An Introduction to the ASN.1 MACRO Replacement Notation

Nilotpal Mitra

Abstract syntax notation one (ASN.1) is used to specify application-layer protocols for open-systems interconnection (OSI) applications. These applications include message-handling services (MHS) and OSI-based integrated services digital network (ISDN) signaling protocols, such as transaction capabilities (TC). One common aspect of these protocols is that they use the remote-operations service element (ROSE), which is based on a request/reply paradigm. In particular, ROSE and its users employ the ASN.1 MACRO notation to capture the application-specific syntax and semantics of a request (to perform some action) and the report of its outcome. The MACRO notation is difficult to understand, however, and is ambiguous when used in defining a message's data structure. Furthermore, it is not possible to build general-purpose ASN.1 compilers that can interpret the semantics of arbitrary MACRO definitions, as each definition is application specific. Recently, a replacement for the MACRO notation, which also corrects the ambiguous aspects of the ASN.1 notation, has been standardized. The new ASN.1 standards provide a notation for defining information-object classes—the means to specify complex concepts used in defining an application-layer protocol. Notations are also provided to show relationships between data in different parts of a protocol message, and to parameterize ASN.1 type definitions. This paper discusses the principal features of the new ASN.1 notation. Rather than describe the new notation in general, the text guides users of ROSE—who are the major users of the MACRO notation—on the replacement of the `OPERATION` and `ERROR` MACROs of ROSE.

Introduction

Abstract syntax notation one (ASN.1) is designed to describe the message structure of application-layer protocols in open-systems interconnection (OSI). It provides a way to specify the data structures of the application-protocol data units (APDUs), without specifying how these APDUs are encoded for transfer during actual communications. To date, almost all OSI application-layer protocols have been specified using ASN.1, as have some integrated services digital network (ISDN) signaling protocols that incorporate OSI-based protocols, such as the transaction capabilities (TC) protocol of Signaling System 7 (SS7) and the Q.932 protocol of the Digital Subscriber Signaling One protocol suite.

The MACRO notation is one aspect of the original ASN.1[1] that is used extensively by OSI applications and ISDN signaling protocols. It plays a substantial role in defining the communication-protocol messages for OSI applications. These applications include message-handling services (MHS), directory and network management, as well as those requiring ISDN signaling, such as intelligent network and services involving user and terminal mobility.

These applications are all based on use of the remote-operations service element

Panel 1. Abbreviations, Acronyms, and Terms

abstract syntax	the complete set of APDUs of an application protocol
APDU	application-protocol data unit
ASN.1	abstract syntax notation one
CCITT	International Telegraph and Telephone Consultative Committee
IEC	International Electrotechnical Commission
ISDN	integrated services digital network
ISO	International Organization for Standardization
ITU-T	International Telecommunications Union - Telecommunication Standardization Sector
MHS	message-handling services
OSI	open-systems interconnection
Q.932	a communication protocol, which is part of the Digital Subscriber Signaling One protocol suite
ROSE	remote-operations service element
SS7	Signaling System 7, a communication protocol
TC	transaction capabilities, a part of the SS7 protocol suite

(ROSE)[2,3], which is a communications paradigm based on a simple request/reply model. In fact, the MACRO notation was introduced to define a compact way of capturing the syntax and semantics of an arbitrary or application-specific request to perform some action, together with the possible responses describing the outcome. This was provided through the `OPERATION` and `ERROR` MACROs of ROSE.

There are more complex MACROs in the MHS and directory-protocol specifications to capture some other concepts used there. It is safe to say, however, that the vast majority of users—particularly those in the ISDN signaling community—understand the MACRO notation principally through its use in ROSE to describe operations and errors.

Use of the MACRO notation, in any general way, has been plagued by the incomprehensible text that describes the notation's use. Considerable reader perseverance is required to understand the text. It would not be an exaggeration to say that most readers trade the ability to write a few instances of some specific MACRO, such as the `OPERATION` MACRO of ROSE mentioned earlier, for an understanding of the formal notation. In addition, with the development of automated compilers to read protocol written in ASN.1, the MACRO notation cannot generally be processed by machine. Compilers can be built to process the use of specific MACRO definitions, such as the `OPERATION` MACRO for ROSE-based applications. But completing the task requires extra input, which is application specific, if the precise semantics associated with a MACRO are to be extracted.

Another difficulty with use of the original ASN.1 notation is its inability to show the relationship between different parts of an APDU. This is of particular relevance to the ROSE protocol specification, where the ROSE APDUs—being generic carriers of information from other applications—have to leave "holes" in the APDU. The "shape" of the holes is determined—at the instance of use—by the value contained in another component of the APDU. A limited, formal linkage was provided in the original ASN.1 notation through the use of the `ANY DEFINED BY` construction. This does not permit, however, a complete parsing of a received APDU without the use of prior application-specific input.

Since 1990, the International Telecommunications Union - Telecommunication Standardization Sector (ITU-T), International Organization for Standardization (ISO), and International Electrotechnical Commission (IEC) have collaborated in developing a replacement for the MACRO notation, as well as removing the ambiguous aspects of the ASN.1 notation, which prevents complete processing by machine. This work, now at the status of common ITU-T recommendations and ISO/IEC international standards, has resulted in the expansion of the original ASN.1 standard into four parts.[4–7]

This paper provides an introduction to the concepts of the new ASN.1 specifications. Rather than treat the subject with generality, it will probably be most useful if the current users of ROSE-based MACROs are guided to the use of the notation that will replace them. Then the reader, presumably convinced by the new notation's ease of use, will be able to study and create more complex examples that will be required by other OSI and ISDN signaling applications. The paper is divided into the following sections:

- "The MACRO Notation" introduces the earlier notation through the use of the ROSE `OPERATION` and `ERROR` MACROs. By showing the shortcomings of the existing notation, this backward detour will provide the motivation for introducing the new concepts.

Figure 1. The OPERATION MACRO.

```
OPERATION MACRO ::=
BEGIN
TYPE NOTATION             ::= Argument Result Error LinkedOperations
VALUE NOTATION            ::= value(VALUE CHOICE{
                                                localValue INTEGER
                                                globalValue OBJECT IDENTIFIER })
Argument                  ::= "ARGUMENT" NamedType | empty
Result                    ::= "RESULT" ResultType | empty
Error                     ::= "ERRORS" "{"ErrorNames"}" | empty
LinkedOperations          ::= "LINKED" "{"LinkedOperationNames"}" | empty
              .....
              .....
ResultType                ::= NamedType | empty
NamedType                 ::= identifier type | type
              .....
              .....
END
```

- "Information Objects" introduces the concept of information objects and classes, the primary replacement for the MACRO. This section also provides the notation for how the contents of an information object can be accessed, thereby preparing for its subsequent application in defining APDUs.
- "Applying Constraints" describes constraint placement on the various components of an APDU, which will be the replacement for the `ANY DEFINED BY` construction of the current ASN.1.
- "Parameterization of ASN.1 Definitions" describes how ASN.1 type definitions can be parameterized. That is, each type definition contains formal parameters that can be filled in later to achieve a complete definition.
- "Useful Applications of This Notation" discusses the new notation's use.

As previously mentioned, all the examples are taken from ROSE.

The MACRO Notation

This was an extension of the standard ASN.1 notation. The standard notation defines a way to represent certain basic ASN.1 types—for example, `INTEGER`, `BOOLEAN`, and `BIT STRING`—and their values. It also defines the notation that allows ASN.1 users to define more complex types, such as `SET`s and `SEQUENCE`s of these basic types, as well as describe the values that such types can take. In this way, an ASN.1 user (for instance, an application-layer protocol designer) can define the data structures that define the APDU. A user can also represent the values of the APDUs that are present during some instance of communication.

The purpose of providing the MACRO definition in the original ASN.1 notation[1] was to take the user one step further. That is, in addition to being able to write grammatically correct sentences in the standard ASN.1 notation, users were now allowed to construct syntactically correct extensions of the standard notation—known as MACRO definitions—to create nonstandard, user-defined type and value notations to capture some complex concept that was required in a protocol specification. An example of such a complex concept is the notion of a remote operation in ROSE, which will be used both to illustrate the MACRO notation and its shortcomings.

Remote Operations. This concept is a paradigm for interactive communication between two objects—a client and a server, for example. The basic interaction is the invocation of an operation (the request to the other object to perform some action), the action performed by the latter and, possibly, a report of the outcome (whether performed successfully or unsuccessfully).

Each application will have its own set of remote operations, together with their possible outcomes. Rather than have each application designer define a new protocol to convey the specifics of its request/response, the ROSE protocol provides a common, standard method

```
DaysOfTheWeek          ::= ENUMERATED {sunday(0), monday(1),....saturday(6)}
today DaysOfTheWeek    ::= tuesday
```

Figure 2. A typical ASN.1 type and value definition.

to convey arbitrary requests and responses by leaving an appropriate space in the ROSE APDUs. This space is where the data structure, corresponding to the request or response, may be fitted in dynamically when communication actually takes place.

It is important to ensure that this can be done at each instance of communication. So, rather than specify the exact shape (more formally, the type of data structure) of the structure that can be placed in an APDU, the APDU is defined to convey arbitrary structures. In order to allow for this, the users of remote operations are provided a template—the `OPERATION` and `ERROR` MACROs. These MACROs can be filled in an application-specific manner. They are then used, together with ROSE, to complete the application-protocol definition.

Use of MACROs in Remote Operations. The `OPERATION` MACRO collects all the syntactic aspects of the building blocks that comprise a remote operation. A remote operation is characterized by:

- The data type of the argument that specifies the requested operation;
- The data type of the result, if any, returned upon successful completion of the operation;
- A set of errors, one of which may be returned to signal the cause of the unsuccessful operation performance;
- Other operations linked to the one invoked, which may be invoked by the performer before completing the originally invoked operation.

A complete definition of the `OPERATION` MACRO is available.[2] However, Figure 1 is a considerably simplified definition.

What this figure shows, quite simply, is that the ROSE standard has defined a generic template (or MACRO) for all users of remote operations to define values for a type, known as `OPERATION`. This type is considerably more complex than either such standardized ASN.1 types as `INTEGER`, `BOOLEAN`, and `OCTET STRING`, or those that can be formed by combining these into `SET`s or `SEQUENCE`s. In fact, the `TYPE NOTATION` for `OPERATION` consists of four items: `Argument`, `Result`, `Error`, and `LinkedOperations`. Together, these items describe any operation. They can be subsequently expanded to show the syntax to be followed when using this `OPERATION` MACRO definition to write some example of an operation. Thus, an operation's `Argument` is defined by writing the keyword `ARGUMENT`, followed by an ordinary ASN.1 type definition for the argument data type. The choice of data type is at the discretion of the user.

The notation for writing any value for this complex type `OPERATION` is provided by the construction `VALUE NOTATION`. This says that the value that any member of the type `OPERATION` can take is either an `INTEGER` or an `OBJECT IDENTIFIER`.

The `VALUE NOTATION` of the MACRO definition is the first source of confusion for most readers. In the standard ASN.1 notation, the value of a particular type normally has some direct or intuitive relationship to the type as shown in Figure 2, where, on the first line, a type `DaysOfTheWeek` is defined as an `ENUMERATED` type (with the specific values enumerated). On the second line, the value notation is used to select a particular value of that type, known as `tuesday`. (The second line assigns the value `tuesday` to `today`.)

This straightforward relationship between a type definition and its values is missing from the `OPERATION` MACRO. The type of operation, defined by using the `OPERATION` MACRO, is a composite of all four properties of an operation: namely, its `ARGUMENT`, `RESULT`, `ERROR`, and `LINKED` (operations), if any. Using the definition of the `OPERATION` MACRO, the value notation for a hypothetical data-base look-up operation (`get`) to obtain a person's address upon providing a name, appears as:

```
get OPERATION
ARGUMENT       Name
RESULT         Address
ERROR          {unknown, noAccess}
::= localValue:1
```

If everything in bold is considered to be a type definition, the example resembles the value notation for an ordinary ASN.1 type, such as the second line of the preceding example (`DaysOfTheWeek`) shown in Figure 2.

The lack of an obvious relationship between the type and value is the result of permitting user-defined syntax. One reason for creating the general MACRO notation is to permit users to define "fancy" data types and assign them values. The relationship between these fancy types and their values is buried in the user's specifications.

How MACROs Are Used in Protocol. The purpose of designing the `OPERATION` MACRO, as discussed in the previous subsection, is to capture the syntactical aspects of the building blocks needed to define a remote operation. The next aspect is how to include these building blocks in the actual communications protocol.

The ROSE standard provides four APDUs that act as carriers of the data types comprising the building blocks:

- `Invoke` conveys the request to perform some remote operation;
- `ReturnResult` signals the successful completion of the operation, if required;
- `ReturnError` reports the inability to perform the requested operation, if required; and
- `Reject` reports any syntactic errors in the first three APDUs.

For example, the `Invoke` APDU is defined as:

```
Invoke   ::= SEQUENCE
{
    invokeId   INTEGER,
    linkedId   [0] IMPLICIT INTEGER OPTIONAL,
    opcode     OPERATION,
    argument   ANY DEFINED BY opcode OPTIONAL
}
```

where the components—crucial to the following discussion—have been highlighted in bold.

The `Invoke` APDU announces that a request to perform a remote operation has arrived. As the performer may have many such requests from the invoker, each new request is identified by a new integer number, the `invokeId`. The `opcode` identifies the precise remote operation to be performed. The value notation of the type `OPERATION`, as discussed in the previous subsection, requires that `opcode`—the value actually conveyed in the protocol—be an `INTEGER` or a globally unambiguous `OBJECT IDENTIFIER` value.

To ensure that this APDU structure can be used by *any* application, the designers of the generic ROSE protocol had to leave a completely flexible slot for the application-specific operation argument. The argument that may be sent with this invocation is not ***any*** ASN.1 type, as suggested by the key word `ANY`. It is really a type constrained in some specific way—through the use of the `DEFINED BY` construct—to a specific data type by the `opcode` conveyed as a component of this APDU.

The specific way in which one component in an APDU constrains another cannot be expressed by the notation itself. In the case of the `Invoke` APDU, there is a relationship between the `opcode` and the `argument` components. To determine what type to substitute for the `ANY` type, the ROSE specifications must be read. In those specifications, the type that replaces `ANY` is the ASN.1 data type that follows the key word `ARGUMENT` for the specific operation identified by the value in `opcode`. (In the example of the operation `get`, in the previous subsection, the type of the `argument` component in the `Invoke` APDU is some user-defined ASN.1 type known as `Name`.)

Similarly, for the `ReturnResult` APDU,

```
ReturnResult   ::= SEQUENCE
{
    invokeId            INTEGER,
    SEQUENCE
    {
    opcode              OPERATION,
    result              ANY DEFINED BY opcode
    }  OPTIONAL
}
```

the `ANY DEFINED BY` type of the `result` component is replaced by the ASN.1 data type that follows the `RESULT` keyword for the operation referenced by `opcode`. (In the example of the operation `get` in the previous subsection, the type of the `result` component in the `ReturnResult` APDU is the ASN.1 type known as `Address`.) The reader can look up the syntax of the other two APDUs, `ReturnError` and `Reject`.[2,3]

One can imagine a model of an ASN.1 compiler for ROSE-based applications that generates a table having one column for the `opcode`, one for the `ARGUMENT`, and one for the `RESULT`. This model illustrates the second shortcoming of the use of the notation: the inability to specify, ***within the notation***, the precise relationship between components in an APDU. A model compiler, receiving a bit string corresponding to the `Invoke` APDU,

```
OPERATION ::= CLASS
{
          &ArgumentType       OPTIONAL,
          &ResultType         OPTIONAL,
          &Errors             ERROR OPTIONAL,
          &LinkedOperations   OPERATION OPTIONAL,
          &operationCode      Code  UNIQUE  OPTIONAL
}

Code                          ::= CHOICE {localValue INTEGER, globalValue OBJECT IDENTIFIER}
```

Figure 3. The information-object class `OPERATION`.

cannot "decide" whether to parse the component, either after a given received `opcode` against the contents of the ARGUMENT, or the RESULT column for the row accessed by the received opcode, ***unless it has previously been fed this information from outside the notation.***

A compiler must be "taught," therefore, to understand the MACROs that it will encounter. Different standards use different MACRO definitions—the OSI MHS standard[8] defines six MACROs, for instance, in addition to those it borrows from other specifications. Thus, it is not possible to build a general-purpose ASN.1 compiler that has the potential to understand every MACRO definition—including those as yet unwritten—as well as the constraints imposed among the components of an APDU using these MACROs.

Information Objects

The replacement for the MACRO notation retains the idea that applications may have complex concepts, aspects of which need to be expressed in data structures conveyed by protocol. Thus, the ROSE specifications define the concept of a remote operation, while the MHS protocol defines the concept of a port, through which an object's capabilities can be accessed.

Each of these concepts is classified by providing a template, analogous to a MACRO definition, known as an *information-object class.* The template shows the attributes of the objects belonging to a class. A notation is provided to derive instances of a class. Information-object class specifications replace MACRO definitions, therefore, for various objects of interest to an application.

Information-Object Classes. As before, these are best explained by means of examples. With reference to the properties of remote operations discussed earlier, in the subsection titled "Use of MACROs In Remote Operations," the new notation permits the description of a class of objects known as OPERATION[9-11]. This is simplified considerably here to show relevant features only, and is defined in Figure 3.

The new notation defines the template for a class of objects assigned the name OPERATION, which consists of five "slots" or, more formally, *field names.* The descriptors of each field start with an ampersand (&), and are followed by a word beginning with either a lower-case or upper-case letter. The latter distinction helps identify the sort of data that may be placed in the corresponding slot when defining an instance of this class.

Words having all upper-case letters are typically used for new keywords, such as CLASS and UNIQUE, and for names assigned to such object classes as OPERATION and ERROR. If the field name starts with an upper-case letter, it is a place holder for either an arbitrary ASN.1 type, a set of information objects, or a set of values of some type. If it is a set of information objects, the descriptor of the object class to which they belong follows. If the field name begins with a lower-case letter, it takes the *value* of the ASN.1 type or object class that follows. If a field is marked OPTIONAL, such a slot need not be filled when defining an instance of the class. These points are further clarified in the following subsection.

Information-Object-Class Fields. In the case of the information-object-class OPERATION, the first two fields, `&ArgumentType` and `&ResultType`, are place holders for an arbitrary ASN.1 data type. Each is a *type field,* because the name starts with an upper-case letter and there is no object class or specific type defined after it. The actual ASN.1 type is placed here by the application-protocol designer when creating an instance of this class.

Figure 4. The information-object class ERROR

```
ERROR ::= CLASS
{
     &ParameterType     OPTIONAL,
     &errorCode         Code  UNIQUE  OPTIONAL
}
Code               ::= CHOICE {localValue INTEGER, globalValue OBJECT IDENTIFIER}
```

This leaves a completely flexible slot for application-specific information, just as was possible with the MACRO notation. The difference is that, unlike the MACRO notation, a user is not allowed to make up any new, nonstandard, user-defined type notations. All the notational tools that a user will require are contained in the new, four-part ASN.1 standards.[4-7] Thus, the difficulty of understanding user-defined syntax is circumvented. Compilers can now be built that only "comprehend" the new, standard ASN.1 notation.

The third and fourth fields, `&Errors` and `&LinkedOperations`, are specified to be place holders for objects that belong to the classes ERROR and OPERATION, respectively. Such objects can only be a set of errors and a set of operations, so these fields contain *object-set fields*. The general naming rule is that field names starting with an upper-case letter are either type fields or object sets, the latter being distinguished by specifying to which class the objects in the set belong.

The fifth field, `&operationCode`, is an example of a *value field*. It is a place holder for some value of the ASN.1 type that qualifies it. In this case, the value is either some integer or an object-identifier value. The special key word UNIQUE reveals that when instances of the class OPERATION are defined, they should each be assigned a distinct operation-code value. The reason is that should these instances be collected into sets (which, as will be pointed out, they inevitably will be in the course of specifying remote, operations-based application protocols), each is uniquely identifiable within that set.

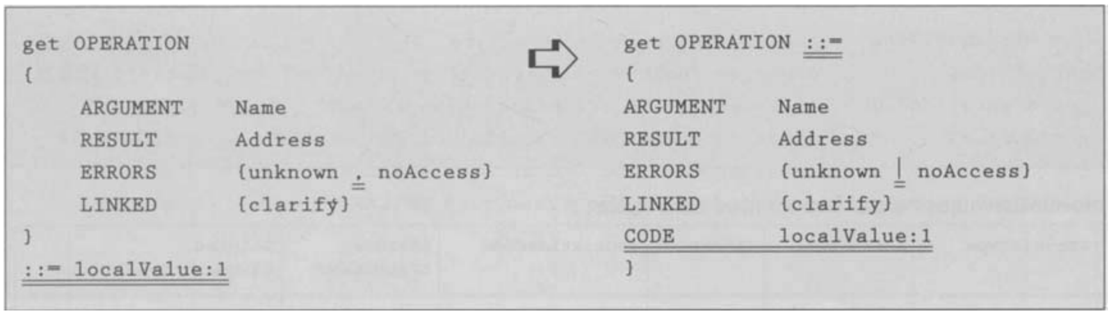
To complete the definitions, the object class ERROR is defined in Figure 4, where `&ParameterType` is a type field and `&errorCode` is a value field.

Defining Instances of an Information-Object Class. Given the foregoing definition of the information-object class OPERATION, an instance of that class, such as a remote operation, can be defined. Using a slightly expanded version of the example `get`, discussed earlier in the subsection titled "Use of MACROs in Remote Operations," by adding a linked operation `clarify`, the new notation permits the writing of:

```
get  OPERATION ::=
{
    &ArgumentType        Name,
    &ResultType          Address,
    &Errors              {unknown | noAccess},
    &LinkedOperations    {clarify},
    &operationCode       localValue:1
}
clarify OPERATION ::=
{
    &ArgumentType        ProvideZipCode,
    &ResultType          RequestedInformation,
    &operationCode       localValue:2
}
unknown ERROR ::=
{
    &errorCode           localValue:1
}
noAccess ERROR ::=
{
    &errorCode           localValue:2
}
```

Therefore, creating an instance of the class OPERATION generally consists of giving a name to that instance, declaring it to be of the class OPERATION, and filling in the fields—as appropriate—with a value of either some ASN.1 type, an arbitrary ASN.1 type, an object set, or a value set.

User-Defined Syntax for Information-Object Classes. The notations discussed in the two preceding subsections can become difficult to read, particularly when defining more complex information-object classes. In this case, the information-object-class designer can provide a user-friendly notation for defining instances of the class. In the OPERATION and ERROR information classes, the



```
get OPERATION                              get OPERATION ::=
{                                          {
    ARGUMENT    Name                       ARGUMENT    Name
    RESULT      Address                    RESULT      Address
    ERRORS      {unknown , noAccess}       ERRORS      {unknown | noAccess}
    LINKED      {clarify}                  LINKED      {clarify}
}                                          CODE        localValue:1
::= localValue:1                           }
```

Figure 5. Going from the OPERATION **MACRO to the** OPERATION **information-object class.**

designers of the new ROSE specifications have defined a special syntax that strongly resembles the old MACRO notation for defining operations and their errors. This special syntax, shown in the following figure in bold for the `OPERATION` information-object class, is appended, using the key words `WITH SYNTAX`, to the definition of the information-object class.

```
OPERATION ::= CLASS
{
    &ArgumentType         OPTIONAL,
    &ResultType           OPTIONAL,
    &Errors               ERROR OPTIONAL,
    &Linked               OPERATION OPTIONAL,
    &operationCode        Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
    [ARGUMENT             &ArgumentType]
    [RESULT               &ResultType]
    [ERRORS               &Errors]
    [LINKED               &Linked]
    [CODE                 &operationCode]
}
```

The optional presence of a field is shown by enclosing the designated field in square brackets ([...]). One cannot determine by looking at the user-defined syntax whether a particular field is a type field or an object-set field. Therefore, it is necessary to keep the standard syntax in mind when writing instances of an information-object class in the user-defined syntax. As discussed later, in the subsection titled "An Unexpected Bonus," the user-defined syntax allows the expression of some relationships not permitted in the standard notation.

Similarly, the ERROR information-object class is defined to allow the following user-friendly syntax:

```
ERROR ::= CLASS
{
    &ParameterType        OPTIONAL,
    &errorCode            Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
    [PARAMETER            &ParameterType]
    [CODE                 &errorCode]
}
```

Using this user-defined syntax, the operation `get`, discussed earlier in the subsection titled "Defining Information-Object Classes," can be written as:

```
get  OPERATION ::=
{
    ARGUMENT              Name
    RESULT                Address
    ERRORS                {unknown | noAccess}
    LINKED                {clarify}
    CODE                  localValue:1
}
clarify OPERATION ::=
{
    ARGUMENT              ProvideZipCode
    RESULT                RequestedInformation
    CODE                  localValue:2
}
unknown ERROR ::=
{
    CODE                  localValue:1
}
noAccess ERROR ::=
{
    CODE                  localValue:2
}
```

Figure 6. Accessing fields of the information-object get.

```
get.&ArgumentType      accesses Name -- an ASN.1 type
get.&Errors            accesses {unknown | noAccess} -- an object set of class ERROR
get.&operationCode     accesses 1 -- a value 1 of type INTEGER
get.&Linked            accesses {clarify} -- an object set of class OPERATION
```

Table I. The OPERATION **information-object class represented as a table**

	&ArgumentType	&ResultType	&Errors	&Linked	&operationCode	&Errors. &Parameter	&Linked. &ArgumentType	...
get	Name	Address	{unknown| noAccess}	{clarify}	1		ProvideZipCode	
clarify	ProvideZipCode	RequestedInfo			2			

Of course, the creator of an object—using the user-defined syntax for the information-object class—must understand the nature of the fields, such as whether it is a type, value, or object-set field. However, the actual instances are defined without the object field names. In the OPERATION and ERROR information-object classes, the user-defined syntax was chosen to resemble the corresponding MACRO realization, shown in the subsection "Use of MACROs in Remote Operations." The contrast between the two forms is shown in Figure 5, with the MACRO notation on the left evolving to the new user-defined syntax of the information-object class definition on the right. These differences are double underlined:

The information-object class OPERATION, as defined in the new ROSE specifications[9], is considerably "richer" in content than that conveyed by the MACRO of the same name. It is not confined, like the MACRO notation, to providing only user-defined type and value notations. The OPERATION information-object class can list various properties of a remote operation, such as whether it is synchronous, whether it always returns a result, and whether the argument and result data types may be excluded at a user's option. These aspects, in the earlier notation, could only be expressed in comments.

Accessing Fields of an Information Object. As previously discussed, a field of an information-object class can contain an arbitrary ASN.1 type, a value or a set of values of an ASN.1 type, an information object, or sets of such objects. A notation is provided for accessing the contents of a particular field of an information object. This is done by appending a dot (".") between the information-object-class name and the particular field type. Thus, for the object class OPERATION, the construction OPERATION.&ArgumentType corresponds to some arbitrary type, while OPERATION.&Errors corresponds to a set of information objects of class ERROR. The type accessed becomes known when a specific operation of the governing class OPERATION is substituted. Using earlier examples of the get and clarify operations results in Figure 6.

Viewing an Information-Object Class as a Table. One advantage of defining information-object classes in the new notation is that they may be viewed conceptually as a table, with the object-field names (starting with an ampersand) being the column headings, and the row headings being instances of the class. For the OPERATION information-object class, for example, Table I can be constructed.

This table could be infinitely long, because the OPERATION information-object class is recursive. That is, one of its fields, &Linked, is of class OPERATION. This, when expanded as a table, leads to a series of columns. One column in this series, &Linked.&Linked, is of class OPERATION, and so on.

The table rows are populated with the contents of the two examples of operations, get and clarify, defined earlier. As can now be seen, the dot notation, described in the preceding subsection, allows access to the contents of a particular cell—the intersection of a row, an object instance, and a column—a field name.

Applying Constraints

The previous sections in this paper described how it is possible to access the contents of various fields of an information object using the dot notation. This process is useful when specifying the fields of some APDUs. Referring back to the Invoke APDU, discussed earlier in the subsection titled "How MACROs Are Used In

```
Invoke  ::= SEQUENCE
{
    invokeId        InvokeID,
    linkedId        [0] IMPLICIT InvokeID OPTIONAL,
    opcode          OPERATION.&operationCode({SomeOperations}),
    argument        OPERATION.&ArgumentType({SomeOperations}{@opcode}) OPTIONAL
}
InvokeID ::=        INTEGER
```

Figure 7. Applying constraints on the `Invoke` **APDU.**

Protocol," it is noted that the value taken by the APDU's argument component is dependent, in some way, on the type of operation that is being invoked—that is, on the value taken by the `opcode` component. This relationship was shown using the `ANY DEFINED BY` construction of the earlier ASN.1 standard.

The dot notation is a way of indicating that when the `opcode` takes the value `get.&operationCode` (or equivalently, the integer 1, which is the value actually encoded in the message), the `argument` component is a value of the type `get.&ArgumentType`—that is, `Name`. However, as the `Invoke` APDU has to be written generically so that it can be used for all operation invocations, each identified by an `opcode` value, a method is needed to constrain the arbitrary ASN.1 type produced by `OPERATION.&ArgumentType`. This arbitrary type corresponds to the operation instance indicated by the `opcode`. This is accomplished using the so-called "at" notation. The arbitrary ASN.1 type, arising from `OPERATION.&ArgumentType`, is constrained by appending—within parentheses—the "@" symbol. This symbol is then followed by a reference to a component within the same construction (`SET` or `SEQUENCE`).

This is best explained by writing the `Invoke` APDU in the new notation, with the changes arising from the use of the new notation in bold (see Figure 7).

Notice that, unlike the `ANY DEFINED BY` construction, this notation tells exactly what the relationship is between the `argument` and the `opcode`. Comparing this definition with the tabular form of the information-object class `OPERATION`, discussed earlier in the subsection titled "Viewing an Information-Object Class as a Table," we see that `OPERATION.&operationCode` refers to the column labeled `&operationCode` in the table for the `OPERATION` information-object class. The parameter `SomeOperations` now becomes a constraint that chooses the rows of this table corresponding to the operations in the set. Therefore, the value `opcode` is constrained to be any of the entries in the resultant cells, which are located at the intersection of the column labeled `&operationCode`, and the rows containing the operations in the set `SomeOperations`.

While this constraint, imposed by `SomeOperations`, also holds for the component `argument`, it was previously pointed out that once a particular value is chosen for `opcode`, the further constraint (`@opcode`) singles out a particular cell; namely, the one containing the argument-type definition for that operation code. So, if the `opcode` takes the value `get`, then the `argument` must be the cell specified by the intersection of the row accessed by `get`, and the column labeled `&ArgumentType` for the table `OPERATION`. The entry in that cell is the type `Name`.

The ROSE APDUs, when encoded after being written in this way, are identical to those in the 1988 standard.[2,3] Only now it is possible to state explicitly—in the new notation—how to parse the various components.

Parameterization of ASN.1 Definitions

In discussing the use of remote operations and ROSE, mention was made of how the ROSE APDUs provide a generic structure for all request/reply types of applications. This is a frequent situation in many application-layer protocols, where one standard provides an infrastructure that is used by some other standard, in order to complete the specification of the total application-layer protocol.

In the 1988 ROSE standard, this kind of linkage between standards was created by stating that users of ROSE must import the definitions of the ROSE `OPERATION` and `ERROR` MACROs into the specifications, which is where application-specific operations and errors are defined. The ROSE APDUs, together with the data structures defined in the other specification, form the

Figure 8. Parameterization of the `Invoke` APDU.

```
Invoke {OPERATION:SomeOperations}  ::= SEQUENCE
{
    invokeId        InvokeID,
    linkedId        [0] IMPLICIT InvokeID OPTIONAL,
    opcode          OPERATION.&operationCode ({SomeOperations}),
    argument        OPERATION.&ArgumentType ({SomeOperations}{@opcode}) OPTIONAL
}
InvokeID ::=        INTEGER
```

Figure 9. Further parameterization of the `Invoke` APDU.

```
Invoke {InvokeID:InvokeIDSet, OPERATION:SomeOperations}  ::= SEQUENCE
{
    invokeId        InvokeID ({InvokeIDSet}),
    linkedId        [0] IMPLICIT InvokeID OPTIONAL,
    opcode          OPERATION.&operationCode ({SomeOperations}),
    argument        OPERATION.&ArgumentType ({SomeOperations} ({@opcode}) OPTIONAL
}
InvokeID ::=        INTEGER
```

complete set of APDUs (more formally known as the *abstract syntax*) for the application. This is a cumbersome and confusing method of specifying the abstract syntax, and much of it must be expressed in prose rather than in a more formal manner.

The notation for parameterization of ASN.1 definitions, introduced in Part 4 of the new standard[7], explicitly permits the ability to define some generic ASN.1 data structure—and leave appropriate holes, or place holders within the structure, which can be filled in at some later stage. Unlike the ANY construction in the earlier ASN.1 specification, which simply left the holes without any notational indication of what is placed there (or how), parameterization permits an explicit connection to be made at the level of specification.

Taking the example of ROSE, the ROSE APDUs have place holders for application-specific information pertaining to the operation that is invoked—that is, the argument of the invocation, the result, or the parameter qualifying an error situation. So, rather than repeatedly shaping the ROSE APDUs each time a particular operation is invoked, or its result returned, the APDU is written as a function of a parameter, the set of operations that can be invoked. With only the latest addition in bold, the `Invoke` APDU is shown in Figure 8.

This declares that the object set `SomeOperations`, belonging to the class `OPERATION`, is a formal reference for a parameter that has to be supplied when using the definition of the `Invoke` APDU. (The curly brackets following a type name indicate that the corresponding ASN.1 definition is parameterized.) The object set `SomeOperations` will be specified by an application designer in some standard that uses ROSE.

Useful Applications of This Notation

The new notation permits a number of useful—and in some cases, quite unintentional—applications of ideas that could not be expressed by the MACRO notation. Three of these applications, which are particularly useful to such ISDN signaling applications as the SS7 TC protocol using ROSE, are highlighted in this section.

Alignment of TC with ROSE. The parameterization notion permits a further refinement of the ROSE APDUs. The range of values used for the `invokeId` can be made into a user-defined parameter in the definition of the ROSE APDUs. With this included, the parameterized `Invoke` APDU is defined in Figure 9 (with only the latest addition in bold). The parameter `InvokeIDSet`, to be supplied by the application designer, is a constraint on the integer values that may be taken by the `invokeID` component.

In the 1988 ROSE standards, the `InvokeID` was defined as an `INTEGER`, with no range specified. In reality, most implementations will use integers in some finite

```
recode {OPERATION:operation, OPERATION.&operationCode:code} OPERATION ::=
{
     &ArgumentType     operation.&ArgumentType,
     &ResultType       operation.&ResultType,
     &Errors           operation.&Errors,
     &Linked           operation.&Linked,
     &operationCode    code
}
```

Figure 10. The `recode` operation

range. In the OSI-based TC standards, which contain the ROSE protocol as a subset, the `invokeID` component is explicitly confined to the range (-128, 127). TC can now use the generic, parameterized ROSE APDUs defined in the new ROSE standard, with the parameter `InvokeIDSet` defined as:

```
InvokeIDSet ::= INTEGER(-128..127)
```

Thus, the new notation allows such standards as TC to be fully compatible with the generic APDUs of ROSE, defined earlier in this subsection.

Replacing the Two-Stage Operations Definition. Another useful example that provides an application of both the dot notation and parameterized definitions is an operation known as `recode`, which takes two parameters. The first parameter, `operation`, is some instance of the class `OPERATION`. The second parameter, `code`, is the new operation code to be assigned to the operation. Its definition is shown in Figure 10. Obviously, the `recode{}` operation retains exactly all the fields of the previously defined operation `operation`, but simply replaces the previous value of the operation code with the new value `code`.

There are many occasions when groups of operations (each group defined independently) are found to be useful, if collected together in some module, to form a new application. As operations in each group are typically assigned locally unique integer values, there is the possibility of operation-code value clashes if two or more groups are collected together.

In the past, a far-sighted application designer, who wanted to avoid such *potential* code-value clashes, could assign a globally unique `OBJECT IDENTIFIER` value to each operation. For a variety of reasons not germane to this discussion, such a technique is not always preferred. The new notation circumvents this problem by placing only one requirement: that these values be unique when operation instances are collected into object sets (see the earlier subsection titled "Defining Information-Object Classes"). If it is found—when such a collection is created—that there is the possibility of a clash in code values, the `recode{}` operation can be used to change the code of some operations, so that each operation within a set is uniquely identifiable.

An Unexpected Bonus. The `OPERATION` MACRO, defined in the earlier subsection titled "Use of MACROs in Remote Operations," requires that the data type of the argument, or result of an operation, always be present or absent. In other words, the data type following the key words `ARGUMENT` and `RESULT` *cannot* be followed by the key word `OPTIONAL`. However, it is precisely this situation that is needed for ISDN signaling applications, such as the intelligent-network application protocol. This is where a requirement has been expressed to find a way to allow an intelligent-network application—based on the state of the call processing—to insert (or not to insert) a data type as an argument or result to, respectively, an `Invoke` or a `ReturnResult` APDU.

The user-defined syntax of the `OPERATION` and `ERROR` information-object classes allows a very effective way to express this requirement. The addition of two new fields, `&argumentTypeOptional` and `&resultTypeOptional`, to the `OPERATION` information-object class, with the additions in bold, is shown in Figure 11. This allows the user-defined syntax to show explicitly when a particular argument or result data type can be present as an option. Such an unexpected bonus now permits writers of other information-object classes (not related to ROSE) to express, in their user-defined syntax, the general notion of relationships between fields of an information-object class. That is, a particular field in an object class, such as `&A`, can be present only if another field, `&B`, is also present. This can be done, in general, as:

```
[IF-THAT-IS-PRESENT  &B  [THIS-IS-PRESENT  &A]]
```

Figure 11. The OPERATION **information-object class with useful additions.**

```
OPERATION ::= CLASS
{
    &ArgumentType               OPTIONAL,
    &argumentTypeOptional       BOOLEAN DEFAULT FALSE,
    &ResultType                 OPTIONAL,
    &resultTypeOptional         BOOLEAN DEFAULT FALSE,
    &Errors                     ERROR OPTIONAL,
    &Linked                     OPERATION OPTIONAL,
    &operationCode              Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
    [ARGUMENT                   &ArgumentType  [OPTIONAL  &argumentTypeOptional] ]
    [RESULT                     &ResultType  [OPTIONAL  &resultTypeOptional] ]
    [ERRORS                     &Errors]
    [LINKED                     &Linked]
    [CODE                       &operationCode]
}
```

The new ROSE standards[9–11] contain many other useful definitions of importance to designers of remote-operations-based applications.

Conclusions

This paper is only an introduction to the elegance and user-friendliness of the new ASN.1 specification, which will replace the now-deprecated MACRO notation. It is expected that standards written using the MACRO notation will be converted, in the next few years, to the use of information-object classes, parameterization of ASN.1 definitions, and constraint specifications.

Certainly, new applications, such as protocols for OSI-generic upper-layer security, are being written in the updated notation. Thus, it is imperative that the use of this new notation is understood. Its application to the ROSE protocol is, perhaps, the best place to begin, because of the large number of familiar and popular OSI and ISDN applications that use the ROSE protocol. It is hoped that this paper will also provide an easy entrée to the new ROSE standards.

Acknowledgments

The author would like to thank Suzanne Usiskin and Herb Bertine of AT&T Bell Laboratories for a careful reading of this manuscript. The author also acknowledges many conversations on the new ASN.1 standards with Bancroft Scott, of Open Software Solutions, Inc., who also reviewed this paper. The author is also indebted to Doug Steedman, of GeneralMagic, Inc., for tutorials and discussions on ROSE and ASN.1.

References

1. CCITT Recommendation X.208 | ISO/IEC 8824: Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1), 1987.
2. CCITT Recommendation X.219 | ISO/IEC 9072-1: Information processing systems - Text communications - Remote Operations - Part 1: Model, notation and service definition, 1989.
3. CCITT Recommendation X.229 | ISO/IEC 9072-2: Information processing systems - Text communications - Remote Operations - Part 2: Protocol specification, 1989.
4. ITU-T Rec. X.680 | ISO/IEC 8824-1: Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation, 1994.
5. ITU-T Rec. X.681 | ISO/IEC 8824-2: Information technology - Abstract Syntax Notation One (ASN.1): Information object specification, 1994.
6. ITU-T Rec. X.682 | ISO/IEC 8824-3: Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification, 1994.
7. ITU-T Rec. X.683 | ISO/IEC 8824-4: Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications, 1994.
8. CCITT Recommendation X.407, Message Handling Systems: Abstract Service Definition Conventions, 1988.

9. ITU-T Rec. X.880 | ISO/IEC 13712-1: Information technology - Remote Operations: Concepts, model & notation, 1994.
10. ITU-T Rec. X.881 | ISO/IEC 13712-2: Information technology - Remote Operations: OSI realizations - Remote Operations Service Element (ROSE) service definition, 1994.
11. ITU-T Rec. X.882 | ISO/IEC 13712-3: Information technology - Remote Operations: OSI realizations - Remote Operations Service Element (ROSE) protocol specification, 1994.

(Manuscript approved April 1994)

Nilotpal Mitra *is a member of the technical staff in the Standards Planning Department at AT&T Bell Laboratories in Holmdel, New Jersey. His work involves developing architectures and standards on ISDN signaling for broadband and intelligent networks, as well as research on OSI-based data communications. He also represents AT&T in these areas in the ITU-T and ISO/IEC standards bodies. Mr. Mitra received a Ph.D. in theoretical physics from Columbia University in New York City. He joined AT&T in 1983.*