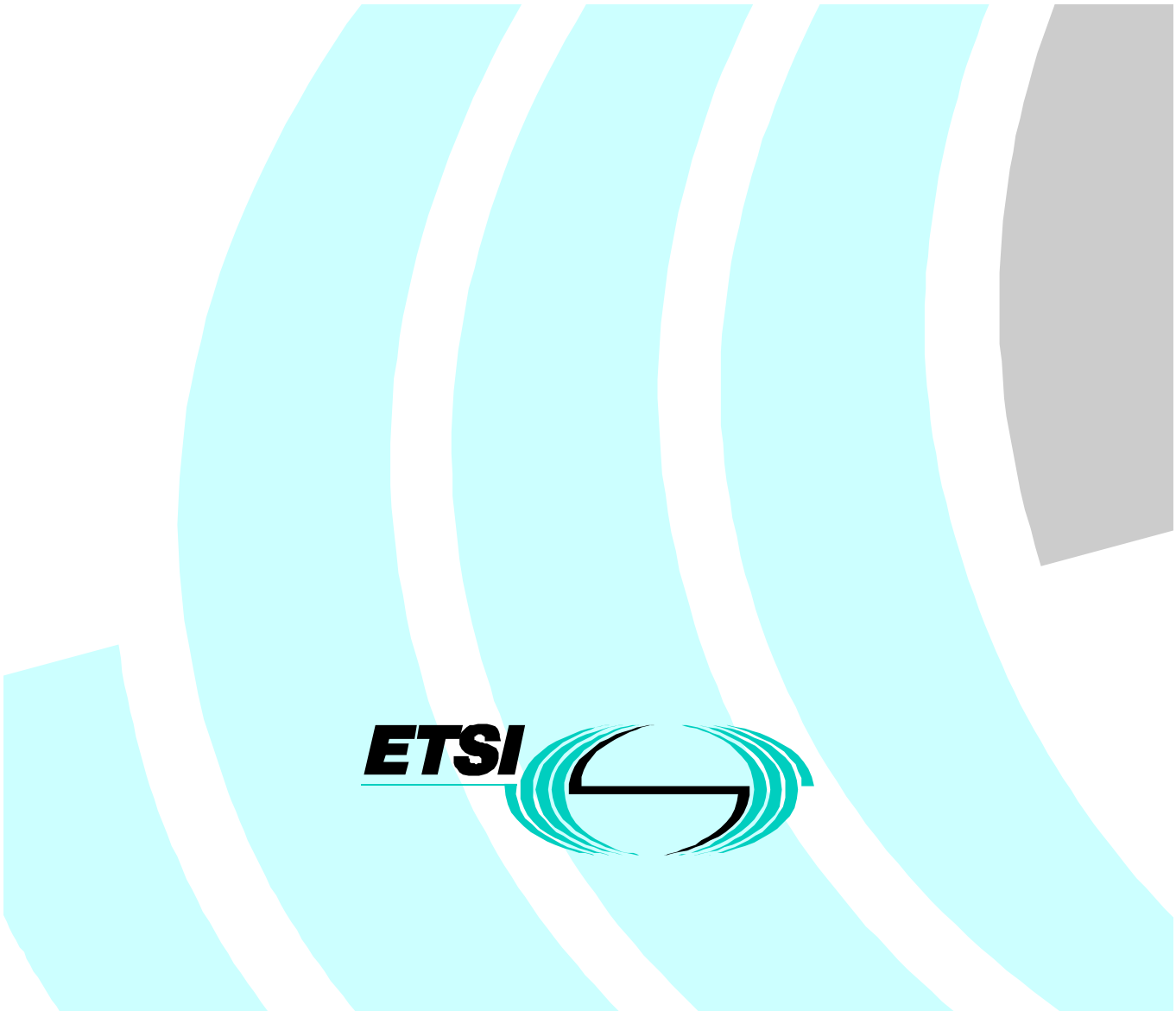
TR 101 295 V1.1.1 (1998-09)

Technical Report

Methods for Testing and Specification (MTS); Rules for the transformation of ASN.1 definitions using ITU-T Recommendations X.681, X.682 and X.683 to equivalent ITU-T Recommendation X.680 constructs



Reference
DTR/MTS-00048 (bzo00ics.PDF)

Keywords
ASN.1, SDL, TTCN

ETSI

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

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

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 SR 000 314: *"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 or http://www.etsi.org/ipr).

Pursuant to the ETSI 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 SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Foreword

This ETSI Technical Report (TR) has been prepared by 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 ASN.1, SDL and, for testing specifications, TTCN. A growing number of ETSI deliverables and those from other standardization bodies are already using the new features offered by ASN.1 94, such as extensibility, information object classes and parameterization.

The current versions of the TTCN and SDL only support a part of the ASN.1 94 language, i.e. X.680 [1]. The remaining part of ASN.1 94 (X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X.682 [5] and X.683 [6]) including the new features such as extensibility, information objects and parameterization, are not supported by SDL and TTCN. Consequently, the use of ASN.1 94 definitions in a SDL or TTCN context requires some transformations of these ASN.1 definitions to remove the use of unsupported features and when possible to replace them by functionally equivalent constructs. This Technical Report identifies and describes this set of transformations.

It should be noted that the transformation rules presented in the technical report are only intended to provide a short term "pragmatic" solution until the associated languages and tools can be extended to support the full features of ASN.1 94.

The present document specifies a set of generic transformation rules which are applicable to all domains where the full ASN.1 94 language must be converted to "pure" ASN.1 X.680 [1]. In addition there are two context specific appendices which provide further transformation rules for SDL Z.105 [8] and TTCN Edition 2 as defined in TR 101 101 [10].

To transform an existing ASN.1 94 module for use within SDL, first apply the generic transformation rules defined in clause 4 and then apply the SDL specific transformations defined in annex A.

The transformations described in clause 4 of the present document are particularly relevant when there is a need to simulate down to layer 1 or generate code, which is achieved with the help of SDL tools supporting Z.105 [8]. In situations where the SDL model is used in isolation, i.e., the transfer syntax is implicit to the SDL tool, some of the limitations for the transformation rules defined in clause 4 are removed. This relaxation of transformation rule limitations is explicitly stated in the text

For transformation of existing ASN.1 94 modules for use with TTCN Edition 2, first apply the generic transformation rules defined in clause 4 and then apply the set of TTCN specific transformation rules defined in annex B.

Also included in the present document is an appendix which defines the necessary feature extensions required to convert an existing ASN.1 90 tool into an ASN.1 X.680 tool capable of using the ASN.1 94 material which has undergone the transformation rules defined in the present document.

1 Scope

The present document describes a set of transformation rules to convert an ASN.1 94 module which uses of the features defined in X.680 [1], X.680 Amendment 1 [2], X.681 [3], X.681 Amendment 1 [4], X682 [5] and X.683 [6], into an equivalent or partially equivalent module which uses only the ASN.1 language features defined in X.680 [1].

In addition to the generic transformation rules defined in main body of this report there are two language specific appendices which provide additional transformation rules for SDL Z.105 [8] and TTCN Edition 2 as defined in ISO/IEC 9646-3 [10] respectively.

The present document complements rather than supplants ETR 60 [13] and Z.105 [8].

2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non-specific.
- For a specific reference, subsequent revisions do not apply.
- For a non-specific reference, 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): "Information technology-Abstract Syntax Notation One (ASN.1): Specification of basic notation".

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

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

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

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

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

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

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

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

[10] 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)".

[11] ISO/IEC 10646-1 (1993): "Information technology, - Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane".

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

[13] ETR 60 (1995) "Signalling Protocols and Switching (SPS);Guidelines for using Abstract Syntax Notation one (ASN.1) in telecommunication application protocols".

3 Definitions and abbreviations

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 Recommendations X.208 [9]

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

3.2 Abbreviations

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

ASN.1	Abstract Syntax Notation 1
BER	Basic Encoding Rules
PER	Packed Encoding Rules
SDL	Specification and Description Language
TTCN	Tree and Tabular Combined Notation

4 Transformation of ASN.1 94 material to ASN.1 X.680

4.1 Introduction

This clause defines a set of generic transformation rules to convert ASN.1 94 modules to "pure" X.680 [1] constructs. Each sub-clause addresses a particular ASN.1 94 feature which is not supported in X.680 [1]. For each feature the transformation rules together with any limitations effecting the validity of the conversions are described. Each transformation rule is illustrated with one or more examples. The examples use underlining to identify the elements that must be changed.

To apply these transformation rules every instance of the specified features must be identified within the ASN.1 94 module and the corresponding transformation rules, associated with that feature, applied.

4.2 Extensibility

4.2.1 Description

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. If extensibility is activated by use of the optional module header field all definition involving the relevant types will be extensible within that module.

No transformation rules can preserve the full semantics of extensibility. The specified transformation rules can only generate a syntax that will provide backwards compatibility for the extension series up to and including the ASN.1 94 version being converted. All forwards compatibility is lost apart from that explicitly added in rule 3.

4.2.2 Transformation rules

Rule 1: If present, remove extensibility field from ASN.1 module header.

Example: 1

```
Example-Module DEFINITIONS
AUTOMATIC TAGS EXTENSIBILITY IMPLIED
::=
BEGIN
    ModuleBody
END
```

Transforms to:

```
Example-Module DEFINITIONS
AUTOMATIC TAGS
::=
BEGIN
    ModuleBody
END
```

Rule 2: Remove all extension markers. Any types defined after the extension marker in SET or SEQUENCE definitions should be made OPTIONAL

Example:2

```
MyType ::= SEQUENCE {
    a   INTEGER,
    ...,
    b   BOOLEAN
}
```

transforms to:

```
MyType ::= SEQUENCE {
    a   INTEGER,
    b   BOOLEAN OPTIONAL
}
```

Rule 3: Add known future components as optional (in SEQUENCE, SET and CHOICE).

Add known future enumeration items (in ENUMERATED)

Example 3:

If it is already known that the next version of the protocol will add an OCTET STRING field to *MyType*, defined in Example 2, update it as follows:

```
MyType ::= SEQUENCE {
    a   INTEGER,
    b   BOOLEAN         OPTIONAL,
    c   OCTETSTRING  OPTIONAL
}
```

4.2.3 Transformation limitations

The transformation rules do not retain any forward compatibility provided by the extension marker in the original ASN.1 94 module.

These transformations also require consideration of any associated transfer syntax. For some encoding rules, most notably PER, the extension marker is visible in the transmitted bytes. In such a case these transformation rules cannot be validly applied. The resultant ASN.1 X.680 [1] definitions would be incompatible with the original ASN.1 94 module. It should be noted that this limitation is only applicable when firstly we are dealing with transmitted bytes (conformance testing, code generation) and secondly when we are using an encoding rule where the extension marker is visible in the transfer syntax.

NOTE: It follows that for any tool which cannot support extension markers the transfer syntax of certain ASN.1 94 systems might be impossible to reproduce.

The transformation rules provide no solution for extensibility markers used within information object set definitions.

4.3 Parameterization

4.3.1 Description

ASN.1 94 supports value parameterization for value notation and value parameterization in type notation for definition of constraints.

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

```
MESSAGE { PDUType } ::= SEQUENCE
{
    asp ASPType,
    pdu PDUType
}
```

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

```
SetupMessage ::= MESSAGE { SetupPDU }
```

4.3.2 Transformation rules

Rule 2: Expand out the parameterized types and values.

Example 4:

```
-- Parameterized value in value notation
genericGreeting{ IA5String : name} IA5String ::= {"Hello", name}

firstString IA5String ::= genericGreeting{ "World"}

-- Parameterized value in type notation
Message1{ INTEGER:maxSize, INTEGER:minSize} ::= SEQUENCE
{
    asp         INTEGER,
    pdu     OCTET STRING (SIZE (minSize..maxSize))
}

ExampleMsg ::= Message1{10, 40}

-- Parameterized type definition
Message2{ PDUType } ::= SEQUENCE
{
    asp         ASPType,
    pdu     PDUType
}

SetupMessage ::= Message2{ SetupPDU }
```

transforms to:

```
-- Transformed Parameterized value in value notation
firstString IA5String ::= "HelloWorld"

-- Transformed Parameterized value in type notation
ExampleMsg ::= SEQUENCE
{
```

```
    asp         INTEGER,
    pdu     OCTET STRING (SIZE (10..40))
}


-- Transformed Parameterized type definition
SetupMessage ::= SEQUENCE
{
    asp         ASPType,
    pdu     SetupPDU
}
```

4.3.3 Transformation limitations

When information objects classes, objects and object sets are used in parameterization, the transformation rules for information objects must be applied to these definitions before the parameterization is expanded out.

4.4 Information objects

4.4.1 Description

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

In ASN.1 94 some of the defined types within the language are defined in terms of information objects (these types and classes are INSTANCE-OF, TYPE-IDENTIFIER and ABSTRACT-SYNTAX).

In principle the transformation rules must replace information extracted from information objects or class, i.e. corresponding to notation *ObjectClassFieldType, TypeFromObject, ValueSetFromObjects, ObjectClassFieldValue, ValueFromObject* by the information itself in the abstract syntax definitions.

4.4.2 Transformation rules

Rule 5: Replace use of "InstanceOfType" notation by the associated sequence.

Example 5:

```
ACCESS-CONTROL-CLASS ::= TYPE-IDENTIFIER


AccessControl ::= INSTANCE OF ACCESS-CONTROL-CLASS ({PossibleTypes})
```

transforms to:

```
AccessControl ::= [UNIVERSAL 8] IMPLICIT SEQUENCE
```

```
        {
            type-id     ACCESS-COTROL-CLASS.&id ({PossibleTypes}),
            value       [0] ACCESS-CONTROL-CLASS ({PossibleTypes}{@.type-id})
        }
```

Rule 6: Remove any Component relation constraints in type declarations

Example 6:

Given the following Information object definition

```
MESSAGE ::= CLASS
{
    &msgCode        INTEGER UNIQUE,
    &msgLength      INTEGER,
    &MsgDataType    OPTIONAL
}
WITH SYNTAX
 {
    CODE            &msgCode,
    LENGTH          &msgLength,
    [DATA TYPE      &MsgDataType]
}
```

used to define the object set ConnectPhaseMsgs:

```
ConnectPhaseMsgs MESSAGE ::=
{
    setup | setupAck | release | relAck
}

setup MESSAGE ::=
{
    CODE            1
    LENGTH          12
```

```
    DATA TYPE     OCTET STRING
}

setupAck MESSAGE ::=
{
    CODE          2
    LENGTH        5
    DATA TYPE     INTEGER
}

release MESSAGE ::=
{
    CODE          3
    LENGTH        1
}

relAck MESSAGE ::=
{
    CODE          4
    LENGTH        1
}
```

The associated type definition:

```
ConnectPhasePDU ::= SEQUENCE
{
    id      MESSAGE.&msgCode
            ({ConnectPhaseMsgs}),
    size    MESSAGE.&msgLength
            ({ConnectPhaseMsgs } {@id}),
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs } {@id}) OPTIONAL
}
```

transforms to:

```
ConnectPhasePDU ::= SEQUENCE
```

```
{
    id      MESSAGE.&msgCode
            ({ConnectPhaseMsgs}),
    size    MESSAGE.&msgLength
            ({ConnectPhaseMsgs }),
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs }) OPTIONAL
}
```

Rule 7: Replace references to object set value fields in type definitions by using generic type plus subtyping or by defining a new type to represent the possible value set. The required generic type and possible value set must be extracted from the information object set.

Example 7:

Given the information object set and associated ConnectPDU type definition from Example 6.

```
ConnectPhasePDU ::= SEQUENCE
{
    id      MESSAGE.&msgCode
            ({ConnectPhaseMsgs}),
    size    MESSAGE.&msgLength
            ({ConnectPhaseMsgs }),
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs }) OPTIONAL
}
```

transforms to:

```
ConnectPhasePDU ::= SEQUENCE
{
    id      INTEGER
            (1..4),
    size    INTEGER
            (1 | 5 | 12),
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs }) OPTIONAL
}
```

Or the cleaner solution declaring new types, transforms to:

```
MsgCodes ::= INTEGER(1..4)
MsgSizes ::=   INTEGER(1 | 5 | 12)

ConnectPhasePDU ::= SEQUENCE
{
    id      MsgCodes,
    size    MsgSizes,
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs }) OPTIONAL
}
```

Rule 8: Replace references to information object type fields in type definitions by transforming into CHOICE construct.

Example 8:

Given the ConnectPDU type definition from the end of EXAMPLE 7.

```
ConnectPhasePDU ::= SEQUENCE
{
    id      MsgCodes,
    size    MsgSizes,
    data    MESSAGE.&MsgDataType
            ({ConnectPhaseMsgs }) OPTIONAL
}
```

transforms to:

```
ConnectPhasePDU ::= SEQUENCE
{
    id      INTEGER
            (1..4),
    size    INTEGER
            (1 | 5 | 12),
    data    CHOICE { setupBody OCTET STRING, setupAckBody INTEGER} OPTIONAL
}
```

Or the cleaner solution declaring a new type:

```
MsgBody ::= CHOICE
                {
```

```
                setupBody       OCTET STRING,
                setupAckBody    INTEGER
            }


ConnectPhasePDU ::= SEQUENCE
{
    id      MsgCodes,
    size    MsgSizes,
    data    MsgBody     OPTIONAL
}
```

Rule 9: Remove or change to ASN.1 comments all definition of classes, information objects and object sets.

Example 9:

Considering Example 6 all the following definitions should be deleted or converted into comments:

```
MESSAGE ::= CLASS
{
    &msgCode        INTEGER UNIQUE,
    &msgLength      INTEGER,
    &MsgDataType    OPTIONAL
}
WITH SYNTAX
 {
    CODE                &msgCode,
    LENGTH              &msgLength,
    [DATA TYPE          &MsgDataType]
}


setup MESSAGE ::=
{
    CODE            1
    LENGTH          12
    DATA TYPE       OCTET STRING
}
```

```
setupAck MESSAGE ::=
{
    CODE            2
    LENGTH          5
    DATA TYPE       INTEGER
}

release MESSAGE ::=
{
    CODE            3
    LENGTH          1
}

relAck MESSAGE ::=
{
    CODE            4
    LENGTH          1
}

ConnectPhaseMsgs MESSAGE ::=
{
    setup | setupAck | release | relAck
}
```

4.4.3 Transformation limitations

The validity of transformation rule 7 which involves converting a reference to an information object set into a type containing a CHOICE is dependant on the transfer syntax. If the required encoding rules make the CHOICE visible in the transfer syntax (PER for example) this transformation is invalid (changes the bits transmitted on the line).

For SDL applications where the transfer syntax is not an issue and for all application areas using a transfer syntax where the CHOICE is not visible (BER for example) transformation rule 7 is still valid.

The second limitation is these transformation rules provide no mechanism for handling situations where it is impossible to construct an associated CHOICE type. For example if we have the type definition:

```
ConnectPhasePDU ::= SEQUENCE
{
    id      MESSAGE.&msgCode
```

```
            ({ConnectPhaseMsgs}),
    size    MESSAGE.&msgLength
            ({ConnectPhaseMsgs } {@id}),
    data    MESSAGE.&MsgDataType OPTIONAL
}
```

The data field is unconstrained, it can be of any type. In this situation it is impossible to implement transformation rule 7 and convert this field to an associated CHOICE type.

The transformation rules presented in this clause cannot reproduce the full semantics of the information objects they replace. The transformation rules cannot preserve component relation constraints. These constraints provide the ability to constrain a type or value with reference to a different field within an information object set.

For example if we consider EXAMPLE 6 in the ConnectPhasePDU type the size field has the component relational constraint {@id}. This means the size field is constrained to the possible values defined in the object set ConnectPhaseMsgs which have the correct id field value.

4.5 User defined constraints

4.5.1 Description

This feature is used to express constraints which are too complex to represent using the normal ASN.1 constraints mechanisms. The ASN.1 94 language does not fully specify how to process these constraints and in effect this feature just provides a special form of ASN.1 comment in which the required constraint mechanism is described in text. User defined constraints need to be removed or converted into an ASN.1 comment.

4.5.2 Transformation rules

Rule 10: Remove user defined constraints or convert to ASN.1 comment.

Example 10:

```
ENCRYPTED {ToBeEnciphered} ::= BIT STRING
    (CONSTRAINED BY
    {-- must be the result of the encipher of some BER-encoded value of -- ToBeEnciphered}
    ! Error : securityViolation)
```

transforms to:

```
ENCRYPTED ::= BIT STRING
```

or

```
ENCRYPTED {ToBeEnciphered} ::= BIT STRING
    -- (CONSTRAINED BY --
    -- { must be the result of the encipher of some BER-encoded value of ToBeEnciphered} --
    -- ! Error : securityViolation) --
```

4.5.3 Transformation limitations

Since user defined constraints can be regarded as a special form of ASN.1 comment this transformation is completely transparent.

Annex A (informative): Additional rules relative to the use of ASN.1 94 in conjunction with SDL

A.1 Identifiers

Case sensitivity is not supported by Z.100 [7] and Z.105 [8]. 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.

Rule A11: Rename the ASN.1 entities so that two entities of the same class shall not be identical when put in lower case.

Example A1:

```
TypeA ::= OCTET STRING (3)

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

can be replaced by:

```
AType ::= OCTET STRING (3)

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

The use of the same identifier for multiple named numbers or multiple named bits of different types in the same scope shall be avoided. The motivation is that named numbers and named bits are mapped onto SDL literals and SDL integer synonyms respectively. Using the same identifier twice would result in illegal SDL (redefinition of the same synonym or same literal). 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.

Rule A12: Rename the numbers of bits so that the identifiers for named numbers or named bits of different types in the same scope shall not be identical.

Example A2:

```
Int1 ::= INTEGER { a (0) }

Int2 ::= INTEGER { a (1) }
```

transforms to:

```
Int1 ::= INTEGER { a1 (0) }

Int2 ::= INTEGER { a2 (1) }
```

The OBJECT IDENTIFIER component values that are assigned by ITU-T, ISO, or both, are not defined in the package called *Predefined*.

Rule A13: Define and import an SDL package containing the definitions of components of the OBJECT IDENTIFIERS used in the ASN.1

Example A3:

In order to use :

```
{ccitt recommendation q 1228 }
```

the component values CCITT, recommendation, and q have to be defined in a user defined package.

Annex B (informative): Additional rules relative to the use of ASN.1 94 in conjunction with TTCN

B.1 Automatic tagging

B.1.1 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).

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).

B.1.2 Transformation rules

These transformation rules need only be applied if the source ASN.1 94 module has the tagging type set to AUTOMATIC in the module header.

Rule B11: Expand out COMPONENTS OF for all SEQUENCE constructs which contain no tagged type.

Example B1:

```
TypeA ::= SEQUENCE
        {
            alpha   INTEGER,
            beta    BOOLEAN
        }

-- SEQUENCE construct with tagged type
TypeB ::= SEQUENCE
        {
            cappa  INTEGER,
            delta   [1]BOOLEAN,
            COMPONENTS OF TypeA
        }

-- SEQUENCE construct without tagged type
TypeC ::= SEQUENCE
        {
```

```
        cappa  INTEGER,
        delta  BOOLEAN,
        COMPONENTS OF TypeA
    }
```

transforms to :

```
-- SEQUENCE construct with tagged type (no change)
TypeB ::= SEQUENCE
    {
        cappa  INTEGER,
        delta  [1]BOOLEAN,
        COMPONENTS OF TypeA
    }

-- SEQUENCE construct without tagged type COMPONENTS OF expanded out
TypeC ::= SEQUENCE
    {
        cappa  INTEGER,
        delta  BOOLEAN,
        alpha  INTEGER,
        beta   BOOLEAN
    }
```

Rule B12: Manually add tag type for all SEQUENCE constructs which contain no tagged type (or contain tagged type due to transformation rule B11).

Example B2:

```
-- SEQUENCE construct without tagged type
TypeC ::= SEQUENCE
    {
        cappa  INTEGER,
        delta  BOOLEAN,
```

```
        alpha   INTEGER,
        beta    BOOLEAN
    }
```

transforms to :

```
TypeC ::= SEQUENCE
    {
        cappa   [0] IMPLICIT INTEGER,
        delta   [1] IMPLICIT BOOLEAN,
        alpha   [2] IMPLICIT INTEGER,
        beta    [3] IMPLICIT BOOLEAN
    }
```

Rule B13: Manually add tag type for all SET constructs which contain no existing tagged type.

Example B3:

```
-- SET construct without tagged type
TypeD ::= SET
    {
        cappa  INTEGER,
        delta   BOOLEAN,
        alpha   INTEGER,
        beta    BOOLEAN
    }
```

transforms to :

```
TypeD ::= SET
    {
        cappa   [0] IMPLICIT INTEGER,
        delta   [1] IMPLICIT BOOLEAN,
```

```
        alpha   [2] IMPLICIT INTEGER,
        beta    [3] IMPLICIT BOOLEAN
    }
```

Rule B14: Manually add tag types for all CHOICE constructs which contain no existing tagged type.

EXAMPLE B4:

```
-- CHOICE construct without tagged type
TypeE ::= CHOICE
    {
        cappa  INTEGER,
        delta   BOOLEAN
    }
```

transforms to :

```
TypeE ::= CHOICE
    {
        cappa   [0] IMPLICIT INTEGER,
        delta   [1] IMPLICIT BOOLEAN
    }
```

B.1.3 Transformation limitations

If the type within the respective construct is a CHOICE type, open type or a DummyReference, AUTOMATIC tagging inserts an EXPLICIT tag type for this element.

Annex C (informative): Additional feature support required to extend tools from ASN.1 90 to ASN.1 X.680

This annex lists the differences between ASN.1 90 and ASN.1 X.680. This list can be used as a check list for upgrading ASN.1 based tools. This appendix is composed of two subclauses: a first one list the features which have been removed or changed. A second one gives the features which have been incorporated in addition.

C.1 Alterations

C.1.1 Removal of Macros

In ASN.1 90 the macro capability allowed the user to extend the notation by providing macros. This capability had the disadvantage that it was not machine processable (redefinition of grammar on the fly) and thus was not fully supported by tools.

This capability has been removed in X.680 [1] and replaced by the information objects and the parameterization capability defined respectively in X.681[3] and X.683 [6].

C.1.2 Removal of ANY and ANY DEFINED BY

The normal use of the any type defined in ASN.1 90 was to leave a "hole" in a specification which would be filled in by some other specification. In X.680 [1] the any type has been superseded by the ability to specify information object classes and then to refer to the fields of information object classes from within type definitions (X.681 [3]). Though described in X.680 [1], the notations AnyType and AnyValue should not be supported by tools.

C.1.3 Local identifiers

Contrary to ASN.1 90, in X.680 [1] identifiers are mandatory in NamedTypes and NamedValues.

The NamedType notation which was defined in ASN.1 90 as

NamedType ::= identifier Type | Type | SelectionType

has been changed to

NamedType ::= identifier Type | SelectionType

In a same way the Named value notation has been changed from

NamedValue ::= identifier Value | Value

to

NamedValue ::= identifier Value

C.1.4 Choice Value syntax

In order to remove some ambiguities, the ChoiceValue now contains a colon. The ChoiceValue notation which was defined in ASN.1 90 as

ChoiceValue ::= NamedValue

has been changed to

ChoiceValue ::= identifier ":" Value

C.1.5 External

The associated type for the EXTERNAL type has been changed from:

EXTERNAL = [UNIVERSAL 8] IMPLICIT SEQUENCE

{ direct-reference OBJECT IDENTIFIER OPTIONAL,

indirect-reference INTEGER OPTIONAL,

data-value-descriptor ObjectDescriptor OPTIONAL,

encoding CHOICE

```
{single-ASN1-type   [0] ANY,
octet-aligned       [1] IMPLICIT OCTET STRING,
arbitrary           [2] IMPLICIT BIT STRING } }
```

to:

```
SEQUENCE {
    identification          CHOICE {
        syntaxes            SEQUENCE {
            abstract            OBJECT IDENTIFIER,
            transfer            OBJECT IDENTIFIER }
                -- Abstract and transfer syntax object identifiers --,
        syntax              OBJECT IDENTIFIER
                -- A single object identifier for identification of the class and encoding --,
        presentation-context-id     INTEGER
                -- (Applicable only to OSI environments)
                -- The negotiated presentation context identifies the class of the value and its encoding --,
        context-negotiation     SEQUENCE {
            presentation-context-id     INTEGER
            transfer-syntax         OBJECT IDENTIFIER }
            -- (Applicable only to OSI environments)
                -- Context-negotiation in progress for a context to identify the class of the value
                -- and its encoding --,
        transfer-syntax         OBJECT IDENTIFIER
                -- The class of the value (for example, specification that it is the value of an ASN.1 type)
                -- is fixed by the application designer (and hence known to both sender and receiver). This
                -- case is provided primarily to support selective-field-encryption (or other encoding
                -- transformations) of an ASN.1 type --,
        fixed               NULL
                -- The data value is the value of a fixed ASN.1 type (and hence known to both sender
                -- and receiver) -- },
    data-value-descriptor       ObjectDescriptor  OPTIONAL
                -- This provides human-readable identification of the class of the value --,
    data-value          CHOICE {
        notation            ABSTRACT-SYNTAX.&Type
```

```
                -- This type notation is defined in X.681[3] and has a value
                -- notation which is any ASN.1 type definition, followed by a colon and the value notation
                -- for that type. This choice alternative is provided to enable the specification using
                -- human-friendly notation of the data values that are values of an ASN.1 type. --,
        encoded         BIT STRING
                -- This choice alternative is provided to enable the specification of data values that are not
                -- values of a single ASN.1 type. -- } }
    ( WITH COMPONENTS {
        ... ,
        identification  (WITH COMPONENTS {
            ... ,
            syntaxes        ABSENT,
            transfer-syntax     ABSENT,
            fixed           ABSENT } ) } )
```

C.2 Additions

C.2.1 Automatic tagging

The AUTOMATIC TAGS keyword has been added to the TagDefault notation. It allows the definition of a module without insertion of tags in the module body.

```
TagDefault ::=
        EXPLICIT TAGS  |
        IMPLICIT TAGS  |
        empty
```

has been replaced by:

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

C.2.2 Embedded PDV

The EmbeddedPDVType, a superset of EXTERNAL with more efficient encoding, has been added to the BuiltinType notation. Similarly the EmbeddePDVValue has been added to the BuiltinValue notation.

The BuiltinType notation is now:

BuiltinType ::=

...

EmbeddedPDVType |

...

EmbeddedPDVType ::= EMBEDDED PDV

The BuiltinValue notation is now:

BuiltinValue ::=

...

EmbeddedPDVValue |

...

EmbeddedPDVValue ::= SequenceValue

C.2.3 String types

Three new string types has been added: UniversalString, BMPString and CHARACTER STRING. The first two type are intended to carry characters defined in ISO 10646-1 [11]. CHARACTER STRING is an unrestricted string type intended to model any string which cannot be represented with any of the other existing ASN.1 string types.

C.2.4 Exception specification

The exception specification indicates special handling in the event of an exceptional condition. Subtype has been replaced by Constrainedtype itself defined as:

ConstrainedType ::= Type Constraint |

TypeWithConsraint

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

ExceptionSpec ::= "!" ExceptionIdentification | empty

ExceptionIdentification ::= SignedNumber |

DefinedValue |

Type ":" Value

C.2.5 New subtype operators

The ASN.1 90 SubType notation has been replaced by the ConstrainedType notation enhancing the possibilities of subtyping with new operators: UNION, INTERSECTION, EXCLUSION.

History

Document history		
V1.1.1	September 1998	Publication

ISBN 2-7437-2379-3
Dépôt légal : Septembre 1998