer Research Institute of Montreal (CRIM) and Bell-Northern-Research on network management, we have defined an object-oriented specification language, called Mondel, which is intended to cover all the specification aspects shown in Figure 1 in a simple framework, which should make it easy to be used. Our experience with this language and the associated simulation and validation tools has been encouraging [Boch 91i, Mond 90, Boch 91].

This paper deals with the problems that arise through the presence of these different specification languages. The notations ASN.1 and ROSE, and the notation for managed objects used for the description of OSI protocols are defined in a relatively ad hoc manner without a precise definition of their semantics, and they do not cover all aspects of specifications, as mentioned above. In order to build tools for the validation of specifications, for the development of their implementations and testing, it is necessary to have a precise language definition, preferably with a formal semantics. Such definitions exists for the FDT's and certain other specifications languages, such as Mondel. However, if one wants to use these languages for OSI protocols, a translation from the notations used in the protocol definitions into these formal specification languages must be provided. This translation issue is the topic of this paper.

Section 2 gives a short introduction to ASN.1 and associated notations, FDT's and Mondel. In the subsequent section, we discuss the general strategies that can be used for an integration between ASN.1 and the formal specification languages. An overview of possible translation from ASN.1 to several of these languages is also given. In Section 4, we discuss the integration of the concepts of entity-relationship modelling into an object-oriented setting including ROSE, using Mondel as an example language. Section 5 deals with the details of translation from ASN.1 into Estelle and the implementation language C, which is generated by some Estelle compilers. The integration of ASN.1 encoding and decoding tools with an Estelle compiler is also described.

2. Overview of ASN.1 and formal description techniques

2.1. ASN.1 and related notations for OSI protocols

ASN.1 [ASN1] is a notation which allows the definition of data types, similar to the data type definitions available in programming languages such as Pascal or ADA. The notation includes a number of predefined types such as integers, reals, booleans, bit strings etc ... It also allows the definition of composed types , such as groups of elements (called SEQUENCE, corresponding to the Pascal RECORD), groups of elements of the same type (called SEQUENCE OF), a list of alternative types (called CHOICE, corresponding to Pascal's variant records). A TAG can be used to for distinguishing different alternatives or optional elements within a group and refers to the coding of the items during transmission.

The main reason for the success of ASN.1 as specification language is probably the fact that it is combined with a standard encoding scheme for PDU's [ASN1 C] which has been adopted for OSI Application layer protocols. Based on the information contained in the ASN.1 definition of the PDU structure, this scheme completely determines the PDU encoding, and can be used for implementing he coding and decoding functions automatically (see for instance [Neuf 90b].

A so-called macro notation has been defined for ASN.1. In contrast to macro expansion rules defined for other languages, however, the ASN.1 macros have no semantic meaning. Nevertheless, this notation has been used to introduce certain additional notations, such as a notation for defining "Remote Operations" and managed OSI objects, which are further discussed in Section 4.

2.2. Overview of Formal Description Techniques

Most protocol specifications are written in natural language, sometimes augmented with state tables and other forms of organized information. In addition, a number of formal techniques have been used to describe protocols in a more precise manner [Boch 90g]. The use of such formal specifications leads to the possibility of automating certain activities of the protocol development cycle [Boch 87c], such as the validation of the protocol specification [Pehr 90], the implementation process [Boch 87h], and the conformance testing of protocol implementations [Sari 89c].

In view of these advantages, ISO and CCITT have developed standardized specification languages, so-called formal description techniques (FDT's), which are intended to be used for the description of OSI protocols and services. They are respectively called Estelle, LOTOS, and SDL (for tutorial introductions, see [Budk 87, Bolo 87, Beli 89]). Although these languages were developed for use within OSI, they have potentially a much broader scope of application.

In Estelle, a specification module is modelled by an extended finite state machine (FSM). The extensions concerning the range of possible values for interaction parameters, and rules for interpreting and selecting values of these parameters are covered by type definitions, expressions and statements of the Pascal programming language. In addition, certain "Estelle statements" cover aspects related to the creation of the overall system structure consisting in general of a hierarchy of module instances. Communication between modules takes place through the interaction points of the modules which have been interconnected by the parent module. Communication is asynchronous, that is, an output message is stored in an input queue of the receiving module before it is processed.

SDL, which has the longest history, is also based on an extended FSM model. For data types, it uses the concept of abstract data types with the addition of a notation of program variables and data structures, similar to what is included in Estelle. However, the notation for the latter aspects is not related to Pascal, but to CHILL, the programming language recommended by CCITT. The communication is asynchronous and the destination process of an output message can be identified by various means, including process identifiers or channel names.

LOTOS is based on an algebraic calculus which includes the concepts of finite state machines plus parallel processes which communicate through a rendezvous mechanism which allows the specification of rendezvous between two or more processes. Asynchronous communication can be modelled by introducing queues explicitly as data types. The interactions are associated with gates which can be passed as parameters to other processes participating in the interactions. These gates play a role similar to the interaction points in Estelle. Data types are described in an algebraic notation for abstract data types, called ACT ONE, which is quite powerful, but would benefit from the introduction of certain abbreviated notations [Scol 87, Boch 89h] for the description of common data structures.

In contrast to the other FDT's, SDL was developed, right from the beginning, with an orientation towards a graphical representation. The language includes graphical elements for the FSM aspects of a process and the overall structure of a specification. The aspects of data types are only represented in the usual linear, program-like form. In addition, a completely program-like form is also defined called (SDL-PR) which is mainly used for the exchange of specifications between different SDL support systems.

In this paper, we are largely concerned with issues related to the implementation process. The languages Estelle and SDL can be considered high-level implementation languages; in fact, a number of FDT compilers exist which translate formal specifications into program code written in a conventional programming language, such as Pascal, C or ADA. For Estelle, such systems are described in [Boch 87h, Vuon 88].

2.3. An object-oriented specification language

We have developed an object-oriented specification language called MONDEL which supports an object-oriented development methodology [Boch 90l]. In many respects, MONDEL resembles other existing object-oriented languages. It has, however, certain properties which make it particularly suitable for describing distributed OSI applications.

In contrast to many other object-oriented languages, MONDEL distinguishes between persistent and non-persistent objects. Persistent objects are like entries in a data base; they remain present until they are explicitly deleted, and they can be interrogated by database-oriented statements which identify the object instances in the database which belong to a given class and have specified properties. Entities and relationships are usually represented as persistent objects.

In contrast to many object-oriented languages, communication between objects is synchronous, that is, an object instance calling the operation of another object is blocked until the called object executes a RETURN statement, which may include the delivery of a result. Synchronous communication is better suited for specifications of higher level of abstraction, since cross-over of messages at interfaces can be largely avoided [Boch 88h]. MONDEL provides a number of statements to express the order in which the operations can be accepted by an object, or for specifying the actions to be performed when an operation call is accepted by the object.

The concept of an ATOMIC operation is introduced which represents a "transaction" in the sense of databases, that is, it represents a set of actions which are either all performed without interference from other "transactions", or undone if an exception condition is encountered. The language has exception handling similar to ADA. The actions of different objects are usually performed in parallel; it is also possible to define several parallel activities within a single object.

The statements of the language have the flavor of a high-level programming language, except for the database-oriented statements mentioned above. However, it is also possible to write assertional specifications by defining input and output assertions for operations, or by defining INVARIANT assertions which must be satisfied at the end of each "transaction".

3. Combining ASN.1 with other description techniques

In order to take advantage of the tools available with formal specification languages for OSI Application layer protocols, in conjunction with the ASN.1 notation and the associated automatic generation of coding and decoding routines, it is necessary to clearly define the relationship between the ASN.1 data structure definitions and the corresponding concepts in the respective formal specification languages. The relationship must be defined at two levels. First, the correspondence between the respective language constructs for defining data structures must be defined. Secondly, issues related to the combination of the respective tools must be addressed.

3.1. Strategies for integrating the ASN.1 notation with other specification languages

3.1.1. Integration at the language level

The following three approaches can be envisioned for the integration of ASN.1 with other languages (e.g. FDT's): (1) substitution, (2) combination, and (3) translation.

In the substitution approach [Alve 87], the ASN.1 notation is used instead of the corresponding FDT concepts for describing the data structures in the specification. It is important to note, however, that ASN.1 does not provide any notation for accessing individual elements contained in a given PDU data structure, and such a notation is required for writing those parts of the specification that describe the semantics of the operations. Therefore, this approach requires the definition of extensions to the ASN.1 notation. Then it is possible to use such an extended notation instead of the data type definition and access constructs provided by the FDT.

In the case of the combination approach [Hase 88], the situation is similar, except that the constructs of the FDT remain valid and the extended ASN.1 notation is allowed as an alternative description method within the same language. Clearly, this leads to a duplication of functionality in the specification language.

In the translation approach [Boch 89h], no access extension for ASN.1 needs be defined. Instead, the ASN.1 notation is translated into equivalent constructs of the FDT. The formal specification of an OSI application layer protocol will therefore include the definition of the PDU structure as obtained by translation from the given ASN.1 definition contained in the protocol standard. For the access to the individual elements of this structure, the available constructs of the FDT can be used.

The substitution and combination approaches require a redesign of the FDT, which is a major undertaking. In the case of the translation approach, the FDT remains unchanged. In order to make this approach successful, however, it is necessary to define the translation in such a manner that the resulting PDU data type definitions are simple to read and natural for the designer of the FDT protocol specification. An overview of possible transitions is given in the next subsection. Details of the translation approach in the case of the Estelle specification language are discussed in Section 5. For the case of LOTOS, the issues are discussed in [Boch 89h].

3.1.2. Integrating language tools in the case of ASN.1 translation

As mentioned above, various tools have been developed independently for ASN.1 and the associated encoding rules, on the one hand, and for the FDT's and other specification languages, on the other hand. The integration of these languages should therefore also lead to an integration of the associated tools. In the case of the translation approach described above, the main issue for tool integration is the compatibility of the respective implementation data structures used in the tools for representing the PDU's. The ASN.1 definition of the PDU, taken from the protocol standard document, is used for generating the ASN.1 encoding routines which translate between the communication format and an internal data structure format used by the ASN.1 tool. The same ASN.1 definition is also translated into FDT data type definitions which become part of the formal protocol specification. This specification is then translated by the FDT tool. For the integration of these tools, it is therefore necessary that the implementation data structures representing the PDU's obtained by the translation from the FDT tool are the same as those used by the encoding routines generated by the ASN.1 tool.

An example of such an integration is discussed in the Section 5. Another example, for the case of LOTOS, is described in [Boch 89h]. In the latter case, the existing systems were basically incompatible; the existing LOTOS interpreter used internally Prolog data structures and LOTOS source code for externally representing constant data structures, while the ASN.1 tools used an internal value tree in C (similar to the ENODE structure described in [Neuf 90b]). Nevertheless, an integration was realized by providing the automatic generation of a translation module which transforms, during run-time, between the internal C value trees of the ASN.1 tool and the external source code supported by the LOTOS interpreter.

It is clear that these problems can be avoided when new tools are designed for an integrated ASN.1-FDT environment.

3.2. Overview of ASN.1 translations

Most concepts of ASN.1 can be translated in a straightforward manner into the different specification languages, as shown in the following table.

ASN.1	**Estelle**	**SDL**	**Lotos**	**Mondel**
INTEGER	integer	Integer	NaturalNumber	integer
BOOLEAN	boolean	Boolean	Boolean	boolean
NULL	...	NEWTYPE Null ENDNEWTYPE Null;	type Null is endtype	none
ANY	...		type Any is endtype	object
BIT STRING	array [1..N] of boolean	String(Boolean,'')	BitString	sequence [boolean]
OCTET STRING	array [1..N] of Octet; (where Octet=0..255)	String(Octet, '') (where Octet=0..255)	OctetString	sequence [Octet] (where octet = integer with invariant 0 <= self <= 255)
SET { x XType, y YType }	record x: XType; y: YType end;	STRUCT x XType; y YType;	TUPLE xyz COMP x: XType, y: YType {abbrev. notation}	passive{object} with x: XType; y: YType
SEQUENCE { x XType, y YType }	same as SET	same as SET	same as SET	same as SET
CHOICE { x [0] XType, y [1] YType }	record (* variant *) case choice: integer of 0: (x: XType); 1: (y: YType) end;	STRUCT choice Integer; x XType; y YType;	ONEOF XType, YType {abbrev. notation}	choice XType or YType {the tags [0] and [1] are coding issues}
SET OF XType	generic list with pointers: LIST=record item: Xtype; next: ^LIST; end;	String (XType,'')	STRINGOF XType {abbrev. notation}	sequence [XType]
SEQUENCE OF XType	same as SET OF	same as SET OF	same as SET OF	same as SET OF

As this table shows, the correspondence between ASN.1 and the other languages is in most cases quite straightforward. (For more details, the reader may refer to the references [Boch 90f], [Boch 89h], and [Boch 90z], respectively). There are, however, certain aspects of ASN.1 which are not directly present in the other languages, such as different kinds of character strings (e.g. PrintableString, NumericString). For enumeration types, such as "color = (red, blue, green)", ASN.1 uses generally an integer representation, such as "Color = Integer { red (0), blue (1), green (2)}", while in the other languages one would prefer a more abstract representation, such as a discriminant union of the form "Color = Choice red or blue or green" without reference to the integer coding of these values.

It is important to note that the ASN.1 notation, although it is defined separately from the ASN.1 data encoding rules, contains certain information elements that relate to coding, such as the "tags" mentioned above. This coding-related information is not required at the logical design level, and is therefore not included in the translations shown in the table above. This information is, however, essential for the compatibility of interworking system, and could be added in a similar manner within the context of the other languages.

The following aspects of the translation may be considered "problematic": (a) Except for Mondel, the ASN.1 concept of "OPTIONAL" leads to somehow lengthy descriptions. (b) The data type definitions in LOTOS, in general, become very lengthy, the translation in the Table is given in terms of an abbreviated notation [Boch 89h]. (c) The ASN.1 list construct "SEQUENCE OF" does not have a correspondence in Estelle and must be "programmed" in a data structure with pointers (see Section 5 for more details).

4. Object-oriented specifications

For applications where many standards and other documents must be considered during the system design, it is important to find a common description formalism which is suitable to express all relevant design issues, which is close to the formalism used in the standard and other documents, and which is formal enough to allow computer-aided design tools. We have tried to make Mondel compatible with several notations used in the context of OSI standardization and other areas, such as explained in the following subsections (see [Boch 90z] for further details).

An important concern is the representation of the entity-relationship (ER) model [Chen 76, OSI ER] which is often used for describing database related applications. It is assumed that the reader is familiar with the ER approach. In addition, various extensions to ASN.1 have been developed within the OSI standardization community for describing such issues as: (a) ports through which connections can be established with other system components [X.407], (b) "Remote Operations" [OSI RO] which correspond to the well-known concept of remote procedure calls, and (c) classes of object with inheritance relations [OSI mb].

4.1. Classes, objects and inheritance

Like object-oriented languages, the ER approach makes the distinction between classes of objects, called types in Mondel, and objects which are the instances of such types. However, the object-oriented approach has the notion of inheritance, which in the area of specifications, we consider in the sense of subtyping [Amer 89, Boch 89f]. This notion corresponds to what is sometimes called the "is-a" relationship. It is important to distinguish this relation between types from the relationships between object instances, which are the usual relations considered in the ER approach.

To express the relation between types, MONDEL has two constructs corresponding to multiple inheritance and non-deterministic choice, respectively. For instance, the definition

type S = A and B endtype S

means that the class **S** inherits all properties of **A** and of **B**; it is therefore a specialization of both. The definition

type G = choice A or B entype G

means that an instance of type **G** is either of type **A** or of type **B**. This construct corresponds to the discriminate union of types, as found in many programming languages.

4.2. Modelling an "entity" and its attributes

MONDEL distinguishes between three basic classes of objects: actors, persistent, and inactive objects. Actors typically represent application programs, users, or system interfaces; persistent objects correspond to instants of entities and relations, which are involved in transactions and for which a persistent copy is retained until the current transaction successfully terminates; inactive objects, finally, include the predefined objects of integers, strings, etc. as well as common data structures and other user-defined objects.

Each MONDEL object has a certain number of fixed acquaintances (usually called "attributes" in MONDEL). Their value does not change during the entire lifetime of the object. It seems natural to model an ER entity type as a persistent MONDEL type where the entity attributes are modelled as inactive acquaintances. In the case that the entity attribute value may change, an acquaintance of type VAR [T] may be used which means that the acquaintance is a variable, to which new values of type T may be assigned during system evolution.

4.3. Modelling relationships using the "database approach"

A relationship between object instances can be most directly be modelled in MONDEL as an object type which has an acquaintance for each role of the relation, as shown for the construct "SEQUENCE" in the table of Section 3. For instance, the notation "type R = persistent with X: XType; Y: Ytype endtype" defines a type of relationship between objects of type XType and YType.

This approach of modelling relationships has the following properties:

(1) Referential integrity (i): An instance of a relationship can not be created without the existence of the related entities.

(2) Referential integrity (ii): If one of the related entities is deleted, the corresponding instance(s) of a relationship is (are) also deleted.

(3) Dynamic relationship: The creation and deletion of instances of a relationship do not affect the existence of the related entities.

(4) Independence: The relationship and the related entities are specified as separate types.

(5) Specialization and relationship attributes: A basic relationship (as exemplified above) may be refined by adding attributes or other pertinent information to the definition of the relation type. For example:

It is possible to specify further properties by defining a subset of the acquaintances to form a key, or by adding an invariant condition which must be satisfied during the creation of each occurrence of the type.

4.4. Modelling relationships using the "data structure approach"

In the context of high-level programming languages, relationships may be represented using various data structures. MONDEL provides certain predefined structures, such as VAR, SET, BAG, and SEQUENCE, which may be used for this purpose. However, the symmetry between the different roles of the relationship is lost in these representations. For instance, the relationship R of Section 4.3 may be represented as an acquaintance of type SET [YType] in each object of type XType. With this approach, the aspects (2), (4) and (5) of Section 4.1.3 are lost. In the case that a type is related to at most one instance of another type, which we call a functional relation, the acquaintance may be of type VAR [], instead of a SET [], which simplifies its use and implementation.

An aggregation relationship, such as "an entity instance of type AType is composed of entities of type BType", is a particular case of a functional relation from BType to AType, and can be modelled as explained above.

4.6. Remote Operations and other object-oriented notations

The "remote operations" [OSI RO] can be modelled naturally in MONDEL in the form of operation calls. In MONDEL, as in any object-oriented language, all communication between objects proceeds through the call of operations which are associated with an object, which must be known to the object that executes the call. In MONDEL, the calling object has to wait for the return of the operation, which may include the result of the operation, or may occur immediately after the acceptance of the operation by the called object. This allows the modelling, in MONDEL, of the synchronous and asynchronous communication foreseen for "remote operations" (see [Boch 91, Boch 90z] for more details). In the languages Estelle and SDL, which have queued interaction primitives, a synchronous remote operation must be represented by two successive queued interactions "request" and "result". The concept of operation errors, as defined for ROSE, can be modelled through exception handling in Mondel.

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

In this section we discuss in more detail the translation from ASN.1 into Estelle and C. Since ASN.1 only covers aspects related to the range of possible PDU parameter values and data structures of PDU parameters, the result of the translation will be data type definitions in the target language. In the case of Estelle, data types are defined in the same notation as in the Pascal programming language. The data typing facilities in many other modern programming languages are similar. This is also the case for C. In the following, we therefore mention the translation into C only in certain cases where its form is not evident from the corresponding discussion for Estelle.

5.1. Overview of translation difficulties

Most constructs of ASN.1 are similar to constructs existing in programming languages, such as Pascal. For these constructs, a natural translation into Estelle and C is relatively straightforward, as discussed in Section 5.2. However, certain particular aspects are often difficult to match, for instance the arbitrary precision of integers in ASN.1 is not compatible with the fixed size integers in Pascal or C. Certain other constructs of ASN.1, such as SEQUENCE OF, have no equivalent in Estelle. It is therefore necessary to build data types in Estelle (and C) which correspond to these ASN.1 structures, as discussed in Section 5.3. A discussion and a translation tool are presented in Section 5.4 and 5.5.

It is important to note that Estelle is not an implementation language. For implementation, Estelle specifications can be automatically translated into parts of the implementation code [Boch 87h]. In Section 5.5, an integrated tool set for the implementation of ASN.1/Estelle specifications in the C language is described. In this case, the PDU definitions written in ASN.1 are translated by the ASN.1 tool directly into C. They are also translated into Estelle and incorporated into the complete Estelle specification of the protocol entity. This specification is then also translated by an Estelle compiler into the C language. As shown in Figure 2, it is important in this context, that the two PDU data type definitions in C obtained by the different translations be identical in order to make the generated encoding and decoding routines compatible with the C data structures used by the remaining part of the protocol entity.

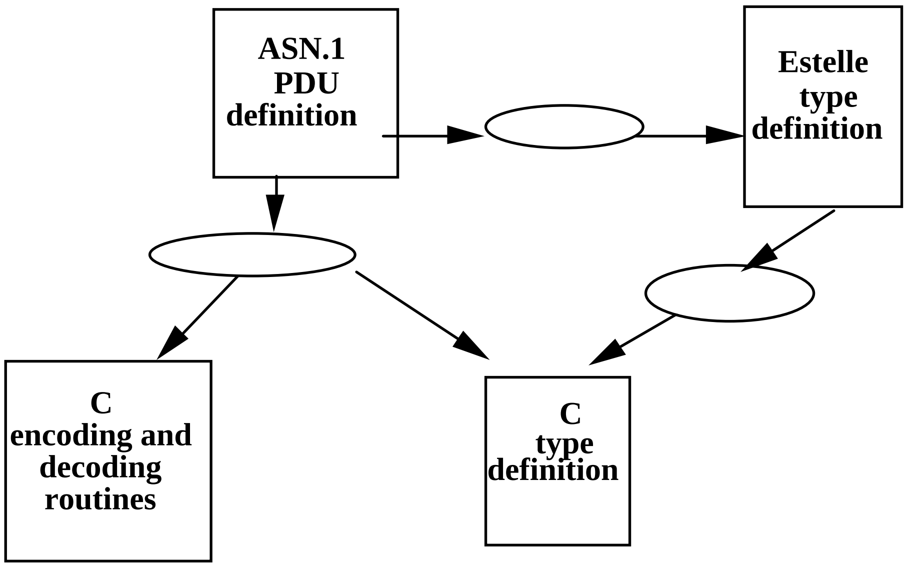

Figure 2 - General overview of translation

5.2. ASN.1 constructs allowing a natural translation

5.2.1. Predefined data types

The predefined data types of INTEGER and BOOLEAN have a corresponding representation in Estelle. The ANY type of ASN.1 is a choice of an arbitrary number of types which is defined at runtime. This type can be represented in Estelle by the notation "..." which means that the type is not yet defined and more information would be provided for an implementation. In 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) can be represented in Estelle and C by a predefined record data structure, containing the fields year, month, day, hour, minute, second, time-difference, and zone.

5.2.2. SEQUENCE and SET

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

```
Type1 ::=           SEQUENCE { a INTEGER,  b BOOLEAN }
```
would be translated into Estelle as:
```
Type1  =            record
                    a: INTEGER;
                    b: BOOLEAN; end;
```
or into C as:
```
typedef struct Type1 {
                    int a;
                    boolean b } Type1;
```
where "boolean" is a type already defined as a "short" integer.

The translation of type structures from ASN.1 into Estelle also determines how the elements of the data types can be accessed by the body of the Estelle specification. Since Estelle includes a notation for accessing Estelle data structures, nothing has to be provided by the translation process. In the case of the above example, for instance, with an additional type definition
```
Type2 ::=              SET { x Type1,  y BOOLEAN }
```
and a variable K2 of type Type2, one would use the Pascal "dot" notation to access the SEQUENCE x by the expression "K2.x", and the integer of x by "K2.x.a".

5.2.3. CHOICE

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

```
Type3 ::=       CHOICE {
        id1     INTEGER,
        id2     [UNIVERSAL 1] IMPLICIT BOOLEAN,
        id3     [APPLICATION 1] OCTET STRING }
```

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

```
Type3 = record
            case choice: integer of
            Type3_id1_tag:      (* = 0x2 *)
                    (id1:integer);
            Type3_id2_tag:      (* = 0x1 *)
                    (id2:boolean);
            Type3_id3_tag:      (* = 0x60000001 *)
                    (id3:OCTSTRING);
            end;
```

while in C the corresponding structure is a so-called union structure of the form

```
typedef struct Type3{
        int         choice;     /* indicate the choice of data */
        union {     /* choices */
                    int         id1;    /* INTEGER */
                    boolean     id2;    /* BOOLEAN */
                    OCTS        id3;    /* OCTET STRING */
                    }           data
        } Type3;
#define Type3_id1_tag 0x2
#define Type3_id2_tag 0x1
#define Type3_id3_tag 0x60000001
```

5.2.4. Tags

Tags are introduced into ASN.1 definitions in order to specify the identifier values that are inserted into the encoded form of the data values. They have no influence on the meaning of the data structures. However, they are clearly important for the encoding and decoding routines, but also for the distinction between the different cases of a CHOICE data structure. The latter is the reason why identifiers have been introduced in the above translation representing the different possible tag values for the given ASN.1 CHOICE.[1] These identifiers can be used in the body of the Estelle specification (or the C-code implementation) to determine the value of a received CHOICE value.

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

5.3. Special translation structures

While the above translations are relatively straightforward, the following ASN.1 concepts have no immediate correspondence in Estelle, and must therefore be modelled by appropriate composed data structures.

5.3.1. Strings

Estelle has no data type directly suitable for strings. Annex B1 of the Estelle standard proposes a set of procedures for manipulating values of a so-called "data_type", which would be natural to consider 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 implementation for string storage and manipulation. Whereas the "data_type" structure of Estelle keeps the entire string in an array, such an array structure is not very effective since first, the total length of a string is often not known in advance, and second, concatenation would require copying at least one of the two parts. Therefore we have adopted a data structure for strings which consists of one or more so-called "chunks", each containing a part of the string. An OCTET STRING in ASN.1 is therefore translated into a pointer to a structure (called OCTS) containing the following three fields: an octet string representing one chunk of the OCTET STRING, an integer specifying the length of that chunk, and a pointer to the next chunk. In order to simplify the translation from Estelle to C, we have adopted a similar data structure to represent OCTET STRING in Estelle. Data manipulation routines are defined for data access in the body of the Estelle specification, and are similar to those defined for "data_type" in the Annex B1 of Estelle standard.

In order to store and manipulate bit strings, we have adopted an approach similar to the case of octet strings with a pointer structure (called BITS).

5.3.2. SEQUENCE OF and SET OF

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

In order to simplify the manipulation of the list structure, we have adopted a translation which is based on a generic list structure including additional information, and a set of standard list manipulation routines which can be used for all lists, independently of the type of the element type. A 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 to keep them consistent, however, a set of manipulation routines have been defined which should be used in the body of the Estelle specification for accessing or updating the list structures.

Using the data structures above and the predefined manipulation routines, we may access the number of items in the list with "C_ListCount (X, result)", and the first element of the list is obtained by the expression "C_ListFirst (X, result_item)". However, the genericity of the data structure poses some problems in Estelle and Pascal because of strong typing. In fact, the items in the list structure are of type UNIV, which is "implementation dependent". They can be converted into elements of the appropriate type by using type casting routines generated by the ASN.1-Estelle compiler. For each type definition of the form SEQUENCE OF <element type>, the routines PutItem_<element type> and GetItem_<element type> are defined which provide type casting from <element type> to UNIV and UNIV to <element type> , respectively.

5.3.3. OPTIONAL

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

(1) There is an additional boolean field included in the data structure for each element which 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 NIL of the pointer means that the element is not present.

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

```
Type5 ::=       SEQUENCE { a INTEGER OPTIONAL }
```

would be translated into:

```
Type5 =         record
     a_IsPresent:        BOOLEAN;
     a:         INTEGER;
```

```
    end;
```

5.4. Discussion

The above list of ASN.1 constructs covers the basic part of ASN.1. The so-called macros and certain recent extensions are not addressed, however. It is interesting to note that most features of ASN.1 can be translated in a natural manner into specification and programming languages, as shown above. The ASN.1 construct that encountered most difficulties is clearly the SEQUENCE/SET OF construct for which no corresponding concept exists in Estelle and most programming languages. This difficulty does not appear for the translation to LOTOS which contains a corresponding "String" construct [Boch 89h]. However, the data structure adopted for the translation into Estelle was partly influenced by considerations for implementation efficiency, since a natural translation into C implementation data structures was a concern. This concern is not addressed by the "String" data structure of LOTOS.

The implementation considerations for the C data structures lead to the use of pointers for various list structures. The access and update routines discussed in the subsections above have the property of completely hiding the pointer structures from the user. However, if the user wants to make a complete copy of a given PDU, which is represented in such a pointer data structure, he/she has to know its detailed structure. In order to avoid this difficulty, the ASN.1-Estelle compiler provides suitable copying routines for each ASN.1 type definition of the form SEQUENCE, SET and CHOICE.

5.5. Overview of a tool environment

In the following, we describe a set of support tools which have the purpose of providing an implementation environment for OSI Application layer protocols in conjunction with the use of the Estelle specification language. It is assumed that an ASN.1 description of the PDU's exist, and that an Estelle specification of the protocol is used in the implementation process. The support tools provide for the following steps:

(1) the automatic generation of encoding and decoding routines for the PDU's, and C type definitions.

(2) the automatic translation of the ASN.1 description of the PDU's into corresponding Estelle data structures, and the generation of declarations for the corresponding manipulation primitives, and

(3) the automatic generation of implementation code from the Estelle specification, and the generation of the algorithmic bodies for the corresponding manipulation primitives.

As indicated in Figure 2, this approach requires the compatibility between the data structures used in the encoding and decoding routines and the corresponding data structures obtained from the Estelle translation. A more detailed configuration of the support environment is shown in Figure 3. It is important to note that the Estelle data structures obtained by the translation step (2) above must be integrated into the Estelle specification of the protocol.

These structures have an impact on the manner in which the processing of the PDU's can be described in the protocol specification. As indicated in Figure 3, the complete Estelle protocol specification is not obtained automatically. In order to simplify the implementation process, it would be useful if formal specifications of the OSI standard protocols would be available.

The ASN.1-Estelle implementation environment comprises three tools corresponding to the three steps identified above, which provide translation into the implementation language C. The first step is provided by the tool CASN1 developed at the University of British Columbia [Neuf 90b]. The third step is provided by the NBS Estelle compiler [Este 87b]. The ASN.1 to Estelle translation of the second step was specially developed for this purpose. The system has been used for several implementations, such as Alternating Bit Protocol, ACSE, ROSE.

NOTE TO THE EDITOR: Figure 3 goes here; it comes from a PowerPoint document

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

A number of incompatibilities had to be bridged and a number of implementation restrictions imposed by the two existing tools had to be accommodated. The necessary adjustments were made for each of these cases by using the following approaches:

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

(b) The automatic introduction of changes into the data structures generated by the CASN1 tool. This approach was realized through the automatic generation of text editing commands during step (2), and the automatic execution of these commands on the source code generated in step (1). The "SED" (standard Unix stream editor) is used for this purpose.

6. Conclusion

ASN.1 is widely used for the definition of OSI Application layer protocols, but it lacks facilities for defining the access to particular components of the data structure and any actions associated with the operations to be performed within the protocol entity. These latter aspects are, however, covered by formal description techniques (FDT's) and other formal specification languages. In order to take full advantage of the automated FDT tools developed for the validation, implementation and testing of communication protocols, it is therefore important that the existing ASN.1 PDU definitions for OSI Application protocols be related to a complete formal specification of the protocols, which may be written in Estelle, Lotos, SDL or any other specification language.

A similar integration problem arises in the context of object-oriented specifications, as they are currently used in the development of many OSI Application layer protocols, in particular for management, distributed processing and transaction systems. We have indicated in Section 4 how the concepts of object-orientation and inheritance can be combined with the entity-relationship database model in the context of an object-oriented specification language, called Mondel. It is not clear at this point whether a suitable object-oriented specification formalism can be obtained based on existing FDT's (see for instance [Cusa 89, Moll 87]) or whether a new language, such as Mondel, would be more suitable for the development of formal specifications of OSI applications and other distributed applications.

The approach for the integration of ASN.1, as discussed in this paper, is oriented toward translation of ASN.1 into a complete formal specification language, e.g. an FDT. This approach has the characteristic of largely automating the implementation process when support tools for these FDT already exist, without the need of changing them. Such automation provides a faster and more reliable implementation process. It is also possible to use ASN.1 type definition for other purpose than PDU definition, such as service parameters or internal types in a specification.

While this paper focus its attention to the use of Estelle or Mondel as a description technique to be combined with ASN.1, a similar approach can be persued for the translation of ASN.1 into other FDT's such as Lotos and SDL, or other object-oriented specification languages. In particular, SDL includes type constructs similar to those of Estelle. Therefore the translation of ASN.1 described in Section 5 could be adapted for SDL as the target language.

References

[Alve 87] Alvey, Formal Methods Applied to Protocols (FORMAT): A Framework for the Underlying Concepts of Communications Protocols, Report no.4 to the Alvey Directorate, UK, February 1987.

[Amer 89] P. America, *A behavioural approach to subtyping in object-oriented programming languages*, Philips J. Res. (Netherlands), Vol. 44, Nos. 2-3, pp.365-383, 1989.

[ASN1 C] ISO 8825, Information Processing - Open systems Interconnection - Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

[ASN1] ISO 8824, Information Processing - Open systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1).

[Beli 89] F. Belina and D. Hogrefe, The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems, Vol. 16, pp.311-341, 1989.

[Boch 87c] G. v. Bochmann, Usage of protocol development tools: the results of a survey, (invited paper), 7-th IFIP Symposium on Protocol Specification, Testing and Verification, Zurich, May 1987, pp.139-161.

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

[Boch 88h] G. v. Bochmann and A. Finkel, *Impact of queued interaction on protocol specification and verification*, Proc. Intern. Symp. Interoperable Inf. Systems (ISIIS), Nov. 1988, Tokyo, pp. 371-382.

[Boch 89f] G. v. Bochmann, *Inheritance for objects with concurrency*, submitted for publication, Technical Report 687, Université de Montréal, 1989.

[Boch 89h] G. v. Bochmann and M. Deslauriers, Combining ASN1 support with the LOTOS language, Proc. IFIP Symp. on Protocol Specification, Testing and Verification IX, June 1989, North Holland Publ., pp.175-186.

[Boch 90f] G. v. Bochmann, D. Ouimet and G. Neufeld, Implementation support tools for OSI application layer protocols, Technical Report no. 720, Université de Montréal, submitted for publication.

[Boch 90g] G. v. Bochmann, Protocol specification for OSI, Computer Networks and ISDN Systems 18 (April 1990), pp.167-184.

[Boch 90l] G. v. Bochmann, M. Barbeau, M. Erradi, L. Lecomte, P. Mondain-Monval and N. Williams, *Mondel: An Object-Oriented Specification Language*, submitted for publication, Technical Report no. 748, Université de Montréal, 1990.

[Boch 90z] G. v. Bochmann et al., System specification with MONDEL and relation with other formalisms, Progress Report No. 13 for CRIM/BNR project, June 1990.

[Boch 91] G. v. Bochmann, S. Poirier and P. Mondain-Monval, *Object-oriented design for distributed systems and OSI standards*, submitted for publication.

[Boch 91i] G. v. Bochmann, L. Lecomte and P. Mondain-Monval, *Formal Description of Network Management Issues*, to be presented at Int. Symp. on Integrated Network Management (IFIP), Arlington, US, April 1991.

[Bolo 87] T. Bolognesi and E. Brinksma, Introduction to the ISO Specification Language Lotos, Computer Networks and ISDN Systems, vol. 14, no. 1, pp.25-59, 1987.

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

[Chen 76] P. P. Chen, *The Entity-Relationship model - Toward a unified view of data*, ACM Trans. on Database Systems, Vol. 1, No. 1, March 1976, pp.9-36.

[Cusa 89] E. Cusak, *Refinement, conformance and inheritance*, Workshop on Theory and Practice of Refinement, Open Univ., UK, Jan. 1989.

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

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

[Moll 87] B. Moller-Pedersen, D. Belsnes and H. P. Dahle, *Rationale and tutorial on OSDL: an Object-Oriented extension of SDL*, Computer Networks and ISDN Systems, Vol. 13, 1987, pp.97-117.

[Mond 90] P. Mondain-Monval, Object-oriented Model for the OSI Reference Model, Technical Report no.736, Université de Montréal, 1990.

[Neuf 90b] G. W. Neufeld and Y. Yang, The Design and Implementation of an ASN..1-C Compiler, IEEE Transactions on Software Engineering, October 1990, Vol 16 Number 10, pages 1209-1220.

[OSI 83] IEEE, Special issue on Open Systems Interworking, Proc. of the IEEE, Dec. 1983.

[OSI ER] ISO, Information processing systems - Concepts and terminology for the conceptual schema and the information base, ISO TR 9007 (1987).

[OSI mb] ISO, Structure of Management Information, Part 4: Guidelines for managed object definition, draft proposal.

[OSI RO] ISO, *Remote Operations*, IS 9072.

[Pehr 90] B. Pehrson, *Protocol verification for OSI,* Computer Networks and ISDN Systems 18 (1989/90) 185-201.

[Sari 89c] B. Sarikaya, *Conformance Testing: Architectures and Test Sequences*, Computer Networks and ISDN Systems 17 (1989), pp. 111-126.

[Scol 87] ISO 97/21 N1540, G.Scollo, *Potential enhancements to Lotos*, 1986.

[Vuon 88] S. T. Vuong, A. C. Lau and R. I. Chan, *Semiautomatic Implementation of Protocol using an Estelle-C Compiler,* IEEE Transaction on Software Engineering, vol. 13, no. 3, pp. 384-393, March 1988.

[X.407] CCITT, X.407 / ISO 8505-4, Message Handling Systems Abstract Service defintion Conventions, 1988.