INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Q.775
(06/97)

SERIES Q: SWITCHING AND SIGNALLING

Specifications of Signalling System No. 7 – Transaction capabilities application part

Guidelines for using transaction capabilities

ITU-T Recommendation Q.775

(Previously CCITT Recommendation)

ITU-T Q-SERIES RECOMMENDATIONS

SWITCHING AND SIGNALLING

SIGNALLING IN THE INTERNATIONAL MANUAL SERVICE	Q.1–Q.3
INTERNATIONAL AUTOMATIC AND SEMI-AUTOMATIC WORKING	Q.4–Q.59
FUNCTIONS AND INFORMATION FLOWS FOR SERVICES IN THE ISDN	Q.60–Q.99
CLAUSES APPLICABLE TO ITU-T STANDARD SYSTEMS	Q.100–Q.119
SPECIFICATIONS OF SIGNALLING SYSTEMS No. 4 AND No. 5	Q.120–Q.249
SPECIFICATIONS OF SIGNALLING SYSTEM No. 6	Q.250–Q.309
SPECIFICATIONS OF SIGNALLING SYSTEM R1	Q.310–Q.399
SPECIFICATIONS OF SIGNALLING SYSTEM R2	Q.400–Q.499
DIGITAL EXCHANGES	Q.500–Q.599
INTERWORKING OF SIGNALLING SYSTEMS	Q.600–Q.699
SPECIFICATIONS OF SIGNALLING SYSTEM No. 7	Q.700–Q.849
General	Q.700
Message transfer part (MTP)	Q.701–Q.709
Signalling connection control part (SCCP)	Q.711–Q.719
Telephone user part (TUP)	Q.720–Q.729
ISDN supplementary services	Q.730–Q.739
Data user part	Q.740–Q.749
Signalling System No. 7 management	Q.750–Q.759
ISDN user part	Q.760–Q.769
Transaction capabilities application part	**Q.770–Q.779**
Test specification	Q.780–Q.799
Q3 interface	Q.800–Q.849
DIGITAL SUBSCRIBER SIGNALLING SYSTEM No. 1	Q.850–Q.999
PUBLIC LAND MOBILE NETWORK	Q.1000–Q.1099
INTERWORKING WITH SATELLITE MOBILE SYSTEMS	Q.1100–Q.1199
INTELLIGENT NETWORK	Q.1200–Q.1999
BROADBAND ISDN	Q.2000–Q.2999

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION Q.775

GUIDELINES FOR USING TRANSACTION CAPABILITIES

Summary

Revised Recommendation Q.775 provides guidance to TC-Users on defining TC-User ASEs.

Source

ITU-T Recommendation Q.775 was revised by ITU-T Study Group 11 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on the 5th of June 1997.

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the 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.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

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

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had/had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Page

Recommendation Q.775

GUIDELINES FOR USING TRANSACTION CAPABILITIES

(revised in 1997)

1 Introduction

1.1 General

The purpose of this Recommendation is to provide guidelines to potential users of Transaction Capabilities (TC-users). The examples given are illustrations only; they indicate how an application may use TCAP, not how TC must be used in all cases. The technical basis of this Recommendation are Recommendations Q.771 to Q.774; in case of misalignment, these should be considered as the primary reference.

The main purpose of TC is to provide support for interactive applications in a distributed environment. TC is based on the Remote Operations concept defined in Recommendation X.880 (ROS) together with some enhancements and additions specific to the Signalling System No. 7 environment to provide the services needed by TC-users. Interactions between distributed application entities are modelled by Operations. An operation is invoked by an (originating) entity: the other (destination) entity attempts to execute the operation and possibly returns the outcome of this attempt.

The semantics of an operation (represented by its name and parameters) is not relevant to TC; TC provides facilities which are independent of any particular operation. The TC-user, when defining an application, must:

1) select operations (this involves defining the semantics and syntax of the data exchanged during the operation invocations and its responses);
2) select TC facilities to support these operations. Such facilities include the handling of individual operations, and the ability to have a number of related operations attached to an association between TC-users, called a dialogue;
3) define the application script (e.g. which of the two peers invokes which operations, the order of message exchange that constitutes the dialogue between the peer TC-users, and their reactions to abnormal situations).

This Recommendation describes the selection process of defining and using operations. The operations appearing hereafter are fictitious, and are taken for illustration purposes only. Also described are the facilities offered by TC for handling one or a sequence of operations in a dialogue. The definition of specific sequences of operations belongs to the application protocol definition and is beyond the scope of this Recommendation; however, clauses 4 and 5 give a brief indication of what information an application specification should contain.

TC services are made accessible to TC-users via primitives; these primitives model the interface between TC and its users, but do not constrain any implementation of this interface.

1.2 Environment

TC defines the end-to-end protocol between TC-users located in a Signalling System No. 7 network. At present, there is no standard interface defined for the use of TC over any other underlying protocol or network (e.g. X.25) than the SCCP of Signalling System No. 7.

TC considers users which are real-time sensitive and do not need to exchange large amounts of data. It is considered that for these users, the standard protocols defined for OSI layers 4 to 6 in the

X-Series of Recommendations would result in excessive overheads and hence are not used.

As a result, TC cannot support all kinds of applications, and a number of applications will still require more elaborate services such as specified in the X-Series of Recommendations. Beside indicating what TC can do, this Recommendation indicates what it cannot do, in order to help the application designer choose how to support an application.

2 Operations

2.1 Definition

An operation is invoked by an originating TC-user to request a destination TC-user to perform a given action.

Each operation belongs to a particular class. This indicates whether either a successful outcome (result), or an unsuccessful outcome (error), or both, or none have to be reported by the destination.

The class of an operation is not signalled to the remote TC-user at the time the operation is invoked; it is assumed that the applications at both ends have a common understanding of the class of each operation in use.

As well as the class, the definition of the operation includes a timer value indicating the maximum time in which the operation should be completed and the result reported.

This timer value is a local matter; it is not conveyed to the remote end through any protocol. It is chosen by the TC-user when defining the operation based on expectations of the round trip time from one TC-user to another and processing delays.

An operation is defined by:

– its operation code and the type of any parameters associated with the operation request;

– its class;

– if the class requires report of success, the possible results corresponding to successful executions are defined by a list of parameters;

– if the class requires report of failure, the possible results corresponding to situations where the operation could not be executed completely by the remote TC-user. Each such situation is identified by a specific error cause; the list of these error causes is part of the operation definition. Diagnostic information can be added to the error cause: if present, it is part of the definition;

– the list of possible linked operations, if replies consisting of linked operations are allowed for this operation. Linked operations have to be described separately;

– a timer value indicating the interval by which the operation has to be completed, if any, returned. This timer value can be one of the factors that is used by an implementation to manage the invoke ID associated with the operation invocation. (When the timer associated with an operation invocation has expired, the invoke ID is returned to the pool of invoke IDs after a suitable, implementation-dependent "freezing" period.)

As a general rule, the choice of the class of an operation should be based on the semantics associated with an operation and operations should not be designed to be carried in any particular message. For instance, if the invoker of the operation does not require any acknowledgement of whether an operation could actually be performed or not, then an operation of Class 4 may be the most appropriate. If an operation invoker does not require explicit acknowledgement that an operation has been performed successfully, but wishes to know if it could not be performed at all, then an operation of Class 2 is suitable. For example, defining an operation such as "Play Announcement" as

Class 2 or 4 suggests different intentions of the operation invoker even though the actions of the operation performer may be identical.

2.2 Examples

2.2.1 Simple operation handlings

NOTE – The operation invocation should fit into one message, and so should the report of a successful outcome. Reports of success may be segmented using Return Result – Not last and Return Result – Last.

Class 1 (both success and failure reported)

Translate a freephone number into a called subscriber number; return the called number if the translation can be performed, otherwise indicate why it cannot; time allocated: 2 seconds.

If the response is not returned for an operation invocation after the timer has expired, the TC-user is informed (operation cancel led by TC); it may assume that either the invocation or the response was lost and, depending on the application requirements, take suitable corrective actions (e.g. invoke the operation again, inform local management, etc.).

Class 2 (only failure reported)

Perform a routine test and send a reply only in case something went wrong; time allocated: 1 minute.

In the case of a Class 2 operation, the TC-user is informed if no result has been received when the timer expires. This is interpreted as a successful outcome, even if the invocation was lost.

This aspect should be considered when selecting Class 2.

Class 3 (only success reported)

Perform a test: this corresponds to a pessimistic view, where failure is considered as the default option, not requiring any reply.

Timer expiry is indicated to the TC-user: this should be interpreted by the TC-user as a failure of the operation (but is considered normal by TC, which considers that the operation has terminated). This aspect should be considered when selecting Class 3.

Class 4 (neither success, nor failure reported)

Send a warning, without expecting a reply or acknowledgement of any kind.

In this case, a result never arises from the invocation of the operation. The TC-user relies upon TC and the network to deliver the invocation. Notification of the timer expiry is a local matter.

Comparisons with ROSE operation classes (Recommendation X.219)

ROSE provides for five classes of operations: Classes 2 to 5, called asynchronous classes, are identical to Classes 1 to 4 of TC. ROSE's Class 1 is a synchronous class; it has no counterpart in TC, where full-duplex exchanges of components are considered.

However, a TC-user can decide to operate in a synchronous manner (see 3.2.1).

2.2.2 More complex operation handling

Operations with segmented results

A successful result may be divided into several segments, each of which is indicated to the originator of the operation by one primitive. This facility, using the TC-RESULT-NL primitive, can be used by TC-users to overcome the absence of segmentation in the underlying layers. The last segment is indicated by the TC-RESULT-L primitive.

The report of an error cannot be segmented.

When the protocol designer can ensure that segmentation is provided by the underlying layers across the signalling routes on which TC messages are transferred, the use of this segmenting facility is deprecated.

TC cannot identify a specific segment in the case of a segmented result.

The TC-user should ensure that each segment can be parsed (i.e. the parameter Parameter of each TC-RESULT-NL/L request primitive should contain enough information to enable the construction of a valid value of the type (or compatible sub-type) associated with the result of the operation.

Example E1: An operation requests the execution of a test. The result of a correct execution is segmented in three parts P1, P2 and P3 to be returned to the originator.

A possible primitive sequence for example E1 is given in Table 1.

Table 1/Q.775

TC-user A	**TC-user B**
TC-INVOKE req (Test, Class = 1)	TC-INVOKE ind (Test) TC-RESULT-NL req (P1)
TC-RESULT-NL ind (P1)	TC-RESULT-NL req (P2)
TC-RESULT-NL ind (P2)	TC-RESULT-L req (P3)
TC-RESULT-L ind (P3)	
Time	
NOTE – The timeout value is specified by the originating TC-user at invocation time. A non-final result does not restart it.	

Linked operations

Another extension to the basic operation scheme is the ability to link an operation invocation to another operation invocation.

Typically, this facility covers situations where the destination of the original (linked-to) operation requires additional information in order to process this operation: this is the case where menu facilities are used (menu facilities allow a user to make a sequence of choices, each being dependent on the previous ones).

Example E2: The operation is the execution of a test with several options; before the test is executed, these options are offered for selection to the test originator (TC-user A). Two operations are nested: operation 1 is the test; operation 2 is the option selection. TC-user A first responds to operation 2 before TC-user B can perform the test with the indicated option(s).

A possible primitive sequence for example E2 is given in Table 2.

There is no limit to the number of operation invocations which may be linked to a given operation invocation.

Note that when an operation B is linked to another operation A, they do not have to be nested. The only condition is that the invocation of B should take place before the outcome of A is reported;

however, operation B does not have to terminate before operation A.

Table 2/Q.775

TC-user A	TC-user B	
TC-INVOKE req (Test, Class = 1)	 TC-INVOKE ind (Test) TC-INVOKE req (Option-selection, Class = 1)	Operation 1 begin Operation 2 begin
TC-INVOKE ind (Option-selection) TC-RESULT-L req (Options)	 TC-RESULT-L ind (Options) TC-RESULT-L req (Test-result)	 Operation 2 end
TC-RESULT-L ind (Test-result)		Operation 1 end
	Time	

2.3 Component-related facilities offered to TC-users

2.3.1 Invocation

So far, operations have been considered from the static point of view. Invocation introduces a dynamic aspect: a specific invocation of an operation has to be differentiated from other possible concurrent invocations of the same or of another operation.

Each particular activation of an operation is identified by an invoke ID. This invoke ID must be unambiguous. It is selected by the TC-user which originates the operation invocation, and passed to the destination TC-user, which will reflect it in its reply (or each segment of a reply) or in a linked invocation: therefore it correlates the reply to an invocation (or each segment of a reply) or a linked invocation, and the invocation itself.

The TC-user is free to assign any value to the invoke ID (index, address, . . .) provided that its value can be mapped to an integer which can be encoded in one octet according to the encoding rules specified in Recommendation Q.773. Note that such an integer takes values between –128 and 127.

The invoke ID associated with an invocation becomes reusable when the last or only segment of a result is received, or when certain abnormal situations are indicated by TC; however, the value should not be reallocated immediately for another operation activation, as immediate reallocation would prevent the correct handling of some situations (see below).

The period during which an invoke ID is released, but cannot be reallocated, is called the freezing period. This period is implementation-dependent.

As invoke IDs receive their value dynamically at the time the operation is invoked, their value cannot appear in the specification of the application protocols; rather, a "logical" value, to which a real value is substituted at execution time, should be indicated in order to identify an operation in a single flow.

Taking invoke IDs into consideration, the sequence of primitives for example E2 above becomes as shown in Table 3:

Table 3/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	TC-INVOKE ind (1, Test) TC-INVOKE req (2, 1, Option-selection, Class = 1)
TC-INVOKE ind (2, 1, Option-selection) TC-RESULT-L req (2, Options)	TC-RESULT-L ind (2, Options) TC-RESULT-L req (1, Test-result)
TC-RESULT-L ind (1, Test-result)	
Time	

where the first parameter of a primitive indicates an invoke ID. When both parameters have to be present, the second one is the linked ID. This is a pure notational convention.

2.3.2 Cancel (by the TC-user)

The TC-user requesting invocation of an operation may stop the activity associated with the corresponding invoke ID, for any reason it finds appropriate. However, cancel should in principle be reserved for abnormal situations: the normal method for terminating an operation is to receive a result or to terminate on timer expiry.

Cancelling has local effect only: it does not prevent the remote TC-user from sending replies to a cancelled operation. When received, these replies will be rejected by TC, as illustrated in the following, which represents a sequence of primitives for the example E1 defined above, where TC-user A cancels the test after receiving the first segment of the result.

In Table 4, segment P2 is not received by TC-user A: TC detects a reject situation (no active Invoke ID) and therefore does not deliver it to TC-user A, and any attempt by TC-user B to send more segments of the reply is rejected at A's side.

Table 4/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	 TC-INVOKE ind (1, Test) TC-RESULT-NL req (1, P1)
TC-RESULT-NL ind (1, P1) Cancel decision: TC-CANCEL req (1) TC-REJECT ind (1, Problem Code) ...	 TC-RESULT-NL req (1, P2) ...
Time	

2.3.3 Reject (by the TC-user)

The TC-user has the sole responsibility of deciding when to send a reject component or when to return an error (failure to perform an operation) indication. The TC-user may reject a component for any reason it finds appropriate, providing that there is a suitable reject problem code defined in the TC specifications (e.g. mandatory information element missing in an operation, error or reply, unexpected operation, unknown operation, etc.) that it can use for the purpose.

Similarly, the TC-user decides which error code and diagnostic information (which is specified as part of the TC-user protocol specifications and agreed to by the two peer TC-users for just this purpose) to use when sending an error component.

The role of the TC-user is illustrated in the following example.

The TC-user at side A expects, in a given situation, to receive operation Y only as a linked operation. When receiving from side B an Invoke component with an operation code referring to Y but no linked ID, the TC-user at side A may elect to:

– not perform the operation and return an error with some previously-determined diagnostic parameter specified in the TC-user application specification just for this purpose;

– reject the component as an "unrecognized operation".

Interpretation of the returned error diagnostic or reject problem code is the responsibility of the TC-user at side B and is not described in the TC Recommendations.

Reject of an operation invocation, or of a result, affects the whole operation: no more replies will be accepted by TC for this invocation. Reject of a linked operation does not affect the linked-to operation as far as TC is concerned. The TC-user should describe their reactions to such abnormal situations as part of their application script.

This is illustrated in Table 5 where, in example E2, TC-user A did not expect the option selection process (it may be an optional feature), and rejects the operation with the Problem Code "Unexpected Linked Operation". TC-user B may then decide to execute the test assuming a default option.

Table 5/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	TC-INVOKE ind (1, Test) TC-INVOKE req (2, 1, Option-selection, Class = 1)
TC-INVOKE ind (2, 1, Option-selection) TC-U-REJECT req (2, Problem Code)	
	TC-U-REJECT ind (2, Problem Code) TC-RESULT-L req (1, Test-result)
TC-RESULT-L ind (1, Test-result)	
Time	

When an operation invocation is rejected, the TC-user may decide to reinvoke it (e.g. the invoke component was corrupted); this would be a new invocation (new Invoke ID). It may also decide to abort the dialogue. A very simple dialogue (a question and a response) may not define any recovery mechanisms, except when the operation is of critical importance (e.g. a database update).

2.3.4 Remote cancel (by the TC-user)

TC does not provide any specific service for cancelling the remote execution of an operation in progress. The cancel service provided by the TC-U-CANCEL req primitive has only a local effect (see 2.3.2).

However, a remote cancel procedure can be defined at the TC-user level, using existing TC services. One solution for the TC-user is to include in one of the ASEs used by the application-context, an operation whose purpose is to cancel existing invocations.

The following ASN.1 module provides a description of such an operation type. This type (and the associated error type) can be imported in one of the modules used by the TC-user so that it can allocate suitable operation and error codes. Alternatively the TC-user may also design its own operation based on the same principles.

```
TCAP-Tools { ccitt recommendation q 775 modules(2) tools(1) version1(1) }

DEFINITIONS ::=
BEGIN
EXPORTS Cancel, CancelFailed, Cancelled;

IMPORTS OPERATION, ERROR, InvokeIdType
FROM TCAPMessages { ccitt recommendation q 773 modules(2) messages(1) version2(2) };

Cancel ::= OPERATION
ARGUMENT InvokeIdType
```

```
-- a TC-user may redefine this type to include
-- an empty result so that it becomes a Class 1 operation
ERRORS { CancelFailed }
-- timer = 15 s

CancelFailed ::= ERROR
PARAMETER SET {
        problem  [0] CancelProblem,
        invokeId  [1] InvokeIdType }

CancelProblem ::= ENUMERATED
{ unknownInvocation(0),
        tooLate (1),
        notCancellable (2) }
-- a TC-user may redefine this type to include application-specific problems

Cancelled ::= ERROR
-- an error of this type should be included in the error list of cancellable operations

END
```

It is necessary to include a "cancelled" error in the list of errors attached to the cancellable operations. In such a case, the TC-user which receives the cancel request will issue a TC-U-ERROR req primitive to terminate the operation. The receipt of an error component with this error code and the corresponding TC-U-ERROR ind will then terminate the operation at the invoking side.

The operation to be cancelled is identified by the invokeId which has been allocated to it at invocation time. The invocation of the Cancel operation does not affect the invocation state machine of the operation to be cancelled, because the invokeId carried in the operation argument is not visible to the component sub-layer.

If the cancellation fails, a user error is reported with three possible diagnostics:

– unknownInvocation: If the invocation has never happened, or has been forgotten;

– tooLate: If the invocation is still known but the execution is at a stage that does not permit a cancellation;

– notCancellable: The invokeId in the argument of the cancel operation corresponds to an operation which has not been agreed by the TC-users being cancellable by the invoking side.

When the cancellation is performed with success and no return error with error code indicating "cancelled" is received for the cancelled operation, the timer associated with the cancelled operation will expire, causing a TC-L-CANCEL ind primitive to be issued by TC.

Alternatively, the TC-user requesting the cancellation may decide to issue a TC-U-CANCEL req immediately after having sent the cancel request so that no further local activities exist for the operation to be cancelled.

The use of the cancel mechanism makes sense if the Cancel operation is invoked before the expiration of the timer associated with the operation to be cancelled. After this, the invoke Id may not be recognized at the performing side as the operation execution may have completed and the response primitive issued. The ability to cancel the execution of an operation after timer expiry is outside the scope of these guidelines because it would mean that the objective is not to cancel the operation execution but the subsequent actions which may have been triggered at the performing end by the invocation.

2.3.5 Reset of operation timer by the TC-user

The selection of an appropriate timer value is part of the definition of an operation. The timer supervision is started by TC when the invoke component to which it relates is sent. However, the TC-user has the ability to request TC to reset this timer at any time before it expires.

The protocol designer selects the timer value according to an estimation of the time required for the completion of the execution of the operation and the transfer of the result. However there are cases where a significant dispersion of the actual execution time is foreseen, making difficult the selection of an appropriate value.

In order to prevent that the operation timer expires while the request is still in progress, the protocol designer can choose one of the two following approaches:

a) select a timer value which is greater than estimated completion time in the worst case;

b) select a timer value which corresponds to the most likely case and arrange the TC-user in such a way that it uses the TC-RESET-TIMER service when it detects that the completion of the operation will take longer than expected. In that case, appropriate procedures need to be defined in order to ensure that the originating TC-user is informed of this situation (e.g. using linked operations).

In the following example, the TC-user A invokes several times the "query" operation to retrieve three different pieces of information from a database system which is distributed to two other locations B1 and B2. Since the TC-user A has no knowledge about how the data are distributed over these two locations, it sends all the requests to the TC-user in B1.

In the first case, the requested information is locally available in B1. In the two other cases, the requested information has to be retrieved from B2. The TC-user B1 notifies the TC-user A that the completion of the request will take more time than expected, invoking a "wait" operation. In the second case, the TC-user A accepts to wait longer than expected and requests TC to reset the operation timer. In the third case, the TC-user A prefers to abandon the operation with a user-specific procedure similar to those described in section 2.3.4. See Table 5 *bis*.

Table 5 *bis*/Q.775

TC-user A	TC-user B1	TC-user B2
First Query TC-INVOKE req (1, query, Timeout = 5 s)	TC-INVOKE ind (1, query) requested information locally available TC-RESULT-L req (1, query-result)	
TC-RESULT-L ind (1, query-result) Second Query TC-INVOKE req (2, query, Timeout = 5 s)	TC-INVOKE ind (2, query) requested information not locally available TC-INVOKE req (3, 2, wait) TC-INVOKE req (1, query)	

Table 5 *bis*/Q.775 (*concluded*)

TC-user A	TC-user B1	TC-user B2
TC-INVOKE ind (3, 2, wait) TC-RESET-TIMER req (2)		TC-INVOKE ind (1, query) TC-RESULT-L req (1, query-result)
	TC-RESULT-L ind (1, query-result) forward received result to A TC-RESULT-L req (2, query-result)	
TC-RESULT-L ind (2, query-result) Third Query TC-INVOKE req (3, query, Timeout = 5 s)	TC-INVOKE ind (3, query) requested information not locally available TC-INVOKE req (4, 3, wait) TC-INVOKE req (2, query)	
TC-INVOKE ind (4, 3, wait) TC-INVOKE req [5, cancel (3)]		TC-INVOKE ind (2, query) TC-RESULT-L req (2, query-result)
	TC-INVOKE ind [5, cancel (3)] TC-U-ERROR req (3, cancelled) TC-RESULT-L ind (1, query-result) discard received result to A	
TC-U-ERROR ind (3, cancelled)		

2.4 Component-related abnormal situations

2.4.1 Component loss

TC assumes a very low probability of message loss in the network; if this probability is too high for an application, it should use the connection-oriented network service approach. If some protocol information needs an upgraded Quality of Service (e.g. charging information), the application should introduce its own mechanisms to obtain higher reliability for this information.

Loss of an operation invocation

Table 6 illustrates the case, in example E1, where no response to the test is received before the time limit expires.

Table 6/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	
Time limit: TC-L-CANCEL ind (1)	
Time	

When a Class 1 operation is lost, the TC-user is informed when the timer associated with the operation expires. When a Class 1 operation with a single result (in a single segment) is lost, TC cannot indicate whether either the operation invocation, or the reply, was lost. If the application needs to discriminate between these two cases, it should do it in the application protocol (e.g. using the time-stamping or acknowledging the operation invocation before replying to it).

For a Class 2 operation, loss will be considered as a success (whether the invocation, or the failure report, was lost). This, considering the probability of loss, may be acceptable for non-critical operations (e.g. statistical measurements).

For a Class 3 operation, loss is treated in the same way as operation failure, whether the invocation, or the success report, has been lost.

For a Class 4 operation, loss will not be visible to TC.

Loss of a result

– Loss of a non-final result is never detected by TC.
– Loss of a final result will eventually be indicated to the TC-user when the time limit is reached, but cannot always be unambiguously interpreted as the loss of a reply; if no non-final result has been received, it may be that the invocation was lost.

Loss of a linked operation

The loss of a linked operation has the same effect as the loss of a non-linked operation. It has no effect on the linked-to operation.

Loss of a reject component

This case should be extremely infrequent, and no application should try to recover from such a situation. If the lost reject concerns an operation invocation, then when the operation times out, the TC-user who invoked the operation will consider that the invocation (or the reply) was lost, and react accordingly; if it concerns a reply, the originator of the reply will not be aware that it was incorrect: it will be up to the originator of the operation to detect the loss.

2.4.2 Component duplication

As message duplication is very infrequent in the Signalling System No. 7 network, scripts for No. 7 applications need not define sophisticated scenarios in anticipation of such situations. However, any application in which duplication would be unacceptable should either define its own duplication detection mechanism or use a connection-oriented service.

Duplicate operation invocation

When an operation invocation is duplicated (by the service provider), the destination TC-user (B) may, or may not, detect the duplication:

– TC-user B detects the duplication: the duplicate may be rejected using the problem code "duplicated invoke ID". In the case of such a rejection, this can be interpreted by the remote TC-user as rejection of the original invocation;

– TC-user B does not detect the duplication: this may happen when there is a master-slave relationship between A and B, and B executes the operation with no knowledge of the context.

Assuming the second case in example E1, a possible sequence could be as given in Table 7.

Table 7/Q.775

<table>
<tr><th>TC-user A</th><th colspan="2">TC-user B</th></tr>
<tr><td>TC-INVOKE req
(1, Test, Class = 1)</td><td rowspan="2">TC-INVOKE ind
(1, Test)
TC-INVOKE ind
(1, Test)
TC-RESULT-NL req
(1, P1)
TC-RESULT-NL req
(1, P1)

TC-RESULT-NL req
(1, P2)
TC-U-REJECT ind
(1, Problem Code)</td><td rowspan="2">Undetected
duplication of
invocation</td></tr>
<tr><td>TC-RESULT-NL ind
(1, P1)
TC-RESULT-NL ind
(1, P1)
A detects an abnormal situation and rejects:
TC-U-REJECT req
(1, Problem Code)
TC detects an abnormal situation and rejects P2:
TC-L-REJECT ind
(1, Problem Code)</td></tr>
<tr><td></td><td colspan="2">TC-R-REJECT ind
(1, Problem Code)</td></tr>
<tr><td colspan="3">Time</td></tr>
</table>

In this sequence, TC-user B considers two independent test invocations, and responds to each of them. The first result P1 is accepted; TC-user A detects that P1 is received a second time, and rejects it; this terminates the operation, and causes result P2 to be rejected when received (reject by TC). Therefore, both activities at B's side will terminate on receipt of rejects.

Duplicate non-final result

If a non-final result is duplicated, TC cannot detect it, and will deliver it twice to the TC-user. Detection of this situation is left to the application.

Duplicate final result

If a final result (RR-L) is duplicated, TC can detect the situation: the second final result is considered as abnormal (the operation has been terminated by the first "final" result), and TC rejects it.

Table 8 shows a sequence for example E1 where the third segment of the result is duplicated (by the network).

Comment: Discarding of duplicates in all cases by TC would probably appear to be a helpful feature. However, it should be noted that:

1) it would require another degree of complexity in TC, which contradicts the basic characteristics of TC in the connectionless approach;

2) it corresponds to a situation which is extremely infrequent, at least in the No. 7 network.

To cover these situations when required by an application, it would be better to use a connection-oriented network service approach, since duplication could then be detected and handled at the lower layers.

Table 8/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	
	TC-INVOKE ind (1, Test)
	TC-RESULT-NL req (1, P1)
TC-RESULT-NL ind (1, P1)	TC-RESULT-NL req (1, P2)
TC-RESULT-NL ind (1, P2)	TC-RESULT-L req (1, P3)
TC-RESULT-L ind (1, P3)	
TC receives RR-L (1, P3) Duplication of P3: TC-L-REJECT ind (1, Problem Code)	
	TC-R-REJECT ind (1, Problem Code)
Time	

2.4.3 Component missequencing

For TC, the order of segmented results is not relevant: if the order is important to the TC-user, appropriate mechanisms should be defined in the application protocol (e.g. by introducing a numbering scheme to identify intermediate replies in a parameter of these replies, or by using a connection-oriented service).

Due to missequencing, a non-final result may arrive after a final result: when this occurs the non-final result is rejected by TC. This is because the final result causes TC to close the invocation state machine associated with this operation; so when the delayed non-final result is received it cannot be associated with any active invocation state machine.

The sequence in Table 9 illustrates what happens in example E1 when the last part of the result is received before the second one: both TC-users are informed.

If a linked operation invocation is received after the final result of the linked-to operation (as a result of a missequencing), the linked operation is rejected.

TC assumes a very low probability of missequencing; if the supporting network is not satisfactory in this respect, the connection-oriented network service approach should be considered.

2.4.4 Reject of a component by TC

A general principle when TC receives a component (operation invocation or reply) which is either not formatted correctly, or received out of context (e.g. a reply without a prior operation invocation), is to reject it, which means that:

1) TC forms a Reject component to inform the originator of the faulty component and informs the local TC-user of the Reject component waiting to be sent to the remote end; TC provides whatever information is available on the nature of the component being rejected (if the dialogue has not already been terminated by the remote TC-user).

2) In reaction to this, the local TC-user may decide to abort, continue, or end the dialogue. In the last two cases, when the TC-user notifies TC of its decision, the peer TC-user is informed of the reject.

Table 9/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	 TC-INVOKE ind (1, Test) TC-RESULT-NL req (1, P1)
TC-RESULT-NL ind (1, P1) TC-RESULT-L ind (1, P3) RR-NL (1, P2) PDU arrives Missequenced result: reject (no active state machine) TC-L-REJECT ind (1, Problem Code)	 TC-RESULT-NL req (1, P2) TC-RESULT-L req (1, P3)
	TC-R-REJECT ind (1, Problem Code)
Time	

Possible cases of reject by TC have been encountered in the previous clauses. Whenever the invoke ID is recognized, rejection by TC causes the termination of the operation: a possible recovery by the TC-user is a new invocation of the terminated operation. When the rejected component is not identifiable, the local TC-user is informed of the received malformed component, and a NULL invoke ID value is included in the Reject component waiting to be sent. However, as this Reject component cannot be associated with any known invocation, abort of the dialogue may be the appropriate reaction.

2.4.5 Operation timer expiry

When TC informs the TC-user of timer expiry (TC-L-CANCEL indication), it indicates that no more information related to the operation invocation (in particular, no reject) can be received. If the peer entity still sends information in relation to this invocation, this information will be discarded when received, provided that the invoke ID of the cancelled operation has not been reallocated. Premature reallocation of invoke ID values is normally avoided by correctly setting timer values and choosing the time an invoke ID is "frozen" after it has been released. In order to compensate for uncertainties in the amount of time required to send information from TC-user to another without accounting for the absolute worst case (which is also in general the most unlikely), an implementation-dependent mechanism avoiding premature reallocation of invoke IDs is required.

Timer expiry indication corresponds to an abnormal situation only in the case of a Class 1 operation. The TC-user is then aware that either the invocation, or the reply, was lost. If no undesirable side effects arise, another invocation of the same operation can take place after timer expiry. This is illustrated by the sequence in Table 10 for example E1.

Timer expiry for a Class 2 operation indicates that no failure was received nor will be accepted for this invocation: it is a definite indication of success (for Class 2) if it is assumed that there is no possibility of message loss in the network. A parallel situation applies to Class 3 in case of failure. The indication of timer expiry for a Class 4 operation is a local decision.

Table 10/Q.775

TC-user A	TC-user B
TC-INVOKE req (1, Test, Class = 1)	TC-INVOKE ind (1, Test)
Timer expiry: TC-L-CANCEL ind (1) TC-INVOKE req (2, Test, Class = 1)	
Time	

3 Dialogues

Whenever one of the operation handling primitives considered in Clause 2 is issued, a request is passed to TC, but nothing is sent to the remote TC-user until a primitive requesting transmission is issued. These primitives, and their relation with operation handling primitives, are considered now.

3.1 Grouping of components in a message

The effect of TC-user issuing a component handling primitive (unless this primitive has local effect only), is to build a component to be included in a message. The message is not transmitted until the TC-user requests it.

Note that a component may also be generated as a consequence of a TC reject procedure: in this case this component is put in the next message for the dialogue unless this dialogue is aborted.

Provided that the maximum size of a message is not exceeded, several components can be grouped and sent to the remote end as a single message, thereby saving transmission overhead. This is done under control of the TC-user, who explicitly specifies when it wants (all) the component(s) awaiting transmission to be sent. The components awaiting transmission are those for which the TC-user has previously issued a component-handling primitive with the same dialogue ID.

Until such time when the SS No. 7 SCCP layer provides a segmenting and reassemble capability, the TC-user has to ensure that the maximum size of an SS No. 7 message will not be exceeded.

Example E3, as given in Table 11, shows the beginning of a dialogue with a network service centre where a switch requests instructions (operation 1) and receives a request to connect the call to a given destination address, and a request to send information (e.g. announcement or message to be displayed) to the calling party. Both components are contained in a single message.

TC-BEGIN and TC-CONTINUE are transmission primitives described in 3.2.

There may be one transmission primitive for each component (thus there is only one component maximum in each message), or fewer transmission primitives than components which allows the grouping of components within a message. In addition, the information contained in the parameters of the transmission primitives (e.g. addressing information) applies to all the components included in the message.

At the originating side, the primitive requesting transmission appears after the component handling primitives; this indicates that the transmission of all the preceding components has to take place immediately; it avoids indicating specific components to be transmitted with a given transmission primitive, and allows transmission primitives without any associated component.

At the destination side, the primitive indicating the reception of transmitted components appears first: it contains control information which is necessary for TC to deliver each of the components (if any) in the message; the last component of the message is indicated to the TC-user by the "Last Component" parameter. The components are delivered to the destination TC-user in the same order as they were passed to TC by the originating TC-user.

Table 11/Q.775

TC-user A	**TC-user B**
TC-INVOKE req (1, Provide-Instructions, Class = 1) TC-BEGIN req (Control parameters)	
	TC-BEGIN ind (Control parameters) TC-INVOKE ind (1, Provide-Instructions) TC-INVOKE req (2, 1, Connect-Call) TC-RESULT-L req (1, Send-Info) TC-CONTINUE req (Control parameters)
TC-CONTINUE ind (Control parameters) TC-INVOKE ind (2, 1, Connect-Call) TC-RESULT-L ind (1, Send-Info)	
Time	

3.2 Dialogue handling facilities

When two TC-users co-operate in an application, more than one operation invocation is generally required. The resulting flow of components has to be identified so that:

1) components related to the same flow can be identified;
2) flows corresponding to several instances of the same application can be identified and allowed to run in parallel.

Each such flow is called a dialogue by the TC-user and is identified by a corresponding Dialogue ID parameter. The dialogue handling facility provided for this purpose is the structured dialogue.

When only a single message is required to complete a distributed application, the Unidirectional message of the unstructured dialogue may be used. The originator does not expect a report of the outcome of the operation (i.e. may only invoke Class 4 operations), but may receive a report of a protocol error if one occurs. This report of a protocol error will also be carried in a Unidirectional message.

3.2.1 Structured dialogue

3.2.1.1 General

The use of dialogues allows several independent flows of components to coexist between two TC-users. The Dialogue ID parameter is used in both operation handling and transmission (dialogue) handling primitives to determine which component(s) pertain(s) to which dialogue.

In the following examples, the Dialogue ID parameter is represented (by convention) by the first parameter in these primitives, starting with letter D. Each TC-user has its own reference for a given dialogue. Local references (those used on the interface) are represented here; mapping of these local references onto protocol references (called Transaction IDs) included in messages is done by TC.

Three primitives have been defined for handling dialogues under normal circumstances; they indicate dialogue begin (TC-BEGIN), continuation (TC-CONTINUE) or end (TC-END). Each of these primitives may be used to request transmission of 0, 1 or several components; these components may contain information relating to one or several operations.

Table 12 illustrates a possible sequence for example E2, where the test request starts the dialogue, which ends when the test result has been sent.

Table 12/Q.775

TC-user A	TC-user B
TC-INVOKE req (D1, 1, Test, Class = 1) TC-BEGIN req (D1, Address)	
	TC-BEGIN ind (D2, Address) TC-INVOKE ind (D2, 1, Test) TC-INVOKE req (D2, 2, 1, Option-selection, Class = 1) TC-CONTINUE req (D2)
TC-CONTINUE ind (D1) TC-INVOKE ind (D1, 2, 1, Option-selection) TC-RESULT-req (D1, 2, Options) TC-CONTINUE req (D1)	
	TC-CONTINUE ind (D2) TC-RESULT-L ind (D2, 2, Options) TC-RESULT-L req (D2, 1, Test-result) TC-END req (D2)
TC-END ind (D1, normal) TC-RESULT-L ind (D1, 1, Test-result)	
Time	
NOTE – D1 and D2 are local references for the same dialogue and map onto transaction IDs which appear in the messages.	

Any grouping of components is allowed in the messages of a dialogue: TC does not check, for instance, that a message terminating a dialogue does not include operation invocations of Class 1.

Full-duplex exchange of components is assumed: if a TC-user wants to introduce some restrictions, e.g. working in a synchronous mode as defined for ROSE users, it would have to introduce the necessary procedures itself.

3.2.1.2 Exchange of messages

Transmission of messages is accomplished with the Quality of Service of the underlying layer services: no flow control or error recovery mechanisms are provided by TC.

– The first dialogue handling primitive of a dialogue must indicate dialogue begin (TC-BEGIN). Further messages must not be sent from the side originating the dialogue until a message is received in the backward direction, indicating dialogue continuation.

– If a TC-user tries to send a large number of messages in a short amount of time, no flow control mechanism in TC will prevent it.

– SCCP Class 1 in-sequence delivery can be requested as an option, indicated by the Quality of Service parameter. Note that this option may not be available end-to-end when interworking with a network which does not provide it.

3.2.1.3 Dialogue end

TC places no restriction on the ability for a TC-user to request dialogue end. It follows that messages may be lost if no precautions are taken in the application on when the dialogue may end.

In particular, if the application protocol allows both TC-users to issue TC-END primitives at about the same time, and if these primitives trigger transmission of components, it is likely that some (if not all) of these components will not be delivered to their respective destination TC-users.

It is up to the application to define, if necessary, its own rules concerning the right to end a dialogue: TC will not check them.

Any message received for a terminated dialogue is discarded if it requests dialogue end, and any message other than an End or an Abort causes the dialogue to be aborted at the remote entity.

It should be noted that a TC-user cannot reject, by means of a TC-U-REJECT request primitive, any component received in an END message. If it is important for an application to be able to reject any component received or be notified of rejections by the remote end, then all components must be placed in either the initial BEGIN message or subsequent CONTINUE messages.

The dialogue is terminated by either the pre-arranged method, or by sending an END message containing no components, or REJECT components (if applicable).

The differences between the three ways of ending a dialogue are as follows.

Prearranged end

A typical application is the access to a distributed database, where the requesting user (TC-user A) does not know where the information it seeks is located. TC-user A broadcasts a request to each location which might have the information required, and will eventually receive a response from the TC-user which holds this information. Prearranged end avoids messages from the other destinations saying: "I do not have this information". Only the responding destination may continue the dialogue (if so wished); all other destination will, by convention, end the dialogue locally; the originator of the requests will also end the dialogues with the non-responding destinations locally, when it receives the response to its request. Note that the convention is between applications: TC does not check that it is respected, nor is it indicated in the TC protocol.

Example E4 in Table 13 illustrates this situation, with two destinations B1 and B2; two dialogues (D1, D2) and (D3, D4) are started; B1 happens to own the requested information, and decides to continue the dialogue.

Prearranged end may also be used when a TC-user wants to send information, and does not expect a reply of any kind afterwards.

Table 13/Q.775

<table>
<tr><th>TC-user A</th><th>TC-user B1</th><th>TC-user B2</th></tr>
<tr><td>TC-INVOKE req
(D1, 1, Question)
TC-BEGIN req
(D1, Address)
TC-INVOKE req
(D3, 1, Question)
TC-BEGIN req
(D3, Address)</td><td>TC-BEGIN ind
(D2, Address)
TC-INVOKE ind
(D2, 1, Question)

TC-RESULT-L req
(D2, 1, Response)
TC-CONTINUE req
(D2)
......</td><td rowspan="2">TC-BEGIN ind
(D4, Address)
TC-INVOKE ind
(D4, 1, Question)
B2 does not have the information:
TC-END req
(D4, local)</td></tr>
<tr><td>TC-CONTINUE ind
(D1)
TC-RESULT-L ind
(D1, 1, Response)
D1 goes on
D3 ends locally
TC-END req
(D3, Local)</td><td></td></tr>
<tr><td colspan="3">Time</td></tr>
</table>

Basic end

When a TC-user issues the TC-END request primitive, it causes transmission of any pending components to the remote end. TC does not check that all operation invocations have received a response when dialogue end is requested: no notification is given to the TC-user that any pending operation invocations have not received a final result.

At the receiving end, the dialogue is considered terminated when all the components received within the message indicating the end have been delivered to the TC-user.

Example: the dialogue ends when the test in example E1, Table 14, receives a response.

Table 14/Q.775

TC-user A	TC-user B
......	......
	TC-RESULT-NL req
	(D2, 1, P1)
	TC-RESULT-NL req
	(D2, 1, P2)
	TC-RESULT-L req
	(D2, 1, P3)
	TC-END req
	(D2, normal)
TC-END ind	End of dialogue for B
(D1)	
TC-RESULT-NL ind	
(D1, 1, P1)	
TC-RESULT-NL ind	
(D1, 1, P2)	
TC-RESULT-L ind	
(D1, 1, P3)	
End of dialogue for A	
Time	

Abort by the TC-user

The abort facility allows the TC-user to stop the dialogue at any time. A typical case is when the user abandons the service. The main differences between this and normal ending are:

– any components for which transmission is pending are not sent to the peer entity;

– peer-to-peer TC-user information can be transmitted at the time the abort is issued, and this is delivered to the remote TC-user.

The sequence given in Table 15 shows a user abandonment in example E2.

Table 15/Q.775

TC-user A	TC-user B
TC-INVOKE req (D1, 1, Test, Class = 1) TC-BEGIN req (D1, Address)	
	TC-BEGIN ind (D2, Address) TC-INVOKE ind (D2, 1, Test) TC-INVOKE req (D2, 2, 1, Option-selection, Class = 1) TC-CONTINUE req (D2)
TC-CONTINUE ind (D1) TC-INVOKE ind (D1, 2, 1, Option-selection) TC-U-ABORT req (D1, Cause)	
	TC-U-ABORT ind (D2, Cause)
Time	

3.2.1.4 Message-related abnormal situations

These are considered independently from the effects of such events in the Component sub-layer.

Message loss

TC provides no protection against message loss. Three cases are identified:

1) the message begins a new dialogue: the dialogue will exist at the originating side only, and no message will be allowed in either direction. Eventually, an implementation-dependent mechanism at the originating end ends the dialogue;

2) the message continues an existing dialogue: loss is not detected. TC will react (or not) to the loss of included components as indicated in 2.4.1;

3) the message ends a dialogue: TC will eventually react (via a timer expiry as described in 2.4.1) if this message contained a response to a Class 1 operation: otherwise an implementation-dependent mechanism ends the dialogue at the destination end.

Message duplication

Duplication of a BEGIN message causes two transactions to be opened, as indicated below: each of these transactions has its own local ID, and the same destination ID. The TC-user eventually detects that something is wrong, and both dialogues are aborted.

The sequence given in Table 16 illustrates a duplication of the BEGIN message in Example E2.

Table 16/Q.775

TC-user A	TC-user B
TC-INVOKE req (D1, 1, Test, Class = 1) TC-BEGIN req (D1, Address)	
TC-CONTINUE ind (D1) TC-INVOKE ind (D1, 2, 1, Option-select)	TC-BEGIN ind (D2, Address) TC-INVOKE ind (D2, 1, Test) Duplicated BEGIN: TC-BEGIN ind (D3, Address) TC-INVOKE ind (D3, 1, Test) Response to the first Begin TC-INVOKE req (D2, 2, 1, Option-select, Class = 1) TC-CONTINUE req (D2) Response to the second Begin TC-INVOKE req (D3, 2, 1, Option-select, Class = 1) TC-CONTINUE req (D3)
TC-CONTINUE ind (D1) TC-INVOKE ind (D1, 2, 1, Option-select) TC-user considers that this invocation is abnormal, and may reject it, or abort one of the dialogues: TC-U-ABORT req (D1, Cause)	
	TC-U-ABORT ind (D3, Cause)
Time	

At that moment, there is still one dialogue (with local ID D2) at TC-user B's side, but no dialogue at A's side. TC-user B will receive an indication from TC when operation 2 of dialogue D2 times-out with no reply (TC-L-CANCEL ind), and may then decide to abort D2. Note that the situation would be more difficult to detect, had TC-user B not invoked a Class 1 operation.

Duplication of a CONTINUE message is not detected by TC.

When an END message is duplicated, the second message is received with an ID which does not correspond to an active dialogue: TC reacts by discarding the duplicate message.

Missequencing of messages

When the missequenced messages involve neither the beginning, nor the end of a dialogue, missequencing is not detected by TC, and may result in component missequencing, to which TC would react as indicated in 2.4.3.

When a message indicating dialogue continuation arrives after a message indicating the end of the same dialogue, it is not delivered, and causes TC to abort the dialogue; the TC-user will probably detect the loss when receiving a premature dialogue end indication. If the application needs to recover from this case, a new dialogue should be started.

Message corruption

When receiving a corrupted message, TC reacts as indicated in Recommendation Q.774.

Table 17 shows the sequence of primitives when TC decides to abort the dialogue after receiving a corrupted message in Example E2.

Table 17/Q.775

TC-user A	TC-user B
TC-INVOKE req (D1, 1, Test, Class = 1) TC-BEGIN req (D1, Address)	
	TC-BEGIN ind (D2, Address) TC-INVOKE ind (D2, 1, Test) TC-INVOKE req (D2, 2, 1, Option-select, Class = 1) TC-CONTINUE req (D2)
Corrupted message: TC-P-ABORT ind (D1, Cause)	TC-P-ABORT ind (D2, Cause)
Time	

3.2.1.5 Relations between dialogue and component handling

The following gives some guidelines on when dialogue end can be requested; if these are not respected, TC will not refuse the request for dialogue end.

The problems that may result from the collision of messages requesting dialogue end have been considered above.

Normal end should not be requested when:

– there are operation invocations pending for the dialogue;

– the application protocol anticipates that replies being transmitted with the termination request could be rejected.

Many applications might not define recovery scenarios in response to a rejected reply. This legitimizes the transmission of replies or of Class 4 operations in a message indicating dialogue end. The other applications should either use the connection-oriented network service approach, or end the dialogue with a message containing no component, that would be sent only when a reject indication can no longer be received.

It is recommended that an operation for which a reply is expected not be sent in an END message (i.e. that the TC-user does not issue a TC-END req primitive to trigger the associated component). It should be noted that this is not a requirement of the TC protocol nor, except as a sensible guideline, of the TC-user. If a TC-user chooses to send an invoke component for an operation of Class 1, 2 or 3 in an END message, it has no pathological implications for the peer TC protocol machines or the TC-users. It is a local matter on how the performing side discards any components generated as a result of performing the operation when there is no active dialogue. Indeed, the fact that the TC-user invoking the operation has chosen to include it in an END message means that it has decided for this instance to override the operation's inherent semantics and does not care to be told if the operation could be performed or not.

3.2.1.6 Addressing issues

The TC-user initiating a dialogue must give the Destination Address and Originating Address to TC. The TC-user providing the Originating Address is responsible for giving this address in such a form that the remote TC can use it, without checking, to reach it.

TC above connectionless SCCP uses any of the addressing options provided by the SCCP. So any combination of types of address for the destination and Origination addresses is allowed.

During the establishment of a dialogue the combination of addresses may be optimized by both the B-side TC-user and TC.

The optional originating address parameter in the first TC-CONTINUE request primitive can be used to change the address which should be used for routing the subsequent messages associated with the dialogue. The actual destination itself should not be changed by this modification. If a new destination is to be reached, the dialogue should be terminated and a new one started to the new destination.

Example of address changes which do not modify the actual destination are:

– Initial address was a global title, subsequent address is a PC and SSN for the same destination to allow optimal routing (generally not crossing network boundary).
– A general global title is used to select a database from a set of replicated databases. The responding database returns its own specific Global Title.

3.2.1.7 Quality of Service

The dialogue handling request primitives allow the TC-user to request a particular Quality of Service from the network layer. If no specific request is made by the TC-user, default options are selected by SCCP (i.e. no return option, no sequencing).

If sequence control is requested, the TC-user is also responsible for providing information enabling the network layer to identify a flow of related messages to be delivered in sequence.

It is recommended that sequence control be requested by TC-users when using the TC-RESULT-NL services.

When the return option is requested, the TC-user is notified of the non-delivery of the associated message, through the TC-NOTICE ind primitive.

Although the TC-NOTICE ind is primarily intended to be processed by a management function (e.g. when it indicates that a failure occurred during the translation of a global title), there are cases where the return option may be requested in a TC-BEGIN req or TC-UNI req primitive for determining whether the destination Application Entity (identified by the subsystem number in the destination address) exists at the receiving entity. In such a case, the TC-NOTICE ind might be interpreted in real time.

In the following example, the TC-user A which resides in a local exchange starts a look-ahead procedure before setting up a call. It requests the return option when issuing the TC-BEGIN req to open a dialogue towards the destination local exchange B. The look-ahead procedure is not implemented in node B. Hence, the subsystem number which corresponds to the look-ahead procedure does not exist and the SCCP in node B returns the SCCP message which contained the BEGIN message. The TC-user A is confirmed via the TC-NOTICE ind primitive and starts a basic call (circuit) establishment procedure.

Table 18/Q.775

TC-user A		TC-user B
Call processing starts TC-INVOKE req (D1, 1, LookAhead) TC-BEGIN req (D1, return option, Dest = B-Address, Orig = A-Address)		
	UNITDATA (return option, B-Address, A-Address, BEGIN)	
		SSN-B does not exist
	UNITDATA SERVICE (Return cause, A-Address, B-Address, BEGIN)	
N-NOTICE ind (D1, return cause, Dest = A-Address, Orig = B-Address)		

3.2.2 Unstructured dialogue

A Unidirectional message will contain only Class 4 operation invocations or reports of protocol errors in such invocations. Multiple components can be transmitted in a Unidirectional message provided that the maximum message size of a message is not exceeded.

3.3 Enhanced dialogue control facilities

3.3.1 Overview

As the number of signalling applications using TC grows, it will become necessary to be able to differentiate between them during an instance of communication – particularly when a number of them reside at the same location within a SS No. 7 node.

The ability to signal at the start of the dialogue which application protocol (among potentially many) is involved in the subsequent exchange of messages is provided by the optional functions and protocol of the Dialogue Portion. The optional Dialogue Portion allows the negotiation of the Application Context and, as a further option within it, the transparent transfer of user data which are not components. The latter might be used to convey, for example, initialization data, versions of user protocols, further refinement of the application context, passwords, etc.

3.3.2 Use of the Application-Context

The TC-user which begins a dialogue can propose an application-context to its peer by including an application-context-name in the TC-BEGIN req primitive. The application-context refers to the set of ASEs and the associated coordinating rules which may be required during the dialogue.

If the Application-context-name is acceptable, the responding TC-user can either decide to continue or terminate the dialogue in a normal manner. Except if it requests a prearranged termination, it shall include the same AC-name in the first (or unique) dialogue handling request primitive it uses.

If the AC-name is not acceptable, the TC-user may elect to:

i) Discard the received components and issue a TC-U-ABORT req primitive to indicate that it refuses the dialogue. It shall include an application-context-name in this primitive which is either the one received or an alternative one to be used by the dialogue-initiator to make a new attempt. TC does not provide a standard feature to enable the TC-user to propose more than one alternative AC-name. However, such a procedure can be defined outside the scope of TC using the user-information parameter (see 3.3.3).

ii) Continue the dialogue but indicate that it makes use of an alternative AC [e.g. one which does not use the ASE(s) it does not support], by including another application-context-name in the first TC-CONTINUE req primitive. The received components may or may not be discarded, depending on prearranged agreement between TC-users.

iii) Terminate the dialogue in a normal manner but indicate that the response(s) included in the END message are based on an alternative AC [e.g. one which does not use the ASE(s) it does not support], by including another application-context-name in the TC-END req primitive.

The TC-user may also provide an application-context-name when using the TC-UNI service. In such a case the AC-Name indicates to the peer TC-user how to interpret the received components.

It is important to note that the application context information conveyed in the dialogue handling APDUs is a name of the type OBJECT IDENTIFIER. Such a name is a reference to a specification (document) where the description of the application context is provided. Such a document can make references to other specifications where, for instance, the abstract syntax of some application protocol is provided. Such specifications can be provided in some formal or semi-formal notation, or plain text.

The specification of application contexts, their semantics, the assignment of an object identifier value and the dissemination of this information to all parties that wish to communicate is the process of registering an application context. In the case where application contexts are registered as a part of ISDN signalling Recommendations, an example of a typical value might be {ccitt recommendation q xxx ac-name(y)} for the yth application context described in Recommendation q.xxx.

While in principle, it is possible for the application context description to be very detailed, and to add new application contexts to cover every situation that may occur, it may be easier to maintain such specifications if the number of application contexts are kept within reasonable limits. For example, let us assume that a particular application context name refers to the combined use of ASEs A and B. In certain circumstances, it may be necessary to signal that only a subset of the capabilities of A and/or B will be used for a particular instance of communications. Instead of registering a new application context name to cover this case, the same information can be conveyed in some mutually acceptable syntax in the user information field of the AARQ and AARE APDUs.

3.3.3 Transfer of user data

TC dialogue handling primitives allow the TC-user to request TC to transfer information not related to component handling facilities (i.e. not based on the Remote Operation paradigm). This information is carried either in the user-information field of dialogue control PDUs or directly in the dialogue portion, once the dialogue is established.

A typical situation where such facility is required is at dialogue establishment to send some initialization data to the peer (refinement of the application-context, authentication data, identification of a destination sub-process within the TC-user, etc.).

In addition, this facility can be used for application context-negotiation: when the TC-user refuses a dialogue (user-abort in dialogue pending state with abort-reason = application-context-not-supported), it can insert a list of alternative application-context-name in the user data field of the TC-U-ABORT req primitive. These names are then carried as part of the user-data of the dialogue protocol data unit (ABRT). The TC-user which is at the origin of the dialogue establishment request can make a new attempt with one of these contexts.

In order to use this facility the two TC-users will have to define the syntax and semantics of the information to be conveyed, if any, in each dialogue APDU for every application context. As the ASN.1 type of this user information is EXTERNAL, the syntax of this information can be written using ASN.1 or any other user specific notation. The manner in which this information is encoded can also be user-specific. The EXTERNAL type allows the embedding of a data value from one Abstract Syntax (in this case some user-specific syntax) within another (that of the dialogue APDUs).

Recommendation X.208 defines the EXTERNAL type as follows:

```
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 }}
```

Of the three forms of reference to identify the type and encoding of the data value that is contained by the EXTERNAL construction, the TC-user must use the direct-reference. The direct-reference-name will provide the key to identifying both the abstract syntax of the data value and the encoding rules which apply to it. The indirect-reference is used to identify the Presentation Context, the use of which is not supported at present in SS No. 7 signalling. In addition to the direct-reference, the TC-user may also provide an explicit description of the data value in an informal notation through the use of the data-value-descriptor.

If the external data value is a single ASN.1 type and the Basic Encoding Rules are used to encode this value, any of the choices for the "encoding" field can be used. If the agreed-to encoding of this data value results in an integral number of octets, the encoding choice can use either the "octet-aligned" or "arbitrary" encoding. If the agreed-to encoding of this external data value does not result in an integral number of octets, then the encoding choice of "arbitrary" should be used.

As the protocol allows a SEQUENCE OF EXTERNAL to be present in the optional user information field of the dialogue control APDUs, the two TC-users are not restricted, when defining an application context, to any particular number in the sequence. (If the segmentation is not provided by the SCCP, TC-users will have to ensure that the Signalling System No. 7 message size restriction is not violated.)

3.3.4 Backward compatibility issues

The new functions and protocol described in the above subclauses are optional and the specification of their protocol procedures in Recommendations Q.771 through Q.774 are easily distinguishable and can be easily removed from procurement documents, implementation specifications and interface specifications between networks which use these Recommendations as a basis. In such a case, one is left with the TC messages defined in the 1988 Recommendations. No network is obliged to support these new features if it offers no services that require these capabilities.

A node supporting the 1988 version of TC will not understand the APDUs associated with the Application Context generated by a node conforming to the 1992 Recommendation (or any latter version) and will therefore abort the transaction using the P-ABORT cause "incorrect transaction portion". If the TC-user at a node which supports TC conforming to the 1992 Recommendation (or any latter version) receives an Abort message with the above-mentioned cause in response to a dialogue initiation using Application Context information, it must interpret it, at least in the situations where a network supports a mixture of TC implementations based on the 1992 Recommendation (or any latter version) and the 1988 Recommendations, as the result of communicating with a node that supports only the 1988 TC Recommendations and not as the result of a true syntax error (which is an extremely unlikely event). The TC-user may therefore attempt a retransmission of the message without the additional Application Context information provided, of course, that this information is not crucial to the application.

It is likely that there will be a period of time during which a network supports both TC implementations conforming to the 1988 Recommendations and to the 1992 Recommendation (or any latter version), as well as applications that will or will not require the support of the Application Context mechanism. Deployment of such capabilities is outside the scope of standards, but it should be kept in mind when deploying services if the inefficiency of transmitting the initial message twice is to be avoided.

4 Guidance for writing TC-users protocol specifications

4.1 Introduction

Recommendation Q.1400 describes how Application Service Elements (ASEs), Application-Contexts (ACs) and Application Entities (AEs) are structured and how an AE is addressed in Signalling System No. 7. This clause illustrates that architecture, considering the functional decomposition of an application, and describes how AEs, ACs, ASEs, operations and errors should be defined.

4.2 Decomposition of functionality

4.2.1 Application process and application entity

A signalling Application Process (AP) communicates through a portion of its software devoted exclusively to communications which is called the Application Entity (AE). An AE, therefore, contains all the functions needed to enable the communications between distributed APs. To summarize subclause 4.1/Q.1400, an AE type is a collection of application-specific communications protocols. The definition of the AE type is a local matter. An AE is the realization of the corresponding AE type at a physical entity.

4.2.2 Application service element

On many occasions, it is found that the communications functions for a variety of applications can be grouped into integrated sets of actions such that each such set can be used in more than one AE. Such an integrated set of actions that has the potential of being used in several AEs is called an Application Service Element (ASE). Of course, there are always some application-specific communications functions that can only be used to fulfil the communications needs of the specific application for which it has been defined.

TC provides a generic means for all signalling applications to communicate using the remote operations paradigm over a connectionless network service.

A TC-user ASE includes a collection of remote operations that, together with TC, collectively provide some overall communications protocol to a signalling application. The ASE definition also specifies which peer TC-user may invoke which operation and in which order. If either TC-user may invoke any of the operations, the ASE is said to be symmetric. The means by which operations are defined and grouped is described in 4.5.

From the perspective of a TC-user, the mechanism for obtaining the services of a TC-user ASE is the invocation of the latter's operations. Each operation provides a part of the ASE's service in an inherently asymmetric manner as it is invoked by one TC-user and executed by another remote peer. However, the TC-users are not always asymmetric (i.e. one limited to always performing operations and the other to invoking them) but may each be able to invoke or perform the same or different operations. [Indeed the service interface from the TC-user's point of view (which is the subject of further study) may appear quite different from that provided by TC. For example, the invocation of a class 1 operation may be seen by the TC-user as the invocation of a confirmed service while, from the perspective of the TC service interface, it is the consequence of two unconfirmed services, viz. TC-INVOKE and TC-RESULT.]

Some application contexts may require the use of a specific TC-user ASE involved in the establishment and release of a dialogue. Such an ASE embodies the knowledge of two specific operations (known as bind and unbind operations) which form a connection-package.

Other applications may also require the use of additional ASEs which are not based on the use of operations. The PDUs of such ASEs are conveyed in the dialogue portion.

4.2.3 Communications between peer AEs/ASEs

Communications between two AEs at different nodes are possible as long as the two AEs are made aware of their peer's address and support a common Application Context. An application context is the specification of the rules, procedures and behaviour at the external interface between two AEs during an instance of communication between them. The means to specify an application context is defined in 4.3.

For TC-based applications using the 1988 (*Blue Book*) Recommendations, which rely on the SCCP UNITDATA service, the SCCP SubSystem Number (SSN) provides the role of routing an SS No. 7 message to the AE within a physical entity which supports the precise application protocol being used and also, implicitly, the encoding being used for its messages. That is, the SSN, in addition to being a piece of addressing information, also implicitly defined the application context (roughly, the message set) and the presentation context (the encoding of the messages).

With the introduction of the protocol for the Dialogue Portion of TC, which contains the application context negotiation mechanism, there is no direct relationship between the AE and the SSN. The application context allows an explicit way of recognizing granularity at the application layer. As (see 6.1.3/Q.1400) one AE may support any number of ASEs, the Application Context identifies the appropriate set of ASEs to be used during the SS No. 7 dialogue.

These points are illustrated in the figure below.

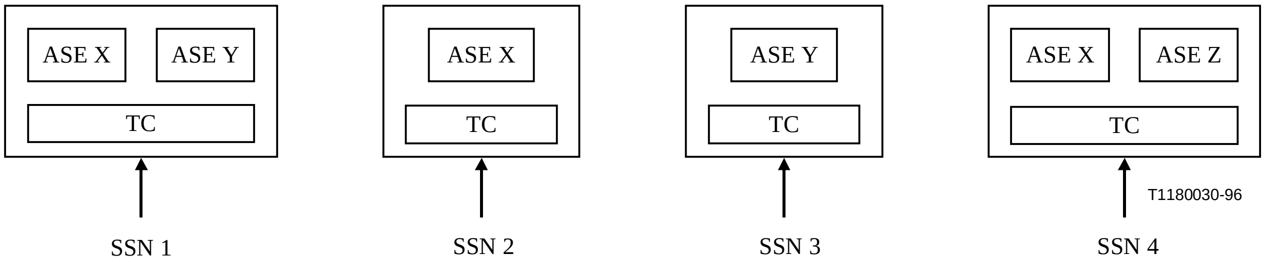

Figure A Figure B Figure C Figure D

Each outer box shows an AE type, which consists of several ASEs, one of which, in the case of interest to us, is always TC. ASEs X, Y and Z represent various TC-user ASEs, such as those for supplementary services. Each AE type is accessed by a distinct SSN.

Figures A and B show two possible local implementations of ASE X. In one case, shown in Figure A, ASE X has been placed together with ASE Y in an AE type accessed by a particular (possibly locally selected) SSN. In Figure B, ASE X is kept separate. These are two possible implementations for the services provided by ASE X and this choice is not subject to standardization. In Figure D, for instance, some other implementation has chosen to group together ASE X with a completely different ASE, ASE Z, to form an AE type accessed at SSN 4.

Note that the AE types in the above figures are different to emphasize that AE types are *not* standardized, so that each node does not have to choose the same implementation. Communications to provide the services of ASE X can still take place between two different implementations, one as in Figure A, say, and another as in Figures B or D so long as the SCCP routing data are correctly populated (which is a network management and administration issue) and as long as the operations/parameters/error codes of ASE X are distinct from those of ASEs Y and Z (which is a matter decided at specification time; see 4.5.8 for details).

The only subject of standardization is the external behaviour of a system. Application contexts define the visible external behaviour between two communicating AEs. Some ACs may be standardized. ASEs are often standardized to permit their reuse in many different application contexts.

4.3 How to specify an application context

During an instance of communication, the interactions between two AEs as well as the interactions between the ASEs within an AE are governed by the rules of an application-context (AC).

The definition of an AC should contain, at least:

– a general description;

– a definition of the complete application protocol between the peer AEs by:

  i) identifying each ASE used by the AC, and indicating which of the peer AEs initiates the service;

  ii) any coordinating rules between these ASEs (e.g. concatenation of the PDUs from different TC-user ASEs, any constraints on the order in which operations from different TC-user ASEs may be invoked, etc.) beside the rules which are an inherent part of the ASE specifications;

  iii) the abstract syntax(es) required by the ASEs;

– any special constraints to ensure that peer AEs with different versions are compatible.

A name shall be assigned to each application-context. Such a name is a value of type OBJECT IDENTIFIER which is carried (if required) as a value of the application-context-name information element in the dialogue portion.

Subclause 5.4 specifies an ASN.1 information object class which may be used for defining the static aspect of application contexts based on TC.

4.4 How to specify an ASE

A TC-user ASE specification should provide at least:

– a general description of the purpose of the ASE and its procedures;

– the list of supported operations as well as which side (or both) can invoke which operation;

– any rules on the sequence in which operations may be invoked;

– the detailed description of the procedures;

– how different protocol versions interwork;

– the description of the interactions between the ASE and TC in terms of TC service primitives;

– SDL diagrams.

The OPERATION-PACKAGE and CONNECT-PACKAGE information object classes defined in Recommendation X.880 may be used for specifying the static aspects of the definition of ASEs which are based on the use of operations.

4.5 How to specify Operations and Errors

4.5.1 General considerations

The set of Operations and Errors which forms a TC user protocol specification can be described using one or several ASN.1 modules. The number of ASN.1 modules to be used is left to the protocol designer and is further discussed in 4.5.5.

A possible notation for defining Operations and Errors is based on the ASN.1 MACRO facility defined in Recommendation X.208. The OPERATION MACRO and the ERROR MACRO are the data types respectively associated to an Operation and to an Error. However, as the MACRO notation is being phased out from ASN.1, Recommendation X.880 provides an alternative notation using two equivalent information object classes. It also explains how one can migrate from the MACRO notation to the object class notation.

Each operation (error) belongs to an operation type (error type) which is derived from the OPERATION MACRO (ERROR MACRO) type.

Each operation type or error type should be given a name (an ASN.1 type reference, which starts with a capital letter).

Each operation or error should be given a name (an ASN.1 value reference which starts with a lower case letter).

Recommendation X.219 states that values for a set of operations and errors have to be unique within an abstract syntax. For TC, this currently means that they have to be unique within the scope of a subsystem number, a group of related subsystem numbers or an application-context.

Type definition can be combined with value allocation or performed in two steps, as illustrated in the following example.

– Type specification and value assignment are separated:

```
OperationTypeExample1 ::= OPERATION
ARGUMENT        ParameterType1
RESULT          ResultType1
ERRORS          { error1, error2 }
```

operationExample1 OperationTypeExample1 ::= localValue 1

– Type specification and value assignment are combined:

```
OperationExample1 ::= OPERATION
ARGUMENT        ParameterType1
RESULT          ResultType1
ERRORS          { error1, error2 }
::= localValue 1
```

Whether it is more appropriate to combine type and by value specifications, as well as the use of global or local value, is further discussed in 4.5.6.

The notation defined in Recommendation X.880 does not support the two steps' approach. However, the code allocated to an operation or an error can be easily modified using the recode {} predefined parameterized operation. In the following example, operationExample2 is defined as being identical to operationExample1 (same operation type) but with a different value.

OperationExample2 ::= recode {operationExample1, 2}

4.5.2 Use of the OPERATION MACRO notation

4.5.2.1 Use of the type notation

An Operation type is fully defined as an instance of the OPERATION MACRO type supplemented by an ASN.1 comment indicating the associated timer value.

The following subclauses provide guidance on the use of the various ASN.1 productions which form the OPERATION MACRO description.

4.5.2.1.1 Specification of the operation argument

The following ASN.1 productions indicate how the argument of an operation has to be specified:

```
Parameter ::= ArgKeyword ParameterType | empty
ArgKeyword ::= "PARAMETER" | "ARGUMENT"
ParameterType ::= NamedType | NamedType "OPTIONAL"
```

If information can be provided at Operation invocation, the keyword PARAMETER or ARGUMENT has to be inserted and followed by the NamedType which corresponds to the data structure to be provided, otherwise the keyword shall not be present in the Operation description.

Both keywords are allowed for backward compatibility purposes with TC-user specifications based on older versions of TC. However, the use of the "PARAMETER" keyword is deprecated for defining new applications.

Although the operation argument is always an optional element of an Invoke component, the specification of the ParameterType indicates whether its presence at invocation time is mandatory or optional for the completion of the operation execution. In the latter case, the keyword "OPTIONAL" follows the ASN.1 type.

4.5.2.1.2 Specification of positive outcomes

The following ASN.1 productions indicate how to specify operations which report success:

Result ::= "RESULT" ResultType | empty
ResultType ::= ParameterType | empty

If information can be returned as a result of a successful Operation execution, the keyword RESULT is to be followed by the NamedType associated with the data structure to be sent. If no information is to be provided but the operation class indicate that the operation report success, the RESULT keyword is to be present in the Operation description but the empty alternative of the Result production is used. If the keyword RESULT is not included in an operation description, this indicates that it does not report success (i.e. Class 2 or 4 operation).

Although the operation result parameter is always an optional element of a Return Result component, the specification of the ParameterType indicates whether its presence is mandatory or optional from a functional point of view. In the latter case, the keyword "OPTIONAL" follows the ASN.1 type.

4.5.2.1.3 Associated Errors

The following ASN.1 productions indicate how to specify operations which report failure:

Errors ::= "ERRORS" "{ " ErrorNames" }" | empty
ErrorNames ::= ErrorList | empty
ErrorList ::= Error | ErrorList "," Error
Error ::= value (ERROR) | type

If the Operation reports failure, the keyword ERRORS should be included and followed by the list of associated errors, otherwise this keyword should not be present. The errors included in the list can be referenced either using a type reference or a value reference (i.e. an error code).

4.5.2.1.4 Specification of linked operations

The following ASN.1 productions indicate how to specify linked operations:

LinkedOperations ::= "LINKED" "{ "LinkedOperationNames "}" | empty
LinkedOperationNames ::= OperationList | empty
OperationList ::= Operation | OperationList "," Operation
Operation ::= value (OPERATION) | type

If the Operation is the parent operation of a set of linked-operations, the keyword LINKED should be included and followed by the list of child operations. The child operations included in the list can be referenced either using a type reference or a value reference (i.e. an operation code).

4.5.2.2 Use of the value notation

The value notation for Operation is either the notation for the value of an element of type INTEGER or the notation for an element of type OBJECT IDENTIFIER. This depends on whether the Operation is allocated a local value or a global value.

4.5.2.3 Specification of timers

The timer value associated with an operation type has to be indicated as an ASN.1 comment against the ASN.1 MACRO description of the operation type.

4.5.3 Use of the ERROR MACRO notation

The type notation for an error is the keyword ERROR optionally followed by the keyword PARAMETER and the ParameterType associated to the information which may be sent as error's parameter. The keyword PARAMETER should not be present if no information is associated with the error condition.

Although the error parameter is always an optional element of a Return Error component, the specification of the ParameterType indicates whether its presence is mandatory or optional from a functional point of view. In the latter case, the keyword "OPTIONAL" follows the ASN.1 type.

The value notation for an error is either the notation for the value of an element of type INTEGER or the notation for an element of type OBJECT IDENTIFIER. This depends on whether the Operation is allocated a local value or a global value.

4.5.4 Use of the (information object) CLASS notation

The replacement of the MACRO notation (defined in Recommendation X.209) with the (information object) CLASS notation (defined in Recommendations X.680 to X.683) retains the idea that applications have complex concepts, aspects of which need to be expressed as data structures which are to be conveyed by protocols during communications. Each of these concepts is classified by providing a template, analogous to the MACRO definition, known as an **information object class**. The template for a class shows the attributes of the objects belonging to that class. Recommendations X.680 to X.683 remove the MACRO notation and replace it with the information object CLASS definition.

4.5.4.1 The OPERATION (information object) CLASS

Recommendation X.880 defines the OPERATION (information object) CLASS. A slightly modified version is presented in ASN.1 definition below, by removing fields not relevant for TC-based applications.

The notation defines the template for a class of objects (remote operations) which are assigned the name OPERATION, which consists of ten fields. Each field starts with an ampersand (&), which is an indication that property, and is followed by a field-name beginning with either a lower-case or an upper-case letter. This distinction helps identify the sort of data that may populate the fields when defining an instance of this class.

Words which are all capitalized are used for standardized keywords such as CLASS and UNIQUE, as well as names to object classes such as OPERATION and ERROR. If a field-name starts with an upper-case letter, it is a placeholder for either an arbitrary ASN.1 type (e.g. &Argument, see ASN.1 below), a set of information objects (e.g. &Errors), or a set of values of some type. If it is a set of information objects, the descriptor of the object class to which they belong follows. If, on the other hand, the field-name starts with a lower-case letter (e.g. &returnResult), it takes the *value* of the ASN.1 type (e.g. BOOLEAN) or an information object class that follows. If a field is marked OPTIONAL, it need not be populated when defining instances of the class. The keyword UNIQUE following a field (e.g. &operationCode) means that the field is a "handle" by which instances of the class in question are identified, and should therefore be unique within any given set or collection of such objects.

Finally, designers of the ASN.1 notation have permitted a limited amount of user-defined notation through the use of the WITH SYNTAX construct which is appended to the definition of the information object class. It permits a more user-friendly notation for defining instances of a CLASS. In Recommendation X.880, the user-defined syntax for the OPERATION and ERROR CLASSes have deliberately been chosen to strongly resemble the earlier MACRO notation. Subclause 4.5.7 defines the (small) changes that are required to convert an existing operation or error definition from the MACRO definition to one employing the information object CLASS definitions.

```
OPERATION ::= CLASS
{
        &ArgumentType            OPTIONAL,
        &argumentTypeOptional    BOOLEAN OPTIONAL,
        &ResultType              OPTIONAL,
        &resultTypeOptional      BOOLEAN OPTIONAL,
        &returnResult            BOOLEAN DEFAULT TRUE,
        &Errors                  ERROR OPTIONAL,
        &Linked                  OPERATION OPTIONAL,
        &synchronous             BOOLEAN DEFAULT FALSE,
        &alwaysReturns           BOOLEAN DEFAULT TRUE,
        &operationCode  Code UNIQUE OPTIONAL
}
WITH SYNTAX
{
        [ARGUMENT                &ArgumentType [OPTIONAL
        &argumentTypeOptional]]
        [RESULT                  &ResultType [OPTIONAL
        &resultTypeOptional]]
        [RETURN RESULT           &returnResult]
        [ERRORS                          &Errors]
        [LINKED                  &Linked]
        [SYNCHRONOUS                     &synchronous]
        [ALWAYS RESPONDS         &alwaysReturns]
        [CODE                    &operationCode]
}
Code ::= CHOICE
        {
                local            INTEGER,
                global           OBJECT IDENTIFIER
        }
```

4.5.4.1.1 Specification of the operation argument

The field &Argument is an optional type field where an arbitrary ASN.1 type can be placed to define the argument of the remote operation invoked. The application designer provides the type to be used when defining a specific operation. In the user-defined syntax, the type of the accompanying argument follows the keyword ARGUMENT.

The field &argumentTypeOptional is an optional value field, which takes the value TRUE if a defined argument may optionally be absent from a functional point of view, or FALSE if it must always be present. In the user-defined syntax, the two cases are shown by the keywords OPTIONAL TRUE (respectively OPTIONAL FALSE) succeeding the type definition of the ARGUMENT.

4.5.4.1.2 Specification of the operation result

The field &ResultType is a type field where an arbitrary ASN.1 type can be placed to define the result of performing the remote operation. The application designer provides the type to be used for the result when defining a specific operation. In the user-defined syntax, the type of the accompanying result follows the keyword RESULT.

The field &resultTypeOptional is an optional value field, which takes the value TRUE if a defined result may optionally be absent from a functional point of view, or FALSE if it must always be present. In the user-defined syntax, the two cases are shown by the keywords OPTIONAL TRUE (respectively OPTIONAL FALSE) succeeding the type definition of the RESULT.

4.5.4.1.3 Specification of positive outcomes

The field &returnResult is an optional value field of type BOOLEAN which specifies whether the positive outcome of a remote operation is always reported. In the user-defined syntax, such a field is recognized by the keyword RETURN RESULT. Its absence implies that the operation always returns a result.

NOTE – This field ensures the explicit recognition of whether an operation returns upon successful completion, even if a result type is not defined for it.

4.5.4.1.4 Associated errors

The field &Errors is an optional field where a set of objects (errors) defined by the CLASS ERROR (see 4.5.5) can be placed to define the set of errors which may be returned if the remote operation fails. In the user-defined syntax, the set of errors, if present, are enclosed within braces "{...}" and follow the keyword ERRORS.

4.5.4.1.5 Specification of linked operations

The field &Linked is an optional field where a set of objects (operations) defined by the CLASS OPERATION can be placed to define the set of operations which may be linked to the particular operation being defined. In the user-defined syntax, the set of linked operations, if present, are enclosed within braces "{...}" and follow the keyword LINKED.

4.5.4.1.6 Synchronous nature of the operation

The field &synchronous is an optional value field which takes the value TRUE if the operation is synchronous (i.e. the invoker must wait for the operation to return before invoking another on the same performer) or FALSE otherwise. In the user-defined notation, the operation code follows the keyword SYNCHRONOUS.

NOTE – If an operation is synchronous, the field &alwaysReturns (see 4.5.4.1) must be TRUE.

TC-user operations are asynchronous. This field has been defined to default to FALSE (i.e. asynchronous) if this keyword is absent; so existing and future TC-user applications need not concern themselves about its presence. However, other operations, such as bind and unbind (see 5.2.2.1 and 5.2.2.2) explicitly use this field in their definitions.

4.5.4.1.7 Operation code

The field &operationCode is an optional value field which is either an integer (a "locally" unique value) or an OBJECT IDENTIFIER (a globally unique value), and must be chosen to be different from those of any other operations in a given set of operations. In the user-defined notation, the operation code follows the keyword CODE.

If an operation code value is not defined, it means that this operation cannot be invoked using the Invoke PDU. An example of such an operation is the bind operation (see 5.2.2.1) which is invoked using the bind-invoke PDU defined in 5.2.2.1.2.

4.5.4.1.8 Classes of operation

The field &alwaysReturns is a value field of type BOOLEAN which denotes whether the operation always returns or not. In the user-defined syntax, it is denoted by the true or false value accompanying the keyword ALWAYS RETURNS. If absent, it implies that the operation always returns.

If an operation always returns, it is of Class 1, or 2, or 3. The field &returnResult helps decide if an operation is of Class 1 or 3. The presence or absence of the field &Errors narrows down the previous choice to Class 1 (respectively Class 3).

4.5.4.1.9 Specification of timers

There is no notational means to specify timers for operations except as ASN.1 comments.

4.5.5 The ERROR (information object) CLASS

Recommendation X.880 defines the ERROR (information object) CLASS. A slightly modified version is presented in ASN.1 description below, by removing a field not relevant for TC-based applications.

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

4.5.5.1 Specification of the parameter accompanying an error

The field &ParameterType is an optional type field where an arbitrary ASN.1 type can be placed to define the parameter accompanying an error report for an operation invocation. The application designer provides the type to be used when defining a specific error. In the user-defined syntax, the type of the accompanying parameter follows the keyword PARAMETER.

The field &parameterTypeOptional is an optional value field, which takes the value TRUE if a defined parameter may optionally be absent from a functional point of view, or FALSE if it must always be present. In the user-defined syntax, the two cases are shown by the keywords OPTIONAL TRUE (respectively OPTIONAL FALSE) succeeding the type definition of the PARAMETER.

4.5.5.2 Error code

The field &errorCode is a value field which is either an integer (a "locally" unique value) or an OBJECT IDENTIFIER (a globally unique value), and must be chosen to be different from those of any other errors in any given set of errors. In the user-defined notation, the error code follows the keyword CODE.

If an error code value is not defined, it means that this error cannot be returned using the Return Error PDU. An example of such an error is the error indicating the ability to bind (see 5.2.2.1) which is returned using the bind-error PDU defined in 5.2.2.1.2.

4.5.6 Examples of Operations and Errors description

This subclause illustrates the part of protocol specification which deals with Operations and associated Errors definitions for a simple TC-user ASE. The purpose of Operations and Errors is briefly described in textual form. Then Operations and Errors, as well as the associated data types, are formally described in one ASN.1 module.

The following example is based on fictitious freephone like dialogue, between a switching centre and a freephone database.

4.5.6.1 Operations and Errors purposes

4.5.6.1.1 Provide routing information

This operation is invoked by a switching centre to request a remote entity to provide routing information in order to establish a call to a subscriber. The routing information provided may be a forwarded-to number and may depend on the calling party number and/or the requested basic service. In the latest case the getCallingPartyNumber child operation is invoked.

4.5.6.1.2 Get calling party number

This operation is invoked by a network element to request a switching centre to provide the calling party number associated with a call set-up request.

4.5.6.1.3 Invalid called number

This error is returned by a network element to indicate that the received called number does not comply with the supported numbering scheme.

4.5.6.1.4 Subscriber not reachable

This error is returned by a network element to indicate that there is currently no routing information available corresponding to a called number.

4.5.6.1.5 Called barred

This error is returned by a network element to indicate that a call cannot be set up because the calling number conflicts with the barring conditions attached to the called party.

4.5.6.1.6 Calling party number not available

This error is returned by a switching centre to indicate that the calling party number cannot be provided.

4.5.6.1.7 Processing failure

This error is returned by a network element to indicate a processing failure.

4.5.6.2 ASN.1 specification

The following ASN.1 module specifies the operations and associated errors and data types which correspond to the protocol elements described above. In this example type definition of Operations and Errors is combined with value assignment.

```
TCAP-Examples { ccitt recommendation q 775 modules(2) examples(2) version1(1) }
DEFINITIONS ::=
BEGIN

IMPORTS OPERATION, ERROR
FROM TCAPMessages { ccitt recommendation q 773 modules(2) messages(1) version2(2) };

provideRoutingInformation           OPERATION
ARGUMENT                            RequestArgument

RESULT                              RoutingInformation

ERRORS                              { invalidCalledNumber,
                                    subscriberNotReachable,
                                    callBarred,
                                    processingFailure }

LINKED                              { getCallingPartyAddress }
-- timer T-pi = 10 s
::= localValue : 1
```

```
getCallingPartyAddress                          OPERATION
RESULT                                          CallingPartyAddress

ERRORS                                          { callingPartyAddressNotAvailable,
                                                processingFailure }

--  timer T-gp = 5 s
::= localValue : 2

invalidCalledNumber ERROR ::= localValue : 1
subscriberNotReachable ERROR ::= localValue : 2
calledBarred ERROR ::= localValue : 3
callingPartyAddressNotAvailable ERROR ::= localValue : 4
processingFailure ERROR ::= localValue : 5
-- data types

RequestArgument  ::= SEQUENCE {
calledNumber              IsdnNumber,
basicService        BasicServiceIndicator OPTIONAL
}

RoutingInformation ::= CHOICE {
        reroutingNumber         [0] IMPLICIT IsdnNumber,
        forwardedToNumber       [1] IMPLICIT IsdnNumber }

BasicServiceIndicator  ::= ENUMERATED {
speech (0),
unrestrictedDigital (1) }

CallingPartyAddress ::= IsdnNumber

IsdnNumber  ::= SEQUENCE {
typeOfAddress             TypeOfAddress,
digits                    TelephonyString }

TypeOfAddress ::= ENUMERATED {
national (0),
international (1),
private (2) }

TelephonyString ::= IA5String  (FROM ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"|"*"|"#")) (SIZE (1..15))

END
```

4.5.7 Moving from the MACRO notation to the (information object) CLASS notation and use of ASN.1 modules

All TC-user specifications have hitherto been written using the ASN.1 MACRO notation defined in Recommendation X.208. This subclause shows how the use of these macros can be transformed into the use of the information object CLASS notation. As mentioned in 4.5.4, the user-defined syntax of the OPERATION/ERROR CLASSes have been deliberately chosen in Recommendation X.880 to allow the greatest commonality with the previous construct. From a user's point of view, there are a number of small changes, some of which are a re-positioning of various symbols while others allow for a greater indication within the notation of aspects of an operation which, in the MACRO notation, could only be expressed as comments.

In this subclause, the *provideRoutingInformation* operation and the *processingFailure* error of 4.5.6.2 are shown in both notations. In each case, on the left-hand side, which shows the macro notation, symbols underlined are to be deleted to form the new notation (the right-hand side), while symbols in ***bold italics*** are to be inserted into the old notation to form the new notation.

provideRoutingInformation OPERATION ***::=*** ***{*** **ARGUMENT RequestArgument** **RESULT RoutingInformation** **ERRORS {invalidCalledNumber** **/ subscriberNotReachable** **/ callBarred** **/ processingFailure }** **LINKED** **{ getCallingPartyAddress }** ***::= CODE*** **local : 1** ***}***	⇒	**provideRoutingInformation OPERATION** **::=** **{ ARGUMENT RequestArgument** **RESULT RoutingInformation** **ERRORS {invalidCalledNumber** **/ subscriberNotReachable** **/ callBarred** **/ processingFailure }** **LINKED** **{ getCallingPartyAddress }** **CODE local : 1** **}**
processingFailure ERROR ***::=*** ***{*** ***::= CODE*** **local : 5** ***}***	⇒	**processingFailure ERROR ::=** **{** **CODE local : 5** **}**

Use of ASN.1 modules

A module is an ASN.1 construction where a protocol designer collects several types and values definitions.

Theoretically, ASN.1 imposes no constraints upon the number of modules used to define a protocol. All the definitions may be contained in one module or many modules. However, if definitions contained in one module are required in another (e.g. the error used by an operation is defined in another module), then the corresponding definition is made available by EXPORTing it from the module in which it is defined and IMPORTing it into the module in which it is used. This applies to all ASN.1 objects, whether defined by type or value.

This gives the application designer the freedom to structure the modules according to needs, or a self-imposed convention. For example, a single module could contain all the definitions particularly in a single AE, single ASE environment. Alternatively, there could be one module for each ASE definition, each module containing all the operation and errors used exclusively by that ASE. At the other extreme, all the operations and errors could be defined in one "central registry" module and exported for use in the other modules where the ASEs are defined.

It may be necessary to employ a "mixed mode" of ASN.1 notation when defining modules. This arises from the use of the ASN.1:1994 notation defined in Recommendations X.680 to X.683 for some cases where definitions are borrowed from existing ITU-T Recommendations employing the new notation. An example of this is the use of some concepts of ROSE (Recommendation X.880) such as the static definitions of application contexts (see 5.1.1), and the use of X.500 Directory operations for particular Intelligent Network interfaces. It is important to ensure that older modules written in ASN.1:1989 do not have to be rewritten, and can coexist with modules that are written using constructs of ASN.1:1994.

Detailed guidelines for working in this "mixed mode" are provided in Recommendation X.680, but are summarized here:

- A specification may comprise modules in both ASN.1:1989 and ASN.1:1994. However, any given module MUST conform to either the 1989 or the 1994 notation, and comments may be included to indicate which version of the notation is being used in which module.

- Type and value references may be imported into an ASN.1:1994 module from an ASN.1:1989 module so long as:
  a) ASN.1:1989 MACRO definitions are NOT imported into a module using ASN.1:1994 notation; and
  b) identifiers for SET, SEQUENCE and CHOICE values are present.
- Type and value references may be imported into an ASN.1:1989 module from an ASN.1:1994 module so long as:
  a) new ASN.1:1994 types (CHARACTER STRING, UniversalString, BMPString, EMBEDDED-PDV) are NOT imported.

4.5.8 Allocation and Management of Operation and Error Codes

4.5.8.1 General considerations

Subclauses 4.1 to 4.5.4 describe how ACs, ASEs, operations and errors may be specified. Also discussed is how ASEs use operations and errors, and how the ASEs themselves are used in defining the application context between two peer AEs.

ACs and ASEs are convenient modelling and specification tools used to aid in the design of application protocols. At the end, during a dialogue between two TC-users, any TC-based application protocol is made up of the exchange of data values for operations and errors types identified by, respectively, their operation or error codes. The only requirement of Signalling System No. 7 (and ROSE) is that operation and error codes be unique within an Abstract syntax. As there is currently no way to explicitly signal the abstract syntax to which a given operation or error code belongs, the application designer has to ensure that these codes are unique within the scope a subsystem number or of an application-context. In the case where the scope of operation and error codes is an application-context, the protocol designer shall also ensure that the application-context-name is conveyed to the remote end using the protocol of the dialogue portion.

There are many possible schemes regarding the allocation and management of operation and error codes and many factors need to be considered. Two major considerations are AC/ASE structure and the reuse of operations and errors.

4.5.8.2 Import and Export of Operations and Errors

As any other ASN.1 type, Operations and Errors can be exported and imported between ASN.1 modules. This method can be used when there is a need to define an operation whose type corresponds to an existing operation but when the value to be allocated to this new operation is different from the one allocated to the existing one (i.e. for uniqueness purposes). This is illustrated by the following example, where objectIdentifier1 and objectIdentifier2 are fictitious identifiers.

```
ExportingModule { objectIdentifier1 }  DEFINITIONS ::=
BEGIN
EXPORTS operation1, OperationTypeA, error1, ErrorTypeA;

IMPORTS OPERATION, ERROR FROM TCAPMessages
{ ccitt recommendation q 773 modules(2) messages(1) version2(2) };

operation1                    OPERATION
ARGUMENT                      ParameterType1
RESULT                        ResultType1
ERRORS                        { error1 }
::= localValue 1

OperationTypeA ::=            OPERATION
```

```
ARGUMENT                    ParameterTypeA
RESULT                      ResultTypeA
ERRORS                      { ErrorTypeA}

operation2 OperationTypeA ::= localValue : 2

error1  ERROR
PARAMETER DiagnosticType1
::= localValue : 1

ErrorTypeA ::= ERROR
PARAMETER DiagnosticTypeA

error2 ErrorTypeA ::= localValue : 2
-- Note that ParameterType1, ResultType1, ParameterTypeA, ResultTypeA,
-- DiagnosticType1 and DiagnosticTypeA have to be defined somewhere if they are not defined
-- within this module, they have to be imported from the module where they are defined.

END

ImportingModule { objectIdentifier2 }  DEFINITIONS ::=
BEGIN
IMPORTS OPERATION, ERROR FROM TCAPMessages
{ ccitt recommendation q 773 modules(2) messages(1) version2(2) };
operation1, OperationTypeA, error1, ErrorTypeA
FROM ExportingModule { objectIdentifier1 };

operation2 OPERATION
ARGUMENT ParameterTypeX -- to be defined somewhere in the module
::= localValue : 2

error2 ERROR ::= localValue : 2
-- Value 2 is already in use. Thus value 3 is allocated to the imported objects
operationA OperationTypeA ::= localValue : 3
errorA ErrorTypeA ::= localValue : 3

END
```

4.5.8.3 Impact of ASE/AC structure on operation and error code administration

Regarding the AC/ASE structure the options are:

4.5.8.3.1 Monolithic approach – One AC, one ASE

Conceptually, this is the simplest approach. The application protocol is defined by an AC which comprises only one ASE (in addition to TC). All the operations used in that ASE could be defined in one ASN.1 module, which also contains the definition of the ASE's operation package and of the AC. Within the protocol, all operations and errors are identified uniquely by being assigned a unique local (integer) value.

The advantage of this scheme is its simplicity. Its disadvantage is that it does not permit to identify independent building blocks which can evolve separately within the AC definition.

4.5.8.3.2 One AC comprising more than one ASE

In defining an application protocol the designer may choose to structure an AC such that it comprises two or more ASEs. For example, it may be decided to group those elements of protocol concerned with user authentication into one separate ASE (which could be reused in another protocol), and those concerned with the actual data base enquiry into another ASE. This can facilitate modular system design but, when all the constituent ASEs are combined to form the AC, care must be taken to ensure that different operations/errors contained in different ASEs have not been allocated the same value.

Using the same operation/error in two different ASEs within the same AC does not cause a problem. If the values allocated to this same operation are the same in each case, then within the protocol there will be only one operation/error associated with that value. If different values are allocated, then although it will appear in the protocol that there are two different operations/errors, in the implementation of the application these two different values will cause the same operation/error to be invoked/identified.

However, if within the same AC, an operation defined in one ASE is allocated the same value as a different operation in another ASE, then this will obviously cause a problem. When an ASE is used in only one AC, a simple code allocation scheme can avoid this problem. But when the same ASE is used in several ACs, this can become difficult to administer, and the only "safe" approaches are those described in i) to iii) below.

i) Two or more ASE protocols share common local operation/error values:

   When the ASEs are defined, the values are allocated by the protocol designer(s) so that no clashes can occur. This requires a coordination in the tasks of defining the ASE. This means that the ASEs share the same abstract syntax.

   A disadvantage of this scheme is that if one of the ASEs is used in more than one context (i.e. together with different set of ASEs), it is more or less impossible to avoid value clashes for any combination.

ii) Assign global values (object identifiers) to operations and errors:

   Since an object identifier is unique throughout SS No. 7, there is no danger of clashes in values when an ASE is combined with any other one.

   One of the disadvantages of this scheme is that when encoded, an object identifier is longer than a simple integer.

iii) Sharing operations/errors by assigning types when defining operations/errors, instead of values:

   This approach assumes that type definition of the operations and errors have been defined independently from value assignment.

   When a protocol designer defines an application-context, it collects all the operation and error types used by the required ASEs and allocates suitable values so that no clashes occur.

   By doing this, one can consider that the protocol designer defines a new set of ASEs isomorphic to the existing ones which differ only by the values of their operations and errors.

4.5.8.4 Reuse of operation and errors

Regardless of the number of ASEs included in a protocol, there are situations where it appears suitable to include an existing operation or error when defining a new ASE.

The operation or error can be reused in one of the following manners.

The operation is imported in one of the module defining one of the ASEs. This is only feasible if it is ensured that there is no value clashes.

This can be achieved if:

i) There is a central registry of operation and errors which uses only values in a reserved range never used by ASE specific operations. This approach imposes a constraint on TC-user ASEs which may not be satisfied in a broader environment (i.e. if Operations or Errors from

DSS 1 or ISO protocols are to be used).

ii) The operation and errors have been allocated global values. The disadvantage of this approach is that a global value requires more octets to be encoded than a local one and requires official registration within the Object Identifier Tree.

iii) The operation or error type is imported in one of the modules defining one of the ASEs, where a suitable value is assigned. This assumes that the exporting protocol uses the two-step approach for the definition of operation and errors or that the required operations and error types are included in a central registry.

iv) The operation or error is completely re-defined. Although part of the original definition (e.g. the type of the argument) may be imported.

4.6 Data types specifications

4.6.1 General

As stated in the previous clause, the type of the information which may accompany an operation invocation, the report of a success or the report of a failure is specified as an ASN.1 data type. This is also valid for the information which may be exchanged as user data of the dialogue portion.

This data type can be a simple built-in type (e.g. integer type, boolean type, null type, octet string type, etc.) or structured one (e.g. sequence type, sequence-of type, choice type, etc). It may also be derived from these types by sub-typing (e.g. size constraint, value range) or tagging.

4.6.2 Use of tags

ASN.1 provides a tagging mechanism which allows for defining a type isomorphic to an existing one which thus differs only by virtue of its tag.

As clearly stated in Recommendation X.208 (ASN.1), tags are intended for machine use mainly to ease the decoding process.

Tags are not intended to be used for direct identification of information elements as they are seen from a local application process point of view. How these information elements are locally identified is an implementation matter and depends on the software design and on the language used to manipulate the internal data representation. In this respect, it should be noticed that distinct tags are mainly required in one of the following situations:

– the information elements are members of a (non-ordered) set (i.e. a set type) and therefore their relative position cannot be used to discriminate between two information elements of the same type (thus with the same tag);

– the information elements are members of an ordered set (i.e. a sequence type) but the presence or absence of optional elements makes it impossible to discriminate between the presence of an optional element and the presence of an immediately following information element of the same type;

– when two occurrences of the same base type appear in a choice type.

There are four classes of tags. In addition to the Universal class which is used to identify a built-in type, three classes are defined to enable the definition of isomorphic types for decoding purposes:

– The *APPLICATION-WIDE class* – Tags allocated in this class can be used to identify directly the structure of the data type to be decoded. Tags allocated in this class are significant across an application and shall not be used when there is a risk of clash between values. The APPLICATION-WIDE class should be used only if the application is a "closed" domain or if there is a common registry.

– The *CONTEXT-SPECIFIC class* – Tags allocated in this class are only significant in a defined domain. Therefore the decoding process identifies the data structure to be decoded both from the tag value and the context in which it appears. There is a common understanding to consider the context to be restricted to the next higher construction.

– The *PRIVATE class* which has very similar property to the APPLICATION-WIDE class but is outside the scope of standardization.

It should be noticed that the CONTEXT-SPECIFIC class is the only one (when used correctly) which ensures that there will never be any conflict between values, when data types are imported and exported between modules or protocols.

4.6.3 Instances and types

There is a need to clearly differentiate a data type from an instance of a data type (i.e. the actual information elements carried in a message or a sub-structure). For specification and readability purposes, ASN.1 provides a NamedType notation which enables to qualify a specific instance of a data type using an ASN.1 identifier.

It should be noticed that there is no need to define one data type per information element. When two information elements are syntactically equivalent it is obviously more convenient to represent them as two instances of the same data type, or if required for decoding, as instances of two types derived from the same data type by CONTEXT-SPECIFIC tagging and whose definitions will thus appear only within the higher construction definition (i.e. the tagged are only defined in the specific context of the higher construction).

4.6.4 Exporting and importing data types

TC based signalling protocols may have to make use of information elements defined in other signalling protocol specifications. Rather than defining a new information element it should be preferred to import the associated type from the specifications where it has been defined first. Data types can be imported formally or informally depending on the way the exporting protocol is specified.

– The exporting protocol is specified using an ASN.1 module which exports the required data types: these data types can be formally imported in one of the modules which define the new protocol.

– The exporting protocol is not specified using ASN.1 modules: a convenient solution is to define a data type isomorphic to the "octet string" type and to specify informally its internal structure using a reference to the specification where it is defined (i.e. using a comment statements).

4.7 How to specify abstract syntaxes

ASE and AC specifications imply that a reference is made to one or several abstract-syntaxes. Each of them represents at an abstract level (i.e. independently of the encoding techniques) sets of data values which may be exchanged during the communication.

There is currently no need for explicitly assigning a name to the abstract-syntax formed by TC messages for a given application because this abstract-syntax is implicitly identified by the subsystem number which addresses the AE. However the structure of the user information conveyed in the dialogue portion shall be defined as part or one or several other(s) abstract-syntax(es).

Thus a protocol designer which wants user-information which are not components to be conveyed by TC shall first define one or several abstract syntax(es) which encompasses all the data types whose values may be conveyed.

It shall also assign a name to each of these abstract-syntaxes. Such a name which is a value of type OBJECT IDENTIFIER will serve as a direct-reference when the actual value will be carried as part of a constructed value of type EXTERNAL, as specified in Recommendation Q.773.

There is currently no formal way to specify an abstract-syntax; however, when this syntax can be described using ASN.1 the simplest manner is to define a choice type built from all the data types which form the abstract syntax.

An abstract syntax can then be informally defined by the following sentence to be included in the protocol specifications:

"The set of data values of type Module-X. Type-A forms an abstract syntax which is identified by the following abstract-syntax-name: <objectIdentifierValue>".

Where in the preceding sentence, Type-A is the name of the choice type and Module-X is the name of the module where it is defined.

In this context, the abstract-syntax-name also implicitly refers to the encoding rules to be applied to the abstract-syntax. Such encoding rules, which may (but not need) be the one defined by Recommendation X.209, must be agreed *a priori* between the TC-users.

The ABSTRACT-SYNTAX ASN.1 information object class specified in Recommendation X.681 may also be used for defining abstract syntaxes. The following example illustrates such a use by defining an abstract syntax which encompasses the value of the InitData type which is a collection of three protocol data units used at dialogue establishment to transfer either a list of supported functional units or authentication information:

```
InitModule DEFINITIONS ::=
BEGIN

InitData ::= CHOICE {
functionalUnits    [0] IMPLICIT FunctionalUnits,
authenticationInfo  [1] IMPLICIT AuthenticationInfo }

FunctionalUnits ::= SEQUENCE OF FunctionalUnit

FunctionalUnit ::= ENUMERATED {unit(1), unit2(2), unit3(3) }

AuthenticationInfo ::= SEQUENCE {
        algorithm OBJECT IDENTIFIER,
        signature OCTET STRING }
init-abstract-syntax ABSTRACT-SYNTAX ::=
{
InitData            IDENTIFIED BY            { -- some object identifier value -- }
}

END
```

4.8 Encoding rules

The concrete syntax of TC messages (i.e. the bit stream exchanged between peer TCs as user data of SCCP messages) is derived by applying the Basic Encoding Rules to the Abstract Syntax Description of TC messages [including TC-user elements except those conveyed as value of an EXTERNAL type (e.g. user information field of a dialogue control APDU)]. The Basic Encoding Rules are described in Recommendation X.209, some minor restrictions are stated in Recommendation Q.773 for the encoding of the TC portion.

The user information conveyed as value of an EXTERNAL type may also (but not need to) be encoded according to the Basic Encoding Rules. In the latter case, the associated abstract-syntax-name serves also as an implicit reference to the applied encoding rules (see 3.3.3).

It should be noticed that the Basic Encoding Rules allows for several options, especially for the encoding of lengths. This means that an implementation must be able to decode a data unit regardless of the encoding options selected by the sending entity.

5 Mapping of the generic ROS concepts onto TC services

5.1 Overview

Recommendation X.880 defines a generic model for interactive communication between objects, where the basic interaction involves the invocation of an operation by one object (the invoker) and its performance by another (the performer). This model, known as Remote Operations (ROS), comes with a set of ASN.1 information object classes to be used by protocol designers in the specification of ROS-based applications.

Recommendation X.800 recognizes that there are multiple possible realizations of this model, as far as communication is concerned. The aim of this clause is to show how and why TC can be considered as one of these realizations, by providing a mapping of the generic concepts onto TC services.

5.1.1 Notation and concept for the generic ROS model

Recommendation X.880 defines several information object classes that are useful in the specification of ROS-based application protocols. These object classes are defined using the information object specification ASN.1 notation defined in Recommendation X.681.

The OPERATION class is used to define an operation. It is merely equivalent to the OPERATION MACRO defined in Recommendations X.219 and Q.773. This class may be used by designers of TC-user applications, as an alternative to the MACRO notation described in clause 4. Guidance for migrating from the MACRO notation to this notation is provided in Annex C/X.880.

The ERROR class is used to define an operation. It is merely equivalent to the ERROR MACRO defined in Recommendations X.219 and Q.773. This class may be used by designers of TC-user applications, as an alternative to the MACRO notation described in clause 4. Guidance for migrating from the MACRO notation to this notation is provided in Annex C/X.880.

The OPERATION-PACKAGE class is used to define a set of operations which may only be invoked by a ROS-object assuming the role of "consumer", the operations which may only be invoked by a ROS-object assuming the role of "supplier", and the operations which may be invoked by both ROS-objects. When using the communication services of SS No. 7 or OSI, an operation package is realized as an Application Service Element (ASE).

The CONNECTION-PACKAGE class is used to define the bind and unbind operations used as part of the establishment and release an association. When realized using the communication services of SS No. 7, a connection package is realized as the procedures that use the transaction capabilities structured dialogue handling services. Application-contexts which do not require the explicit invocation of bind and unbind operations can still be considered as including a connection package which uses the emptyBind and emptyUnbind predefined operations.

The CONTRACT class is used to define an association contract in terms of a connection package and one or more operation packages. When specifying the contract, the packages in which only the association initiator assumes the role of consumer, only the association responder assumes the role of consumer, and either may assume the role of consumer are identified. When using the communication services of SS No. 7 or OSI, a contract is realized as an application context.

The ROS-OBJECT-CLASS class is used to define a set of common capabilities of a set of ROS-objects in terms of the (association) contracts they support as initiators and/or responders. When realized using TC or OSI, a ROS-object maps to a portion of an application process.

These classes provide a notation which is available for the design of ROS-based applications independently of any particular realization. The actual protocol specification requires the definition of an application context which indicates how the operation contract is realized. Subclause 5.4 defines an APPLICATION-CONTEXT information object class which is available for the specification of TC-based realizations of an operation contract.

5.1.2 Communication model

The realization of ROS involves the selection of a suitable medium to convey invocations and replies between a pair of ROS-objects.

The possible media can be classified in two broad categories:

- whose required when the invoker and the performer are to be implemented in a single physical equipment;
- whose required when the invoker and the performer are to be implemented in separated physical equipments.

The first category can be further devised in message-passing and procedure calling facilities.

The medium in the second category depends on the type of network which interconnects the two objects and on some quality of service criteria.

Recommendation X.880 models the medium as being composed of two stub objects (one for the invoker, one for the performer) and one information transfer object (see Figure 1). The information transfer object capabilities also includes the association control functionalities which might be required to set up an association between the application entities involved in the communication.

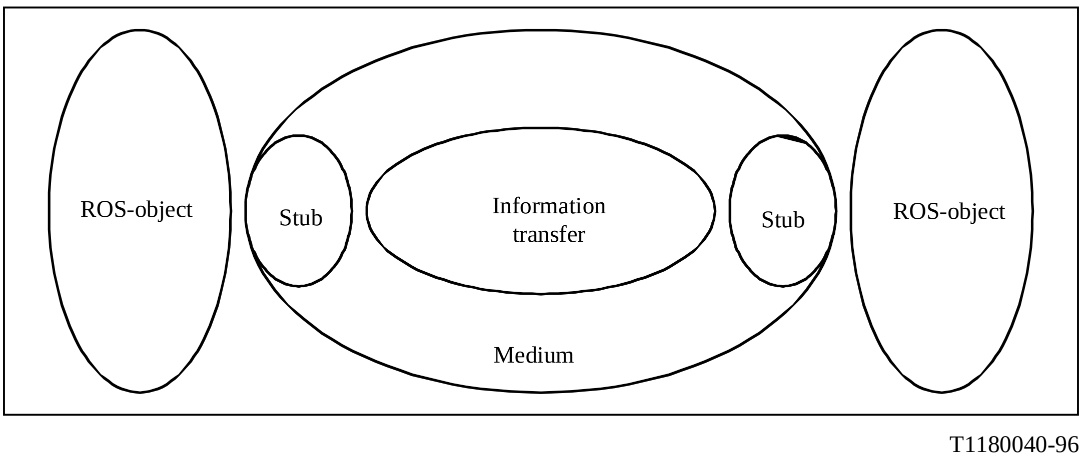

Figure 1/Q.775 – Generic ROS communication model

The role of each stub object is merely to transform invocations and replies into protocol data units (and vice versa) they exchange using the information transfer object. For a given type of stub objects, there are several possible types of information transfer objects.

In the context of OSI, the stub objects are realized by the Remote Operation Service Element (ROSE) while several information transfer realizations are available, using suitable combination of ACSE, RTSE and the presentation service.

The stub objects are realized by the Component Handling Bloc of the TC Component Sublayer (see Recommendation Q.774) together with a collection of operation specific ASEs (the TC-user ASEs). The CHA whose services are defined in 3.1.3/Q.771 drives the generic protocol required to invoke and reports returns of arbitrary operations.

Each TC-user ASE embodies knowledge of the definitions of the specific operations involved in some operation package. Collectively the CSL and the TC-user ASEs have knowledge of all the operations of the association contract. See Figure 2.

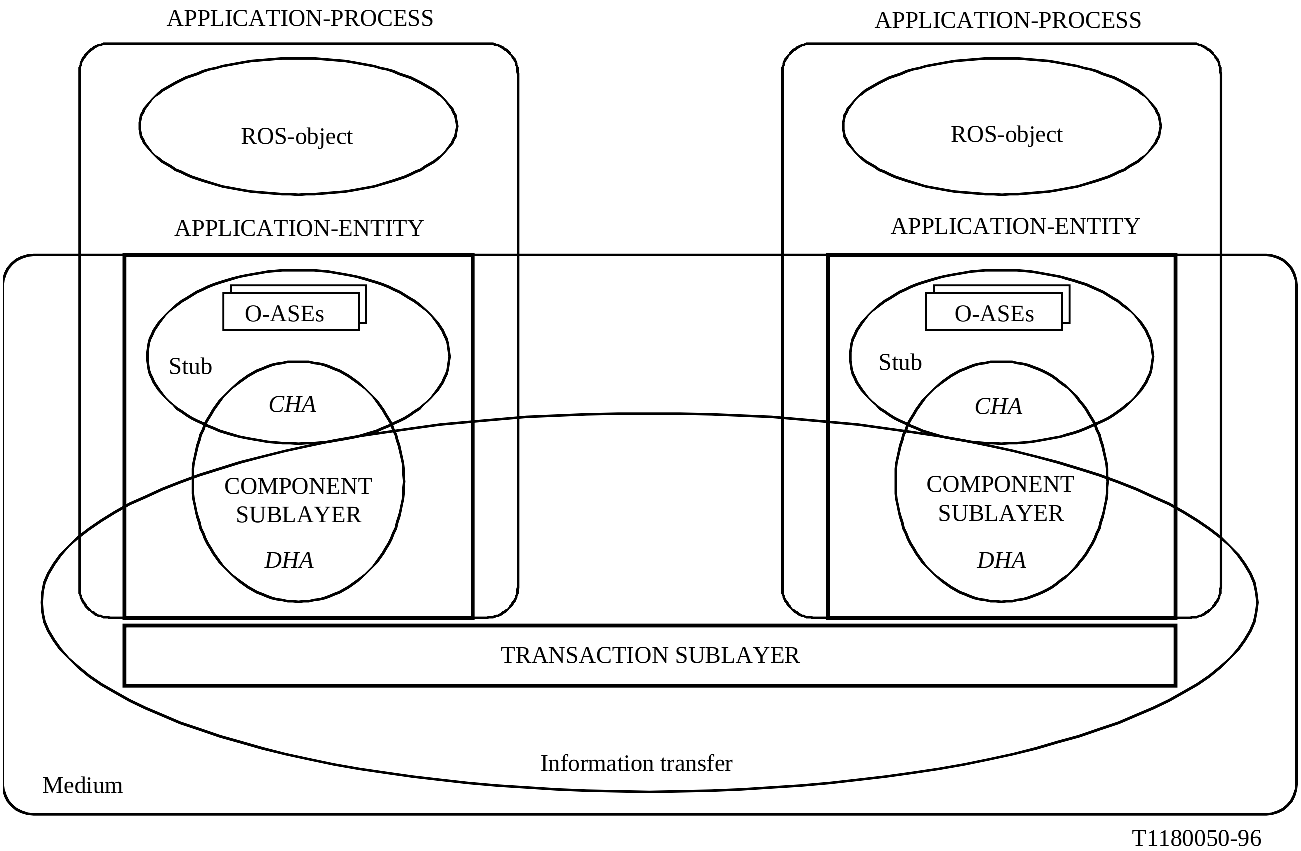

Figure 2/Q.775 – TC realization of ROS

5.2 Remote operation service realization

5.2.1 Basic services (stub)

The TC Component Sublayer provides the necessary services for supporting the invocation of operations and reporting responses. It also provides additional local services for cancelling operation (TC-U-CANCEL request, TC-L-CANCEL indication) or reporting locally-detected protocol error (TC-L-REJECT indication).

The following restrictions should be noted:

- The set of allowed InvokedIds is restricted to the integer range (–128 to 127).

- The &synchronous field of an operation definition is not taken into account. From a TC point of view, operations are always considered as being asynchronous. However, the TC-user might behave in a synchronous manner.
- The priority fields of an operation definition are not taken into account[1].

5.2.2 Bind and unbind operations

TC does not provide any specific mechanism for invoking bind and unbind operations. It is up to the TC-user to construct the bind and unbind ADPUs and transfer them to TC as any other kind of user information. As a consequence, TC is not aware that these operations are being invoked and does not check that they are used consistently with regards to the dialogue service and component handling service (e.g. it cannot verify that no operation is requested after an unbind operation has been invoked).

5.2.2.1 Bind operation

When an application-context definition includes a connection package, the initiating TC-user invokes a bind operation to be executed as part of the dialogue establishment procedure, prior to the execution of any other operation. Failure of the execution of this operation leads to the rejection of the dialogue.

If the TC-user does not really need to invoke an explicit bind operation, it is assumed that it uses the emptyBind predefined operation.

5.2.2.1.1 Invoking a bind operation

The TC-user can invoke a bind operation using the TC-BEGIN request primitive. If the definition of the bind operation includes an &ArgumentType field, the TC-user constructs a bind-invoke PDU from this information and transfers it as the first (or only) part of the user-information parameter of the TC-BEGIN request primitive. Otherwise, no bind-invoke PDU is sent.

NOTE – This should ensure that the bind-request PDU will be included in the first external field of the user-information element of the Dialogue Request APDU (AARQ).

5.2.2.1.2 Responding to a bind operation

The TC-user reports the outcome of a bind operation using the first dialogue handling primitive it issues.

Successful execution of the bind operation is reported using a TC-CONTINUE request primitive or a TC-END request primitive if there is no need to continue the dialogue. In the latter case, it shall also invoke an unbind operation.

NOTE – Use the TC-END request primitive at this stage places restriction of the use of unbind operations. It implies that only the responder can unbind and that the unbind operation definition does not include an &ResultType field and that the definition of its associated error does not include an &ParameterType field (e.g. as the emptyUnbind operation).

If the bind operation definition includes an &ResultType field, the TC-user constructs a bind-result PDU from this information and transfers it as the first (or only) part of the user-information parameter of the TC-CONTINUE request primitive or TC-END request primitive. Otherwise no bind-result PDU is sent.

---

[1] This might evolve as the studies on priority handling in SS No. 7 will progress.

The TC-user reports unsuccessful execution of a bind operation using a TC-U-ABORT request primitive issued as an immediate response to the TC-BEGIN indication. The abort reason parameter takes the value "dialogue-refused".

If the definition of the associated error includes an &ParameterType field, the TC-user constructs a bind-error PDU from this information and transfers it as the first (or only) part of the user-information parameter of the TC-CONTINUE request primitive or TC-END request primitive. Otherwise no bind-error PDU is sent.

The emptyBind operation and the bind-invoke, bind-result and bind-error PDUs are defined in Recommendation X.880. For convenience their ASN.1 definitions are reproduced below:

```
Bind {OPERATION:operation} ::= CHOICE
{
	bind-invoke		[16]	OPERATION.&ArgumentType (operation),
	bind-result		[17]	OPERATION.&ResultType (operation),
	bind-error		[18]	OPERATION.&Errors.&ParameterType (operation)
}

emptyBind OPERATION ::= {ERRORS {refuse} SYNCHRONOUS TRUE}
```

Where *operation* refers to the bind operation.

5.2.2.2 Unbind operation

5.2.2.2.1 Invoking an unbind operation

If the application-context definition includes a connection package, the TC-user invokes an unbind operation as part of the dialogue termination procedure.

The mapping onto TC services depends on the type of this unbind operation:

a) If the unbind operation definition does not include an &ResultType field and the definition of its associated error does not include an &ParameterType field, the operation can be invoked using the TC-END request primitive.

b) If the unbind operation definition includes an &ResultType field or the definition of its associated error includes an &ParameterType field, the operation must be invoked using the last TC-CONTINUE request primitive issued by the unbind requestor.

In both cases, if the unbind operation definition includes an &ArgumentType field, the TC-user constructs an unbind-request APDU which is transferred as the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-request APDU is sent.

5.2.2.2.2 Responding to an unbind operation

When accepting an unbind operation, the TC-user issues a TC-END request primitive. If the unbind operation definition includes an &ResultType field, the TC-user constructs an unbind-result APDU which is transferred in the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-result APDU is sent.

When refusing an unbind operation, the TC-user issues a TC-CONTINUE request primitive. If the definition of the associated error includes an &ParameterType field, the TC-user constructs an unbind-error APDU which is transferred in the last (or only) part of the user-information parameter of a TC-END request primitive. Otherwise no unbind-result APDU is sent.

NOTE – This should ensure that the unbind-result PDU will be included in the last external field of the user-information element of the Dialogue Response APDU (AARE) when the TC-END request primitive is issued as an immediate response to the TC-BEGIN indication primitive, or otherwise in the single EXTERNAL field of the Dialogue Portion.

If the association contract includes a connection package but the TC-user does not need to explicitly invoke an unbind operation, it is assumed that the emptyUnbind operation is used. This operation is conceptually mapped onto the TC-END request primitive, however no unbind PDU is sent.

The emptyUnbind operation, the unbind-invoke, unbind-result and unbind-error PDUs are defined in Recommendation X.880. For convenience they are reproduced below:

```
Unbind {OPERATION:operation} ::= CHOICE
{
	unbind-invoke	[19]	OPERATION.&ArgumentType (operation),
	unbind-result	[20]	OPERATION.&ResultType (operation),
	unbind-error	[21]	OPERATION.&Errors.&ParameterType (operation)
}

emptyUnbind OPERATION ::= {SYNCHRONOUS TRUE}
```

Where *operation* refers to the unbind operation.

5.3 Information transfer

5.3.1 Association realizations

TC provides two association realizations through its dialogue handling function: the structured mode and the unstructured mode which are defined in Recommendation Q.771.

5.3.2 Transfer realization

As far as Remote Operations are concerned, TC provides the following information transfer capabilities to its user:

- Bind and unbind PDUs are transferred in a user-information in the Dialogue Portion.
- Basic ROS PDUs (plus the return result not last) are transferred in the component portion of any message.

TC provides only one type of transfer realization, irrespective of the type of association realization chosen. However, from a sender's point of view, this realization offers some flexibility to the TC-users as far as PDU concatenation is concerned.

Besides Remote Operations, TC also provides means to transfer any kind of user information through the use of dialogue handling service primitives.

5.4 TC-based application context

The static aspects of a TC-based application context definition realizing some particular association contract can be described as an information object of class APPLICATION-CONTEXT, which is specified as follows:

```
APPLICATION-CONTEXT ::=			CLASS
{
	&associationContract			CONTRACT,
	&dialogueMode				DialogueMode,
	&termination				Termination OPTIONAL,
	&componentGrouping			BOOLEAN DEFAULT TRUE,
	&dialogueAndComponentGrouping		BOOLEAN DEFAULT TRUE,
	&AdditionalASEs				OBJECT IDENTIFIER OPTIONAL,
	&AbstractSyntaxes			ABSTRACT-SYNTAX,
```

```
        &applicationContextName                          OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX
{
        CONTRACT                                         &associationContract
        DIALOGUE MODE                                    &dialogueMode
        [TERMINATION                                     &termination]
        [COMPONENT GROUPING ALLOWED                      &componentGrouping]
        [DIALOGUE WITH COMPONENTS ALLOWED                &dialogueAndComponentGrouping]
        [ADDITIONAL ASEs                                 &AdditionalASEs]
        ABSTRACT SYNTAXES                                &AbstractSyntaxes
        APPLICATION CONTEXT NAME                         &applicationContextName
}
DialogueMode ::= ENUMERATED {structured (1), unstructured (2)}

Termination ::= ENUMERATED {basic (1), prearranged (2)}
```

The &associationContract field identifies the association contract which is realized by this application context.

The &dialogueMode field indicates whether this application context makes use of the dialogue structured mode facilities or the dialogue unstructured mode facilities. If the association contract definition includes a connection package, the &dialogueMode field shall indicate "structured".

The &termination field indicates whether basic or prearranged termination is used to end the dialogue. If this field is absent, the application-context definition does not place any constraint on which end method is used.

The &componentGrouping field indicates whether components might be grouped in a single message. If this field is absent, the application-context definition does not place any restrictions on this issue.

The &dialogueAndComponentGrouping field indicates whether bind and unbind PDUs can be sent in messages which also contain components. If this field is absent, the application-context definition does not place any restrictions on this issue.

The &AdditionalASEs field contains the object identifiers of the ASEs required by the application context (if any) which are not based on the use of Remote Operations.

The &AbstractSyntaxes field contains the abstract syntaxes which are required for the conveyance of information between the objects, including the PDUs for invoking and reporting on the operations in the contract.

The &applicationContextName field contains the value which shall be provided to TC to identify the application context.

5.5 Abstract syntaxes

5.5.1 Dialogue control

The &AbstractSyntaxes field of an application-context definition must include the following abstract syntax if the &dialogueMode field indicates "structured".

```
dialogue-abstract-syntax ABSTRACT-SYNTAX ::= {DialoguePDU IDENTIFIED BY dialogue-as-id}
```

The &AbstractSyntaxes field of an application-context definition must include the following abstract syntax if the &dialogueMode field indicates "unstructured".

```
uniDialogue-abstract-syntax ABSTRACT-SYNTAX ::= {UniDialoguePDU IDENTIFIED BY uniDialogue-as-id}
```

5.5.2 User-defined syntaxes

5.5.2.1 General

The &AbstractSyntaxes field of an application-context definition must include one or more abstract syntaxes to represent the TC messages (including the components) and the bind and unbind PDUs. Such abstract syntaxes must be defined by the application designer.

TC messages are defined in Recommendation Q.773 while bind and unbind PDUs are defined in Recommendation X.880.

How many abstract syntaxes are defined to support a particular application context is up to the application designer. However, the following rules shall be followed:

a) If the application context realizes an association contract which includes a connection package, the values of the data types:

   Bind{ac.&associationContract.&connection.&bind}
   Unbind{ac.&associationContract.&connection.&unbind}

   shall appear in at least one of these abstract syntaxes.

b) For each operation *op* involved in the set of operation packages used by the application context, there shall be at least one of the abstract syntaxes which include the values of the following types:

   Invoke {TCInvokeIds, OPERATION:op}
   ReturnResult {OPERATION:op}

c) For each error *err* involved in the set of operation packages used by the application context, there shall be at least one of the abstract syntaxes which include the values of the following types:

   ReturnError {ERROR:err}

d) At least one of the abstract syntaxes shall include:

   Reject

5.5.2.2 Defining the abstract syntaxes

Given an operation package, a single abstract syntax which allows the exchange of TC messages carrying invocation and reporting for all of its operations can be defined using the following data type:

TCSingleAS {OPERATION-PACKAGE: package} ::= TCMessage { {AllOperations {package}}, {AllOperations {package}} }

Or alternatively a pair of abstract syntaxes can be defined based upon the pair of types:

TCConsumerAS {OPERATION-PACKAGE: package} ::= TCMessage { {ConsumerPerforms {package}}, {ConsumerPerforms {package}} }

TCSupplierAS {OPERATION-PACKAGE: package} ::= TCMessage { {SupplierPerforms {package}}, {SupplierPerforms {package}} }

A single abstract syntax may accommodate a set of packages, provided that the operation and error codes are unique. For example, the following data type can be used as the basis of a single abstract syntax to accommodate all the operation packages involved in an association contract:

```
AllPackagesAS {APPLICATION-CONTEXT:ac} ::=
		TCSingleAS
		{
			combine
				{
					{
						ac.&associationContract.&OperationsOf
						| ac.&associationContract.&InitiatorConsumerOf
						| ac.&associationContract.&InitiatorSupplierOf
					}
					{},
					{}
				}
		}
```

An independent abstract syntax can be defined to represent values of bind and unbind PDUs based on the following type:

```
ConnectionAS {APPLICATION-CONTEXT:ac} ::= CHOICE
{
	bind		Bind{ac.&associationContract.&connection.&bind},
	unbind		Unbind{ac.&associationContract.&connection.&unbind}
}
```

The object identifier value allocated to this abstract syntax should be included in the &abstract-syntax-name multivalued field of the application-context definition. It is intended to serve as direct reference when values of these PDUs are conveyed in the user-information parameter of Dialogue Control PDUs or directly as value of the Dialogue Portion.

Additional abstract syntaxes may also be defined to represent values of the PDUs associated with non ROS-based ASEs (see the example in 4.7).

5.6 Notation extension

The following ASN.1 module contains definitions which enable the designer of a TC-user protocol to specify application contexts and abstract syntaxes as instances of appropriate information object classes.

```
TC-Notation-Extensions   {ccitt recommendation q 775 modules(2) notation-extension(4) version1(1)}

DEFINITIONS ::=

BEGIN

IMPORTS

TCMessage{} FROM TCAPMessages {ccitt recommendation q 773 modules(2) messages(1) version3(3)}

Bind{}, Unbind{} FROM Remote-Operations-Generic-ROS-PDUs {joint-iso-ccitt remote-operations(4) generic-
ROS-PDUs(6) version1(0)}

AllOperations{}, ConsumerPerforms{}, SupplierPerforms{}, combine{} FROM Remote-Operations-Useful-
Definitions {joint-iso-ccitt remote-operations(4) useful-definitions(7) version1(0)}

CONTRACT, OPERATION-PACKAGE FROM Remote-Operations-Information-Objects {joint-iso-ccitt remote-
operations(4) informationObjects(5) version1(0)}

UniDialoguePDU, uniDialogue-as-id FROM UnidialoguePDUs
	{ccitt recommendation q 773 modules(2) unidialoguePDUs(3) version1(1)}
```

```
DialoguePDU, dialogue-as-id FROM DialoguePDUs
        {ccitt recommendation q 773 modules(2) dialoguePDUs(2) version1(1)}

APPLICATION-CONTEXT ::=                                   CLASS
{
        &associationContract                              CONTRACT,
        &dialogueMode                                     DialogueMode,
        &termination                                      Termination OPTIONAL,
        &componentGrouping                                BOOLEAN DEFAULT TRUE,
        &dialogueAndComponentGrouping                     BOOLEAN DEFAULT TRUE,
        &AdditionalASEs                                   OBJECT IDENTIFIER OPTIONAL,
        &AbstractSyntaxes                                 ABSTRACT-SYNTAX,
        &applicationContextName                           OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX
{
        CONTRACT                                          &associationContract
        DIALOGUE MODE                                     &dialogueMode
        [TERMINATION                                      &termination]
        [COMPONENT GROUPING ALLOWED                       &componentGrouping]
        [DIALOGUE WITH COMPONENTS ALLOWED                 &dialogueAndComponentGrouping]
        [ADDITIONAL ASEs                                  &AdditionalASEs]
        ABSTRACT SYNTAXES                                 &AbstractSyntaxes
        APPLICATION CONTEXT NAME                          &applicationContextName
}

DialogueMode ::= ENUMERATED {structured (1), unstructured (2)}

Termination ::= ENUMERATED {basic (1), prearranged (2)}

dialogue-abstract-syntax ABSTRACT-SYNTAX ::= {DialoguePDU IDENTIFIED BY dialogue-as-id}

uniDialogue-abstract-syntax ABSTRACT-SYNTAX ::= {UniDialoguePDU IDENTIFIED BY uniDialogue-as-id

TCSingleAS {OPERATION-PACKAGE: package} ::= TCMessage { {AllOperations {package}}, {AllOperations
{package}} }

TCConsumerAS {OPERATION-PACKAGE: package} ::= TCMessage { {ConsumerPerforms {package}},
{ConsumerPerforms {package}} }

TCSupplierAS {OPERATION-PACKAGE: package} ::= TCMessage { {SupplierPerforms {package}},
{SupplierPerforms {package}} }

AllPackagesAS {APPLICATION-CONTEXT:ac} ::=
        TCSingleAS
                {
                        combine
                                {
                                        {
                                                ac.&associationContract.&OperationsOf
                                                | ac.&associationContract.&InitiatorConsumerOf
                                                | ac.&associationContract.&InitiatorSupplierOf
                                        },
                                        {},
                                        {}
                                }
                }
```

```
ConnectionAS {APPLICATION-CONTEXT:ac} ::= CHOICE
{
	bind		Bind{ac.&associationContract.&connection.&bind},
	unbind		Unbind{ac.&associationContract.&connection.&unbind}
}
END
```

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	**Switching and signalling**
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communication
Series Z	Programming languages