TR 101 114 V1.1.1 (1997-11)

Technical Report

Methods for Testing and Specification (MTS); Analysis of the use of ASN.1 94 with TTCN and SDL in ETSI deliverables

European Telecommunications Standards Institute

Reference
DTR/MTS-00047 (abc00ics.PDF)

Keywords
ASN.1, SDL, TTCN

ETSI Secretariat

Postal address
F-06921 Sophia Antipolis Cedex - FRANCE

Office address
650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16
Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

X.400
c= fr; a=atlas; p=etsi; s=secretariat

Internet
secretariat@etsi.fr
http://www.etsi.fr

Contents

Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETR 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available **free of charge** from the ETSI Secretariat. Latest updates are available on the ETSI Web server (http://www.etsi.fr/ipr).

Pursuant to the ETSI Interim IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETR 314 (or the updates on http://www.etsi.fr/ipr) which are, or may be, or may become, essential to the present document.

Foreword

This Technical Report (TR) has been produced by the ETSI Technical Committee Methods for Testing and Specification (MTS).

Introduction

As telecommunications specifications become more complex, ETSI deliverables are increasingly making use of languages such as Abstract Syntax Notation One (ASN.1), Specification and Description Language (SDL) and, for testing specifications, the Tree and Tabular Combined Notation (TTCN). A growing number of ETSI deliverables are already using the new features offered by the 1994 version of ASN.1, such as extensibility and information object classes.

ETR 60 [12] recommends the use of ASN.1 94 in preference to older versions of the language whenever new work involving ASN.1 is undertaken by ETSI. However, ETR 60 [12] does not take into account the use of ASN.1 94 *together* with SDL, neither does it take into account the use of ASN.1 94 *together* with TTCN.

At the time of writing, the integration of ASN.1 94 and SDL is incomplete and the integration of ASN.1 94 and TTCN is not well-defined. If ETSI is to successfully and efficiently use various combinations of these languages then it is essential that this integration is correct, consistent and in accordance with ETSI's needs.

The present document identifies the key technical issues for integrating ASN.1 94 and SDL and for integrating ASN.1 94 and TTCN. It is intended that the technical content of this document be used as a basis for further work by ETSI (or others) in defining the actual integration of the above mentioned languages.

1 Scope

The present document identifies the key technical issues and problems of completing the integration of ASN.1 (as defined in the ITU-T Recommendations X.680 [1], X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X.682 [5] and X.683 [6]) and SDL (as defined in ITU-T Recommendations Z.100 [7] and Z.100 Addendum 1 [8]).

The present document also identifies the key technical issues and problems of integrating ASN.1 (as defined in the ITU-T Recommendations X.680 [1], X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X.682 [5]) and X.683 [6] and the second version of TTCN (as defined in ISO/IEC 9646-3 [11]).

The present document focuses on types and does not investigate representation of values.

It is intended that the proposed solutions stated in the present document be used as a basis for further work by ETSI (or others). The present document complements rather than supplants ETR 60 [12] and ITU-T Recommendation Z.105 [9].

2 References

References may be made to:

a) specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or

b) all versions up to and including the identified version (identified by "up to and including" before the version identity); or

c) all versions subsequent to and including the identified version (identified by "onwards" following the version identity); or

d) publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

[1] ITU-T Recommendation X.680 (1994): "Abstract Syntax Notation One (ASN.1): Specification of basic notation".

[2] ITU-T Recommendation X.680 Amendment 1 (1994): "Abstract Syntax Notation One (ASN.1): Specification of basic notation: Rules of extensibility".

[3] ITU-T Recommendation X.681 (1994): "Abstract Syntax Notation One (ASN.1): Information object specification".

[4] ITU-T Recommendation X.681 Amendment 1 (1994): "Abstract Syntax Notation One (ASN.1): Rules of extensibility".

[5] ITU-T Recommendation X.682 (1994): "Abstract Syntax Notation One (ASN.1): constraint specification".

[6] ITU-T Recommendation X.683 (1994): "Abstract Syntax Notation One (ASN.1): parametrization of ASN.1 specifications".

[7] ITU-T Recommendation Z.100 (1993): "Specification and Description Language (SDL)".

[8] ITU-T Recommendation Z.100 Addendum 1 (1996): "Corrections to Recommendation Z.100, CCITT Specification and Description Language (SDL)".

[9] ITU-T Recommendation Z.105 (1994): "SDL combined with ASN.1".

[10] CCITT Recommendation X.208 (1990) : "Specification of the Abstract Syntax Notation One (ASN.1)".

[11] TR 101 101 (1997) "Methods for Testing and Specification (MTS); TTCN interim version including ASN.1 1994 support [ISO/IEC 9646-3] (Second Edition Mock-up for JTC1/SC21 Review)".

[12] ETR 60 (1995) "Guidelines for using Abstract Syntax Notation one (ASN.1) in telecommunication application protocols".

[13] ETS 300 771-1 (1997):"Broadband Integrated Services Digital Network (B-ISDN); Digital Subscriber Signalling System No. two (DSS2) protocol; B-ISDN user-network interface layer 3 specification for point-to-multipoint call/connection control; Part 1: Protocol specification; [ITU-T Recommendation Q.2971 (1995), modified]".

[14] ITU-T Recommendation X.691 (1995): "ASN.1Encoding Rules: specification of Packed Encoding Rules (PER)".

[15] ITU-T COM-10-1-E, Questions allocated to ITU-T study group 10 (Language for telecommunication application) for the study period 1997-2000.

3 Definitions and abbreviations

3.1 Definitions

For the purposes of the present document, the following definitions apply:

ASN.1 94: ASN.1 as defined in the 1994 ITU-T Recommendations X.680 [1], X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X.682 [5] and X.683 [6].

ASN.1 90: ASN.1 as defined in the 1990 ITU-T Recommendation X.208 [10].

3.2 Abbreviations

For the purposes of the present document, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
ASP	Abstract Service Primitive
ATS	Abstract Test Suite
BER	Basic Encoding Rules
BNF	Backus-Nauer Form
IN	Intelligent Network
INAP	Intelligent Networks Application Protocol
IUT	Implementation Under Test
PER	Packed Encoding Rules
PDU	Protocol Data Units
PIXIT	Protocol Implementation eXtra Information for Testing
SDL	Specification and Description Language
SUT	System Under Test
TTCN	Tree and Tabular Combined Notation

4 Use of ASN.1 94 in ETSI deliverables

An examination of approximately 1000 ETSI standards produced over the last five years shows that there is a small, but significant, use of ASN.1 (roughly 10%) to describe such things as Protocol Data Units, Remote Operations, Object Identifiers, data for test suites etc. When one considers just higher-layer, management and application protocols the proportionate use of ASN.1 is significantly higher.

When ASN.1 is used it is used extensively, for example, in the specification of ISDN supplementary services and of IN protocols, and could not easily be replaced by other techniques. In that sense, ASN.1 is a vital aspect in the specification of ETSI standards.

There is also a tendency to use ASN.1 in the specification of test suites, even though the base specifications may not use ASN.1. An example of this is the test suites for B-ISDN DSS2 UNI [13] where ASN.1 is used to specify explicitly that Information Elements may appear in any order in a message. This is difficult to express when ASN.1 is not used.

ETR 60 [12] recommends the use of ASN.1 94 in preference to older versions of the language. This recommendation is strongly supported by ETSI TC MTS. While many standards still use older versions of ASN.1 there is a growing trend towards the use of ASN.1 94 not only in ETSI standards but also in ITU-T recommendations. For example,

- Information Object Classes instead of macros in the definition of Remote Operations;
- use of extensibility to leave standards open to future development;
- parametrization of ASN.1 specifications for compactness and flexibility;
- specification of constraints.

One factor that will be significant to the use and acceptance of ASN.1 94 will be the availability of tool support for the new features.

The present document identifies the key technical issues and problems of integrating ASN.1 94 and SDL and of integrating ASN.1 94 and TTCN. Clause 5 discusses the issues of integrating ASN.1 94 with SDL. Clause 6 discusses the issues of integrating ASN.1 with TTCN.

5 Problems and issues associated with the use of ASN.1 94 and SDL

This clause considers the problems associated with the integration of ASN.1 94 with SDL. The ITU-T Recommendation Z.105 [9] defines how ASN.1 can be used in combination with SDL. However this recommendation only supports partial use of the ASN.1 94, i.e., ITU-T Recommendation X.680 [1] with some restrictions but not the extensions defined in the ITU-T Recommendations X681 [3], X.682 [5] and X.683 [6].

This clause is divided into two parts: the first part considers problems associated with the current ITU-T Recommendation Z.105 [9] definition and the limited ASN.1 94 it is designed to support. The second part considers extensions to the ITU-T Recommendation Z.105 [9] specification necessary to encompass the ASN.1 94 functionality at present not supported. For each problem or feature there is a section containing a description together with some possible solutions and consequences of these solutions. The end of this clause includes a tabular summary of the problems and extensions described.

5.1 Current ITU-T Recommendation Z.105 Issues

The intention of ITU-T Recommendation Z.105 [9] is that the structure and the behaviour of systems are described with SDL, while parameters of exchanged messages and internally used data are described with ASN.1.

Basically, ITU-T Recommendation Z.105 [9], which is an extension to Z.100 [7], defines:

- a new syntax merging the ITU-T Recommendations Z.100 [7] and X.680 [1] syntaxes;
- an equivalence between ASN.1 constructs and SDL constructs.

ITU-T Recommendation Z.105 [9] allows:

- the import of ASN.1 types and values from ASN.1 modules to SDL packages;
- the expression of types and values using the ASN.1 syntax embedded in SDL specifications.

To a large degree, the version of ASN.1 as defined in ITU-T Recommendation X.680 [1] is supported. The features of ITU-T Recommendations X.681 [3], X.682 [5], and X.683 [6] are not supported.

5.1.1 Restrictions on ITU-T Recommendation X.680 imposed by Z.105

Because of the integration of the concepts and of the syntaxes of ASN.1 and SDL, ITU-T Recommendation Z.105 [9] defines a set of restrictions on ITU-T Recommendation X.680 [1]. These restrictions limit the possibility to use existing ASN.1 material with SDL.

5.1.1.1 Description

5.1.1.1.1 Mapping restrictions

These restrictions apply both when ASN.1 definitions are imported into SDL from an external module or when the ASN.1 syntax inside SDL is used. This could be a potential problem when there is a need to import existing ASN.1 material into SDL packages, because there is a likelihood that these ASN.1 modules are not compliant with these restrictions.

The mapping restrictions are:

- case sensitivity is not supported. The motivation for this restriction is that SDL is case insensitive. This restriction implies that introducing two types with the same name (apart from case sensitivity) is an error. However, it is allowed to have the same name if they are of different entity classes. Entity classes are for example, type names, value names and identifiers;
- the use of the same identifier for named numbers or named bits of different types in the same scope shall be avoided. The motivation is that named numbers and named bits are mapped on SDL integer synonyms. Using the same identifier twice would result in illegal SDL (redefinition of the same synonym). Double use of the same identifier in different enumerated types, or in an enumerated type and in a named integer or named bit is allowed, because the identifiers in enumerated types are not mapped on integer synonyms;
- the OBJECT IDENTIFIER component values that are assigned by ITU-T, ISO, or both, are not defined in the package called *Predefined*.

5.1.1.1.2 Syntax differences when using ASN.1 syntax inside SDL

The syntax differences when using ASN.1 syntax embedded in SDL are:

- the dash in ASN.1 names is not supported inside SDL descriptions. Dashes are allowed in names within ASN.1 modules that are imported in SDL, but when they are imported in SDL, the dashes should be transformed to underscores. The motivation for this restriction is that the dash is considered as the minus operator in SDL;
- definitions must be ended with a semi-colon;
- ASN.1 type REAL is not represented as a sequence of integers, as is the case in ITU-T Recommendation X.680 [1]. The value notation of ITU-T Recommendation X.208 [10] shall be used instead, i.e., { 314, 10, -2 } should be used instead of { mantissa 314, base 10, exponent -2 }. Alternatively, the SDL syntax for denoting REAL values can be used, i.e., 3.14 is also allowed. As a consequence, no subtyping of mantissa, base, or exponent is allowed, and no operators for accessing or changing the mantissa, base, or exponent of a REAL value are supported;
- in external type and value references, spaces shall be put around the period (.).

5.1.1.2 Solution A: Make recommendations on the use of ITU-T Recommendation X.680

5.1.1.2.1 Solution description

Make recommendations on the use of ASN.1, so as to facilitate the reuse of ASN.1 modules in the scope of SDL specifications. They should apply both to the ASN.1 modules developed on their own and to the ASN.1 embedded in SDL.

Some examples of guidelines are:

- two entities of the same class shall not be identical once put in lower case;
- the use of the same identifier for named numbers or named bits of different types in the same scope should be avoided.

5.1.1.2.2 Solution consequences

Conformance to these rules ensures the possibility to import ASN.1 entities from ASN.1 modules to SDL specifications provided only ITU-T Recommendation X.680 [1] is used. Whereas, SDL specifications importing ASN.1 modules *not* conforming to the mapping restriction could be semantically incorrect from the ITU-T Recommendation Z.105 [9] point of view. For example, consider two types defined in an ASN.1 module with names identical apart from the cases. Import of these two types into SDL will have the effect to define the same type twice.

NOTE: The rules defined above do not prevent name clashes when importing two ASN.1 modules into the same SDL specification. This is a general problem which could be solved by prefixing entity names with module reference.

5.1.2 Non-support of ITU-T Recommendation X.680 Amendment 1, ITU-T Recommendations X.681, X.682 and X.683

5.1.2.1 Description

Consider the case where ASN.1 94 is already used in some ETSI standards and there is a need to import these ASN.1 definitions into SDL specifications.

5.1.2.1.1 Solution A: Limit the use of ASN.1 to ITU-T Recommendation X.680

5.1.2.1.1.1 Solution description

Restrict the use of ASN.194 to ITU-T Recommendation X.680 [1] only.

5.1.2.1.1.2 Solution consequences

This solution will guarantee that the ASN.1 modules can be imported into SDL. But this will limit strongly the use of ASN.1 94 and will be in contradiction with ETR 060 [12].

5.1.2.1.2 Solution B: Make recommendation to ease the reuse of ASN.1 94

5.1.2.1.2.1 Solution description

Transform the ITU-T Recommendation X.680-series features to equivalent constructs in ITU-T Recommendation X.680 [1].

NOTE: This transformation would have to be done manually. Of course, the support of these transformations by a tool would be of interest.

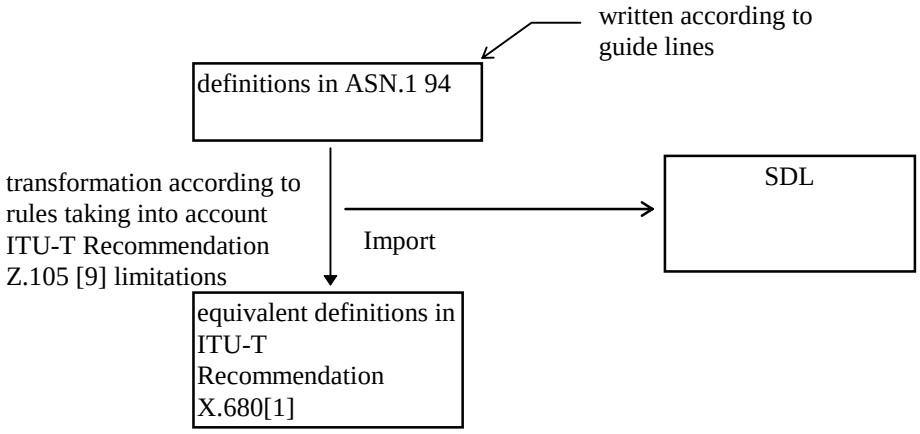

Figure 1: Transformation to ITU-T Recommendation X.680 [1]

The transformations would consist mainly of:

- removing the extension marker (=> no extensibility in the SDL model);
- resolve parametrization (arduous but not difficult);
- resolve constraints. This is probably the most difficult point. Ways to imitate the table constraint mechanism would have to be defined. (probably with CHOICEs) (=> instanciations of information objects);
- replace remaining uses of information objects, object classes and object sets, by appropriate types and values, then remove the declarations of these objects, classes and sets;
- redefine the appropriate types for entities which have not been mapped in ITU-T Recommendation Z.105 [9].

5.1.2.1.2.2 Solution consequences

The use of ASN.1 94 specifications in SDL as defined in ITU-T Recommendation Z.105 [9] will be facilitated (to a degree). Nevertheless manual transformation of ASN.1 definitions will remain a tedious work.

5.1.3 Defects in ITU-T Recommendation Z105

Defects in ITU-T Recommendation Z.105 [9] could constitute a limitation to the use of the present document. A significant number of defects, most of them are editorial, have been identified and are described in the ITU-T document COM-10-1 [15]. ITU-T will maintain a Master list of changes containing corrections or clarification corresponding to these defects. These changes will be incorporated in the next version of ITU-T Recommendation Z.105 [9].

In addition the following problems have been identified:

- Type constraints are supported in a limited way:
  - exceptions are not supported (ITU-T Recommendation X. 680 [1] BNF productions Constraint ::= "(" ConstraintSpec ExceptionSpec")" and ExceptionSpec ::= "!" ExceptionIdentification | empty);
  - intersections in subtype constraints are not supported (but unions are). (ITU-T Recommendation X. 680 BNF productions SubtypeConstraint ::= ElementSetSpec and the following);
  - exclusions in subtype constraints are not supported;
- some predefined types are not supported, e.g., TeletexString.

5.2 Extensions to ITU-T Recommendation Z.105 to support ASN.1 94 more completely

ITU-T Recommendations X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X.682 [5] and X.683 [6] are not addressed by ITU-T Recommendation Z.105 [9]. The support of these extensions is desirable. Moreover, the use of these features is encouraged by ETR 60 [12]. ASN.1 modules using these concepts, especially information objects and parametrization, will be of particular interest.

The table 1 of ITU-T Recommendation Z.105 [9] summarizing the data characteristics of SDL, ASN.1 and their combination could be complemented as follows (the first line has been added):

Table 1: Characteristics of SDL and ASN.1

	SDL	ASN.1	SDL/ASN.1
Definition of object classes, information objects and object sets		X	X
Definition of types	X	X	X
Notation for values	X	X	X
Definition of operators	X		X
Expressions	X		X
Encoding of values		X	X

NOTE: The last line of the table is not strictly correct as ASN.1 94 (i.e., the ITU-T Recommendation X.680-series) does not allow the definition of encoding. However, the ITU-T Recommendation X.690-series does define encoding rules (e.g., ITU-T Recommendation X.691 [14] that can be applied on ASN.1 types and values. ITU-T Recommendation Z.105 [9] models cannot contain specification of encoding, except as informal text.

5.2.1 ITU-T plans for the evolution of ITU-T Recommendation Z.105

At the time of writing, ITU-T plans to issue new versions of ITU-T Recommendations Z.100 [7] and Z.105 [9] by the year 2000 to keep SDL in line with developing technologies.

ITU-T Recommendation Z.105 [9] is concerned by the following ITU-T decisions:

- The underlying data model of this so-called SDL-2000 language will no longer be ACT ONE but a more powerful and more implementable data model, still to be defined. Consequently the mapping rules of ASN.1 definitions to SDL data concepts will have to be changed to reflect the new data model.
- ITU-T Recommendation Z.105 [9] will continue to be a document separate from Z.100 [7].
- ITU-T Recommendation Z.100 [7] will be updated so that Z.105 [9] does not need to redefine ITU-T Recommendation Z.100 [7].

5.2.2 Syntax considerations

ITU-T Recommendation Z.105 [9] has the philosophy to merge completely the ASN.1 and SDL syntaxes, though the complete mixing of the notations is discouraged to the user. Keeping this philosophy to incorporate the ASN.1 94 extensions will result in a complex grammar. Therefore alternatives solutions allowing the import of ASN.1 material from ASN.1 modules but not their definition inside SDL specifications have been investigated.

Concerning the syntax, three levels of integration to allow the use of ASN.1 94 with SDL can be envisaged. Each level defines a way to complement the ITU-T Recommendation Z.105 [9] grammar. The following example illustrates these three possible solutions:

```
ModuleA DEFINITIONS ::=
BEGIN
```

```
-- definition of an object class
PARAMETERS-BOUND ::= CLASS
{
    &minLength      INTEGER,
    &maxLength      INTEGER
}
WITH SYNTAX
{
    MINIMUM-FOR-LENGTH      &minLength
    MAXIMUM-FOR-LENGTH      &maxLength
}

-- definition of a type parameterized by an object class
ParamType {PARAMETERS-BOUND : bound} ::=
    OCTET STRING (SIZE(bound.&minLength..bound.&maxLength))

-- information object, instance of the class PARAMETERS-BOUND
networkSpecificBoundSet PARAMETERS-BOUND ::=
{
    MINIMUM-FOR-LENGTH      3
    MAXIMUM-FOR-LENGTH      5
}

-- definition of a parameterized type
List1 { ElementTypeParam } ::= SEQUENCE {
    elem    ElementTypeParam,
    next    List1 { ElementTypeParam } OPTIONAL
}

-- definition of a parameterized value
genericString { IA5String : name } IA5String ::= { "Name : ", name}

END
```

5.2.2.1 Solution A: Full integration of ASN.1 94 and SDL

5.2.2.1.1 Solution description

This solution consists in complementing the ITU-T Recommendation Z.105 [9] grammar so that the ASN.194 and SDL syntaxes are merged. This solution complies with the syntactic approach taken for the current ITU-T Recommendation Z.105 [9]. Both the import of all classes of ASN.1 entities, e.g., information object set, parametrized type, and their definition inside SDL are permitted.

NOTE: Though it is permitted, ITU-T Recommendation Z.105 [9] discourages the definition of ASN.1 types inside SDL packages.

The ASN.1 module defined above can be imported to SDL as described in solution B, but equivalently the same definitions could be inserted directly in a SDL package as follows:

system example

```
...
/* definition of an object class */
PARAMETERS_BOUND ::= CLASS
{
    &minLength      INTEGER,
    &maxLength      INTEGER,
}
WITH SYNTAX
{
    MINIMUM_FOR_LENGTH      &maxLength
    MAXIMUM_FOR_LENGTH      &maxLength
} ;

/* definition of a type parameterized by an object class */
ParamType {PARAMETERS_BOUND : bound} ::=
    OCTET STRING (SIZE(bound.&minLength..bound.&maxLength)) ;

/* information object, instance of PARAMETERS_BOUND */
networkSpecificBoundSet  PARAMETERS-BOUND ::=
{
    MINIMUM_FOR_LENGTH      3
    MAXIMUM_FOR_LENGTH      5
} ;

/* definition of a parameterized type */
```

```
newtype List1 { ElementTypeParam }
    SEQUENCE {
    elem    ElementTypeParam,
    next    List1 { ElementTypeParam } OPTIONAL
} ;

/* definition of a parameterized value */
genericString { IA5String : name } IA5String ::= { "Name : ", name} ;

...

endsystem example;
```

5.2.2.1.2 Solution consequences

This solution conforms with the philosophy adopted for the current ITU-T Recommendation Z.105 [9]. This results in a homogeneous definition. On the other hand the ITU-T Recommendation Z.105 [9] grammar must be enriched with numerous Backus-Nauer Form (BNF) productions which would result in a more complicated standard.

5.2.2.2 Solution B: Allow import of ASN.1 94 module from SDL, and the use of imported entities in SDL

5.2.2.2.1 Solution description

This solution allows:

- The import of all class of entities (types, values, information objects, object class, object sets) defined in an ASN.1 94 module.
- The use of these imported entities in SDL.

This solution does not permit the definition of ITU-T Recommendations X.681 [3], X.682 [5] and X.683 [6] entities (information objects, object class, object sets, parametrized types, constraints) inside SDL. The import of all class of ASN.1 entity (types, values, classes, object, object sets) from ASN.1 modules to SDL definitions and the use of this material is authorized.

For instance, an SDL model making use of *moduleA* can be:

```
use moduleA ;
...
process myproc;
dcl
    /* use of a type parameterized by an information object */
    mygvns ParamType { networkSpecificBoundSet },
    /* use of a type parameterized by a type */
    myintlist   List1 {INTEGER},
    mymax   integer,
    mystring    IA5String;

start;
/* use of an information object value */
task mymax := networkSpecificBoundSet.&maxLength;
/* use of a parameterized value */
task mystring := genericString { "John" };
endprocess myproc;
```

The affected ITU-T Recommendation Z.105 [9] BNF productions could be:

```
<sort constructor>  ::= ...existing Z.105 non-terminals ...
            <object class field type>
            <from object>
<extended primary> ::= ...existing Z.105 non-terminals ...
            <object class field value>
            <from object>
<object class field value> ::= [ <sort> : ] <expression>
<from object> ::= [ <package name> . ] { <object name> | <object set name> } [ { <actual parameter>
{ , <actual parameter> } * } ] . <field name>
<actual parameter> ::= <sort> | <extended primary>
<object class field type> ::= <defined object class> . <field name>
<defined object class> ::= {[ <package name> . ] <object class name>} | TYPE-IDENTIFIER | ABSTRACT-
SYNTAX
```

5.2.2.2.2 Solution consequences

In this solution the ITU-T Recommendation Z.105 [9] grammar is extended in a limited way. All the possibilities of mixing ASN.1 and SDL, e.g., defining information objects in a SDL package, are not possible but this is discouraged by ITU-T Recommendation Z.105 [9] anyway.

5.2.2.3 Solution C: Allow import of ASN.1 94 module from SDL, keep the ITU-T Recommendation Z.105 syntax unchanged.

5.2.2.3.1 Solution description

This solution consists in only allowing the import of fully defined types and values.

To be able to make use of generic types and values defined by *ModuleA*, an intermediate module must be defined as follows:

```
ModuleB DEFINITIONS ::=
BEGIN
    IMPORTS
        ParamType, PARAMETERS-BOUND,
        networkSpecificBoundSet, List1, genericString
    FROM ModuleA;
NetworkSpecificParamType ::= ParamType { networkSpecificBoundSet },

IntegerList1 ::= List1 { INTEGER }

networkSpecificMaxLength:= networkSpecificBoundSet.&maxLength;
johnBirthdayGreeting := genericString { "John" };
END

The SDL specification becomes:

use moduleB;
...
dcl
    mygvns NetworkSpecificParamType ,
    myintlist   IntegerList1,
    mymax   integer,
    mystring    IA5String;

start;
task mymax := networkSpecificMaxLength;
task mystring := johnBirthdayGreeting;
endprocess myproc;
```

5.2.2.3.2 Solution consequences

The main advantage of this solution is that few changes to the ITU-T Recommendation Z.105 [9] are required, none at the syntax level. On the other hand the user has to define intermediate ASN.1 modules to be able to make use of parametrized entities.

5.2.3 Information objects

ASN.1 94 allow the definition of ASN.1 modules intended to work with any of a number of instances of some class of information objects. ITU-T Recommendation X.681 [3] specifies a notation for defining information object classes and instanciations of these classes, and a notation for extracting information from objects.

5.2.3.1 Solution description

The proposed solution would complement ITU-T Recommendation Z.105 [9] with the support of information objects. Nevertheless from the semantic point of view, no equivalence would be defined between information objects and existing SDL concepts (the equivalence defined by ITU-T Recommendation Z.105 [9] remains at the level of types and values). An exception for this statement could concern table constraints. This issue is discussed in subclause 5.2.3.

The following example illustrates the possible use of information objects in SDL:

Consider the following information object class and information object:

```
MY-OBJECT-CLASS ::= CLASS
{
    &valueField     INTEGER,
    &TypeField
}
WITH SYNTAX {
    VALUE-FIELD     &valueField,
    TYPE-FIELD      &typeField
}
myInfoObject MY-OBJECT-CLASS ::= {
    VALUE-FIELD     96
    TYPE-FIELD      REAL
}
```

Once imported, the following use of *myInfoObject* in SDL will be possible:

```
/* use of ASN.1 types and values */
dcl a Integer := myInfoObject.&valueField;
dcl b myInfoObject.&typeField;
```

The following use will not be correct because no equivalence between ASN.1 information object classes and SDL sorts is defined:

```
/* Incorrect: declaration of a variable for an object class */
dcl c MY_OBJECT_CLASS := myInfoObject;
```

As generally presented in subclause 5.2.1, three solutions for the support of information objects can be envisaged:

- solution A of 5.2.1: inclusion of the notation for defining objects, class and sets, and of the notation for extracting information, into the SDL grammar;
- solution B of 5.2.1: inclusion of the notation for extracting information;
- solution C of 5.2.1: no notation for information objects included in the SDL grammar.

5.2.3.3 Solution consequences

The solution enables the import of modules using information objects which is a key feature of ASN.1 94.

5.2.4 Open types

5.2.4.1 Description

ASN.1 94 define features making types and values generic: parametrization, type openness, specification of constraints. It is essential that the types are fully instanciated when used in SDL.

In replacement of the ANY notation, ASN.1 94 allows the definition of open types. The open type notation consists in the *ObjectClassFieldType* production specified in ITU-T Recommendation X.681 [3], where the *FieldName* denotes either a type field or a variable-type value field.

Consider the following definition of the type *ErrorReturn*

```
ERROR-CLASS ::= CLASS
{
&code   INTEGER,
&Type
}

ErrorReturn ::= SEQUENCE
{
errorCode   ERROR-CLASS.&code,
errorInfo   ERROR-CLASS.&Type
}
```

5.2.4.2 Solution A: No support of open types

5.2.4.2.1 Solution description

As the exact type of *errorInfo* is not known an equivalent SDL definition cannot be determined. The proposed solution consists in not allowing the use of open types.

5.2.4.2.2 Solution consequences

This solution could be felt as a strong limitation. In particular, the use of the *InstanceOfType* notation which is a shorthand notation for defining an open type constrained by an information object set of a class derived from the TYPE-IDENTIFIER predefined class, and is recommended by ETR 60 [12] when there is a need for embedding information from another protocol, will not be possible.

5.2.4.3 Solution B: Support of open types constrained with tables (information object sets)

5.2.4.3.1 Solution description

ITU-T Recommendation X.682 [5] defines the constraint table notation which enables to specify that the type of an item depends on the value of another item with the help of information object sets.

```
ERROR-CLASS ::= CLASS
{
    &code       INTEGER,
    &Type
}
WITH SYNTAX {&code &Type}

ErrorSet ERROR-CLASS ::=
{
    {1 INTEGER} |
    {2 REAL}
}
ErrorReturn ::= SEQUENCE
{
    errorCode   ERROR-CLASS.&code   ({ErrorSet}),
    errorInfo   ERROR-CLASS.&Type   ({ErrorSet}{@errorCode})
}
```

The possible types that *errorInfo* could take are specified (INTEGER or REAL). Consequently an equivalent SDL construct could be defined. The equivalence could be close to the one defined for the CHOICE construct.

The following type definition is not strictly equivalent to the one given above but gives a first idea of the kind of transformation that can be applied on table constraints to implement them in SDL.

```
ErrorReturn ::= SEQUENCE
{
errorCode   INTEGER,
errorInfo   CHOICE { code1 INTEGER, code2 REAL }
}
```

5.2.4.3.2 Solution consequences

Types constrained by object sets will be allowed, thus extending the coverage of ASN.1 94. The use of the *InstanceOfType* notation, which is a shorthand notation for defining an open type constrained by an information object set of a class derived from the TYPE-IDENTIFIER predefined class, will be possible as well. A model, which could be complex, for the translation of these ASN.1 constructs to SDL should be determined.

5.2.5 Extensibility

ITU-T Recommendations X.680 Amendment 1 [2] and X.681 Amendment 1 [4] define rules of extensibility, enabling the definition of extensions for types and object sets respectively.

5.2.5.1 Solution A: Allow use of extension marker in ITU-T Recommendation Z.105 type definitions

The extensibility mechanism is intended to allow the intercommunication of implementations of different versions of a specification, one version extending the definition of a type of another version. As shown in the following example, ITU-T Recommendation Z.105 [9] could benefit from the introduction of extensibility.

Consider the type *Mytype* defined in a protocol specification as follows:

```
/* MyType can be extended */
newtype MyType
    SEQUENCE {
    a   INTEGER,
    ...
} ;
```

A new version of the protocol specification could define *MyType* as follows:

```
newtype MyType
    SEQUENCE {
    a   INTEGER,
    ...,
    b   BOOLEAN
} ;
```

An implementation of the old version and an implementation of the new version would be able to exchange message parameters of type *MyType*. Nevertheless access to field *b* by the new version for messages coming from the old version could result in execution errors. The specification should be written in such a way this cannot occur, for example, by testing first the presence of field *b* in the received message.

Support of extensibility by ITU-T Recommendation Z.105 [9], will affect the grammar (addition of the ellipsis notation), and the modelling of ASN.1 ENUMERATED, SEQUENCE, SET and CHOICE, into SDL equivalent constructs.

A *Present* operator could be added for every field following the extension mark (...). In the above example operator *Bpresent* can be used to test the presence of field *b*.

From the syntactic point of view the impact on ITU-T Recommendation Z.105 [9] will be:

```
<module definition>     ::= <module>  definitions  [<tagdefault>] [<extensiondefault>] ::=
                            begin  [<modulebody>]  end
<extensiondefault>  ::= extensibility implied
```

NOTE: In accordance to the current ITU-T Recommendation Z.105 [9], the possibility to express exceptions has been omitted.

```
<enumerated>        ::= enumerated { {<name number>}+  [, ...]  { , <name number> }* }
<sequence>          ::= { sequence | set }  { { [ { <elementsort> }+ [ , ... ] { ,  <elementsort> }*
                        ] | ... } }
<choice>            ::= choice   {  [ { <namedsort> }+ [ ,  ... ]  { , <namedsort> }* ] }
<range condition>   :=  { <range> }+ [ { ,  |  | }  ... ]  { { ,  |  | }  <range> }*
```

5.2.6 Encoding

TTCN allows the specification of the encoding rules for PDUs (either ASN.1 or tabular ones). The reference to the encoding standard (BER, PER etc.) is in free text. Contrary to TTCN, SDL does not support the definition of the encoding to be used. This problem is not specific to ITU-T Recommendation Z.105 [9] but is more an ITU-T Recommendation Z.100 [7] general problem. Nevertheless it is emphasized when using ASN.1 data for which encoding is usually associated.

The possibility to express encoding in ITU-T Recommendation Z.105 [9] could be a valuable extension. The advantages are:

- in some cases the encoding is part of the protocol specification. SDL, enhanced with encoding, will thus cover more extensively protocol specifications;

- code generators based on SDL could be extended to treat the encoding as shown in figure 2.

This report identifies the need to specify:

- the default encoding for a whole package/system/block;
- specific encoding for a type.

In a way similar to TTCN the encoding could be split into:

- encoding rules = reference to a standard;
- encoding variations = reference to a section of the standard.

ITU-T Recommendation Z.105 [9] explicitly states that tags are ignored. If the encoding is to be incorporated to ITU-T Recommendation Z.105 [9] this would be no more true because encoding needs the tag information. Either the mapping takes into account the tags, or the mapping is considered as incomplete and a code generator cannot rely uniquely on the SDL equivalent form of an ASN.1 type but has to consider the source ASN.1 form.

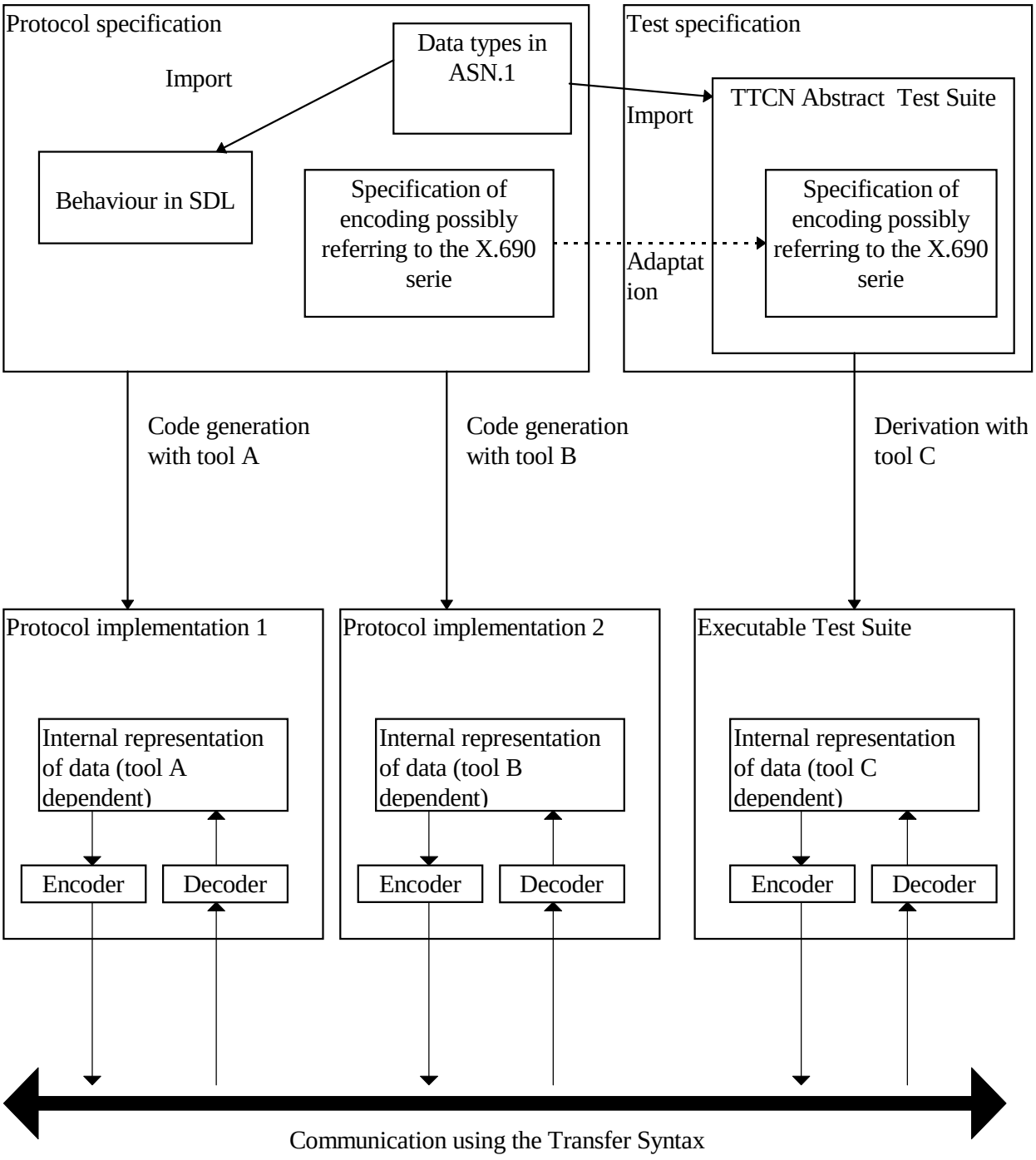

Figure 2: Use of encoding information by code generators

6 The Problems and issues associated with the use of ASN.1 94 and TTCN

This clause considers the problems associated with the integration of ASN.1 94 with TTCN as defined in ISO/IEC 9646-3 [11]. The clause is split into two main parts, "Problems with TTCN Edition 2" and "TTCN Extensions Necessary to Support New ASN.1 94 Features".

"Problems with TTCN Edition 2" considers problems associated with the current TTCN specification and the limited ASN.1 94 it supports. As it stands, the standard is ambiguous as to exactly what is supported and how it interworks with ASN.1.

The "Problems with TTCN Edition 2" clause is structured as a list of problems together with associated proposed solutions. In general the solutions proposed fall into one of two categories shown graphically in figure 3.

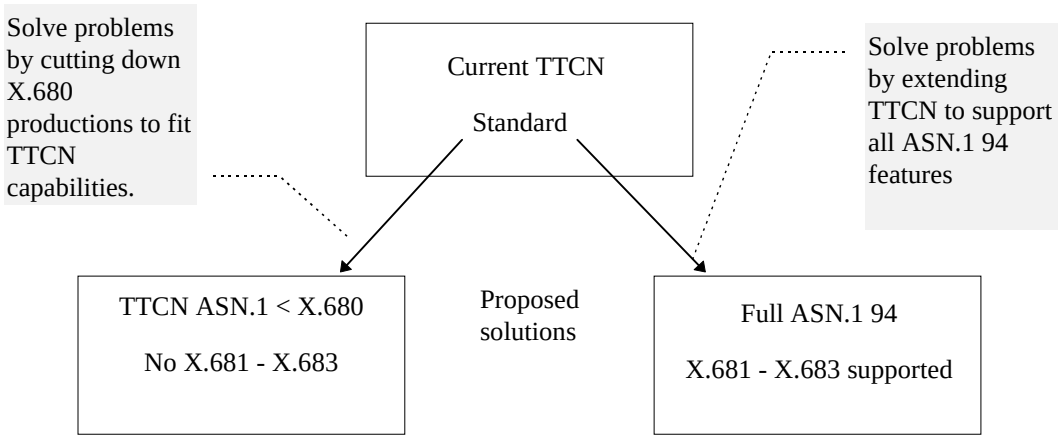

Figure 3: Possible solution paths to rectify TTCN problems

The first of these categories involves cutting out any features of ASN.1 94 that do not fit the current TTCN standard. Implementing such solutions leads to a TTCN standard that only supports a subset of ITU-T Recommendation X.680 [1] and provides no support for ITU-T Recommendation X.681 [3], X.682 [5] and X.683 [6].

The second category of solutions proposes extensions to the TTCN standard to support the new ASN.1 94 features. If these solutions are implemented the TTCN standard could be fully ASN.1 94 compliant.

Solutions in the first category are, in general, less radical and easier to implement. However this solution path should only be seen as a temporary measure because eventually TTCN must support the defined ASN.1 standard which is ASN.1 94.

The second part of the clause goes on to consider extensions to the TTCN specification to encompass the ASN.1 94 functionality at present not supported. For each new feature there is a clause containing a description together with some possible solutions and consequences of these solutions.

6.1 Problems with TTCN Edition 2

This clause categorizes and describes the problems associated with the current TTCN specification and the use of ASN.1 94. Many of the proposed solutions require the TTCN specification to support new features from ASN.1 94.

6.1.1 Realization of TTCN BNF

6.1.1.1 Description

The current TTCN BNF is not consistent with respect to the ASN.1 94 definitions because they include many features which have no support within the TTCN environment. The TTCN BNF references Value and Type definitions from ITU-T Recommendation X.680 [1]. Within the definitions of these terms X.680 [1] uses productions from X.681[3] (Information Object Specification) and X.683 [6] (Parametrization of ASN.1 Specifications). The current TTCN environment provides no support for the former and only partial support for the latter (extending TTCN to support these features is considered in subclause 6.2).

To illustrate the problem, in rule 122 in ISO/IEC 9646-3 [11] specification states :

```
122  ASN1_Type ::= Type
     /* REFERENCE - Where Type is the non-terminal defined in ISO/IEC 8824-1: 1994 ...*/
```

In the ISO/IEC 8824-1: 1994 (ITU-T Recommendation X.680 [2]) specification it is stated:

```
Type ::= BuiltinType | ReferencedType | ConstrainedType

BuiltinType ::=
    BitStringType |
    BooleanType |
    CharacterStringType |
    ChoiceType |
    EmbeddedPDVType |
    EnumeratedType |
    ExternalType |
    InstanceOfType |
    IntegerType |
    NullType |
    ObjectClassFieldType |
    ObjectIdentifierType |
    OctetStringType |
    RealType |
    SequenceType |
    SequenceOfType |
    SetType |
    SetOfType |
    TaggedType

ReferencedType ::=
    DefinedType |
    UsefulType |
    SelectionType |
    TypeFromObject |
    ValueSetFromObjects

DefinedType ::=
    Externaltypereference |
    typereference |
    ParameterizedType |
    ParameterizedValueSetType
```

The underlined identifiers have no meaning within the TTCN environment because information objects and type parametrization are not supported in the current version of TTCN.

6.1.1.2 Solution A: Redefine ASN.1 productions

6.1.1.2.1 Solution description

Redefine a set of ASN.1 productions removing all references to the new features introduced by ASN.1 94. The required production redefinition's are shown in appendix A.

6.1.1.2.2 Solution consequences

The consequences of this solution are:

- this solution has the disadvantage of increasing the divergence between the standardized ASN.1 language and the ASN.1 dialect supported by TTCN. It requires non-trivial redefinition of ASN.1 productions within TTCN;
- any existing ASN.1 94 module that makes use of any of the new features must be rewritten before it can be used within TTCN;
- this solution could be used as a short term pragmatic solution to provide a consistent TTCN specification.

6.1.1.3 Solution B: Extend TTCN to support ASN.1 94 features

6.1.1.3.1 Solution description

Extend TTCN to support the new features introduced in ASN.1 94.

This extension of the TTCN specification to support parametrization and information objects is considered in detail in the second part of this clause.

6.1.1.3.2 Solution consequences

In the medium term this solution has the clear advantage that we are actually using the defined ASN.1 specification within TTCN not a redefined dialect. The consequences however are far ranging and will require careful consideration. The consequences are explained in detail in subclauses 6.2.4 and 6.2.5.

6.1.2 Semantic problem with DefinedValue Extension

6.1.2.1 Description

This clause describes the problems associated with the redefinition in TTCN rule 739 of the ASN.1 identifier *DefinedValue* from:

```
DefinedValue ::= Externalvaluereference | valuereference | ParameterizedValue
```

To:

```
DefinedValue ::= ConstraintValue&Attributes | valuereference | ParameterizedValue
```

DefinedValue is included in the following ASN.1 definitions. In each case it is possible to produce syntactically valid constructs which have no sense or undefined behaviour. In some cases the static semantics defined in ITU-T Recommendation X.680 [1] limit the type that can be produced from *DefinedValue* but since this specification is referring to a different specification of *DefinedValue* the meaning in the TTCN context is not definitive.

6.1.2.1.1 NumberForm

If we consider the production NumberForm within the ASN.1 standard:

```
NumberForm ::= number | DefinedValue
```

From the redefinition of *DefinedValue*, *NumberForm* can be validly defined as an TTCN constraint e.g.,

```
NumberForm ::= "?"
```

The *NumberForm* production is used in the definition of *ObjectIdentifierValue* in the following way:

```
ObjectIdentifierValue ::= "{" ObjIdComponentList "}" |
                          "{" DefinedValue ObjIdComponentList "}"

ObjIdComponentList ::= ObjIdComponent |
                       ObjIDComponent ObjIdComponentList

ObjIdComponent ::= NameForm     |
                   NumberForm |
                   NameAndNumberForm
```

Using these productions it is possible to produce the following:

```
ObjectIdentifierValue ::= "{" "6" "5" "4" "?" "}"
```

If we further consider the following productions for *SymbolsFromModule* which use *ObjectIdentifierValue*:

```
SymbolsFromModule ::= SymbolList FROM GlobalModuleReference

GlobalModuleReference ::= modulereference AssignedIdentifier

AssignedIdentifier ::= ObjectIdentifierValue |
                       DefinedValue |
                       empty
```

Using these productions it is possible to produce the following syntactically valid construct

```
SymbolsFromModule ::= SymbolList FROM "{" "6" "5" "4" "?" "}"
```

In this case the *ObjectIdentifierValue* must consist of a sequence of positive numeric values which uniquely and unambiguously identify an object in the object identifier tree. The concept of adding constraints to such a value, as shown above, has no sense. Which ASN.1 module should these symbols be imported from ?

6.1.2.1.2 CharsDefn

DefinedValue is also used in the definition of *CharsDefn*:

```
CharsDefn ::= cstring | DefinedValue
```

From the definition of *CharacterStringList*:

```
CharacterStringList ::= "{" CharSyms "}"

CharSyms ::= CharsDefn | CharSyms "," CharsDefn
```

The following production is theoretically possible:

```
CharacterStringList ::= "{" "abc", "?" , "def" "}"
```

Such a constraint within a list of characters appears inappropriate. It could be argued that clause 34.8 in ITU-T Recommendation X.680 [1] already provides static semantics to cover this case. It states " The *DefinedValue* in *CharsDefn* shall be a reference to a value of that type." Since this clause is referring to a different definition of *DefinedValue*, i.e. the true ASN.1 definition, its applicability is ambiguous here.

6.1.2.1.3 ExceptionIdentification

The definition of *ExceptionIdentification* within the ASN.1 standard also uses *DefinedValue*:

```
ExceptionIdentification ::= SignedNumber |
                            DefinedValue |
                            Type ":" Value
```

Using the productions:

```
Constraint ::=  "(" ConstraintSpec ExceptionSpec ")"

ExceptionSpec ::= "!" ExceptionIdentification | empty
```

The following valid construct can be produced:

```
Constraint ::=  "(" ConstraintSpec "!" "?" ")"
```

Within ASN.1 the *ExceptionSpec* is used to define either a value or a type and value associated with a constraint violation (the constraint is defined before the *ExceptionSpec*). The ability to define constraints within the *ExceptionSpec* itself makes no sense.

6.1.2.1.4 NamedNumber

In the ASN.1 definition of *NamedNumber* the *DefinedValue* production is also used:

```
NamedNumber ::= identifier "(" SignedNumber ")" |
                identifier "(" DefinedValue ")"
```

Considering the definition of *IntegerType*:

```
IntegerType ::= INTEGER |
                INTEGER "{" NamedNumberList "}"

NamedNumberList ::= NamedNumber |
                    NamedNumberList "," NamedNumber
```

We can define constructs such as:

```
IntegerType ::= INTEGER "{" one "(" "1" ")" "," any "(" "?" ")" "}"
```

In this example names ought to be provided for specific values within the defined type. The ability to use a constraint in this list makes no sense.

The definition *NamedNumber* is also used in the *EnumeratedType* production:

```
EnumeratedType ::= ENUMERATED "{" Enumeration "}"

Enumeration ::= EnumerationItem |
                EnumerationItem "," Enumeration

EnumerationItem ::= identifier | NamedNumber
```

Using these productions it is therefore possible to construct syntactically valid constructs such as:

```
EnumeratedType ::= ENUMERATED "{" one "(" "1" ")" "," any "(" "?".")" "}"
```

In this example the list of possible enumerated values for an enumerated type ought to be defined, again the constraint has no place here.

6.1.2.1.5 NamedBit

The definition of *NamedBit* also references the *DefinedValue* production:

```
NamedBit ::= identifier "(" number ")" |
             identifier "(" DefinedValue ")"
```

The *NamedBit* production is used in turn in the definition of *BitStringType*:

```
BitStringType ::= BIT STRING |
                  BIT STRING "{" NamedBitList "}"

NamedBitList ::= NamedBit | NamedBitList "," NamedBit
```

Using these production the following syntactically valid ASN.1 construct is possible:

```
BitStringType ::= BIT STRING "{" powerOn "(" "1" ")" "," any "(" "?" ")" "}"
```

The specification of a constraint within this type definition makes no sense.

6.1.2.1.6 ClassNumber

The ASN.1 definition of *ClassNumber* uses the production for *DefinedValue*:

```
ClassNumber ::= number | DefinedValue
```

The *ClassNumber* production is in turn used in the definition of Tag:

```
Tag ::= "[" Class ClassNumber "]"

Class ::= UNIVERSAL     |
          APPLICATION |
          PRIVATE          |
          empty
```

Using these production we can obtain:

```
Tag ::= "[" UNIVERSAL "?" "]"
```

This again clearly makes no sense.

6.1.2.2 Solution A: Static semantic checks

6.1.2.2.1 Solution description

Develop a rigorous set of static semantic statements which explicitly exclude the possibility of any of the above described problems.

6.1.2.2.2 Solution consequences

The consequences of this solution are:

- this solution will require effort in the definition of the required static semantics. In effect this solution is trying to patch-up an existing suspect solution (the redefinition of *DefinedValue*) as such it is depreciated as a long term solution;
- this solution might be acceptable as a short term pragmatic approach.

6.1.2.3 Solution B: Separation of value and constraints within TTCN BNF

6.1.2.3.1 Solution description

Rework TTCN productions to remove the problems described above. This could be done by separating out the Value and constraints productions.

The semantic problems with the TTCN *DefinedValue* extension are caused by allowing the use of constraints at inappropriate places within the ASN.1 productions. In the current standard there is the single *ConstraintValue&Attributes* entry point from ASN.1 into TTCN. This entry point provides access to all TTCN values (test case variables, PIXIT values etc.) and constraints (any, any or omit, etc.). If we could separate these two production paths (values and constraints) it might be possible to selectively pass only the appropriate productions into the ASN.1, i.e. redefine the ASN.1 definition to include at various places TTCN values, TTCN constraints or both as semantically valid.

It should be noted that any development of the TTCN language, due to the inherent grammar type, is a difficult task.

6.1.2.3.2 Solution consequences

This solution would require a major reworking of the TTCN specification. The magnitude of the effort required and the quality of the final solution are as yet unclear.

6.1.2.4 Solution C: Re-implement functionality using ASN.1 features

6.1.2.4.1 Solution description

Re-implement the existing functionality using ASN.1 parametrization and ASN.1 constraint extensions. This solution would use ASN.1 parametrization (see subclause 6.2.4) to pass values (such as test suite variables or constants) from TTCN to ASN.1. The TTCN constraint types, such as "omit" or "any" would need to be standardized into the ASN.1 language.

This solution solves the stated problems because it separates the value and constraints paths. TTCN can only pass values to ASN.1 (possibly types also). If the types are consistent for the TTCN and ASN.1 environments these values should always be semantically valid. By defining the constraints with in the ASN.1 the existing BNF restricts the use of these constraints to valid locations.

6.1.2.4.2 Solution consequences

The consequences of this solution are:

- the redefinition of *DefinedValue* would no longer be necessary;
- in principle this solution requires very little extra effort for TTCN apart from support for ASN.1 94 value *parametrization*;
- the main problem is the need to add TTCN constraint types into the ASN.1 standard;
- this is the preferred long term solution as it provides a clean modular syntax between TTCN and ASN.1 while retaining the desirable functionality of the current system without the associated errors.

6.1.3 Syntactic problem with *DefinedValue* extension

6.1.3.1 Description

The TTCN redefinition of *DefinedValue* has the consequence that in certain circumstances the syntax of an ASN.1 clause can be ambiguous. An example of this behaviour can be demonstrated using the *BuiltinValue* production:

```
BuiltinValue ::=
        ... |
        SequenceValue |
        SequenceofValue |
        ... |
```

Where:

```
SequenceValue ::= { NamedValue, NamedValue, ...}

SequenceofValue ::= { Value, Value, ...}

NamedValue ::= identifier Value
```

With the TTCN extension to *DefinedValue* it is possible to reach an expression through an ASN.1 value, i.e.

```
Value :: BuiltinValue | ReferencedValue

ReferencedValue ::= DefinedValue | ValueFromObject

DefinedValue ::= ConstraintValue&Attributes | typereference | ParameterizedType |
                 ParameterizedValueSetType

ConstraintValue&Attributes ::= ConstraintValue ValueAttributes

ConstraintValue ::= ConstraintExpression | MatchingSymbol | ConsRef

ConstraintExpression ::= Expression
```

This relationship leads to the following conflicting situation:

```
BuiltinValue ::= { identifier1 - 73, ...}
```

There are two ways of interpreting this example. If "-" is a unary operator then - 73 is a value, therefore the identifier1 - 73 must be a *NamedValue*. In this case the example resolves to a *SequenceValue*.

On the other hand if we consider "-" as a binary operator then the term identifier1 - 73 is an expression, hence it is equivalent to a Value. In this case the example resolves to a *SequenceofValue*.

6.1.3.2 Solution A: Define TTCN lexical rule

6.1.3.2.1 Solution description

Considering this problem in isolation one solution would be to formalize "normal" mathematical practice into the syntax, i.e.

```
- 73              /* The space between the operator and value indicates this is an expression */

-73              /* There is no space between the operator and value, therefore this a negative
                    value */
```

This rule could perhaps be usefully introduced into the entire syntax. This opportunity could be used to clearly define lexical rules for TTCN

NOTE: If this solution is implemented the problem of visually distinguishing between - 73 and -73 within the graphical form becomes important. Perhaps some visual clue to differentiate these two cases would be useful. This visual clue could be specified in the TTCN standard at the discretion of the tool developers.

6.1.3.2.2 Solution consequences

The consequences of this solution are:

- this change appears straight forward;
- the only foreseen negative consequences is backwards compatibility with existing test-suites.

6.1.4 ASN.1 94 entry point

6.1.4.1 Description

Within the TTCN specification rule 739 references a ASN.1 94 production from ITU-T Recommendation X.680 [1] for *DefinedValue*. The production is referenced as:

```
DefinedValue ::= Externalvaluereference | valuereference
```

However the actual definition is:

```
DefinedValue ::= Externalvaluereference | valuereference | ParameterizedValue
```

6.1.4.2 Solution A: Correct ASN.1 94 production reference

6.1.4.2.1 Solution description

Incorporate the correct definition from ITU-T Recommendation X.680 [1] into the TTCN specification.

6.1.4.2.2 Solution consequences

The consequences of this solution are:

- the consequence of using this definition is that parametrized values are explicitly defined in the ASN.1 productions instead of being coerced into ASN.1 by the redefinition of *DefinedValue*;
- this could be used as the first step in solving the problems associated with the *DefinedValue* redefinition described in clause 6.1.2.

6.1.5 ASN.1 reserved words

6.1.5.1 Description

The ASN.1 reserved words defined in the TTCN specification table A.3 specify the reserved words from ASN.1 90 not ASN.1 94.

6.1.5.2 Solution A: Include subset of ASN.1 94 keywords

6.1.5.2.1 Solution description

Change the TTCN specification to include a subset of the ASN.1 94 keywords. The chosen subset includes all keywords associated with the core ITU-T Recommendation X.680 [1] specification but omits the keywords associated with information objects. The explicit changes to the table are:

Remove:

```
ANY
DEFINED
```

Add:

```
ALL
BMPString
CHARACTER
CONSTRAINED
EXCEPT
INTERSECTION
ISO646String
ObjectDescriptor
UNION
UNIQUE
UniversalString
```

6.1.5.2.2 Solution consequences

The consequences of this solution are:

- the new types introduced as keywords should be supported within TTCN (see clause 6.2.1);
- this solution is still not fully compatible with ASN.1 94 because it does not include all the keywords.

6.1.5.3 Solution B: Include all ASN.1 94 keywords

6.1.5.3.1 Solution description

Add the complete ASN.1 94 keywords list to the TTCN specification. In addition to the changes specified in solution A the following additional keywords should be added. It should be noted that these additional keywords relate to features at present not supported by TTCN.

```
ABSTRACT-SYNTAX
AUTOMATIC
CLASS
EMBEDDED
TYPE-IDENTIFIER
INSTANCE
PDV
SYNTAX
```

6.1.5.3.2 Solution consequences

The consequences of this solution are:

- the list of reserved keywords will be fully compatible with the ASN.1 94 specifications;
- if it is decided not to support all the ASN.1 94 features within TTCN, this solution may define keywords which have no associated meaning within TTCN.

6.2 TTCN extensions to support new ASN.1 94 features

This clause considers extensions to the TTCN standard to support new features introduced in ASN.1 94.

6.2.1 New ASN.1 94 types

6.2.1.1 Description

For ASN.1 and TTCN to function together there must be a clear definition of the type mapping between the two environments. With ASN.1 90 this lead to TTCN using the same type definitions for the base types.

ASN.1 94 introduces a number of new types into the language description. Some of these types are straightforward to introduce into TTCN but others are associated with the new features which currently are not supported by TTCN.

6.2.1.2 Solution A: Support subset of ASN.1 94 types

6.2.1.2.1 Solution description

Extend the TTCN specification to support a subset of the new types introduced by ASN.1 94 . The subset includes all the types which are straight-forward to add to TTCN. These types are :

```
BMPString
UniversalString
```

6.2.1.2.2 Solution consequences

The consequences of this solution are:

- TTCN would not be fully type compatible with ASN.1 94. The incompatible types would be those associated with information objects;
- although requiring considerable effort (especially in the case of universal string) this solution avoids any difficult implementation problems.

6.2.1.3 Solution B: Support all ASN.1 94 types

6.2.1.3.1 Solution description

Extend the TTCN specification to support all of the new types introduced by ASN.1 94. The list of new types would be:

```
BMPString
UniversalString
EMBEDDED PDV
INSTANCE OF
```

6.2.1.3.2 Solution consequences

The consequences of this solution are:

- the TTCN specification would be fully compatible with the ASN.1 94 specification in terms of defined types;
- the last two types are defined in the ASN.1 94 specification in terms of information objects. If it is decided not to support information objects within TTCN then these types could not be fully implemented.

6.2.2 AUTOMATIC tagging

6.2.2.1 Description

ASN.1 94 introduces the feature of AUTOMATIC tagging. This provides a new tagging mode in addition to the existing IMPLICIT and EXPLICIT. When AUTOMATIC tagging is selected the system will automatically insert any necessary tags within the associated module without the need for user intervention (N.B. the user still has the choice to override the AUTOMATIC mechanism for specific constructs by explicitly defining tags).

The ITU-T Recommendation X.680 [1] clause "Guidelines for the use of the ASN.1 notation" recommends always to use AUTOMATIC tagging in the development of new modules. This feature is therefore likely to be used in many, if not all, new ASN.1 modules.

AUTOMATIC tags is selected from the ASN.1 module header. Since TTCN only allows ASN.1 type definitions not module definitions there is no current mechanism for selecting the tagging regime within TTCN (it is by default EXPLICIT). Since the potential effort to convert the type definitions in an ASN.1 module using AUTOMATIC tagging into type definitions using explicit tagging is large and error prone, a mechanism to introduce tagging regimes into TTCN should be found.

6.2.2.2 Solution A: No support for AUTOMATIC tagging

6.2.2.2.1 Solution description

Provide no support for AUTOMATIC tagging.

6.2.2.2.2 Solution consequences

The consequences of this solution are:

- the TTCN ASN.1 type declarations are not fully compatible with type declarations within ASN.1 94 modules;
- ASN.1 type declarations taken from modules using AUTOMATIC tagging will have to be carefully rewritten for the TTCN environment precisely re-implementing the translations to the syntax that the AUTOMATIC tagging produces. For a complex hierarchical type this is a non-trivial task.

6.2.2.3 Solution B: Support for AUTOMATIC tagging

6.2.2.3.1 Solution description

Introduce an option into the TTCN ASN.1 type proforma to enable the user to specify the tagging regime to be used in the table. In terms of the BNF this would be analogous to the ASN1_Encoding option added to the value production, i.e.:

```
ASN1_Type ::= Type [ASN1_Tagging]

ASN1_Tagging ::=   EXPLICIT TAGS |
                   IMPLICIT TAGS |
                   AUTOMATIC TAGS |
                   empty
```

If no tag type is defined, EXPLICIT tagging is assumed.

6.2.2.3.2 Solution consequences

- in terms of tagging this solution will bring TTCN into line with the ASN.1 94 specification. Type declarations can be directly transposed or referenced from ASN.1 94 modules;
- all ASN.1 "type" proformas will need to be extended (new entry in header) to allow specification of the tagging type.

6.2.3 Extensibility

6.2.3.1 Description

In principle extensibility provides a mechanism for future compatibility by defining a syntax which will accept elements not defined in that syntax. ASN.1 94 allows extensibility to be specified within a syntax definition. The extensibility can either be specified explicitly using the extension marker "..." or globally across an ASN.1 module by addition of an optional field in the module header.

The extension marker can be placed in the definition of ENUMERATED TYPE, SEQUENCE, SET and CHOICE. The effect of the extension marker is to disable error generation when the received element does not match the specified syntax of the associated type.

At first sight the idea of extensibility seems to be mutually exclusive in regards to conformance testing. However since it is often the case that the test purpose only requires the checking of certain specific parameters, this mechanism could actually be used as a useful addition to provide future compatibility to test suites especially for postambles and preambles. This feature has the potential to increase flexibility and life span of TTCN test suites.

It is recommended that the optional header field to specify automatic extensibility is not incorporated into TTCN. In a conformance testing environment explicit specification within the type is preferable to implicit definition, i.e. something explicitly written in the type specification is easier to see and understand than an optional header field.

The extensibility feature also requires consideration of the associated transfer syntax. For some encoding rules, most notably PER as defined in ITU-T Recommendation X.691 [14], the extension marker is visible in the transmitted bytes. In such a case if the language used to define the data types for conformance testing cannot support extension markers the transfer syntax of the associated Implementation Under Test (IUT) might be impossible to reproduce within the testing system.

6.2.3.2 Solution A: No support for extensibility

6.2.3.2.1 Solution description

Provide no support for extensibility.

6.2.3.2.2 Solution consequences

The consequences of this solution are:

- the future compatibility of TTCN specified test suites and hence test suite life, might be reduced;
- the ASN.1 defined within TTCN will not be fully compatible with the ASN.1 94 specifications;
- this solution removes any risk associated with using extensibility during conformance testing.
- Test suites for an IUT using encoding rules where the extensibility marker is visibly encoded (e.g., PER) would be extremely difficult or impossible to write using standard TTCN.

6.2.3.3 Solution B: Support for extensibility

6.2.3.3.1 Solution description

Support the extension marker and explicit extensibility within the ASN.1 syntax for TTCN.

6.2.3.3.2 Solution consequences

This solution provides the potential benefits in terms of test suite life and direct use of existing ASN.1 94 module specifications. It also removes limitations on the testable transfer syntax.

6.2.4 Parametrization

6.2.4.1 Description

The current TTCN specification supports only a part of the possible ASN.1 94 parametrization features. This subclause describes the current TTCN support and then considers possible extensions to allow the use of the remaining ASN.1 94 parametrization features.

At present TTCN provides value parametrization for constraint declarations. This means values from constants, test suite variables or PIXITs can be passed into a constraint.

ASN.1 94 supports the idea of value parametrization for the value notation in a similar way to TTCN. In addition however ASN.1 94 allows value parameters to be used in type notation for definition of constraints.

ASN.1 94 also includes the concept of generic type parametrization. For example, consider the following definition:

```
MESSAGE { PDU_Type } ::= SEQUENCE
{
    ASP ASP_Type,
    PDU PDU_Type
}
```

This defines the parametrized type MESSAGE{}. Within the body of the protocol this parametrized type can be used to define further types. For example:

```
SetupMessage ::= MESSAGE { Setup_PDU }
```

Which has the meaning:

```
SetupMessage ::= SEQUENCE
{
    ASP ASP_Type,
    PDU Setup_PDU
}
```

At present TTCN supports a limited form of type parametrization in the form of PDU parameter in ASP type definitions.

The addition of generic type parametrization into TTCN would provide a powerful new feature. For example, it could provide the ability to define elements of a protocol from the Protocol Implementation eXtra Information for Testing (PIXIT) list. This would allow far greater flexibility to be introduced into ATS's allowing straight forward customization for a specific System Under Test (SUT). On the other hand type parametrization has a large impact on TTCN compilation.

6.2.4.2 Solution A: Value parametrization from TTCN

6.2.4.2.1 Solution description

Support value parametrization of constraints only, using the existing TTCN syntax. In this solution the TTCN BNF provides the definition for parameter list and referencing the parameter list.

6.2.4.2.2 Solution consequences

The consequences of this solution are:

- this solution does not support type parametrization;
- this solution is not syntactically compatible with the ASN.1 94 parametrization specification, i.e.,

Using TTCN BNF the parameter list takes the form:

```
( name1:TYPE1; name2:TYPE2)
```

Using the ASN.1 94 specification the same parameter list takes the form:

```
{ TYPE1:name1, TYPE2:name2}
```

This solution therefore requires any parametrization contained within an existing ASN.1 94 modules to be rewritten entirely before these definitions can be used within TTCN;

- this solution is depreciated because it leaves two incompatible mechanisms for providing parametrization.

6.2.4.3 Solution B: Value parametrization from ASN.1 94

6.2.4.3.1 Solution description

Support value parametrization of constraints by altering the TTCN specification to make use of the ASN.1 value parametrization. In this solution the ASN.1 94 specification would define the syntax of parameter lists and in the case of ASN.1 types the referencing of that parameter list.

More effort is required to better define the changes necessary for this solution. In effect this solution moves the value parametrization from TTCN to ASN.1 for ASN.1 types.

6.2.4.3.2 Solution consequences

The consequences of this solution are:

- this solution is ASN.1 94 compliant. It provides a single unified mechanism for passing parameters into ASN.1 value notation;
- the syntax of parameter passing into ASN.1 constraints and non-ASN.1 constraints would potentially be different. This problem would hopefully be rectified during the investigation of the necessary TTCN BNF changes;
- this solution has the potential to provide a unified value parameter syntax for TTCN and ASN.1;
- this solution provides no type parametrization.

6.2.4.4 Solution C: Value and type parametrization from ASN.1 94

6.2.4.4.1 Solution description

Full support for ASN.1 94 parametrization. This involves providing value and type parametrization. The main steps in addition to those described in solution B are:

- change parameter list definitions to ASN.1 94 syntax;
- addition of type parameters to ASN.1 constraint definitions;
- addition of type parameters to ASN.1 types definitions;
- support the definition of types within the PIXIT.

6.2.4.4.2 Solution consequences

The consequences of this solution are:

- the possibility to directly use any parametrized type definitions from pre-defined ASN.1 modules;
- the main consequence of this solution is the TTCN system can no longer resolve all types at compile time of the Abstract Test Suite (ATS). That is to say that if a type parameter is defined from the PIXIT list the definition of this type might not be available at ATS compile time. This introduces the idea of a meta or open type;

- the meta (or open) type is used to represent a type which is unresolved. In many ways this concept is analogous to the ANY type available in ASN.1 90 (but removed in ASN.1 94).

6.2.5 Information objects

6.2.5.1 Description

Information objects are the macro replacement mechanism defined in ASN.1 94. In principle information objects are a form of generic table which allow the association of specific sets of field values or types. The greatest single advantage of Information objects is they are machine processable.

TTCN has never supported macro definitions within ASN.1. However the semantics of the macro notation were not directly machine processable. Looking at the existing use of information object within standards it appears there use is more wide-spread than macros were. Often information objects are used as a fundamental structuring mechanism.

In ASN.1 94 some of the defined types within the language are defined in terms of information objects (these types are TYPE-IDENTIFIER, ABSTRACT-SYNTAX and INSTANCE-OF). Any system than cannot support information objects may have difficulties to provide these built-in types.

If TTCN does not support information objects then any existing ASN.1 modules using this feature must be transformed before it can be used within TTCN. Such a transformation and involves converting an information object set into a type containing a CHOICE with all the possible field types. The validity of this transformation is dependant on the transfer syntax. If the required encoding rules make the CHOICE visible in the transfer syntax (e.g., PER) this transformation is invalid (changes the bits transmitted on the line). It follows that the testing of an IUT using a transfer syntax where the CHOICE is visible will be extremely difficult or impossible using a test specification language which does not support information objects.

6.2.5.2 Solution A: No support for information objects

6.2.5.2.1 Solution description

Provide no support for information objects.

6.2.5.2.2 Solution consequences

The consequences of this solution are:

- any pre-defined ASN.1 modules making use of information objects must have these information objects expanded-out before they can be used within TTCN. This process requires extra effort and is potentially error prone;
- the system defined object classes would not be available;
- support for certain transfer syntax's would be extremely difficult or impossible.

6.2.5.3 Solution B: Support for information objects

6.2.5.3.1 Solution description

Support the definition of information objects within TTCN.

This could be achieved either by using the existing ASN.1 proformas or perhaps cleaner introducing a new proforma to allow specification of information object classes.

6.2.5.3.2 Solution consequences

This solution allows direct use of pre-defined ASN.1 94 modules containing information objects, therefore saving time and effort.

Annex A (informative): Redefinition of ITU-T Recommendation X.680 rules for use with TTCN

This appendix specifies the ASN.1 94 productions rules from ITU-T Recommendation X.680 [1] which must be redefined to allow consistent productions within TTCN as defined in ISO/IEC 9646-3 [11].

TTCN 9.5 rule 122

```
ASN1_Type::=Type

BuiltinType::=
    BitStringType |
    BooleanType |
    CharacterStringType |
    ChoiceType |
    EmbeddedPDVType |
    EnumeratedType |
    ExternalType |
    ~~InstanceOfType |~~
    IntegerType |
    NullType |
    ~~ObjectClassFieldType |~~
    ObjectIdentifierType |
    OctetStringType |
    RealType |
    SequenceType |
    SequenceOfType |
    SetType |
    SetOfType |
    TaggedType

ReferencedType ::=
    DefinedType |
    UsefulType |
    SelectionType |
    ~~TypeFromObject |~~
    ~~ValuesetFromObjects~~

DefinedType ::=
    Externaltypereference |
    typereference |
    ~~ParameterizedType |~~
    ~~ParameterizedValueSetType~~

Elements ::=
    SubtypeElements |
    ~~ObjectSetElements |~~
    "(" ElementSetSpec ")"
```

TTCN 9.5 rule 739

```
 ASN1_Value ::= Value

BuiltinValue ::=
    BitStringValue |
    BooleanValue |
    CharacterStringValue |
    ChoiceValue |
    EmbeddedPDVValue |
    EnumeratedValue |
    ExternalValue |
    ~~InstanceOfValue~~ |
    IntegerValue |
    NullValue |
    ~~ObjectClassFieldValie |~~
    ObjectIdentifierValue |
    OctetStringValue |
    RealValue |
    SequenceValue |
    SequenceOfValue |
    SetValue |
    SetOfValue |
    TaggedValue

ReferencedValue ::=
```

```
DefinedValue |
ValueFromObject
```

Annex B (informative): ASN.1 94 syntax issues

This annex considers issues associated with the defined syntax of ASN.1

B.1 Real type

The syntactic checking of ASN.1 94 appears in some ways to have been weakened from the earlier versions of this language. One example of this is the real type.

In ASN.1 90 the real value was explicitly defined to consist of three values Mantissa, Base and Exponent (when non-zero):

```
NumericRealValue ::= { Mantissa, Base, Exponent } | 0
Mantissa ::=         SignedNumber
Base ::=             2 | 10
Exponent ::=         SignedNumber
```

Within the ASN.1 94 specification real value is defined as a sequence of named values of arbitrary length:

```
NumericRealValue ::= 0- | SequenceValue
SequenceValue ::= "{" ComponentValueList "}" | "{" "}"
ComponentValueList ::= NamedValue | ComponentValueList , NamedValue
```

This latter definition appears to provide considerably less syntactic checking allowing a value sequence of any length and also any value type.

B.2 ASN.1 names

The use of hyphens as separators for multi-word ASN.1 names requires name translation when ASN.1 names are used within the TTCN or SDL environments. The use of the hyphen separator within ASN.1 instead of the more usual underscore appears purely arbitrary.

It would clearly be desirable to have a common multi-word separator for all these tools and avoid this unnecessary name translation. Since there are good reasons to avoid the use of the hyphen in the TTCN and SDL environments the best solution would be to change the multi-word name separator in ASN.1 from hyphen to underscore.

It should be in the interest of all those involved in ASN.1 standardization to incorporate changes into the language to facilitate better interworking with other standard languages (TTCN, SDL). Better integration between these formal tools to allow seamless use of ASN.1 in all phases from specification to conformance testing can only further its appeal and use.

B.3 Language issues

The redefinition of *DefinedValue* in the ASN.1 specification for TTCN creates a unique and incompatible form of ASN.1 within TTCN. This ASN.1 dialect is formed by the coercion of TTCN production rules into the ASN.1 language specification.

This situation requires tool-users to learn and distinguish various versions of the ASN.1 language and also makes it very difficult for the tool-manufacturers to develop a single ASN.1 module for use in all tools. Clearly it would be better if there was only a single Standardized specification of the ASN.1 language and this definition was used unchanged by all relevant tools. For TTCN to support this goal the redefinition of *DefinedValue* would have to be superseded by a new syntactically cleaner mechanism.

Annex C (informative): Bibliography

The following material, though not specifically referenced in the body of the present document, gives supporting information.

ETS 300 414 (1995): "Methods for Testing and Specification (MTS); Use of SDL in European Telecommunications Standards (Rules for testability and facilitating validation)".

History

Document history		
V1.1.1	November 1997	Publication

ISBN 2-7437-1813-7
Dépôt légal : Novembre 1997