INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Addendum 1
Q.1400

(02/95)

INTELLIGENT NETWORK

ARCHITECTURE FRAMEWORK FOR THE DEVELOPMENT OF SIGNALLING AND OAM PROTOCOLS USING OSI CONCEPTS

Addendum 1 to
ITU-T Recommendation Q.1400

(Previously "CCITT Recommendation")

FOREWORD

The ITU-T (Telecommunication Standardization Sector) is a permanent organ of the International Telecommunication Union (ITU). The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1 (Helsinki, March 1-12, 1993).

Addendum 1 to ITU-T Recommendation Q.1400 was prepared by ITU-T Study Group 11 (1993-1996) and was approved under the WTSC Resolution No. 1 procedure on the 7th of February 1995.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Addendum 1 to Recommendation Q.1400

ARCHITECTURE FRAMEWORK FOR THE DEVELOPMENT OF SIGNALLING AND OAM PROTOCOLS USING OSI CONCEPTS

(Geneva, 1995)

New rules on forward and backward compatibility for protocols specified in ASN.1 have been agreed. These rules are simple to use and are based on latest ASN.1, X.680-series Recommendations. Subclause 12.5/Q.1400 (1993) is to be replaced by the following text.

12.5 Compatibility rules for application layer protocols specified in ASN.1

It is envisaged that minor extensions to an application protocol may be needed from time to time. An abstract syntax is extended if its associated type is extended (e.g. if a choice type, it can be extended by adding a new component or extending an existing one). One way of extending a PDU (or any structure type) is to extend the type of any of its components. In supporting such extensions, care needs to be taken to ensure that the extensions are indeed minor. Therefore, the following types of extensions to the abstract syntax might be considered minor:

– addition of an information element which may enhance an activity but is not essential to performing the basic activity (e.g. list of additional routing options); or

– addition of an information element to add a capability which is not essential to the base capability (e.g. addition of “Name” in addition to “Number” for terminal display purposes).

In the above cases, a new application context name need not be defined; however, forward compatibility procedures for dealing with the unknown information must exist at the receiving application process.

The following types of extensions might be considered major:

– addition of a new procedure; or

– fundamental alteration of a procedure (e.g. “do this procedure twice”).

In these cases, a new application context name should be defined.

Following backward and forward compatibility rules are applied to application layer protocols specified using ASN.1.

12.5.1 Backward compatibility rules

Modifications to all future versions of ITU-T Recommendations for ASN.1 based signalling protocols from 1992 onwards shall be made in such a way that backward compatibility with older versions is ensured.

The objective of the following subclauses is to provide guidelines on the types of changes which can be made to a protocol specification while ensuring backward compatibility with the original protocol.

Changes which do not affect the transfer-syntax (i.e. the bits and bytes exchanged between peer entities) or which extend it are backward compatible. Using simple words, backward compatibility means that an encoded PDU of the original protocol is a valid encoded PDU for the new protocol.

Apart from very specific cases, changes which do not affect the abstract syntax or extend it produce a backward-compatible transfer syntax when BER are employed. Such changes are listed in 12.5.1.1 and 12.5.1.2.

However, in the absence of forward compatibility rules, a proper interworking can only be ensured if extended values (i.e. values which belong to the extended abstract syntax but not to the original one) are never sent to an implementation which only supports the original protocol.

Non-backward compatible changes are those which affect the transfer-syntax in a non-compatible way. In this case, an encoded PDU of the original protocol is not necessarily a valid encoded PDU for the new protocol.

In general, a non-compatible change from an abstract-syntax point of view causes a non-compatible change from a transfer-syntax point of view. However, for a given set of encoding rules there may be some exceptions.

An example for such changes are listed in 12.5.1.3.

In addition it is obvious that changing the encoding rules causes in most cases incompatibility from a transfer-syntax point of view.

12.5.1.1 Changes without impact on the abstract-syntax

These changes are purely restricted to the way the abstract syntax and the type used for its definition are specified. They do not affect the set of values defined by the abstract syntax. Such changes may be needed to bring a specification in line with the rules stated in 12.5.1.2 and 12.5.1.3 of this (this list is not necessarily complete).

a) In a set type or sequence type, replace the use of "COMPONENTS OF" with direct inclusion of the equivalent components, or vice versa.

b) In a choice type, replacing nested choice types with direct inclusion of each "NamedType" which appear in the "AlternativeTypeList".

c) Replace a type by a "typereference" representing the same type or vice versa.

d) Replace a value by a "valuereference" representing it or vice versa: This includes replacing a "number" by a "valuereference".

e) Replace a type by an equivalent selection type or vice versa.

f) Add or (if unused) remove one or more "NamedBit" from a bit string type.

g) Add or (if unused) remove one or more "NamedNumber" from an integer type.

h) Change the spelling of a "typereference", a "modulereference" a "valuereference" or an "identifier" consistently throughout all ASN.1 modules. This includes adding identifiers where allowed from the syntax.

i) Split up one ASN.1 module into several ASN.1 modules.

j) Put several ASN.1 modules together into one ASN.1 module.

k) Move parts of one ASN.1 module into another ASN.1 module.

l) Add one or more "Symbol" to the "EXPORTS" list. (or remove the "EXPORTS" statement to indicate that everything is exported).

m) Add one or more "Symbol" to the "IMPORTS" list (symbols from ASN.1 modules already referenced in the "IMPORTS" list as well as symbols from newly referenced ASN.1 modules).

n) Remove one or more existing "Valueassignment" if their associated "valuereference" is never used (even not implicitly in ANY DEFINED BY construction) throughout all ASN.1 modules.

o) Remove one or more existing "Typeassignment" (including those of Type OPERATION and ERROR) if they are never used throughout all ASN.1 modules.

p) Add an ERROR Type or Value which was already included in the abstract syntax (i.e. attached to another OPERATION type) to the list of ERRORS of an OPERATION Type if it had such a list or add a list of ERRORS to an OPERATION Type if it did not have a list.

q) Add an OPERATION Type or Value which was already included in the abstract syntax (e.g. not as a linked Operation) to the list of LINKED OPERATIONS of an OPERATION Type if it had such a list or add a list of LINKED OPERATIONS to an OPERATION Type if it did not have a list.

12.5.1.2 Extension of an abstract syntax

An abstract syntax is extended if its associated type is extended (i.e. if a choice type, it can be extended by adding a new component or extending an existing one). One way of extending a PDU (or any structured type) is to extend the type of any of its components.

One ASN.1 type is considered to be an extension of another if the former includes all the values of the latter and possibly some others.

Given a certain type, its extensions are those types which could be derived by one or more of the following changes, combined with any number of those described in 12.5.1.1.

a) Change a single type into a choice type which includes this single type in the "AlternativeTypeList".

NOTE 1 – The tag of this alternative has to remain unchanged, no additional (EXPLICIT) Tag is allowed. It has to be taken care, that all references to this changed type throughout all ASN.1 modules still meet all ASN.1 requirements – in particular distinctness of tags and uniqueness of identifiers.

b) Add one/more "NamedType" to the "AlternativeTypeList" of a choice type.

NOTE 2 – See Note 1 for changing a single type into a choice type.

c) Add an optional component to a sequence type or a set type.

d) Add a default component to a sequence type or a set type.

e) Extend one or more components of a choice type, sequence type or a set type.

f) Extend the type in terms of which a sequence-of type or a set-of type is defined.

g) Change a mandatory component of a sequence type or set type to an optional or default component.

NOTE 3 – The Tag of this component has to remain unchanged and distinctness of Tags has to be ensured.

h) Add one or more new "NamedNumber" to an enumerated type.

NOTE 4 – Distinctness of values and uniqueness of identifiers has to be ensured.

i) Extend the "ValueRange" of an integer type by decreasing the "LowerEndpoint" and/or increasing the "UpperEndpoint".

j) Extend the "SizeConstraint" of an octet string type, a bit string type or a character string type by decreasing the "LowerEndpoint" and/or increasing the "UpperEndpoint".

k) Extend the "SizeConstraint" of a sequence-of type or a set-of type by decreasing the "LowerEndpoint": and/or increasing the "UpperEndpoint".

l) Change the "Value" assigned to a "valuereference" if the effect for all references still meets all other rules (e.g. increase a Value used only as "UpperEndpoint" in a "ValueRange" or "SizeConstraint").

The following changes to OPERATION and ERROR definition affect the abstract syntax formed by the set of values whose type is the one of TCAP messages or ROSE PDUs parameterized by a specific list of operations.

m) Add new value definition of Type OPERATION and ERROR respectively as long as they are distinct with all other value definitions of Type OPERATION and ERROR respectively.

n) Add a Type as ARGUMENT/PARAMETER to an OPERATION Type if it did not have an ARGUMENT/PARAMETER.

o) Add a Type as RESULT to an OPERATION Type if it had an empty RESULT or no RESULT.

p) Add a Type as PARAMETER to an ERROR Type if it did not have a parameter.

12.5.1.3 Non-compatible changes

Changes cause incompatibility from an abstract-syntax point-of-view when a value of the original abstract syntax is not a valid value for the new abstract syntax.

A non-compatible change to the type(s) in terms of which an abstract syntax is defined causes an incompatibility between the original abstract syntax and the new one.

The following list provides some examples of such non-compatible changes:

- replace a type by another type even if the tag remains the same;
- remove a type definition or a value definition referred either explicitly (IMPORTS) or implicitly (ANY DEFINED BY of TCAP);
- remove a "NamedType" from the "AlternativeTypeList" of a Choice type;
- remove a "NamedNumber" from the "Enumeration" of an enumerated type;
- restrict the "ValueRange" of an integer type;
- restrict the "SizeConstraint" of a string type;
- restrict the "SizeConstraint" of a sequence-of type;
- change the order of elements in the "ElementTypeList" of a sequence type;
- make any combination of the above changes to one or more components of a structured type.

12.5.2 Forward compatibility rules

Forward compatibility is most generally achieved through application-context negotiation. However, in order to minimize the number of protocol fallbacks in the signalling network it will sometimes be necessary to define forward compatibility rules which allow a version of a protocol to accept protocol data units generated by a future version without having to provide a new application-context-name.

This feature is also necessary where application context negotiations is not supported, e.g. the optional dialogue portion of TC is not supported.

If the protocol designer wishes to ensure that a value of a PDU of the new version of a protocol be (at least) always partly recognized by an implementation of an older version of this protocol, the following guidelines shall be followed:

- the new protocol version shall comply with the rules described in 12.5.1 (i.e. the new abstract syntax shall be an extension of the old one);
- the applicable encoding rules (which shall be unchanged) permit the unknown parts of the encoding to be delimited;[1)]
- extensibility rules shall be included in the specification of the original protocol.

If the last Recommendation is not followed, the behaviour of a receiving entity is implementation dependent.

It is recommended to restrict the use of the extensibility rules to:

- the addition of OPTIONAL and DEFAULT components in types derived from the SEQUENCE or SET type;
- the addition of bit assignment in types derived from the BIT STRING type (without extending the size constraint);
- the addition of alternatives to a CHOICE type, providing that it does not correspond to a mandatory element of a higher structure;
- the addition of an enumerated values to an ENUMERATED type, providing that it does not correspond to a mandatory element of a higher structure.

The protocol designer shall be aware that extensibility rules beyond those listed here (e.g. relaxing a size constraint or a value range, or extending a CHOICE type corresponding to a mandatory element) may require special care (e.g. specifying error codes to be used when an unrecognized information element is encountered) to avoid important functional errors.

1) This is always ensured if the BER are employed.

There are two basic ways of providing a specification of the extensibility rules:

i) The specification includes a general statement which globally applies to any PDU of the protocol.

   This is illustrated by the following extract from the ITU-T Transaction Processing (TP), Recommendation X.862.

   "To provide for future compatibility a receiving TPPM shall:

   a) ignore any undefined element of a SET, ENUMERATED or SEQUENCE;

   b) where named bits are used in ASN.1, treat any bit as insignificant if no name is assigned to it."

ii) The specification includes several statements, each of which specifically applies to a particular PDU (or any inner structure) of the protocol.

The 1993 version of ASN.1 provides a new piece of notation to flag the types which can be extended in such a way that unknown additions be ignored. This flag is an ellipsis (...) and is called an extensions marker. It is recommended that users of the 1988 version of ASN.1 include this flag as an ASN.1 comment.

The following examples illustrate a 1993 specification where extra (unknown) element values will be accepted for PDU-A and TypeA, but not for TypeB.

```
PDU-A ::= SEQUENCE {
        element1 TypeA,
        element2 TypeB,
        ...}

TypeA ::= SEQUENCE {
        element3 TypeC,
        element4 TypeD,
        ...
        }

TypeB ::= SEQUENCE {
        element5 TypeE,
        element6 TypeF }
```

The extensibility mechanism may also be required when application layer relays are involved which support a lower version of the protocol than the one used by the actual senders and receivers of the messages (e.g. Vn-Vn TC dialogue relayed by a Vn-1 node). However, in this case, there is an additional requirement that the relay node passes the received unrecognized information to the subsequent node.

The specification of extensibility rules without any other indication does not place any requirement on the receiving entity regarding further processing of the part of encoding which corresponds to unknown values. In the absence of any additional specification, it is assumed that the undecoded parts are ignored.

If there is a need to take any other special action (e.g. return a specific error, retransmit this part of the encoding to a third party, etc.) the protocol designer shall provide an explicit specification of the unexpected behaviour, in addition to the extensibility rules. As a possible option, it can include in the original version of the protocol some information elements which convey dynamically an indication of the behaviour to be followed on receipt of unrecognized information elements.