INTERNATIONAL TELECOMMUNICATION UNION

ITU-T Q.774

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(03/93)

SPECIFICATIONS OF SIGNALLING SYSTEM No. 7

SIGNALLING SYSTEM No. 7 – TRANSACTION CAPABILITIES PROCEDURES

ITU-T Recommendation Q.774

(Previously "CCITT Recommendation")

FOREWORD

The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the International Telecommunication Union. 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, established the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

ITU-T Recommendation Q.774 was revised by the ITU-T Study Group XI (1988-1993) and was approved by the WTSC (Helsinki, March 1-12, 1993).

NOTES

1 As a consequence of a reform process within the International Telecommunication Union (ITU), the CCITT ceased to exist as of 28 February 1993. In its place, the ITU Telecommunication Standardization Sector (ITU-T) was created as of 1 March 1993. Similarly, in this reform process, the CCIR and the IFRB have been replaced by the Radiocommunication Sector.

In order not to delay publication of this Recommendation, no change has been made in the text to references containing the acronyms "CCITT, CCIR or IFRB" or their associated entities such as Plenary Assembly, Secretariat, etc. Future editions of this Recommendation will contain the proper terminology related to the new ITU structure.

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

CONTENTS

SIGNALLING SYSTEM No. 7 – TRANSACTION CAPABILITIES PROCEDURES

(Melbourne, 1988; modified at Helsinki, 1993)

1 Introduction

Transaction Capabilities (TC) allows TC-users to exchange components via Transaction Capabilities Application Part (TCAP) messages. It also allows, as an option, the transfer of Application Context Name and user information (i.e. data which are not components) between two TC-users. Procedures described in this section specify the rules governing the information content and the exchange of TCAP messages between TC-users.

1.1 Basic guideline

To maximize flexibility in service architecture and implementation style, TCAP procedures restrict themselves to supporting the exchange of components and, as an option, Application Context Name and user information (i.e. data which are not components) between TC-users. Application specific (TC-user) procedures are not part of TCAP.

When the selection of a parameter value associated with a primitive that is required by a lower layer (sub-layer) is not relevant to that layer (sub-layer), the value is simply passed down through the primitive interface. The same assumption applies to the parameters received from a lower layer through the primitive interface which are not required for TCAP functions.

1.2 Overview

Clause 2 describes addressing rules for TC messages. Clause 3 describes transaction capabilities based on a connectionless network service.

2 Addressing

In a Signalling System No. 7 environment using a connectionless network service, TC messages will use any of the addressing options afforded by the Signalling Connection Control Part (SCCP). Assignment and use of global titles may be network and/or application specific. They are not a part of this protocol specification and are not discussed further here.

Addressing options when other network providers are used are for further study.

3 Transaction capabilities based on a connectionless network service

3.1 Sub-layering in TCAP

TCAP procedure is divided into component sub-layer procedure and transaction sub-layer procedure. The component sub-layer procedure provides a TC-user with the capability of invoking remote operations and receiving replies. The component sub-layer also receives dialogue control information and user information from a TC-user and generates dialogue control APDUs as appropriate. It uses transaction sub-layer capabilities for transaction control which is the ability to carry a sequence of components and, optionally, a dialogue portion, in transaction sub-layer messages over an end-to-end connection between two TC-users.

3.2 Component sub-layer procedures

The component sub-layer provides two kinds of procedures:

- dialogue handling;
- component handling.

3.2.1 Normal procedure

3.2.1.1 Component handling procedure

3.2.1.1.1 Mapping of TC component handling service primitives to component types

Recommendation Q.771 describes the services provided by the component sub-layer by defining the service interface between the TC-user and the component sub-layer and the interface between the component sub-layer and the transaction sub-layer. Component handling refers to the ability of the TC-user to invoke a remote procedure and receive its response. Component handling procedures map component handling service primitives onto components, which constitute the protocol data units (PDUs) of the component sub-layer. A mapping of these primitives to component sub-layer PDUs is indicated in Table 1.

TABLE 1/Q.774

Mapping of TC component handling service primitives to components

Service Primitive	Abbreviation	Component Type
TC-INVOKE	INV	Invoke (Note 1)
TC-RESULT-L	RR-L	Return Result (Last) (Note 1)
TC-U-ERROR	RE	Return Error (Note 1)
TC-U-REJECT	RJ	Reject (Note 1)
TC-R-REJECT	RJ	Reject (Note 1)
TC-L-REJECT	(Note 2)	
TC-RESULT-NL	RR-NL	Return Result (Not Last)
TC-L-CANCEL	(Note 3)	
TC-U-CANCEL	(Note 3)	

NOTES

1 X.219 and X.229 Compatible.

2 Treatment of this primitive is described in 3.2.2.2.

3 There is no component type associated with this primitive since the effect is purely local.

3.2.1.1.2 Management of invoke IDs

Invoke IDs are assigned by the invoking end at operation invocation time. A TC-user need not wait for one operation to complete before invoking another. At any point in time, a TC-user may have any number of operations in progress at a remote end (although the latter may reject an invoke component for lack of resources).

Each invoke ID value is associated with an operation invocation and its corresponding invoke state machine. Management of this invoke ID state machine takes place only at the end which invokes the operation. The other end reflects this invoke ID in its relies to the operation invocation, and does not manage a state machine for this invoke ID. Note that both ends may invoke operations in a full-duplex manner: each end manages state machines for the operations it has invoked, and is free to allocate invoke IDs independently of the other.

An invoke ID value may be reallocated when the corresponding state machine returns to idle. However, immediate reallocation could result in difficulties when certain abnormal situations arise. A released ID value (when the state machine returns of idle) should therefore not be reallocated immediately; the way this is done is implementation-dependent, and thus is not described in this Recommendation.

The two peer applications must have *a priori* knowledge of the classes and timers associated with each operation.

Component states and state transitions are described in 3.2.1.1.3.

3.2.1.1.3 Operation classes

See Table 2.

TABLE 2/Q.774

Operation Classes

Operation Class	Description
1	Reporting success or failure
2	Reporting failure only
3	Reporting success only
4	Outcome not reported

A different type of state machine is defined for each class of operation, the state transitions of which are represented by Figures 1 to 4. These state machines are described here from a protocol point of view (sent/received components), whereas they are described in Recommendation Q.771 from a service (primitives) point of view.

The states of each component state machine are defined as follows:

- *Idle* – The invoke ID value is not assigned to any pending operation.
- *Operation Sent* – The invoke ID value is assigned to an operation which has not been completed or rejected. The "operation sent" state is triggered when the invoke component is transmitted.
- *Wait for Reject* – When a component indicating the completion of an operation is received, the receiving TC-user may reject this result. The Wait for Reject state is introduced so that the invoke ID is retained for some time, thereby making the rejection possible.

State transitions are triggered by:

- a primitive received from the TC–user, causing a component to be built, and eventually sent;
- receipt of a component from the peer entity;
- a number of situations indicated on Figures 1 to 4, corresponding to the following situations:

Cancel – A timer is associated with an operation invocation. This invocation timer is started when the invoke component is passed to the transaction sub-layer. The TC-INVOKE request primitive indicates a timer value. A cancel situation occurs when the invoking TC-user decides to cancel the operation (TC-U-CANCEL request primitive) before either the final result (if any) is received, or a timeout situation occurs. On receipt of a TC-U-CANCEL request, the component sub-layer stops the timer; any further replies will not be delivered to the TC-user, and TCAP will react according to abnormal situations as described in 3.2.2.2.

End situation – When an End or Abort message is received, or when pre-arranged end is used, TCAP returns any pending operations to Idle.

Invocation timeout – A timeout situation occurs when the timer associated with an operation invocation expires: the state machine returns to Idle, with notification to the TC-user by means of a TC-L-CANCEL indication (in the case of a class 1, 2 or 3 operation). This notification indicates an abnormal situation for a class 1 operation, or gives the definite outcome of a class 2 or 3 operation for which no result has been received (normal situation).

Reject timeout – A Reject timeout situation occurs when the timer associated with the Wait for Reject state expires. If this occurs, the component sub-layer assumes that the TC-user has accepted the component.

In Figures 1 to 4, components contain either single ID values, or ordered pairs of IDs (i, y), where i is the invoke ID and y is the linked ID. The state diagrams are modelled for a single operation invocation with ID i. The value of y is not relevant to the ID i. A linked invoke operation can only be accepted if the linked to state machine is in the Operation Sent state.

Components can be received "well-formed" or "malformed". The diagrams show where this is significant. If it does matter whether the component is received "well-formed" or "malformed", then the diagram indicates "receive" only.

These figures also indicate normal and abnormal transitions. Abnormal transitions result in the abnormal procedures discussed in 3.2.2. For instance, if a class 1 operation times out, this indicates an abnormal situation. The component sub-layer will release the invoke ID and inform the application. It is now up to the application to decide whether to abort the transaction or to report an error to the peer application. In another example, receiving a return error component does not produce a valid transition for a class 3 operation. This type of error would lead the component sub-layer to form a Reject component with a problem code "return error problem – unexpected return error". It would inform the local TC-user of this with a TC-L-REJECT primitive so that the reject component could be sent (if the user so desired) with the next dialogue handling primitive issued.

Class 1 operations report failure or success. A rejection in the case of a protocol error may also occur. Upon invoking a class 1 operation, the invoking end will keep the ID i active until a "last" reply is received and can no longer be rejected. An ID may be released locally, at the option of the TC-user. An ID may also be released upon invocation timeout. This is indicated in Figure 1.

Class 2 operations report failure only. A rejection in the case of a protocol error may also occur. Upon invoking a class 2 operation, the invoking end will keep the ID i active until a reply has been received and can no longer be rejected or until a timeout[1)] cancel or end situation occurs. This is indicated in Figure 2.

Class 3 operations report success only. A rejection in the case of a protocol error may also occur. Upon invoking a class 3 operation, the invoking end will keep the ID i active until a reply has been received and can no longer be rejected or until a timeout[2)] cancel or end situation occurs. This is indicated in Figure 3.

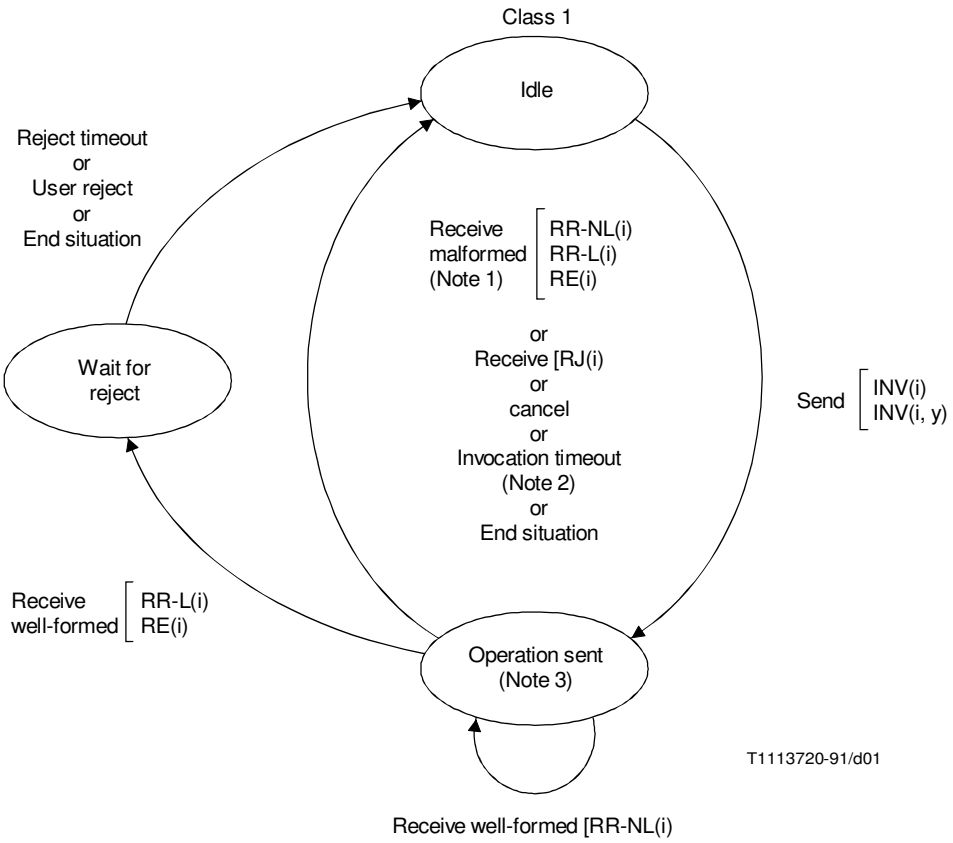
Class 4 operations do not report their outcome. A rejection in the case of a protocol error may also occur. Upon invoking a class 4 operation, the invoking end will keep the ID i active until a reject has been received or until a timeout[3)] cancel or end situation occurs. This is indicated in Figure 4.

---

1) A timeout for a class 2 operation is a "normal" situation.

2) A timeout for a class 3 operation is a "normal" situation.

3) A timeout for a class 4 operation is a "normal" situation.



NOTES

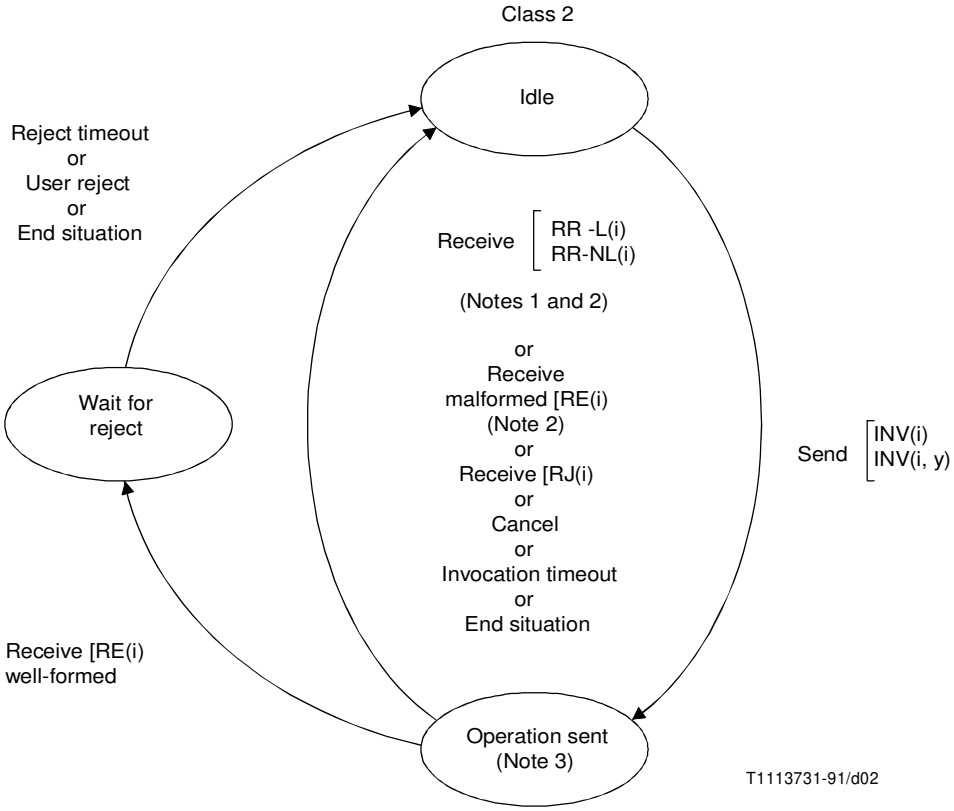
1 In these situations, the TC-user is informed and the transition occurs when the sending of the reject is initiated.

2 These are abnormal situations.

3 When a linked invoke INV(x, i) is received, the existence of the state machine i is checked to ensure it is in the Operation Sent state, but there is no impact on the state machine.

FIGURE 1/Q.774

Operation class 1



NOTES

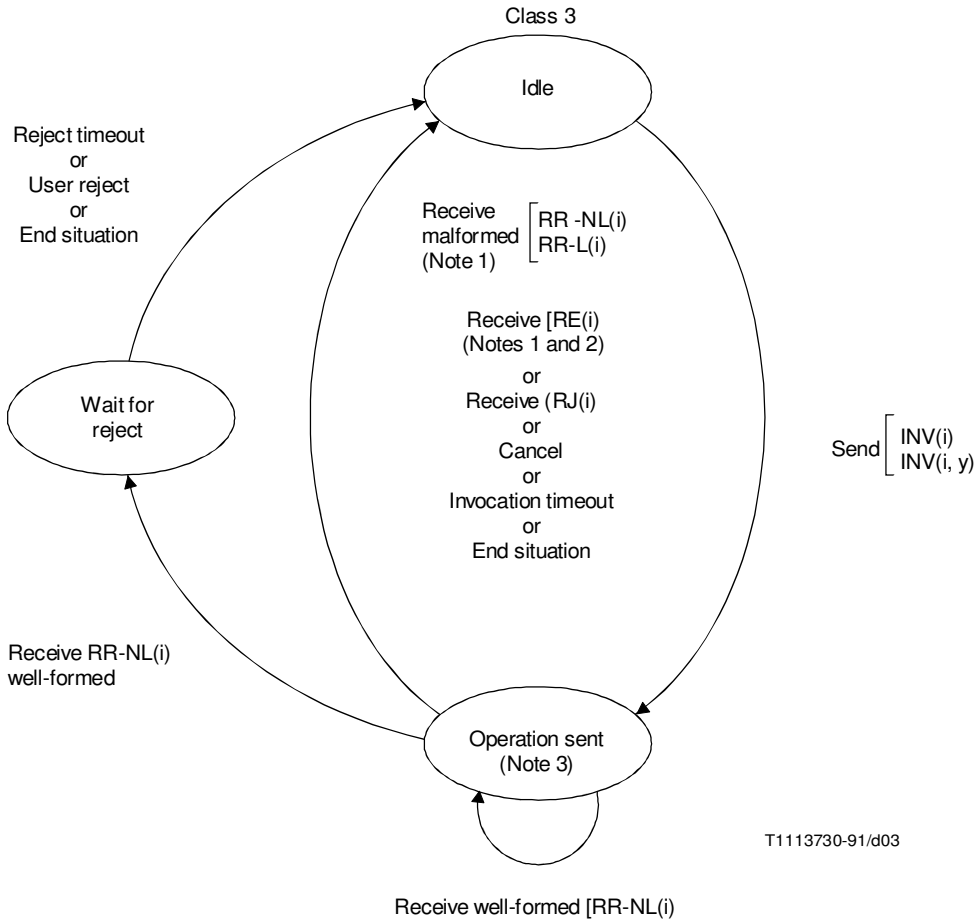
1 These are abnormal situations.

2 In these situations, the TC-user is informed and the transition occurs when the sending of the reject is initiated.

3 When a linked invoke INV(x, i) is received, the existence of the state machine i is checked to ensure it is in the Operation Sent state, but there is no impact on the state machine.

FIGURE 2/Q.774

Operation class 2



NOTES

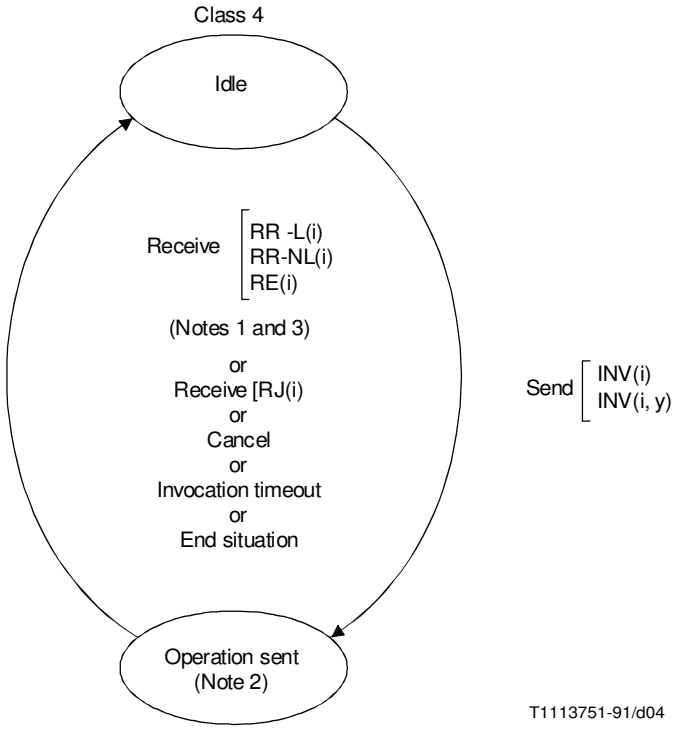
1 In these situations, the TC-user is informed and the transition occurs when the sending of the reject is initiated.

2 These are abnormal situations.

3 When a linked invoke INV(x, i) is received, the existence of the state machine i is checked to ensure it is in the Operation Sent state, but there is no impact on the state machine.

FIGURE 3/Q.774

Operation class 3



NOTES

1 These are abnormal situations.

2 When a linked invoke INV(x, i) is received, the existence of the state machine i is checked to ensure it is in the Operation Sent state, but there is no impact on the state machine.

3 In these situations, the TC-user is informed and the transition occurs when the sending of the reject is initiated.

FIGURE 4/Q.774

Operation class 4

3.2.1.1.4 Sample component flows

Some sample component flows that are compatible with Recommendation X.229 (Remote operations) are indicated in Figure 5. The flows show cases of valid component sequences correlated to an invoked operation.

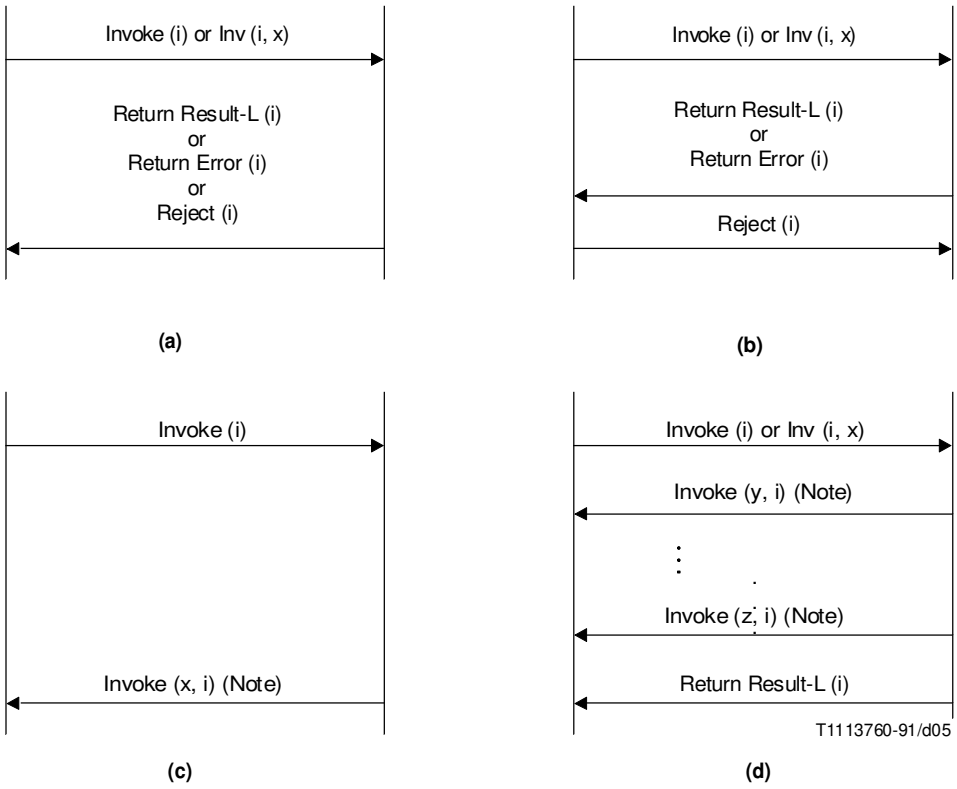
Figure 6 depicts that, as an extension to Recommendations X.219 and X.229, TCAP permits multiple return results to respond to the same Invoke operation for the purpose of segmenting a result over a connectionless network service.

3.2.1.2 Dialogue control via TC primitives

The TC-UNI, TC-BEGIN, TC-CONTINUE and TC-END request primitives are used by a TC-user to control the transfer of components.

Certain TC dialogue control request primitives can also cause a dialogue control APDU to be constructed, if an Application Context Name parameter is included in the TC-BEGIN request primitive.

A mapping of Dialogue handling primitives onto Dialogue Control APDUs is given in Table 3.



NOTE – No change to the component state machine of the original invoke

FIGURE 5/Q.774

X.229 compatible component flows

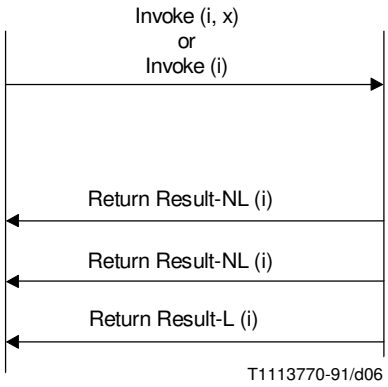


FIGURE 6/Q.774

Segmented result

TABLE 3/Q.774

Mapping of TC Dialogue Handling Service Primitives to Dialogue Control APDUs

TC Primitive (request)	Dialogue Control APDU
TC-UNI	Dialogue UNI (AUDT)
TC-BEGIN	Dialogue Request (AARQ)
TC-CONTINUE	Dialogue Response (AARE[accept]) (Note 1)
TC-END	Dialogue Response (AARE[accept)] (Note 2)
TC-U-ABORT	Dialogue Abort (ABRT) Dialogue Response (AARE[rejected]) (Note 3)

NOTES

1 This is applicable only for the first backward TC-CONTINUE primitive [i.e. when the dialogue is in the Initiation Sent/Initiation Received (IS/IR) state].

2 This is applicable only for the TC-END request primitive issued in response to a TC-BEGIN indication primitive (i.e. when the dialogue is in the IS/IR state).

3 This is applicable only before the dialogue is established (i.e. before the first backward Continue message) and only if the "abort reason" parameter in the TC-U-ABORT request primitive is set to "application-context-name-not-supported".

The dialogue control PDUs are carried in the dialogue portion of the TC message. The dialogue portion, when present, is concatenated with the component portion and transferred to the transaction sub-layer as user-data of the corresponding TR-service primitive.

Components in a message are delivered to the remote TC-user in the same order in which they are received by the originating component sub-layer from the local TC-user. The corresponding indication primitives are employed by the component sub-layer to inform the TC-user at the receiving end of the state of the dialogue.

A TC-user employs a dialogue control request primitive (TC-UNI, TC-BEGIN, TC-CONTINUE or TC-END) to trigger transmission of all previously passed components with the same dialogue identifier except for the TC-ABORT primitives which cause discard of pending components. A component sub-layer dialogue control primitive in turn triggers a corresponding service request to the transaction sub-layer, the sub-layer where the transaction control service is provided. A mapping of TC to TR transaction control primitives is provided in Table 4.

TABLE 4/Q.774

Mapping of TC Dialogue Handling Service Primitives to TR Primitives

TC Primitive	TR Primitive
TC-UNI	TR-UNI
TC-BEGIN	TR-BEGIN
TC-CONTINUE	TR-CONTINUE
TC-END	TR-END
TC-U-ABORT	TR-U-ABORT
TC-P-ABORT	TR-P-ABORT

Dialogue begin

A TC-BEGIN request primitive results in a TR-BEGIN request primitive, which begins a transaction, and transmits any (0 or more) components passed on the interface with the same dialogue ID. If the Application Context Name parameter has been included in the TC-BEGIN request primitive, a Dialogue Request (AARQ) APDU is also sent concatenated with the Component Portion.

The Destination Address and Origination Address provided in the TC-BEGIN request primitive are stored by the transaction sub-layer before sending the Begin message.

At the destination end, a TR-BEGIN indication primitive is received by the component sub-layer. It causes a TC-BEGIN indication primitive together with dialogue information, if any, starting a dialogue to be delivered to the TC-user, followed by component handling primitives associated with each of the components received (if any).

The receiving transaction sub-layer stores the received Originating Address as the destination address for this transaction and its own address as the Originating Address (from memory or from the destination address of the received N-UNITDATA indication primitive). The TC-user receives a TC-BEGIN indication primitive containing the received Destination Address and Originating Address.

Dialogue confirmation

If the TC-user has received an Application Context Name parameter in the TC-BEGIN indication primitive, and this Application Context is acceptable, the TC-user should include the same value in the first backward TC-CONTINUE request primitive. This causes a Dialogue Response (AARE) APDU to be sent concatenated with any components in a Continue message.

If the proposed Application Context Name is not acceptable, the TC-user may still wish to continue the dialogue but offer a different Application Context Name in the first backward TC-CONTINUE request primitive. This causes a Dialogue Response (AARE) APDU to be concatenated with any components in a Continue message.

In both these cases, the “result” field of the AARE APDU is set to “accepted” while the “result-source-diagnostic” field is set to either “dialogue-service-user (null)” or “dialogue-service-user (no reason given)”. The choice of one or the other value is implementation-dependent and are semantically equivalent as far as this Recommendation is concerned.

The responding TC-user issues a TC-CONTINUE request primitive with the possibility of optionally including an Originating Address parameter which is used only if the TC-user decides to change its own address (Originating Address at the B-side).

The transaction sub-layer stores the new Originating Address and sends a Continue message to the initiating TC-user.

The transaction sub-layer at the end initiating the transaction receives the Continue message and, as this is the first backward message, stores the Originating Address in the received N-UNITDATA indication primitive as the Destination Address for the transaction. The Originating Address stored for this transaction remains unchanged. These addresses will be used for all subsequent messages for this transaction and remain unchanged during the lifetime of the transaction.

Dialogue continuation

A TC-CONTINUE request primitive results in a TR-CONTINUE request primitive which transmits any components passed on the interface with the same dialogue ID. If Reject components (see 3.2.2.2) have been built by the Component sub-layer for this dialogue, they are also transmitted.

At the destination end, a TR-CONTINUE indication received by the component sub-layer causes a TC-CONTINUE to be delivered to the TC-user, followed by component handling primitives associated with each of the components received.

No dialogue control APDUs are exchanged during this stage of the transaction. If dialogue control APDUs were exchanged during the establishment of the dialogue, the Application Context Name sent in the AARE APDU is assumed to be the application context in place between the TC-users for the life-time of the dialogue. TC does not verify this except in that it treats the presence of a dialogue control APDU during this stage of the dialogue as an abnormal event. During this phase, a dialogue portion with a user-defined abstract syntax may optionally be present.

Dialogue end

If the TC-user issues a TC-END request primitive for the basic end of the dialogue in immediate response to a TC-BEGIN indication primitive containing an Application Context Name, it causes a Dialogue Response (AARE) APDU to be formed with the "result" field set to "accepted" while the "result-source-diagnostic" field is set to either "dialogue-service-user (null)" or "dialogue-service-user (no reason given)". The choice of one or the other value for the "result-source-diagnostic" field is implementation-dependent and are semantically equivalent as far as this Recommendation is concerned. The AARE APDU concatenated with any components is passed for transmission in a TR-END request primitive.

The TC-END indication primitive causes the dialogue state machine to be idled.

In the case of basic end of a dialogue, any components passed on the interface plus any reject components built by the component sub-layer for this dialogue are passed for transmission to the transaction sub-layer in a TR-END request primitive, then the dialogue is ended.

At the destination end, a dialogue ends when each component (if any) accompanying the TR-END indication primitive have been delivered to the TC-user by an appropriate component handling primitive following the TC-END indication.

The component sub-layer does not check, when a TC-user requests the end of a dialogue, that all the component state machines associated with this dialogue have returned to Idle. Similarly, no check is made by the component sub-layer that all the state machines associated with a dialogue have returned to Idle when it has delivered the components accompanying a TR-END indication primitive. In an end situation, any non-idle-state machine is returned to Idle when the TR-END request primitive is passed to the transaction sub-layer (at the originating side), or when all accompanying components have been delivered to the TC-user at the destination side; any components pending transmission are discarded.

When a TC-user has received a TC-BEGIN indication primitive with an Application Context Name parameter that it finds unacceptable, and does not wish to continue the dialogue, it issues a TC-U-ABORT request primitive. In the TC-U-ABORT request primitive, the setting of the parameter "Abort reason" to the value "application-context-name-not-supported" causes a Dialogue Response (AARE) APDU to be formatted. The setting of the values for various fields in the AARE APDU are as follows: the result field is set to "reject permanent", and the "result-source-diagnostic" is "dialogue-service-user (application-context-name-not-supported)". This APDU is sent concatenated along with any components in the user data field of the TR-U-ABORT request primitive.

If the "Abort reason" parameter in the TC-U-ABORT request primitive is absent or has a value other than "application-context-name-not-supported," this describes an abnormal termination of the dialogue and is described in 3.2.2.

Prearranged end and TC-user abort of a dialogue do not trigger transmission of pending components. All state machines associated with the dialogue are returned to Idle, and the components are discarded.

3.2.2 Abnormal procedures

3.2.2.1 Dialogue control

Any abnormal situation detected by the component sub-layer that is due to a malformed or inopportunely received component results in the rejection of the component, and in notification to the local TC-user. Abort of a dialogue is always the reflection of a decision by

- The component sub-layer in the case when an incorrect dialogue portion is received, i.e. syntactically incorrect or inconsistent with the state of the underlying transaction. The latter case corresponds to a situation when a dialogue portion is missing when its presence is mandatory (e.g. an AARQ APDU was sent in a Begin message, but no AARE APDU was received in the first backward Continue message) or when a dialogue portion is received inopportunely (e.g. a dialogue APDU is received during the active state of a transaction). At the side where the abnormality is detected, a TC-P-ABORT indication primitive is issued to the local TC-user with the "P-Abort" parameter in the primitive set to "abnormal dialogue". At the same time, a TR-U-ABORT request primitive is issued to the transaction sub-layer with an ABRT APDU as user data. The abort-source field of the ABRT APDU is set to "dialogue-service-provider" and the user information field is absent. At the receiving side, a TC-P-ABORT indication is issued by the component sub-layer on the receipt of an ABRT APDU as user data in the TR-U-ABORT indication primitive with the abort-source field of the ABRT APDU set to "dialogue-service-provider".

If any components are received in the message with an incorrect dialogue portion, these are discarded.

– The transaction sub-layer to abort the underlying transaction. The component sub-layer idles the operation state machines of the dialogue, discards any pending component, and passes a TC-P-ABORT indication primitive to the TC-users.

– The TC-user to abort the dialogue. At the originating side, a TC-U-ABORT request is received from the TC-user: active component state machines for this dialogue are idled, and a TR-U-ABORT request is passed to the transaction sub-layer. At the destination side, a corresponding TR-U-ABORT indication is received from the transaction sub-layer, any active component state machines for the dialogue are idled, and a TC-U-ABORT indication is passed to the TC-user.

If a TC-U-ABORT request primitive is issued during the active state of a dialogue, with the "Abort reason" parameter absent or set to "user-defined", a Dialogue Abort (ABRT) APDU is formatted only for the case when the AARQ/AARE APDUs were used during the dialogue establishment state. User data provided in the primitive is then transferred in the User Information field of the ABRT APDU.

When the dialogue is in the "Initiation Sent" state, i.e. a Begin message has been sent but no backward message for this transaction has been received, the result of the TC/TR-U-ABORT request primitive is purely local. Any message subsequently received that is related to this dialogue shall be handled according to the actions indicated in Table 7.

In these cases described above, accompanying information (P-Abort cause, or user-provided information) passes transparently through the component sub-layer.

Handling of the notification of abnormal situations which cannot be related to a particular dialogue is for further study.

3.2.2.2 Abnormal procedures relating to operations

The following abnormal situations are considered:

– no reaction to class 1 operation invocation (see 3.2.1.1.3);

– receipt of a malformed component – The component type and/or the Invoke ID cannot be recognized (i.e. the state machine cannot be identified);

– receipt of a well-formed component in violation of authorized state transitions.

The actions taken by the component sub-layer to report component portion errors are shown in Table 5. The following considerations have guided the choices indicated in this table:

– When a protocol error has been detected by the local TC-user, this TC-user is not subsequently advised via the TC-Reject (as indicated in Table 5) since it is already aware of the protocol error.

– In other cases (reject by component sub-layer), the local TC-user is always advised so that it can issue a dialogue control primitive (see the reject mechanism described below).

– When a component is rejected, the associated state machine returns to Idle.

– The reject mechanism applies whenever possible – Even if the invoke ID is not assigned or not recognized (i.e. no state machine can be identified), the reject mechanism should be initiated. The only case where rejection is purely local is when the component to be rejected is itself a Reject component.

Protocol errors in the component portion of a TCAP message are reported using the Reject component. The Reject component is sent in response to an incorrect component other than Reject. If the incorrect component is itself a Reject component, the component is discarded and the local TC-user is advised of the syntax error in the received Reject component.

When an invoke ID is available in a component to be rejected, this ID is reflected in the Reject component.

Component type abbreviations are identified in Table 1.

In the case of multiple components within a message, when a malformed component is detected by the component sub-layer, subsequent components in the message are discarded.

TABLE 5/Q.774

Action taken on Protocol Errors in Component Portion

Local					Remote (Note)	
Component Type received	Type of error	Local action	Component state machine	Local user advised	Component state machine	Remote user advised
Invoke	Syntax error or invalid linked id. (valid invoke id.)	Initiate Reject	NA	Yes[a)]	Return to Idle	Yes
	Syntax error (invalid invoke id.)	Initiate Reject	NA	Yes[a)]	No action	Yes
Return result (L/NL) or Return error	Syntax error (valid invoke id.)	Initiate Reject	Return to Idle	Yes[a)]	NA	Yes
	Syntax error (invalid invoke id.)	Initiate Reject	No action	Yes[a)]	NA	Yes
Return result (L/NL)	Operation Class 2/4	Initiate Reject	Return to Idle	Yes[a)]	NA	Yes
Return error	Operation Class 3/4	Initiate Reject	Return to Idle	Yes[a)]	NA	Yes
Reject	Syntax error	Initiate Local Reject	No action	Yes	NA	No
Unknown	Syntax error	Initiate Reject	No action/NA[b)]	Yes[a)]	NA/No action[b)]	Yes

NA Not applicable.

a) This is to alert the TC-user so it can issue a dialogue control primitive to send the Reject component formulated by the Component Sub-Layer.

b) It is not possible to decide whether an ISM exists or not, therefore, no action can be taken.

NOTE – Any action at the remote end depends on the local user issuing a dialogue primitive to send the Reject component formulated by the local CSL. Some users may choose not to issue the dialogue primitive in some or all cases. In these cases no action is taken at the remote end.

Rejection of any portion of a segmented result shall be equivalent to rejecting the entire result. The associated state machine is returned to Idle. Subsequent portions of the same segmented result shall also be rejected (on the basis of no active state machine).

The reject mechanism when the component sub-layer detects a situation where (non-local) reject should be initiated (as – per Table 5), it builds a Reject component, stores it, and informs the local TC-user by means of TC-L-REJECT indication primitive. The TC-user may decide

a) to continue the dialogue; or

b) to end the dialogue using the basic scenario; or

c) to abort the dialogue.

In cases a) and b), the first dialogue handling primitive (TC-CONTINUE request or TC-END request respectively) issued by the TC-user triggers transmission of the stored Reject component(s) built for this dialogue by the component sub-layer. The remote component sub-layer receives the Reject component(s) built for this dialogue, idles the corresponding component state machine(s) if possible (as per Table 5) and informs the TC-user of the (remote) rejection via TC-R-REJECT information primitive(s).

If the component sub-layer generated reject combined with accumulated components from the TC-user exceeds the message length limitations, then the TC-user, being aware of the Reject component, must initiate two dialogue handling primitives. The component sub-layer, also being aware of the length problem, will send all the components, except the Reject, with the first primitive. The Reject will be sent with the next dialogue handling primitive together with any further components provided by the TC-user.

3.2.3 Compatibility issues

As the Dialogue Portion of a TC message, as defined in these specifications, is optional from a syntax point of view, a TCAP implementation conforming to these Recommendations will understand a TC-BEGIN message without the dialogue portion, i.e. an implementation that conforms to the 1988 TC Recommendations and will be able to respond with subsequent messages that conform to the latter Recommendations.

A TC implementation conforming to the 1988 Recommendations receiving a BEGIN message with a dialogue portion will treat it as a syntactically invalid message. It will react by sending an ABORT message with a P-Abort cause information element indicating "incorrect transaction portion". It will be up to the TC-User at the initiating side to interpret this abort. One interpretation could be that the abort is due to communication with a *Blue Book* implementation; so the TC-User could issue a new TC-BEGIN request primitive without the parameters related to the dialogue portion. However, such interpretations and subsequent actions are outside the scope of these Recommendations.

The version number field in the Dialogue Request (AARQ) APDU and the Dialogue Response (AARE) APDU will be used to indicate the versions of the Dialogue Portion that are supported by TCAP at the node sending the message. The version described in these Recommendations is "version 1". If the component sub-layer at a node conforming to these Recommendations (including the optional dialogue handling portion) receives a Dialogue Request (AARQ) APDU with a version field which indicates that version 1 is not in the list of supported versions, the component sub-layer builds a Dialogue Response (AARE) APDU with its fields set as follows:

- protocol-version = version 1
- application-context-name = the one received in the AARQ APDU
- result = reject (permanent)
- result-source-diagnostic = dialogue-service-provider (no-common-dialogue-version)
- user-information = absent.

The component sub-layer issues a TR-U-ABORT request indication with the AARE APDU as user abort information. Informing its local TC-user is not required and any received components are discarded.

The component sub-layer at the node receiving an AARE APDU thus formatted as user abort information in a TR-U-ABORT indication primitive, informs its TC-User via a TC-P-ABORT indication primitive with the "P-Abort" parameter in the primitive set to "no common dialogue portion".

The above-mentioned forward compatibility mechanism is included to ensure interworking with future versions resulting from the evolution (if any) of the dialogue portion. For the situation where only the current version of the dialogue portion (as defined in these Recommendations) is implemented, the receipt of an AARE APDU with the version field set to any value other than "version 1" shall be considered a syntax error and the procedures described in 3.2.2.1 will be followed.

3.3 Transaction sub-layer procedures

3.3.1 General

In the case of a structured dialogue, the transaction sub-layer provides for an end-to-end connection between its users (TR-users). This end-to-end connection is called a transaction.

The transaction sub-layer procedure associates each TCAP message and, therefore, all the contained components and the dialogue portion, if present, with a particular transaction.

The transaction sub-layer processes the transaction portion (message type and transaction ID) of a TCAP message. Transaction IDs identify a transaction. Each end assigns a local transaction identification; these local transaction IDs are exchanged in the transaction portion of messages as indicated in Recommendation Q.773.

The component portion of a TCAP message is passed between the component sub-layer and the transaction sub-layer as user data in the transaction sub-layer primitives.

In the case of an unstructured dialogue, no transaction ID is assigned. The component portion of the UNIDIRECTIONAL message is received as user data in the TR-UNI request primitive. The transaction portion of the UNIDIRECTIONAL message is formatted and the message transmitted.

3.3.2 Mapping of TR service primitives to message types

Recommendation Q.771 describes the services performed by the transaction sub-layer by defining the service interface between the TR user and the transaction sub-layer and the transaction sub-layer and the SCCP. Similarly, state transition diagrams appear in Recommendation Q.771 based on service primitives. In this subclause, a message based description of the protocol is provided. A mapping of TR-primitives to transaction sub-layer protocol data units is indicated in Table 6.

TABLE 6/Q.774

Mapping of TR Service Primitives to Messages

Service Primitive	Message Type
TR-UNI	UNIDIRECTIONAL
TR-P-ABORT	ABORT (Note 1)
TR-BEGIN	BEGIN
TR-CONTINUE	CONTINUE
TR-U-ABORT	ABORT (Note 2)
TR-END	END

NOTES

1 With P-Abort cause information element.

2 Empty or with a User Abort information element.

3.3.3 Normal procedures

3.3.3.1 Message transfer without establishing a transaction

3.3.3.1.1 Actions of the sending end

The TR-UNI request primitive is used when a TR-user sends a message to another TR-user but does not need to enter into a transaction. A Unidirectional message, which does not have a transaction ID, is used in this case.

3.3.3.1.2 Actions of the receiving end

The receipt of a Unidirectional message causes a TR-UNI indication primitive to be passed to the TR-user. No further action is taken by the transaction sub-layer.

3.3.3.2 Message transfer within a transaction

3.3.3.2.1 Transaction begin

In the following discussion, the sending node of the first TCAP message is labelled node "A", and the receiving node is labelled node "B".

3.3.3.2.1.1 Actions of the initiating end

The TR-user at node "A" initiates a transaction by using a TR-BEGIN request primitive, which causes a Begin message to be sent from node "A" to node "B".

The Begin message contains an originating transaction ID. This transaction ID value, when included in any future message from node "A" as the originating transaction ID or in a message to node "A" as the destination transaction ID, identifies the transaction to node "A".

Once the transaction sub-layer at node "A" has sent a Begin message it cannot send another message to the transaction sub-layer at node "B" for the same transaction until it receives a Continue message from node "B" for this transaction.

3.3.3.2.1.2 Actions of the receiving end

The receipt of a Begin message causes a TR-BEGIN indication primitive to be passed to the TR-user at node "B". In response to a TR-BEGIN indication primitive, the TR-user at node "B" decides whether or not to establish a transaction. If the TR-user does want to establish a transaction, it passes a TR-CONTINUE request primitive to the transaction sub-layer; otherwise, it terminates the transaction (see 3.3.3.2.3). These conditions are defined by the TR-user.

The Begin message contains only an originating transaction ID. If, after receiving a Begin message with a given originating transaction ID, the transaction sub-layer receives another Begin message with the same originating transaction ID, the transaction sub–layer does not consider this as an abnormal situation: a second transaction is initiated at node "B".

3.3.3.2.2 Transaction continuation

A Continue message is sent from one node to another when a TR-CONTINUE request primitive is passed from the TR-user to the transaction sub–layer at the sending node.

A Continue message includes the destination transaction ID which is identical – i.e. of the same octet length and value – to the originating transaction ID received in the first messages from the peer node. Each node assigns its own originating transaction ID at transaction initiation time. The transaction IDs remain constant for the life of the transaction.

A Continue message includes both an originating transaction ID and a destination transaction ID. The originating transaction ID in successive Continue messages is not examined.

Receipt of a Continue message causes a TR-CONTINUE indication primitive to be passed to the destination TR-user.

Once the user at node "B" has responded with a TR-CONTINUE request primitive to establish a transaction, all subsequent interactions at either end between the TR-user and the transaction sub-layer are via TR-CONTINUE primitives until the transaction is to be terminated. In message terms, once a Continue message is sent from node "B", all subsequent messages shall be Continue messages until the transaction is to be terminated.

3.3.3.2.3 Transaction termination

– The basic method – A TR-user at either end may terminate a transaction by passing a TR-END request primitive (indicating basic end) to the transaction sub-layer. An End message is sent to the peer entity which, in turn, passes a TR-END indication primitive to its TR-user. The End message contains a destination transaction ID which is identical – i.e. of the same octet length and value – to the originating transaction ID received in the first message from the peer node.

– The pre-arranged method – This method implies that the peer entities know *a priori* – at a given point in the application script – that the transaction will be released. In this case, the TR-user passes a TR-END request primitive (indicating pre-arranged end) to its transaction sub-layer, and no End message is sent.

3.3.3.2.4 Abort by the TR-user

When a TR-user wants to abort a transaction, it passes a TR-U-ABORT request primitive to the transaction sub-layer, which sends an Abort message with user-provided (cause and diagnostic) information.

At the receiving side, the transaction sub-layer receiving an Abort message containing user-provided information passes this information without analyzing it to the TR-user in a TR-U-ABORT indication primitive.

3.3.3.2.5 Example of message exchange

Figure 7 depicts an example of exchanges of TCAP messages between two TR-users.

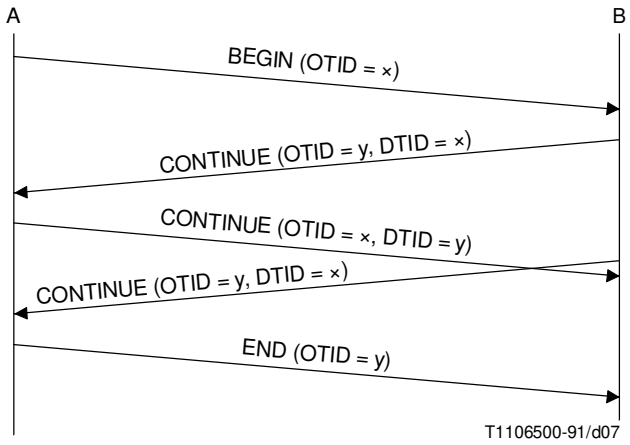


FIGURE 7/Q.774

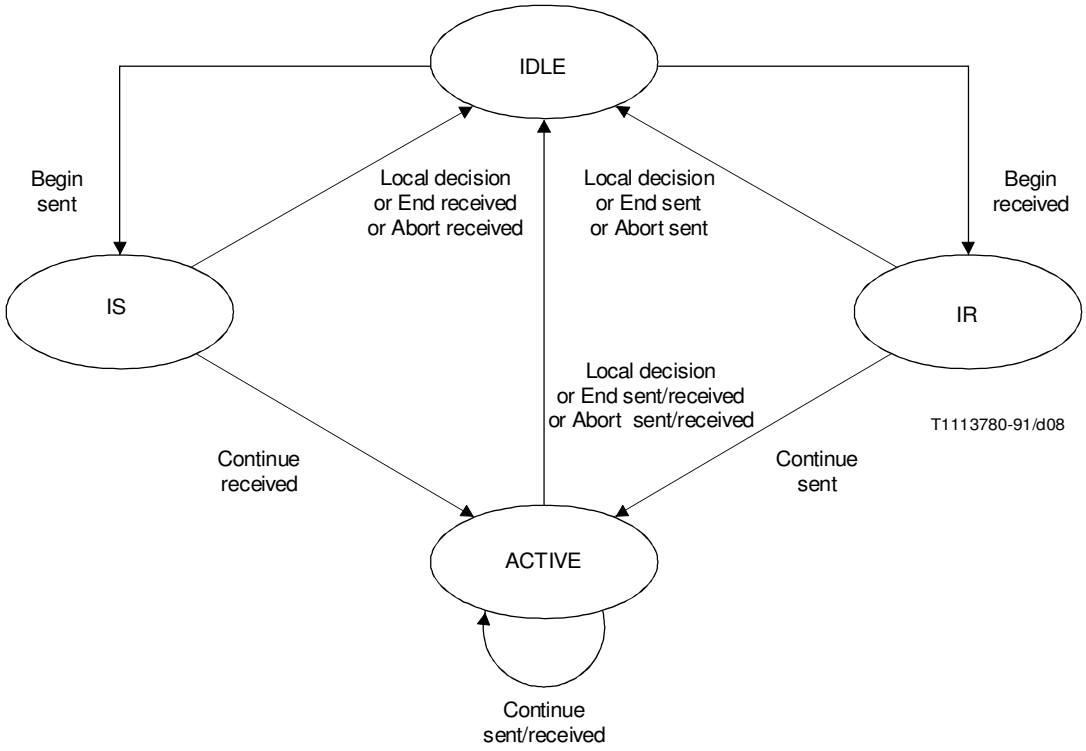
A simple example of exchange of TCAP messages

3.3.3.2.6 Transaction state transition diagrams

A state machine is associated with a transaction at each end of this transaction. Four transaction states are introduced:

- *Idle* – No state machine exists.
- *Init Sent (IS)* – A Begin message has been sent: an indication from the peer entity whether the transaction has been established or not is awaited.
- *Init Received (IR)* – A Begin message has been received: a request from the TR-user either to continue the transaction, or to terminate it, is awaited.
- *Active* – The transaction is established: Continue messages can be exchanged in both directions simultaneously.

Figure 8 shows the transaction state transition diagram.



NOTE – Local decision:
1 pre-arranged end;
2 see 3.3.4.

FIGURE 8/Q.774

Transaction state transition diagram

3.3.4 Abnormal procedures relating to transaction control

The following abnormal situations are covered by the transaction sub-layer:

1) No reaction to transaction (initiated or established).

2) Receipt of an indication of abnormal situation from the underlying layer.

3) Receipt of a message with an unassigned or non-derivable destination transaction ID so that the message cannot be associated with a transaction (non-derivable means that the information is not found or not recognized; unassigned means that the ID is derivable but it has not been assigned to a transaction).

4) Receipt of a message with a recognized destination transaction ID – The message can be associated with a transaction, but the message type is not compatible with the transaction state.

Case 1 considers those situations where one node is in the Idle state and the other node is in a non-Idle state, for example, due to message loss.

This is covered by a local, implementation-dependent, mechanism which results in aborting the transaction locally, as described below.

Case 2 is for further study.

When a transaction portion error is found (cases 3 and 4 above), the transaction sub-layer should take the following actions.

The status of the originating transaction ID should be checked. Actions are the following:

1) if the originating transaction ID is not derivable, the local end (which received the message) discards the message and does not take any other action, e.g. it does not send an Abort message or terminate the transaction; or

2) if the originating transaction ID is derivable, the following actions are taken:

   i) The transaction sub-layer should form an Abort message with an appropriate P-Abort cause information element and transmit it to the originating end. The originating end will then take the appropriate action to terminate the transaction if the originating transaction ID is assigned.

   ii) If the destination transaction ID is not derivable or derivable but not assigned, the transaction sub-layer takes no action to terminate the transaction at its end.

   iii) If the destination transaction ID is derivable and assigned:

      a) the transaction sub-layer terminates the transaction at its end, i.e. return to idle;

      b) the transaction sub-layer informs the component sub-layer of the abort of the transaction via the transaction sub-layer abort; and

      c) the component sub-layer should

         – release all invoke IDs associated with this transaction;

         – discard any pending components for that transaction;

         – inform the TC-user of the transaction abort.

Finally, regardless of the disposition of the transaction IDs, the entire erroneous TCAP message should be discarded.

When receiving an Abort message, the destination transaction sub-layer does the following:

– if the Abort message contains user information (or no information), inform the TR-user by means of the TR-U-ABORT indication primitive;

– if the Abort message contains a P-Abort cause information element, inform the TR-user by means of the TR-P-ABORT indication primitive. Notification to the management is for further study;

– in both cases, discard any pending messages for that transaction and return the transaction state machine to Idle.

TABLE 7/Q.774

Actions when an Abnormal Transaction Portion is Received

Local End (detects protocol error)						Remote End	
Message Type Received	Originating Transaction ID [d]	Destination Transaction ID [d]	Action	Transaction State Machine	Local User Advised	Transaction State Machine	User Advised
UNIDIREC-TIONAL	–	–	Discard	– [c]	No	– [c]	No
BEGIN	Not der.	–	Discard	NA [b]	No	NA [b]	No
	Der.	–	Abort	NA [b]	No	Return to Idle [a]	Yes [a]
CONTINUE	Not der.	–	Discard	NA [b]	No	NA [b]	No
	Der.	Not der. unass.	Abort	NA [b]	No	Return to Idle [a]	Yes [a]
	Der.	Ass.	Abort	Return to Idle	Yes	Return to Idle [a]	Yes [a]
END/ABORT	–	Not der. unass.	Discard	NA [b]	No	NA [b]	No
	–	Ass.	Discard	Return to Idle	Yes	NA [b]	No
UNKNOWN	Not der.	–	Discard	NA [b]	No	NA [b]	No
	Der.	Not der. unass.	Abort	NA [b]	No	Return to Idle [a]	Yes [a]
	Der.	Ass.	Abort	Return to Idle	Yes	Return to Idle [a]	Yes [a]

NA Transition to the Idle state is Not Applicable [see item b] below]

Not der. Not derivable

Der. Derivable (Whether a particular abnormality renders a TID not derivable is implementation-dependent)

Ass. Derivable and assigned

Unass. Derivable but unassigned

Abort Send ABORT message

a) If the Transaction ID is assigned at this end, otherwise the state transition is not applicable, and the user is not informed.

b) The expression NA is used in those cases where the normal procedure of Return to Idle at both ends following the appearance of an abnormal situation is Not Applicable because it is impossible to identify the Transaction ID(s) and therefore to relate the damaged message to a specific transaction at either ends (Local and/or Remote end).

c) The Unidirectional message does not refer to an explicit transaction and therefore it does not affect the Transaction State Machine.

d) The derivability of the Transaction IDs is implementation dependent.

Annex A

Transaction capabilities SDLs

(This annex forms an integral part of this Recommendation)

A.1 General

This annex contains the description of the transaction capability procedures described in this Recommendation by means of SDLs according to the CCITT specification and description language. In order to facilitate the functional description as well as the understanding of the behaviour of the signalling system, the transaction capabilities application part (TCAP) is divided into the component sub-layer and the transaction sub-layer (see Figure A.1). The transaction sub-layer is divided into two functional blocks: a Transaction Coordinator (TCO) and a transaction state machine(s) Block (TSM). The component sub-layer again is divided into a component handling block (CHA) and a dialogue handling block (DHA) (see Figure A.2).

The SDL is provided according to this functional partitioning which is used only to facilitate understanding and is not intended to be adopted in a practical implementation of the TCAP. The functional blocks and their associated service primitives are shown in Figure A.2.

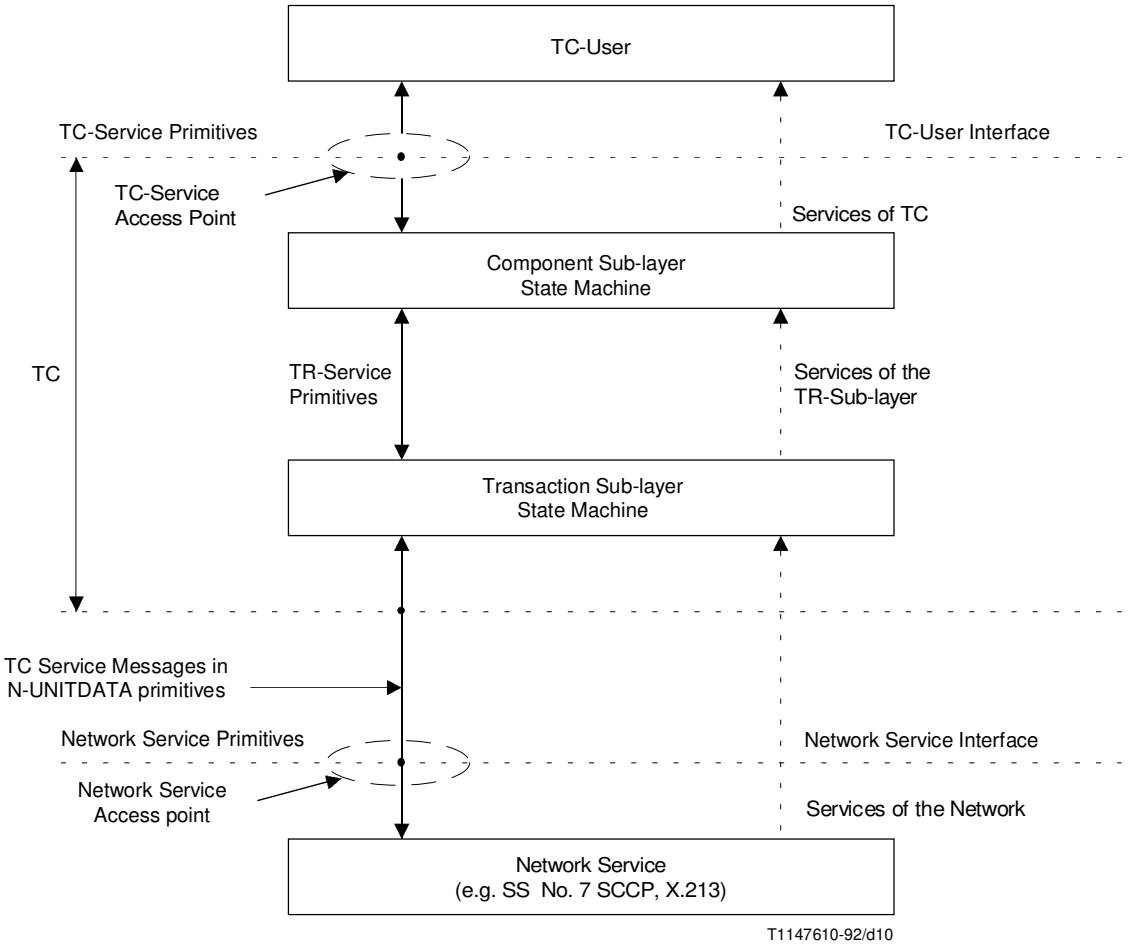
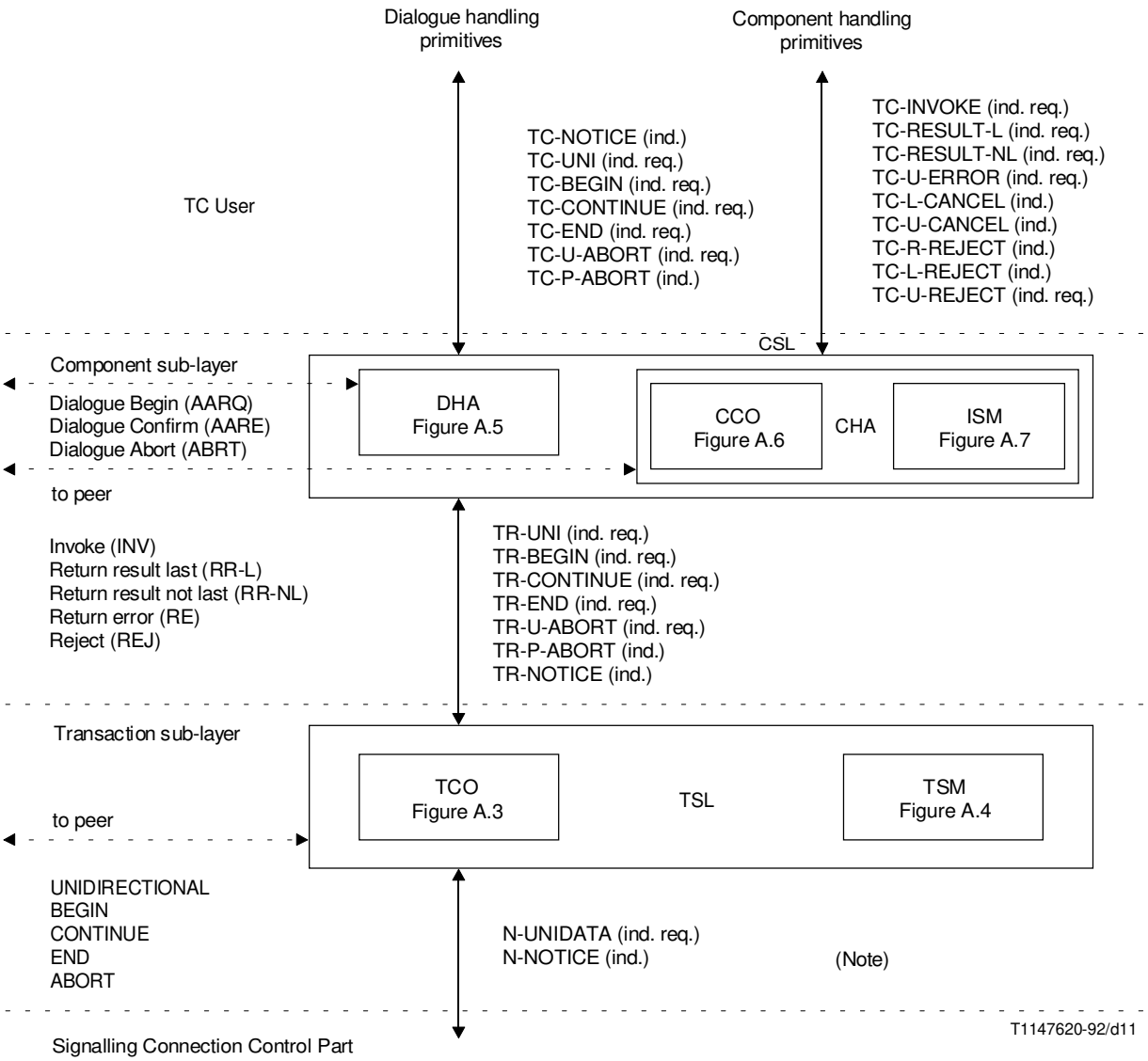


FIGURE A.1/Q.774

Interfaces of the component and transaction sub-layers (state machines) and service primitives

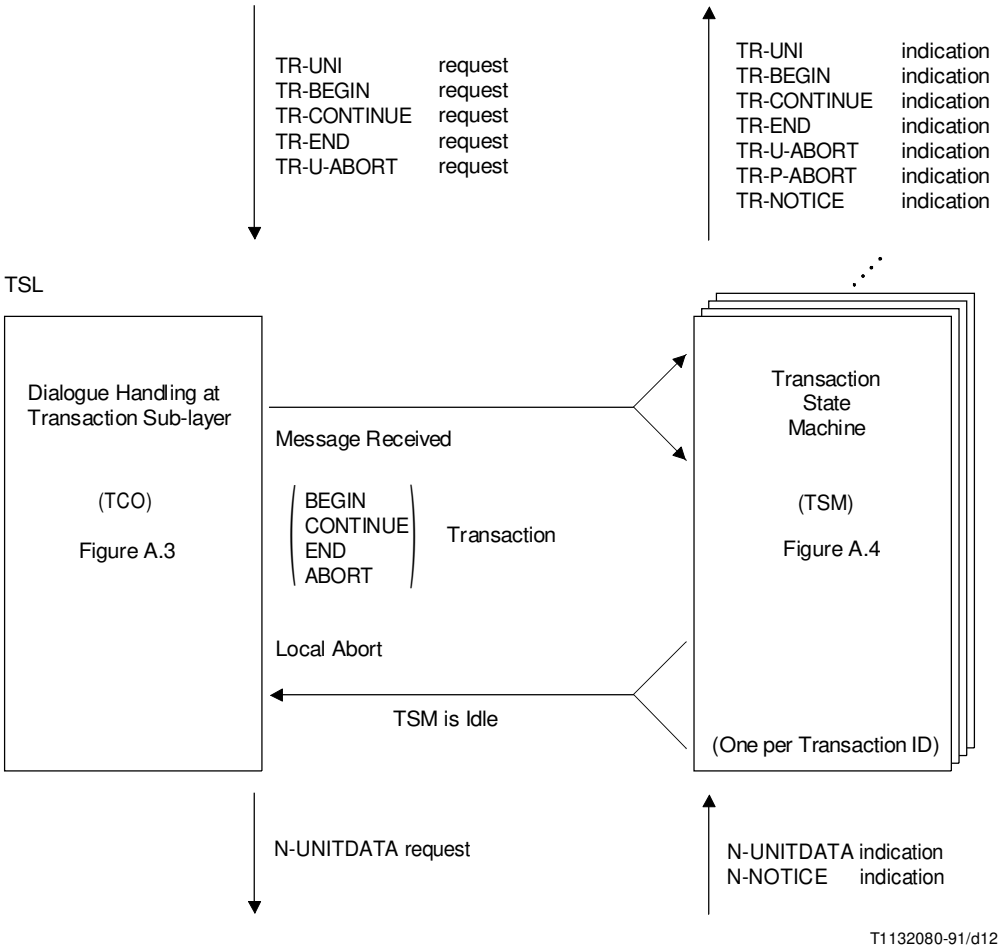


CHA	Component Handling
DHA	Dialogue Handling (CSL)
TCO	Transaction Coordinator
TSL	Transaction Sub-Layer
CCO	Component Coordinator
ISM	Invocation State-Machine
TSM	Transaction State Machine
CSL	Component Sub-layer

NOTE – Other Network Service Primitives are for further study (Q.700-Series Recommendations).

FIGURE A.2a/Q.774

Overview block diagram of TC



NOTE – The functions of TSL are shared to two functional blocks: TCO and TSMs. The TCO handles syntax checking of incoming TCAP-messages and interactions with individual TSMs. For each transaction, there is a TSM that provides state information, issues primitives to TR-User and assembles and sends the TCAP-messages. UNIDIRECTIONAL and ABORT messages that are not related to any local TSM, are assembled and sent by TCO.

FIGURE A.2b/Q.774

Overview block diagram of transaction sub-layer

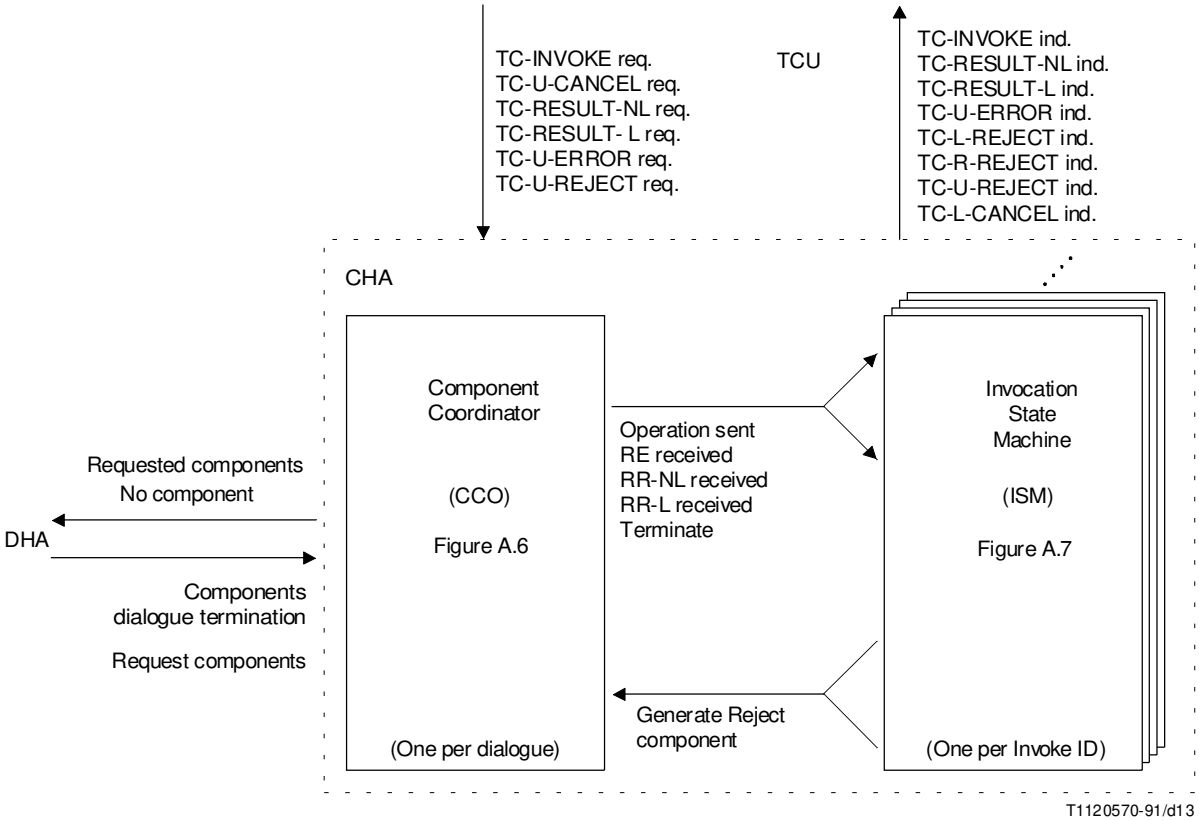
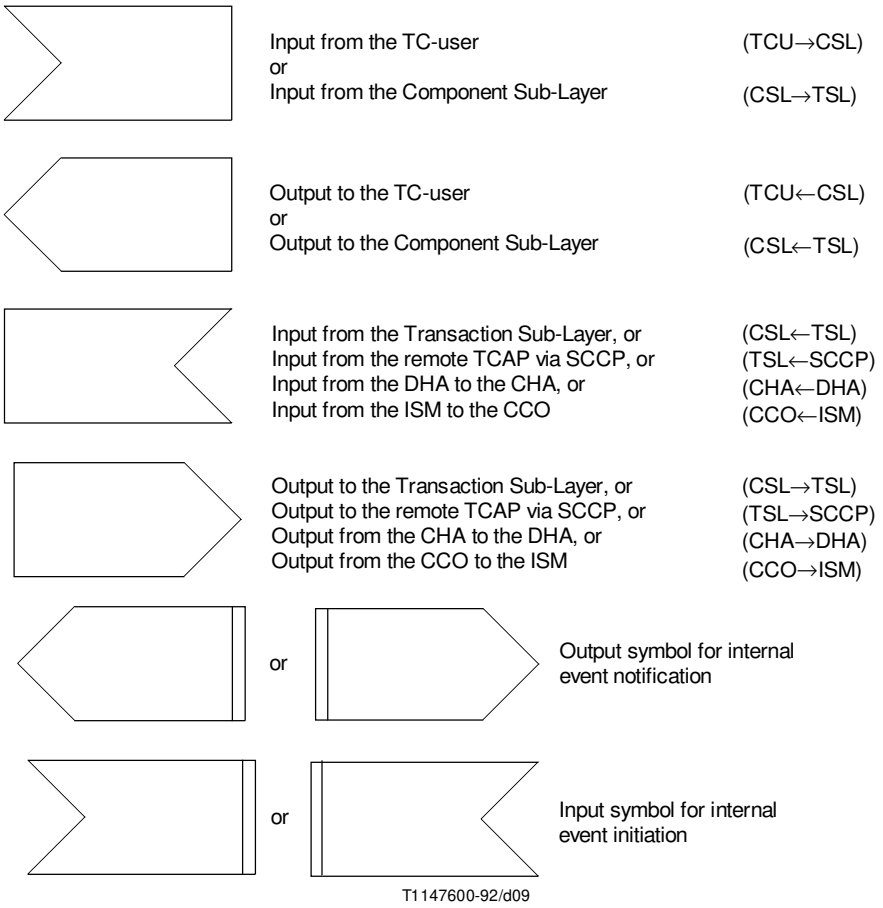


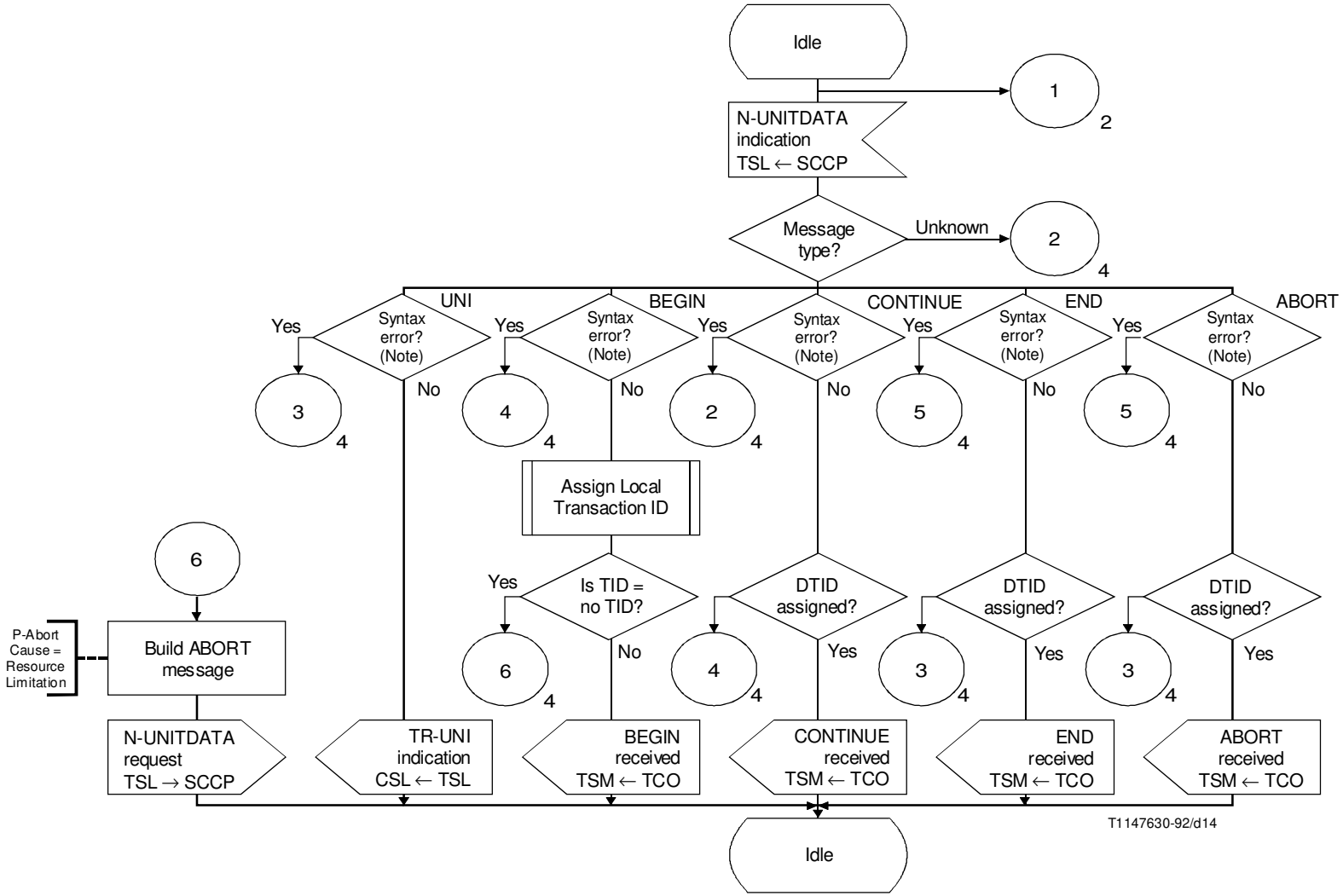
FIGURE A.2c/Q.774

Overview block diagram of CHA

A.2 Drafting conventions

To indicate the direction of each interaction the symbols are used as shown below:





NOTE – Checks for correctly formatted TR-portion information elements for this message type.

FIGURE A.3/Q.774 (sheet 1 of 4)

Transaction Coordinator

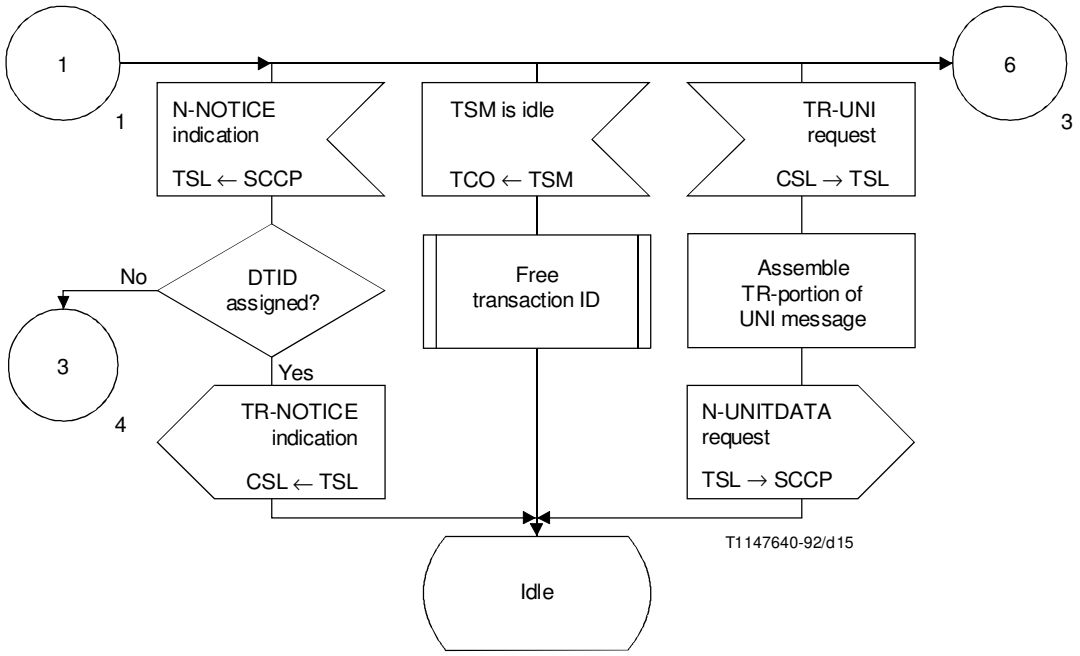


FIGURE A.3/Q.774 (sheet 2 of 4)

Transaction Coordinator

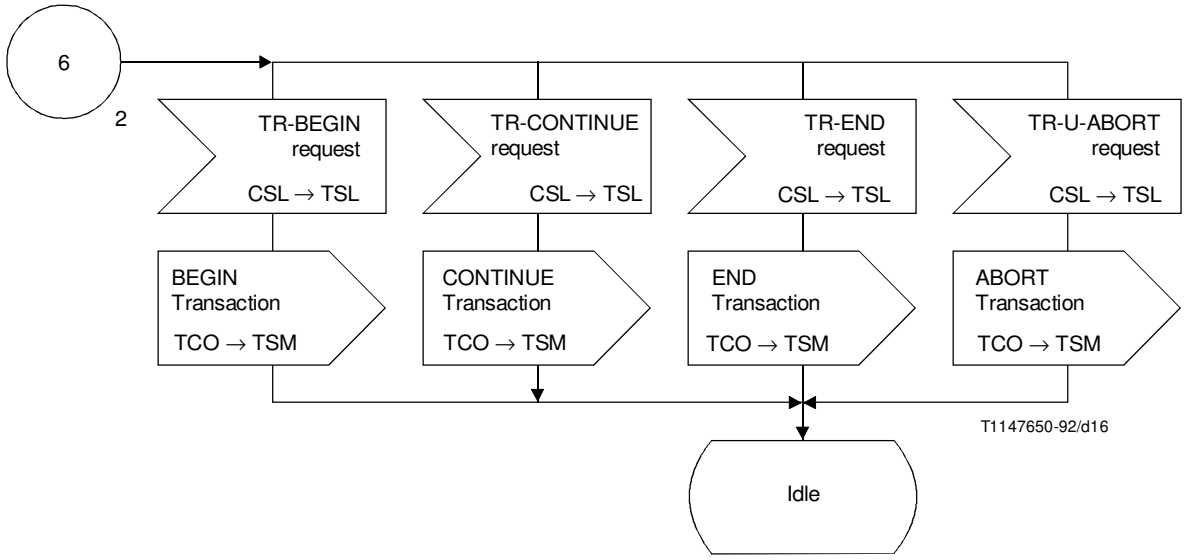
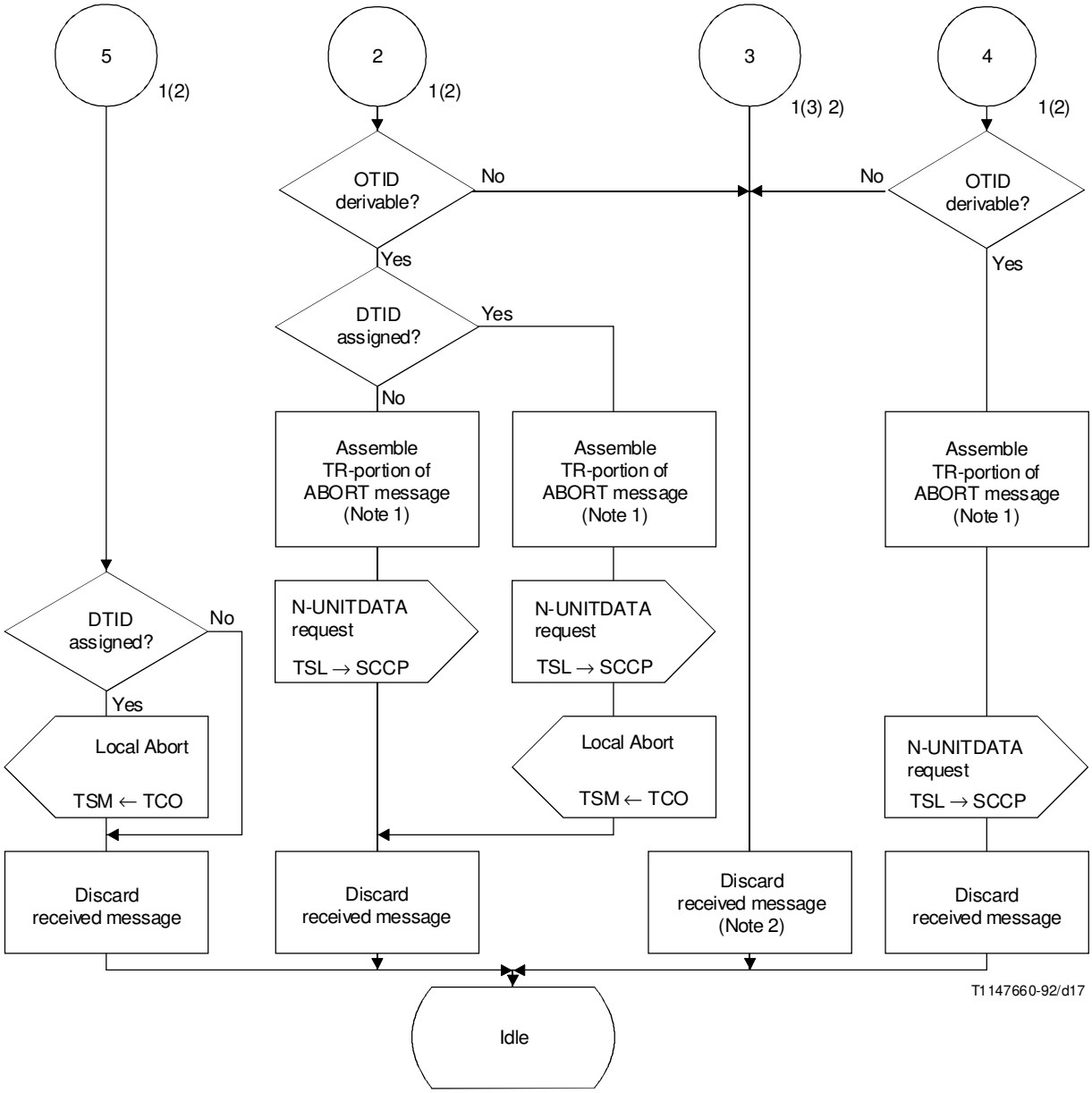


FIGURE A.3/Q.774 (sheet 3 of 4)

Transaction Coordinator



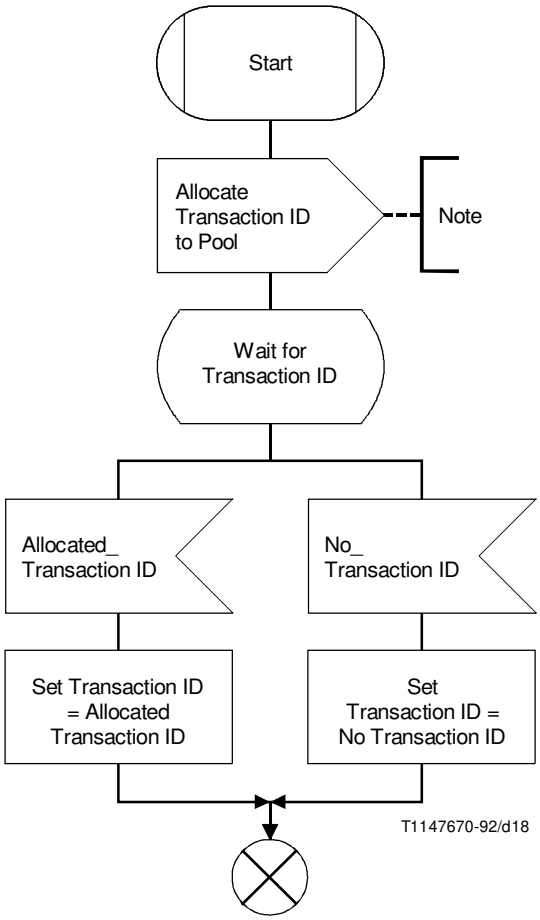
NOTES

1 Using appropriate P-Abort Cause values as defined in Recommendation Q.773.

2 Notification to TC-user is a local implementation option (for only UNI message)

FIGURE A.3/Q.774 (sheet 4 of 4)

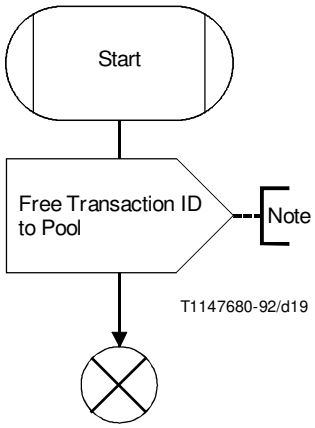
Transaction Coordinator



NOTE – The pool realization is implementation-dependent

FIGURE A.3 *bis*/Q.774

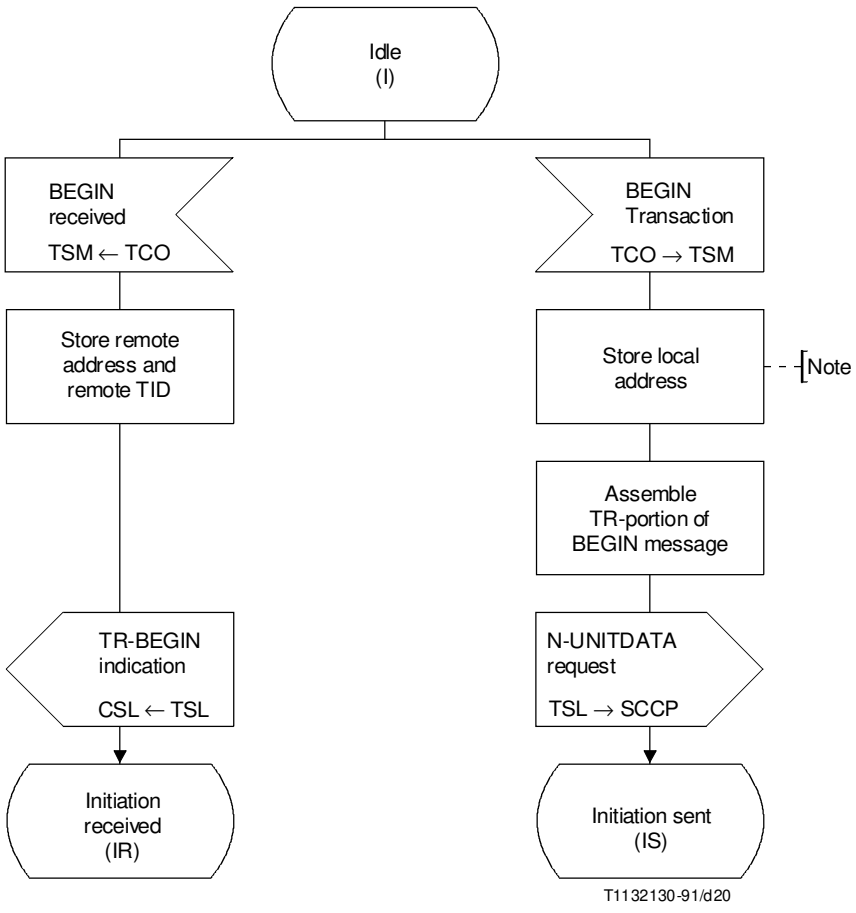
Procedure ASSIGN TRANSACTION ID



NOTE – The pool realization is implementation-dependent

FIGURE A.3 *ter*/Q.774

Procedure FREE TRANSACTION ID



NOTE – This may be provided by TC-user or be implicitly associated with the access point at which the N-UNITDATA primitive is issued.

FIGURE A.4/Q.774 (sheet 1 of 5)

Transaction state machine

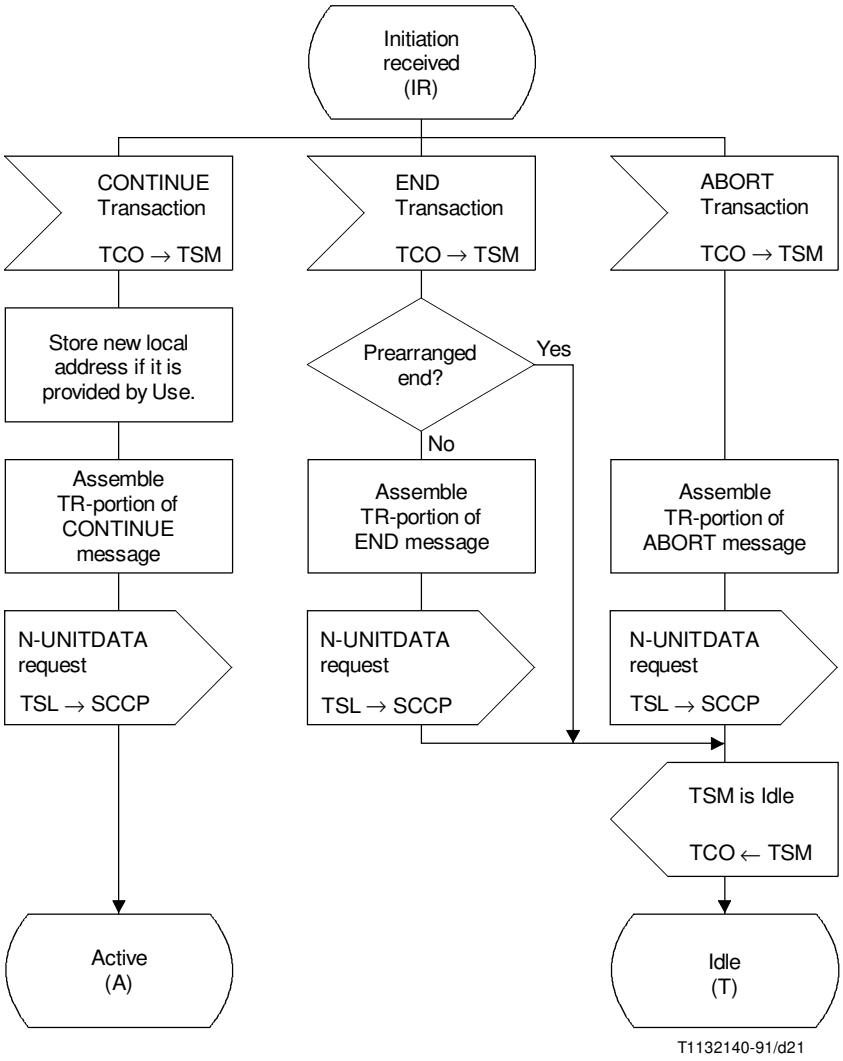


FIGURE A.4/Q.774 (sheet 2 of 5)

Transaction state machine

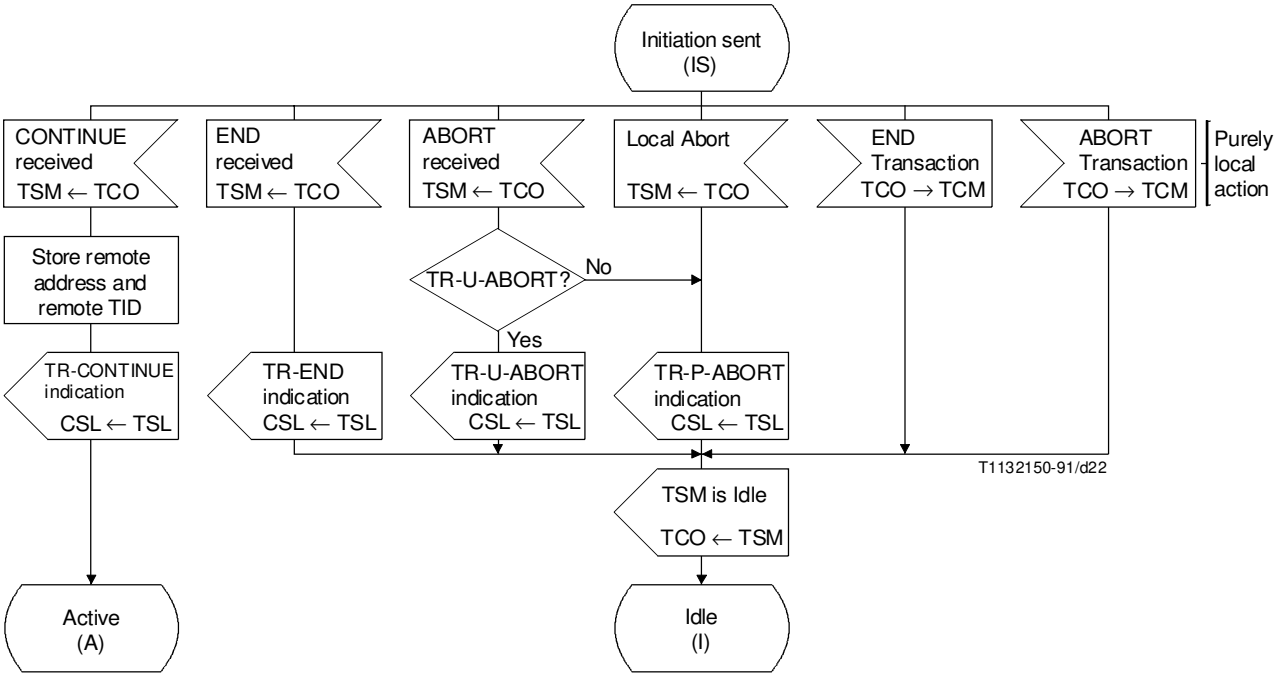


FIGURE A.4/Q.774 (sheet 3 of 5)

Transaction state machine

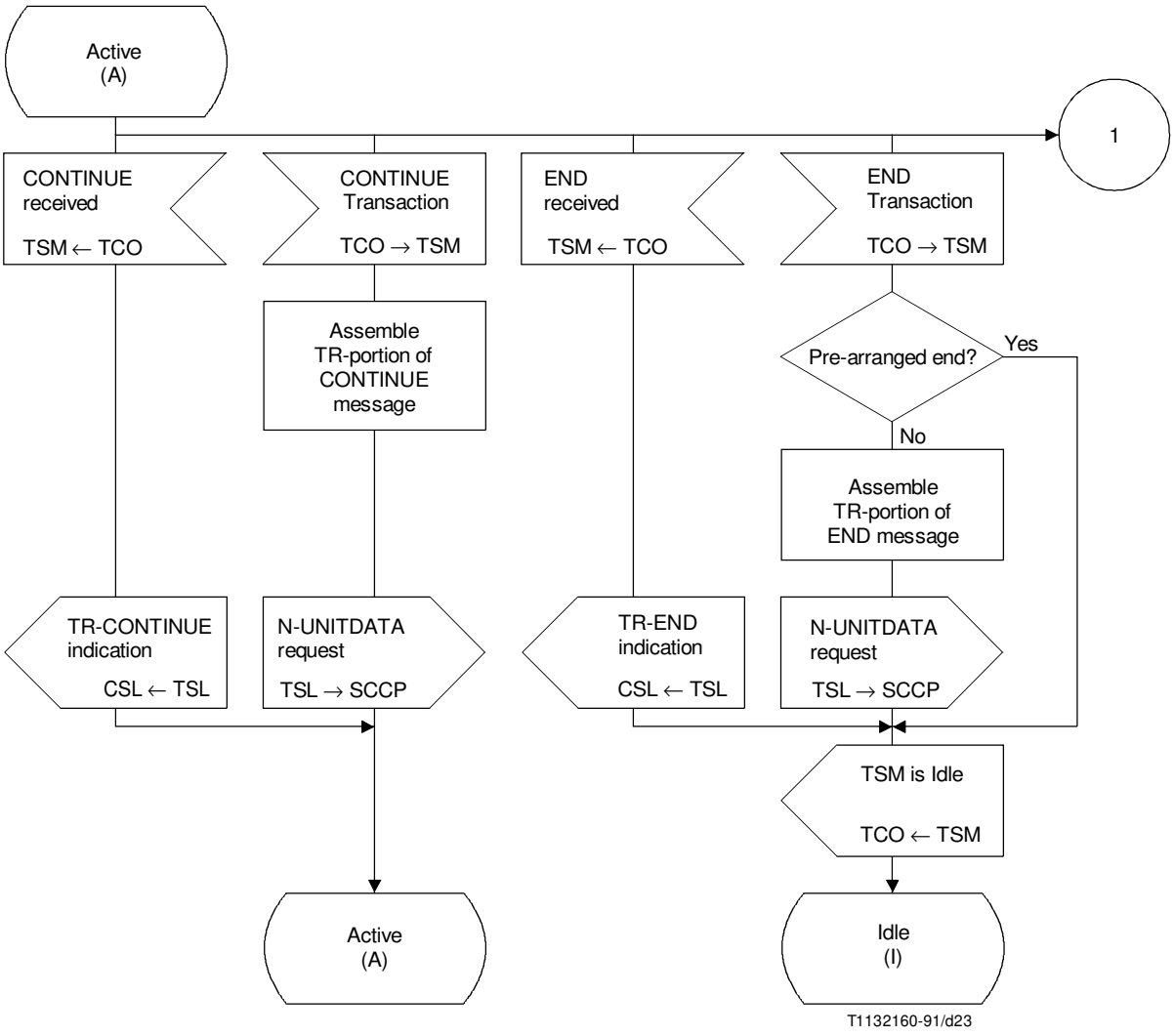


FIGURE A.4/Q.774 (sheet 4 of 5)

Transaction state machine

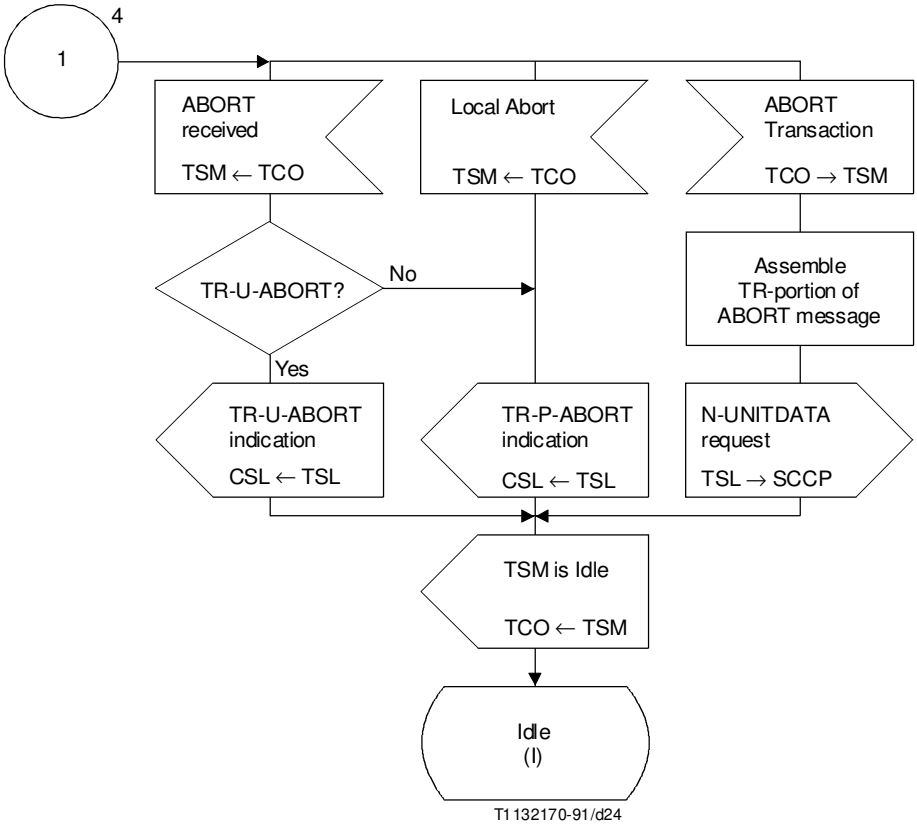


FIGURE A.4/Q.774 (sheet 5 of 5)

Transaction state machine

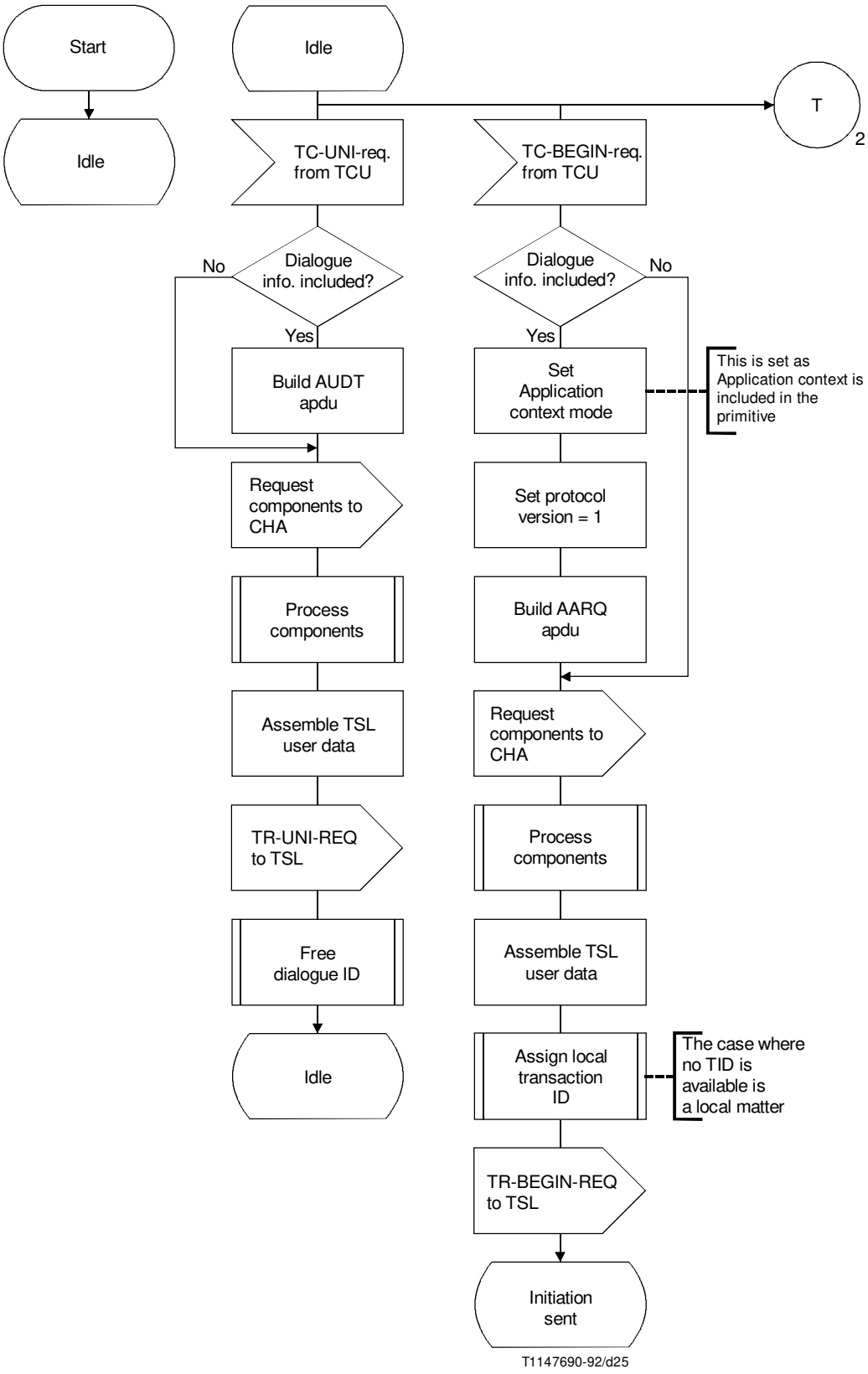


FIGURE A.5/Q.774 (sheet 1 of 11)

Dialogue handling at CSL

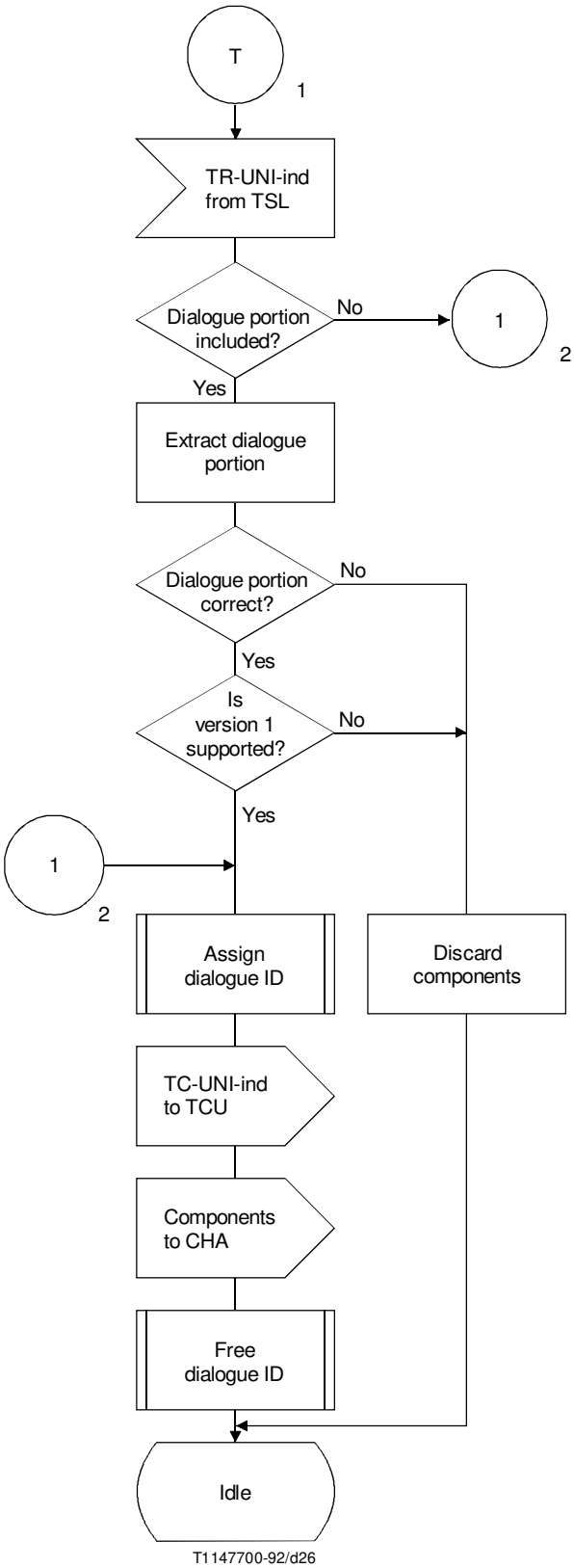


FIGURE A.5/Q.774 (sheet 2 of 11)

Dialogue handling at CSL

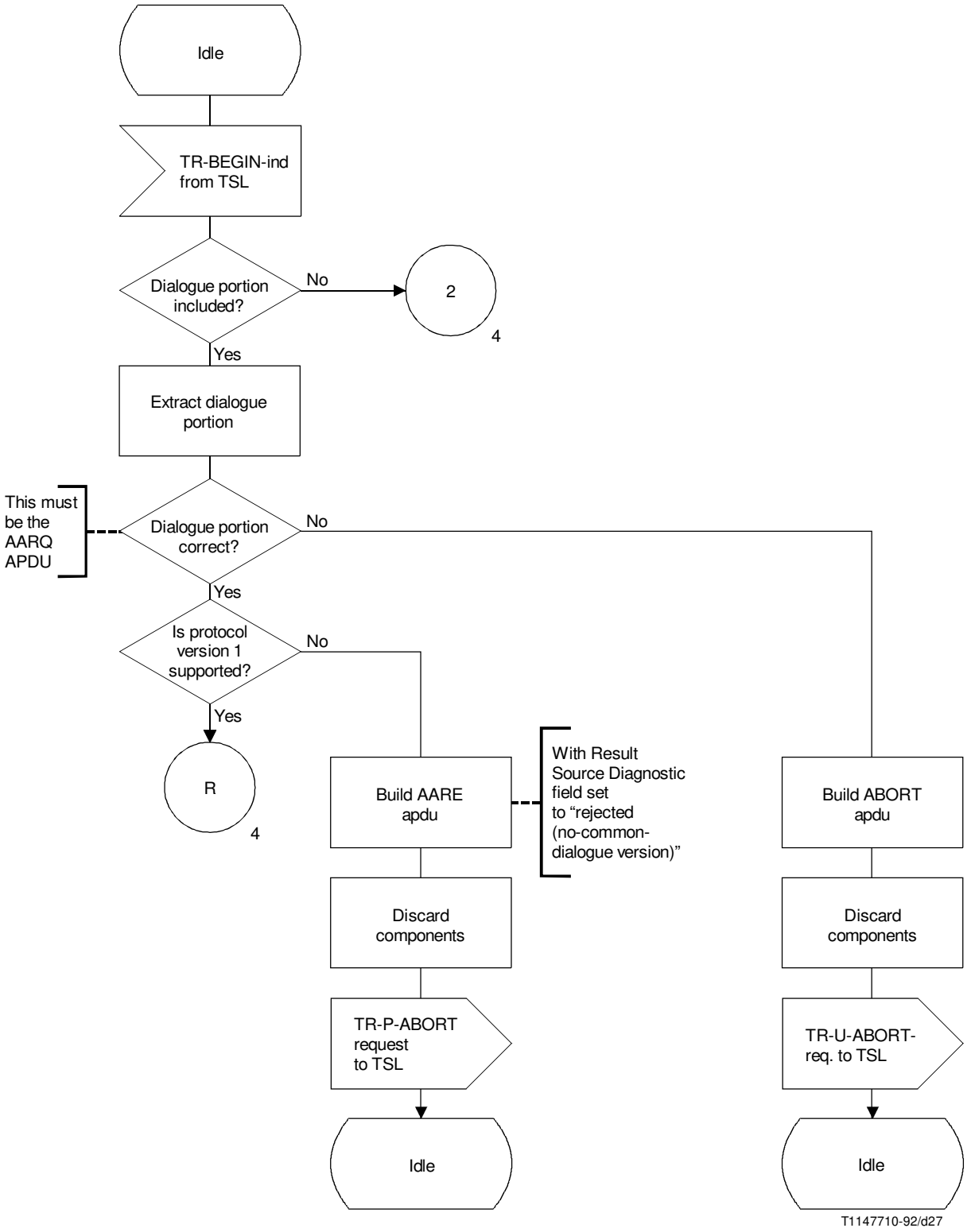


FIGURE A.5/Q.774 (sheet 3 of 11)

Dialogue handling at CSL

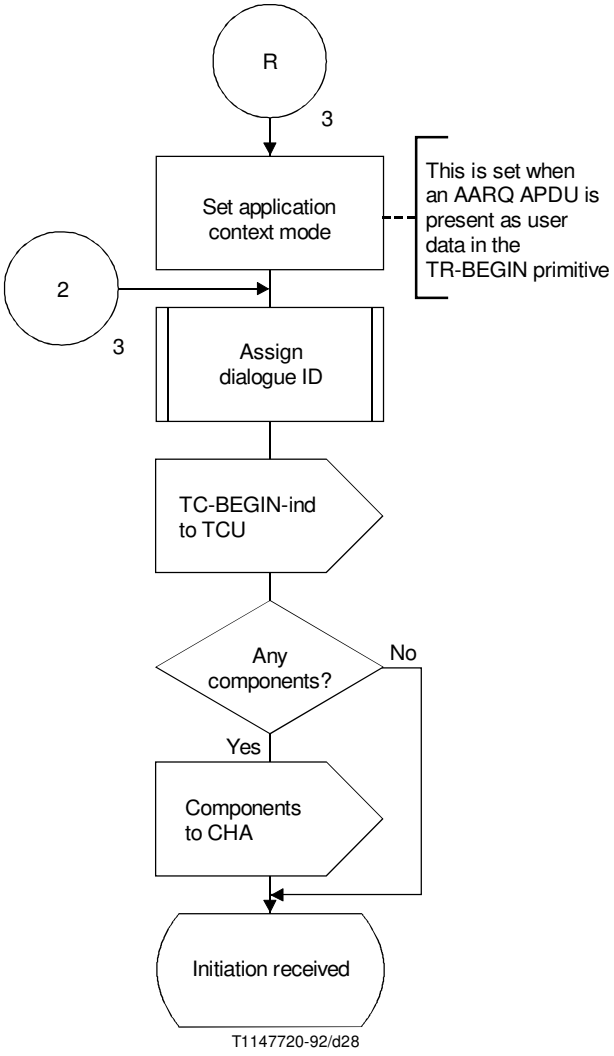


FIGURE A.5/Q.774 (sheet 4 of 11)

Dialogue handling at CSL

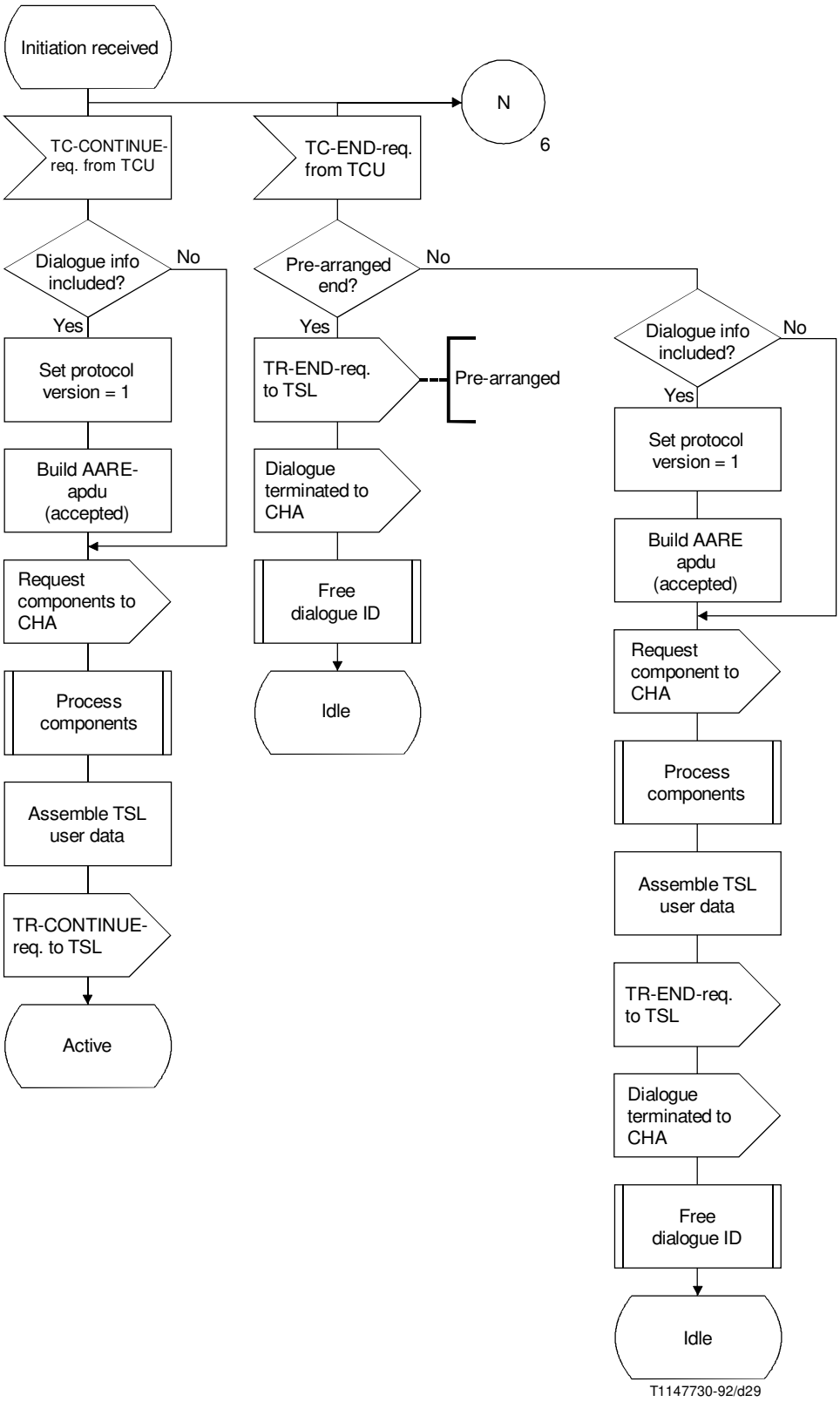


FIGURE A.5/Q.774 (sheet 5 of 11)

Dialogue handling at CSL

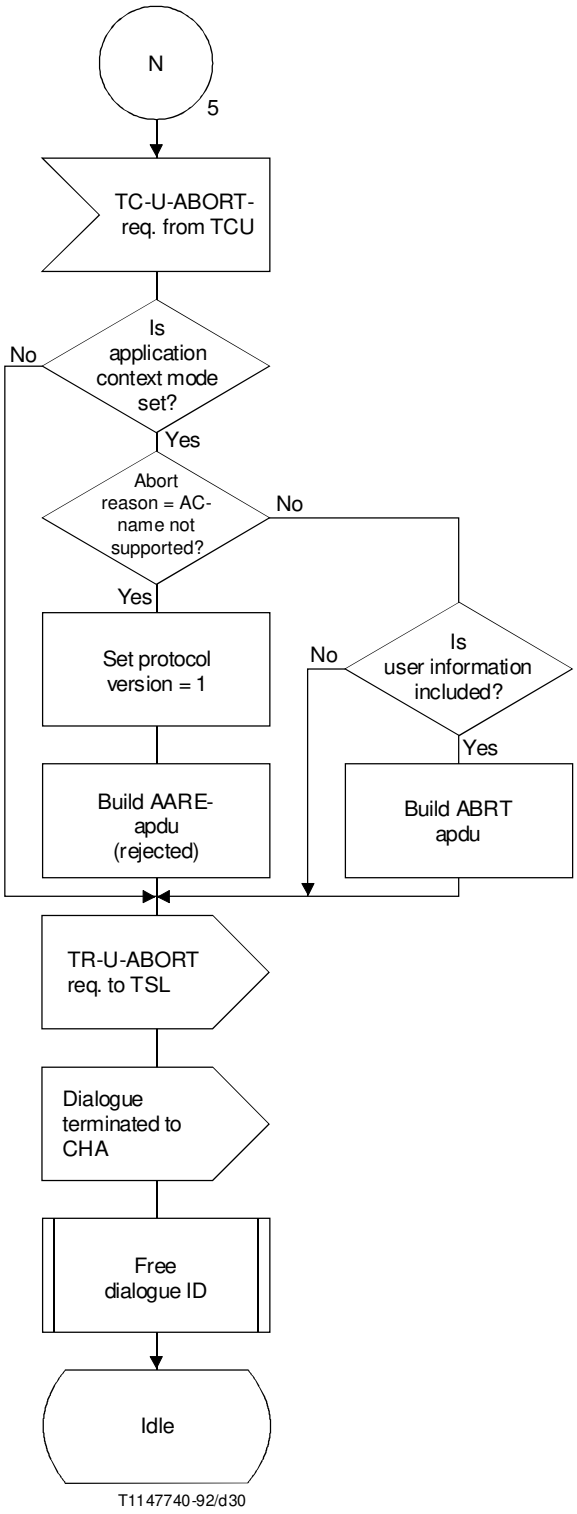


FIGURE A.5/Q.774 (sheet 6 of 11)

Dialogue handling at CSL

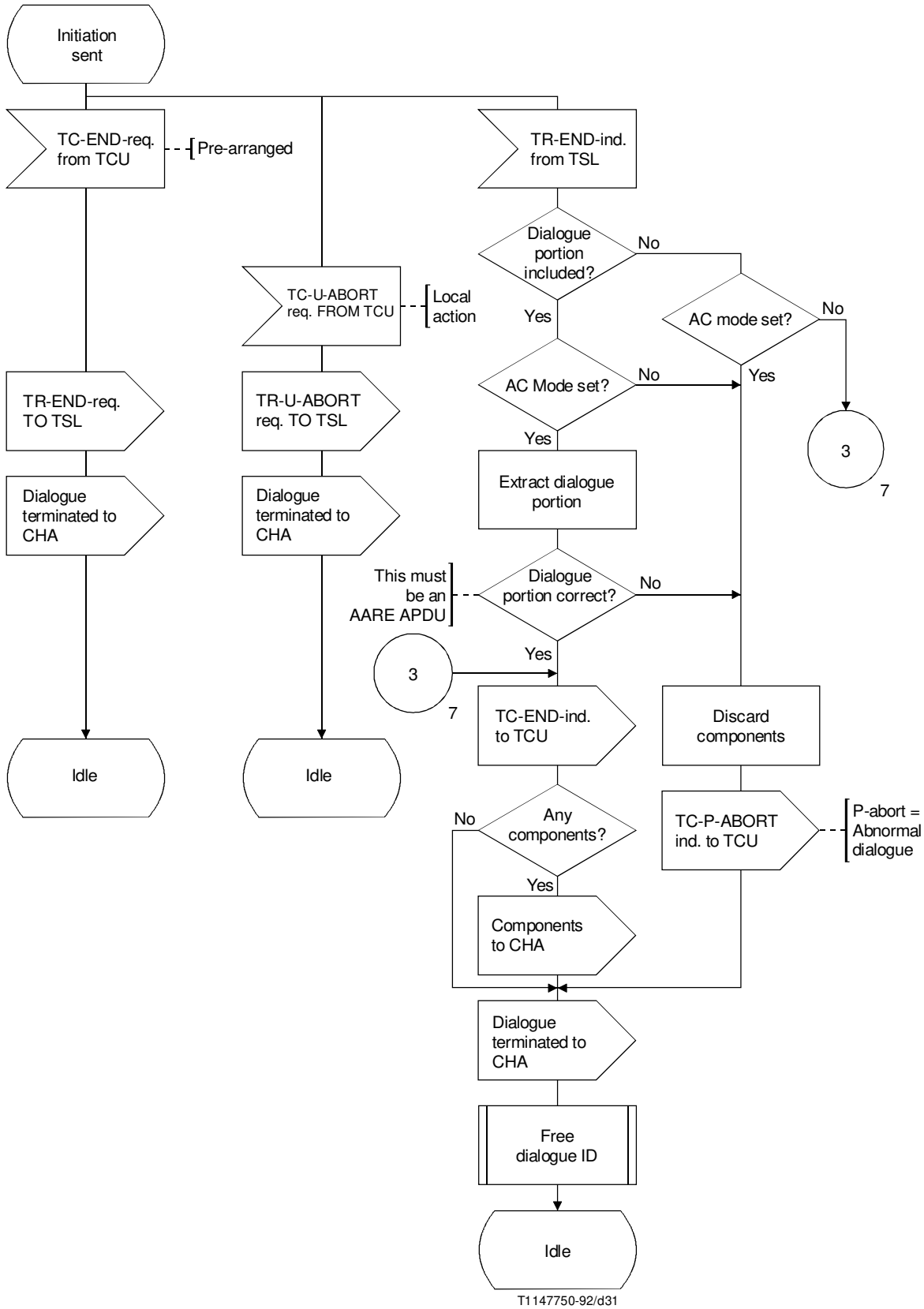


FIGURE A.5/Q.774 (sheet 7 of 11)

Dialogue handling at CSL

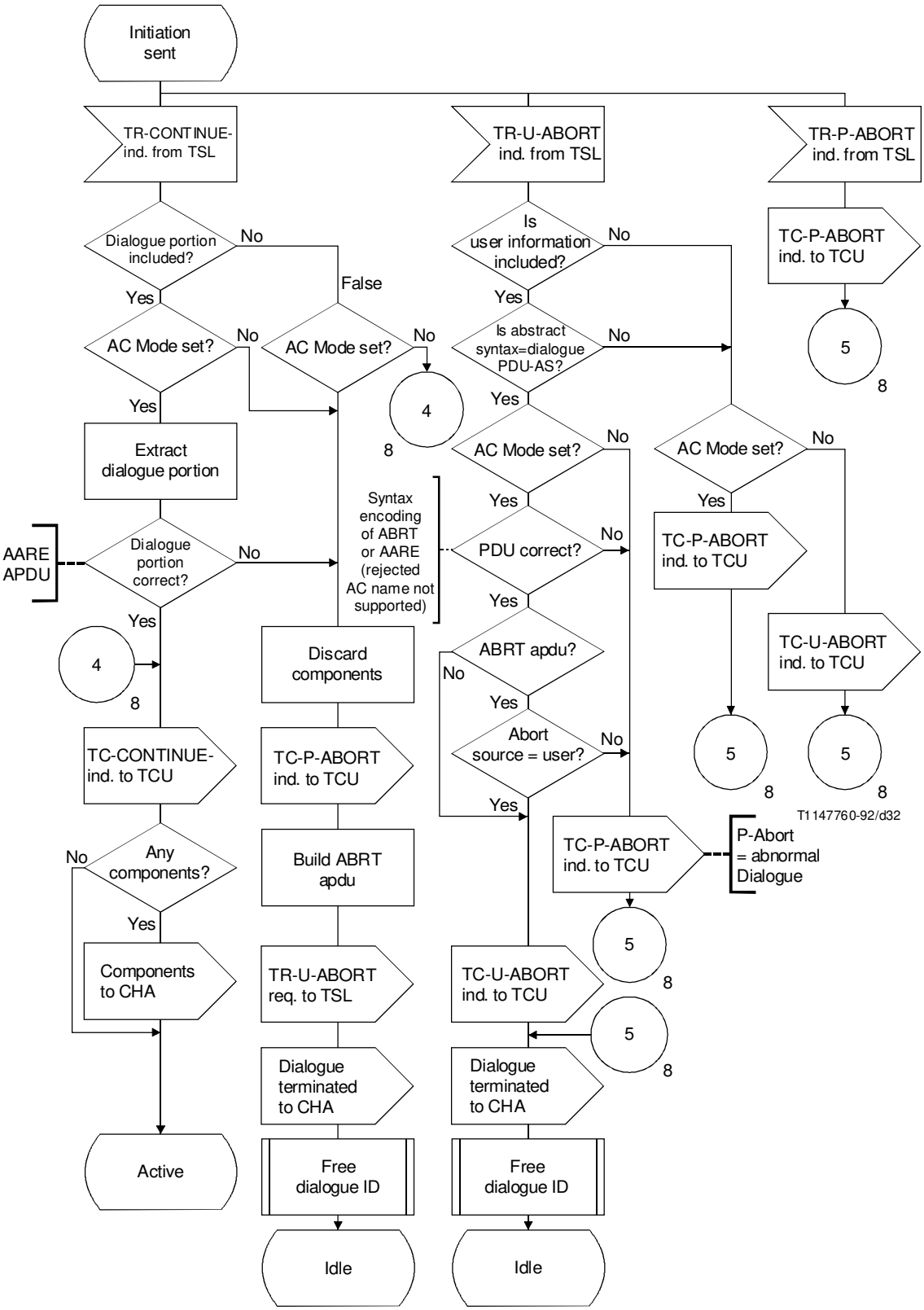


FIGURE A.5/Q.774 (sheet 8 of 11)

Dialogue handling at CSL

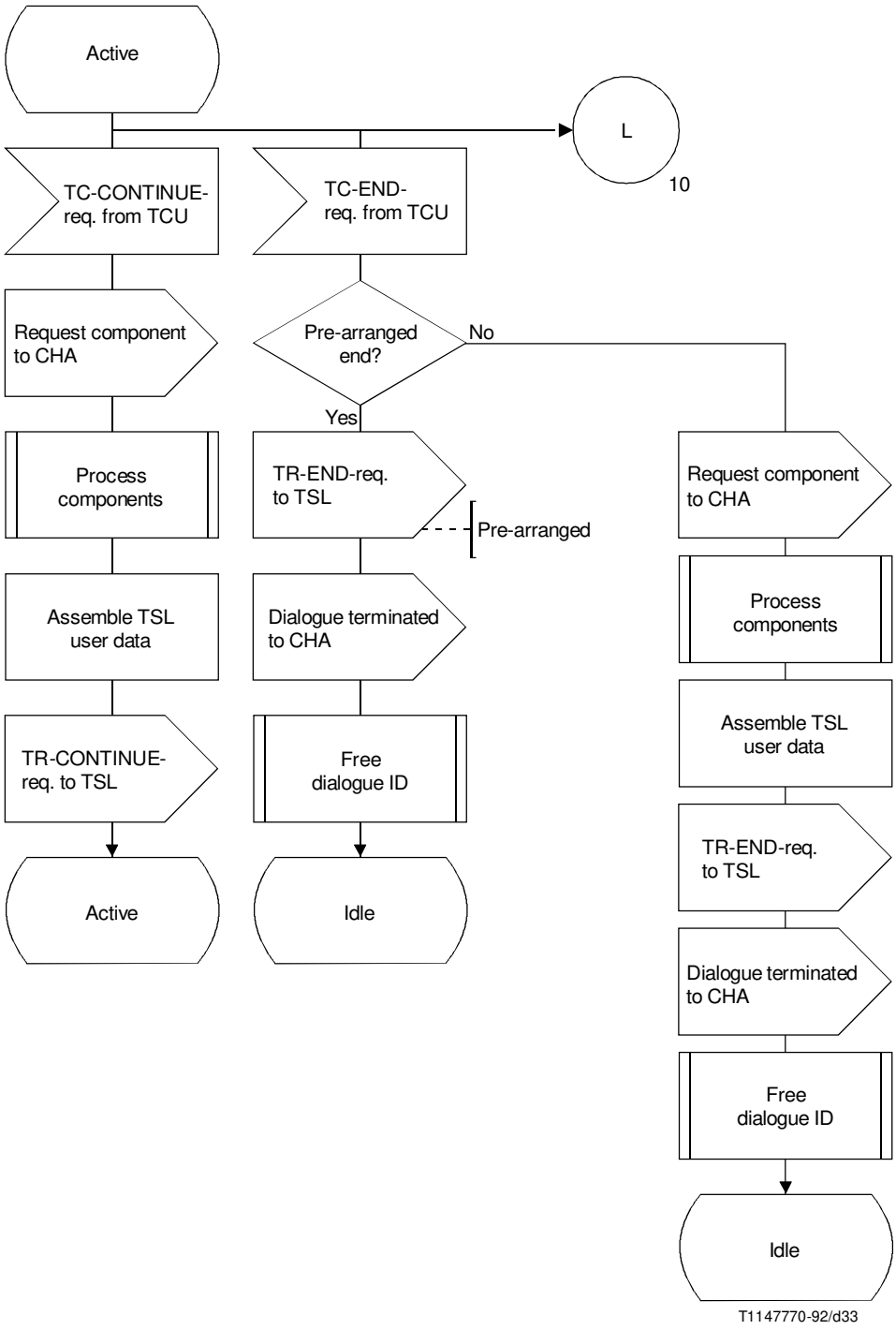


FIGURE A.5/Q.774 (sheet 9 of 11)

Dialogue handling at CSL

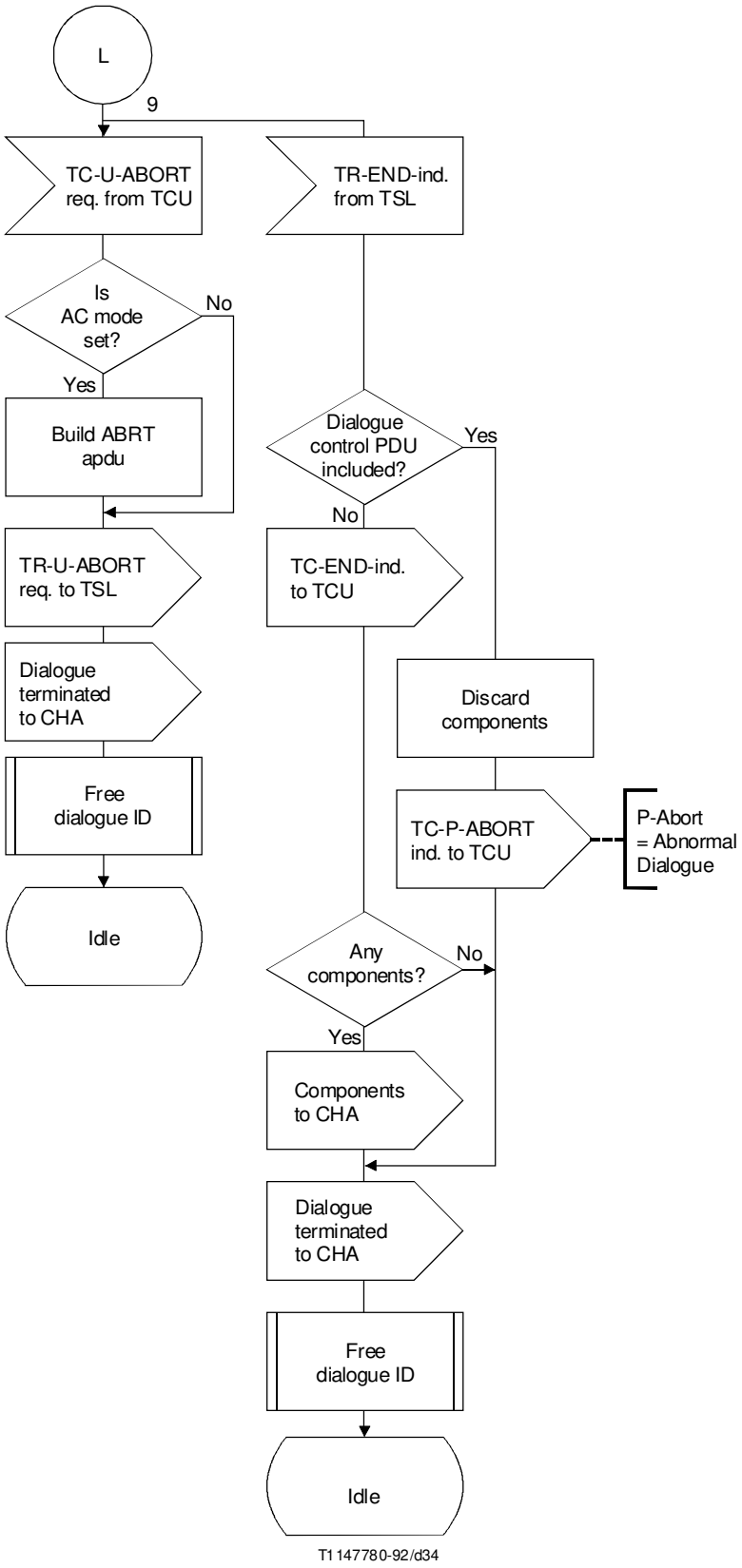


FIGURE A.5/Q.774 (sheet 10 of 11)

Dialogue handling at CSL

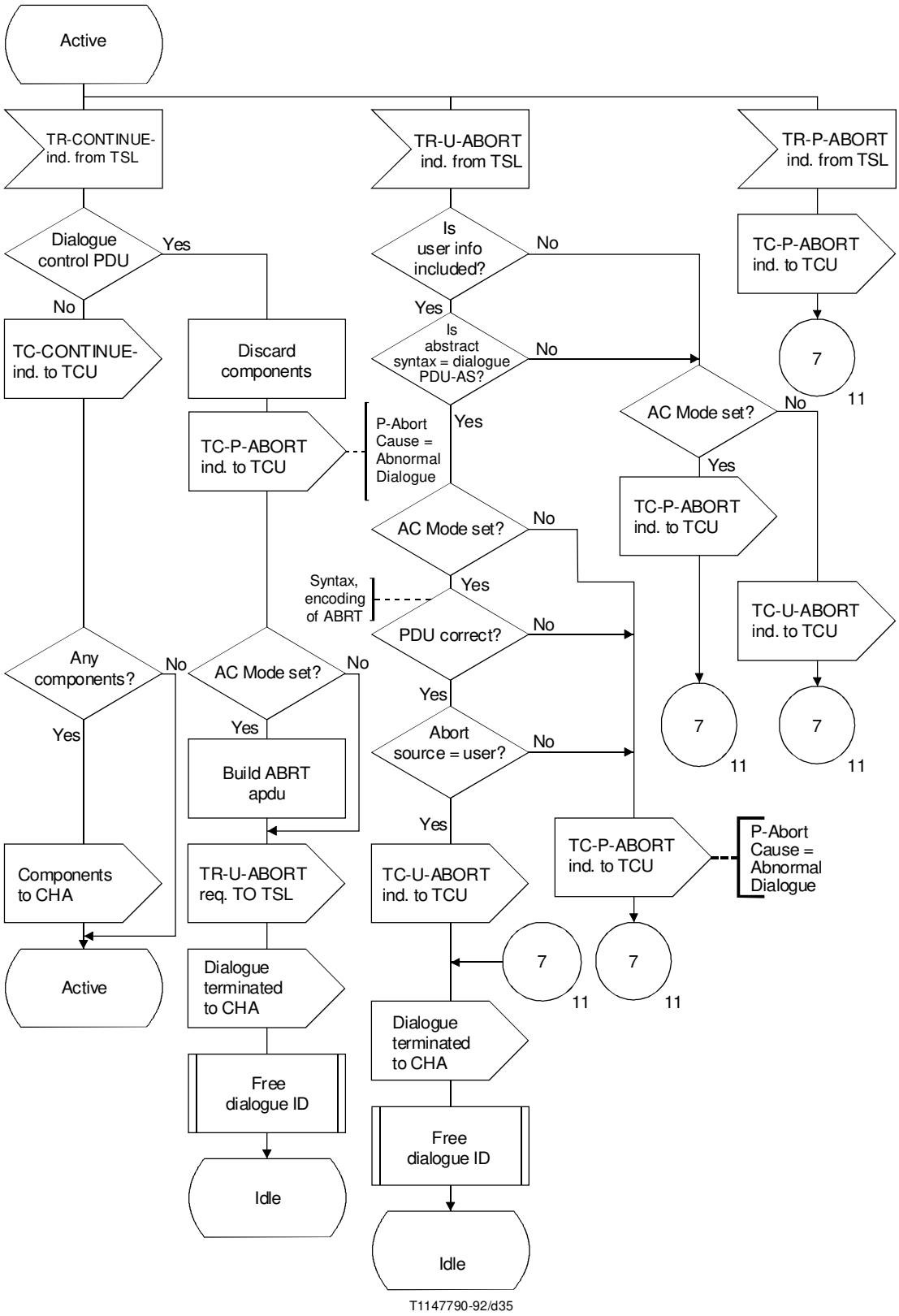


FIGURE A.5/Q.774 (sheet 11 of 11)

Dialogue handling at CSL

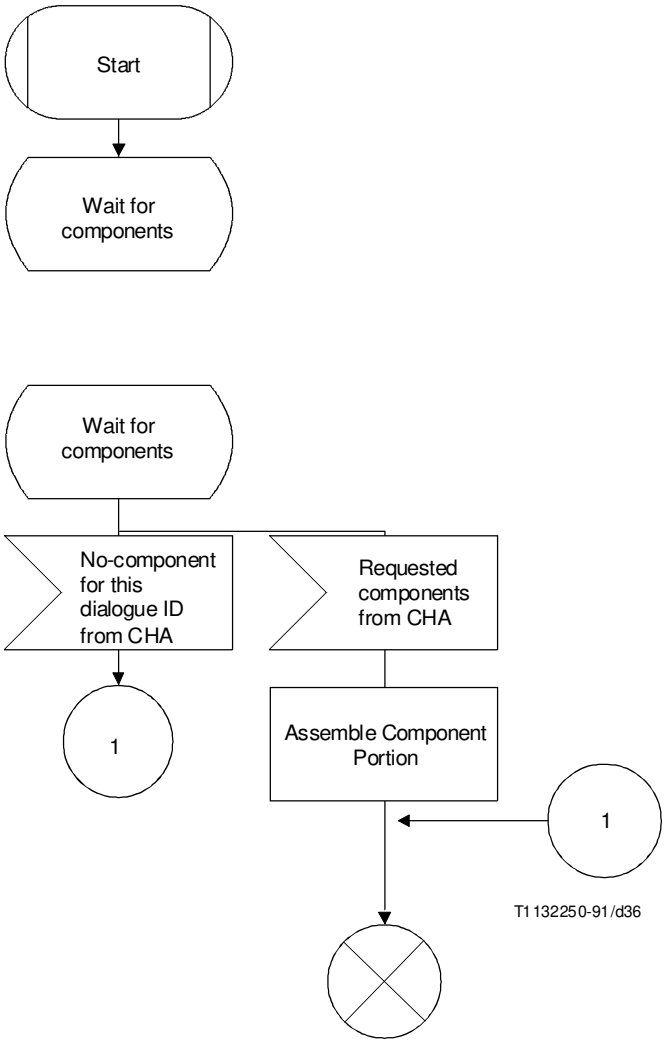
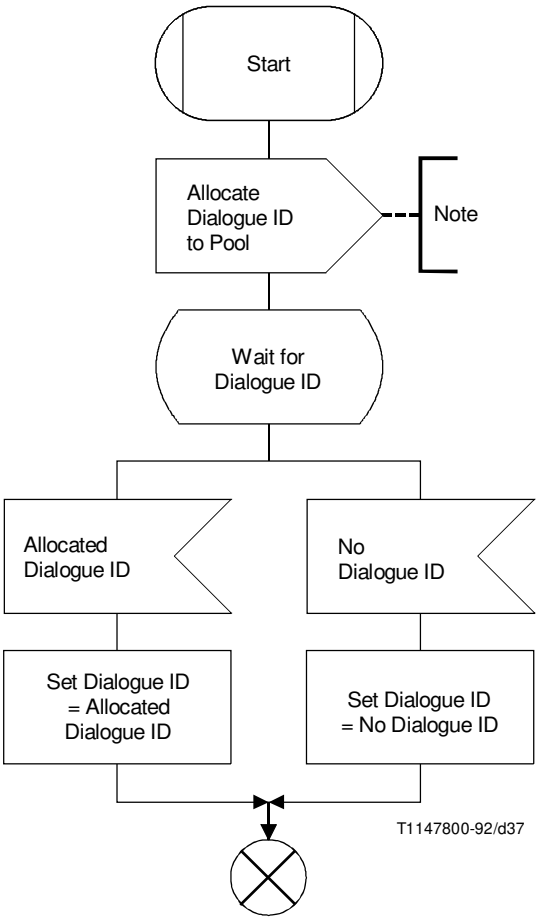


FIGURE A.5 *bis*/Q.774

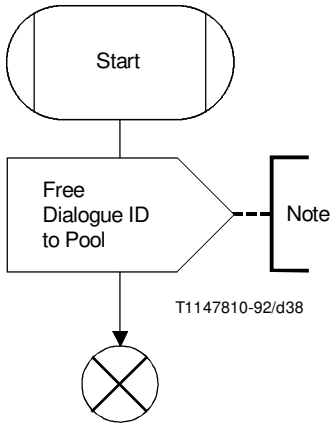
Procedure process components



NOTE – The pool realisation is implementation-dependent. The case of "No Dialogue ID" is not described in the calling process as the handling of this situation is an internal matter.

FIGURE A.5 *ter*/Q.774

Procedure ASSIGN DIALOGUE ID



NOTE – The pool realisation is implementation-dependent.

FIGURE A.5 *quater*/Q.774

Procedure FREE DIALOGUE ID

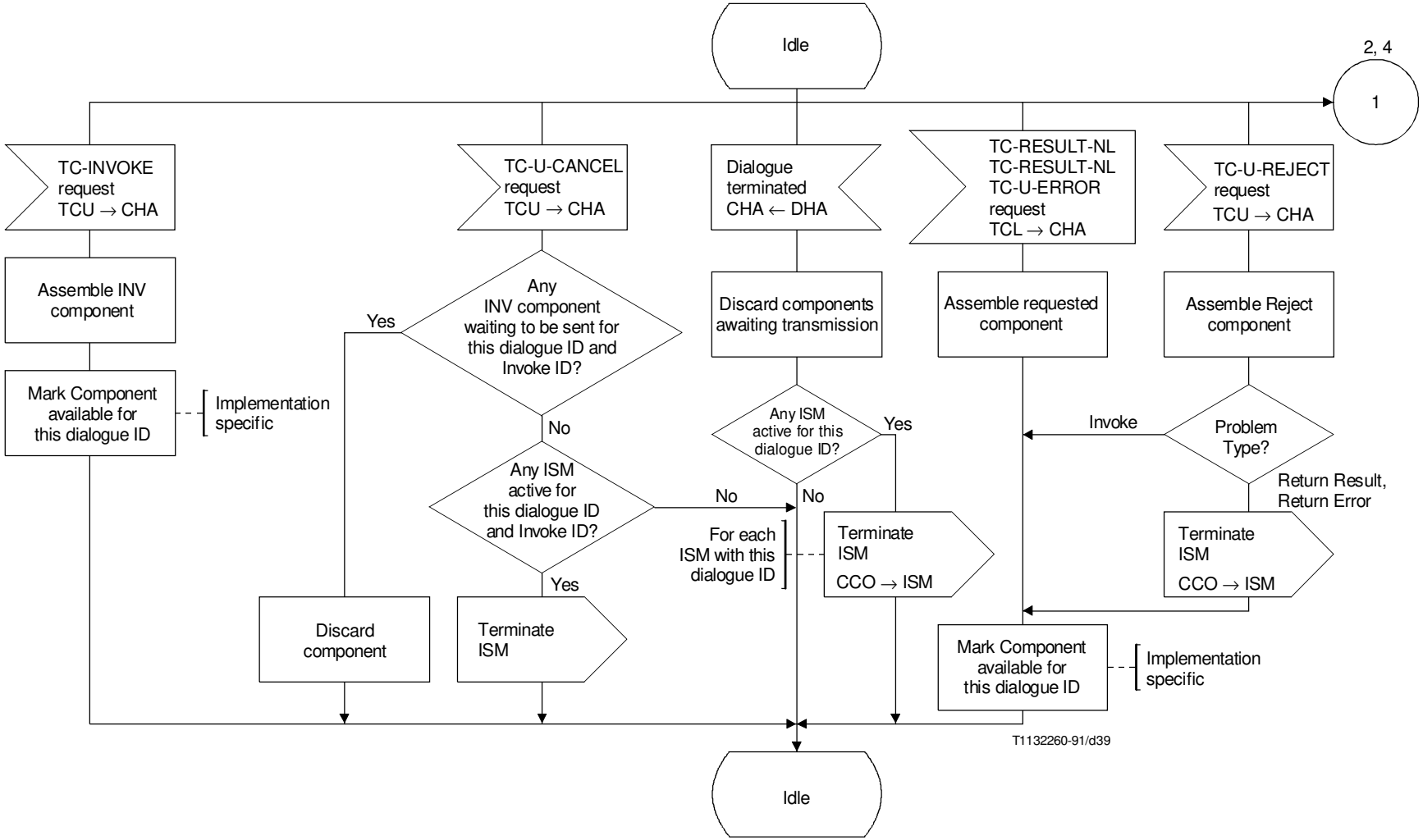
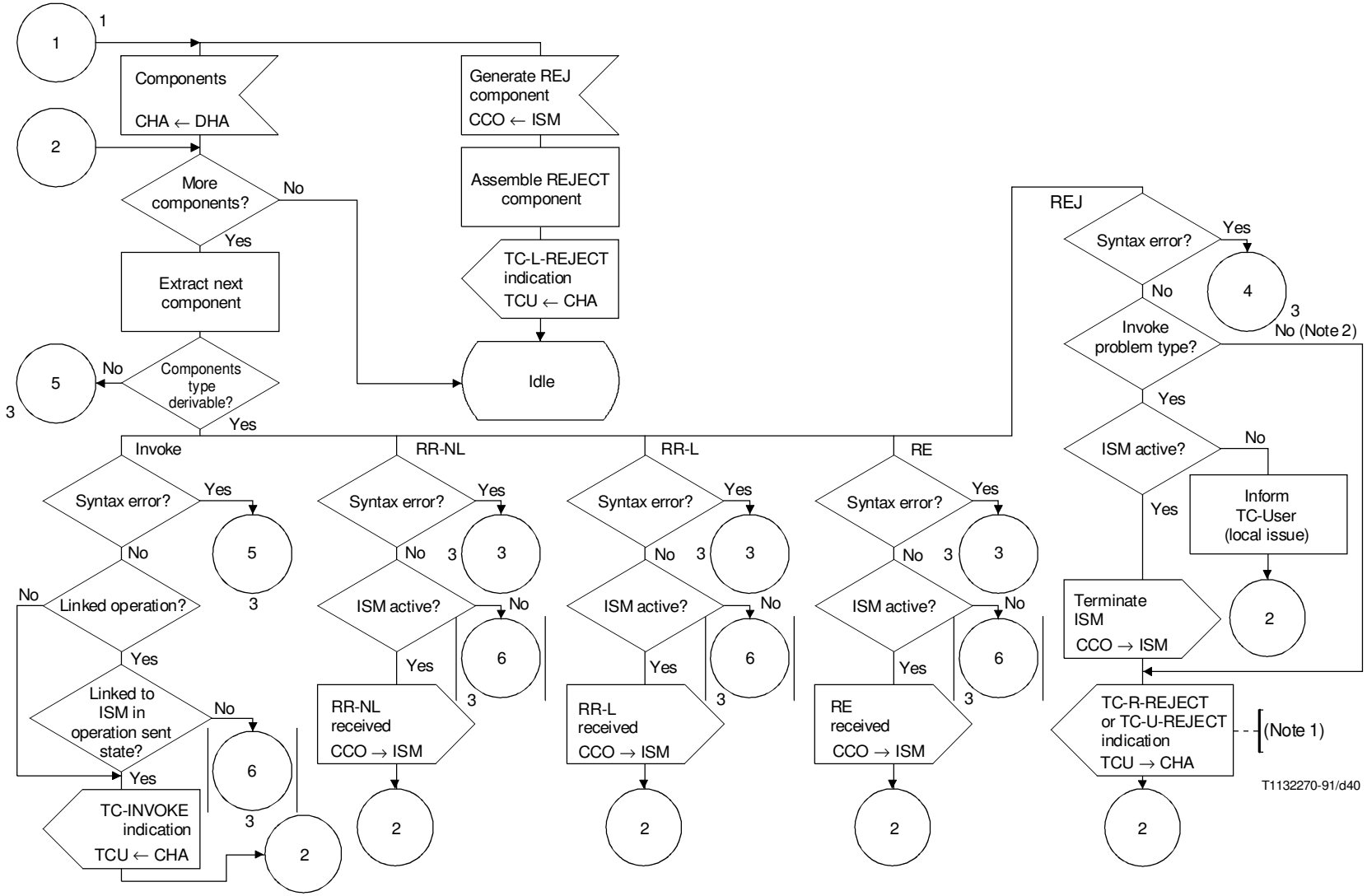


FIGURE A.6/Q.774 (sheet 1 of 4)

Component coordinator



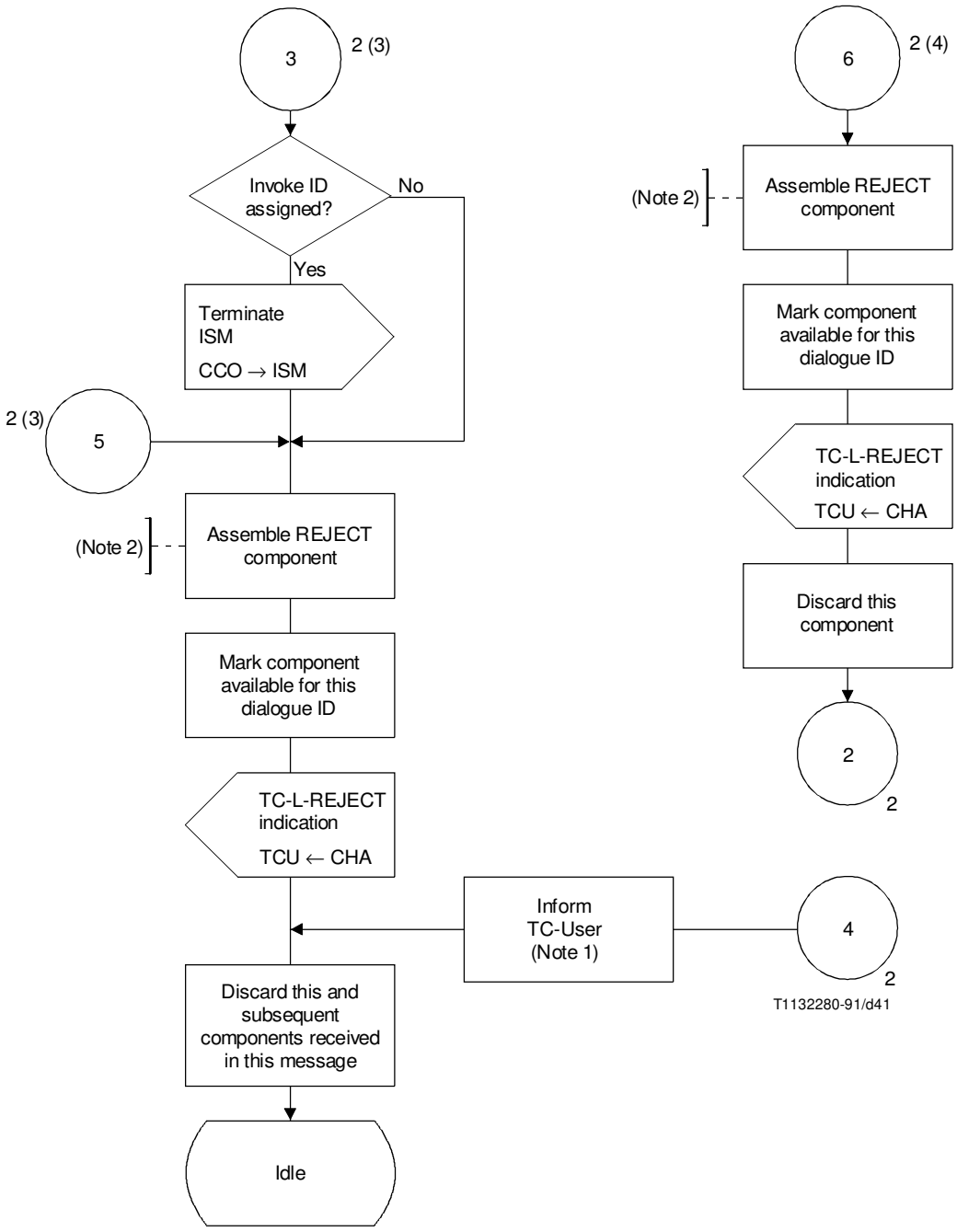
NOTES

1 Choice of TC-U-REJ or TC-R-REJ indication is determined from the Problem code value.

2 If "general" problem received, ID may refer to an ISM run by the other end. Therefore, a local ISM is not terminated.

FIGURE A.6/Q.774 (sheet 2 of 4)

Component coordinator



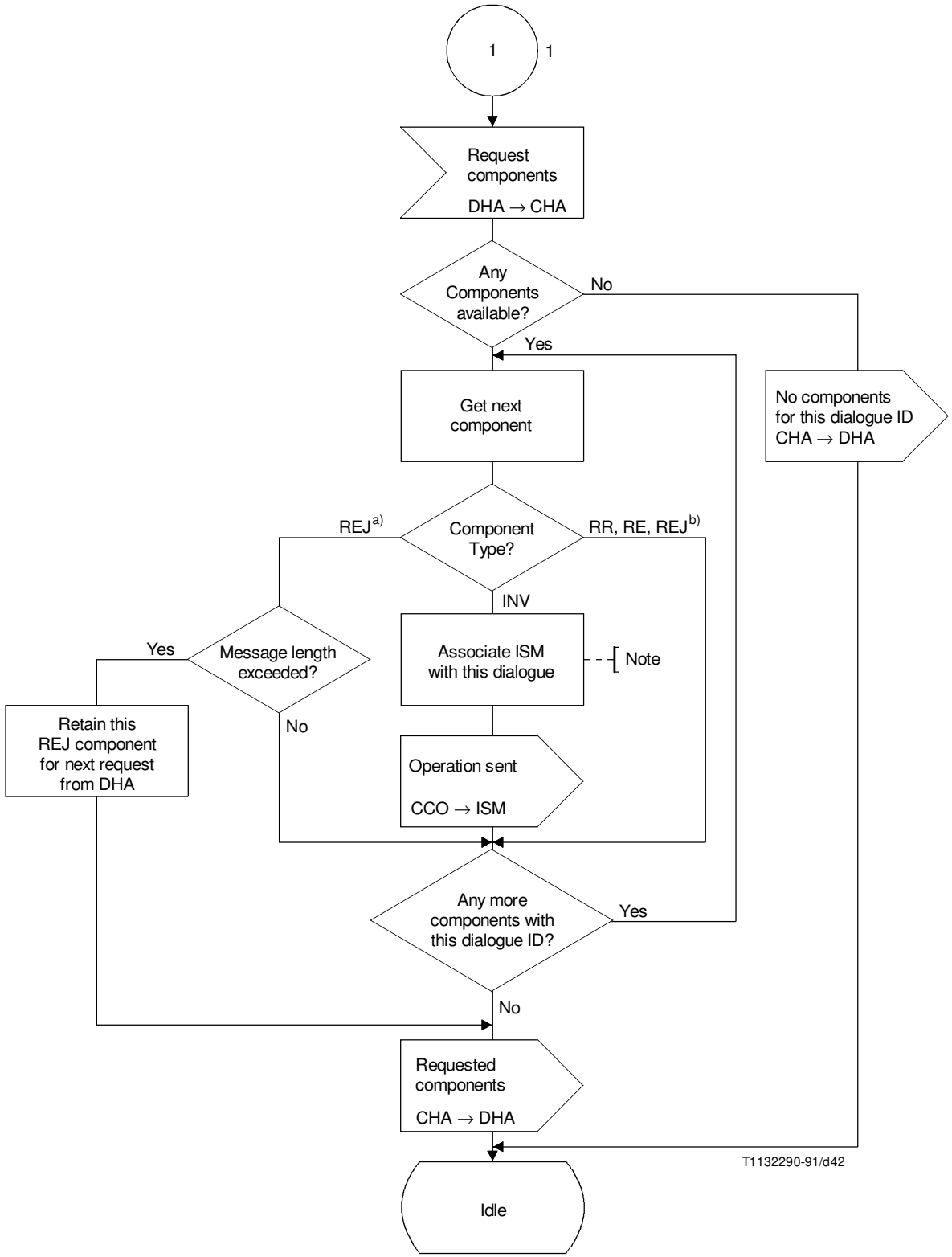
NOTES

1 Inform (local implementation) TC-user of syntax error in received Reject component.

2 Choose an appropriate problem code from values defined in Recommendation Q.772.

FIGURE A.6/Q.774 (sheet 3 of 4)

Component coordinator



a) Component sub-layer generated Reject.
b) TC-user generated Reject.

NOTE – Implementation specific.

FIGURE A.6/Q.774 (sheet 4 of 4)

Component coordinator

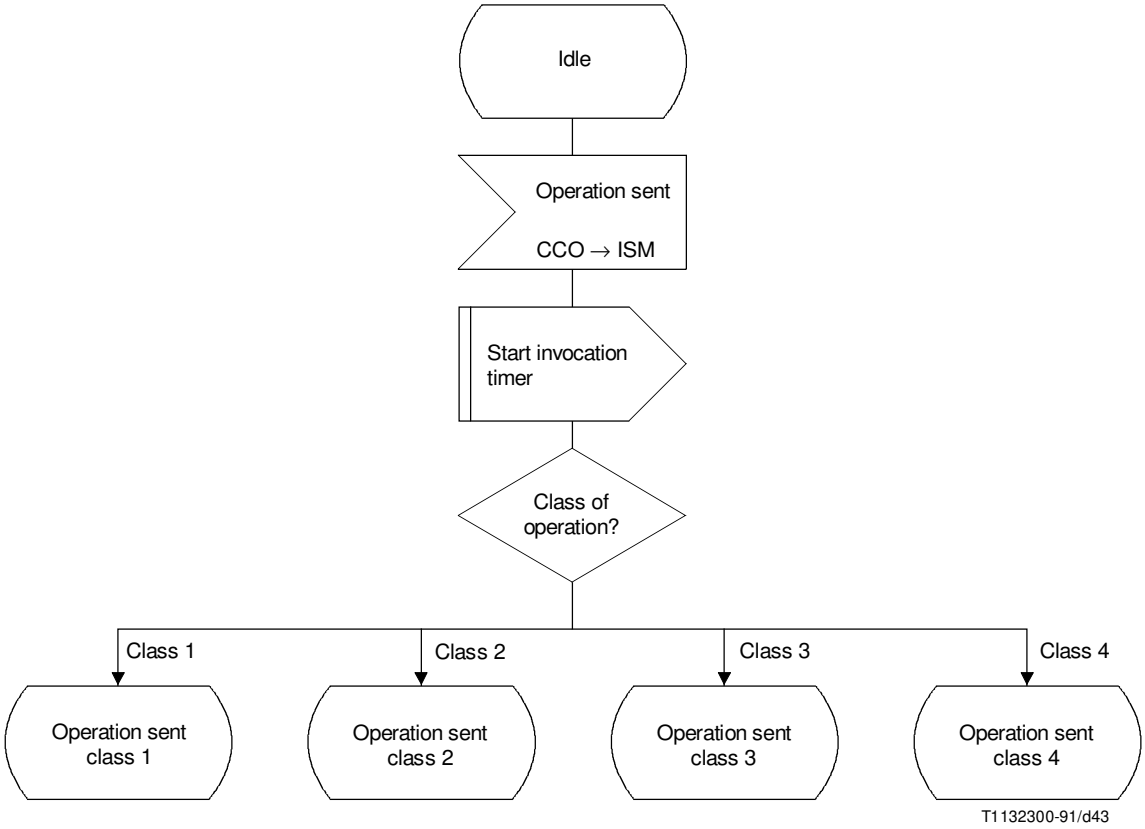


FIGURE A.7/Q.774 (sheet 1 of 6)

Invocation state machine

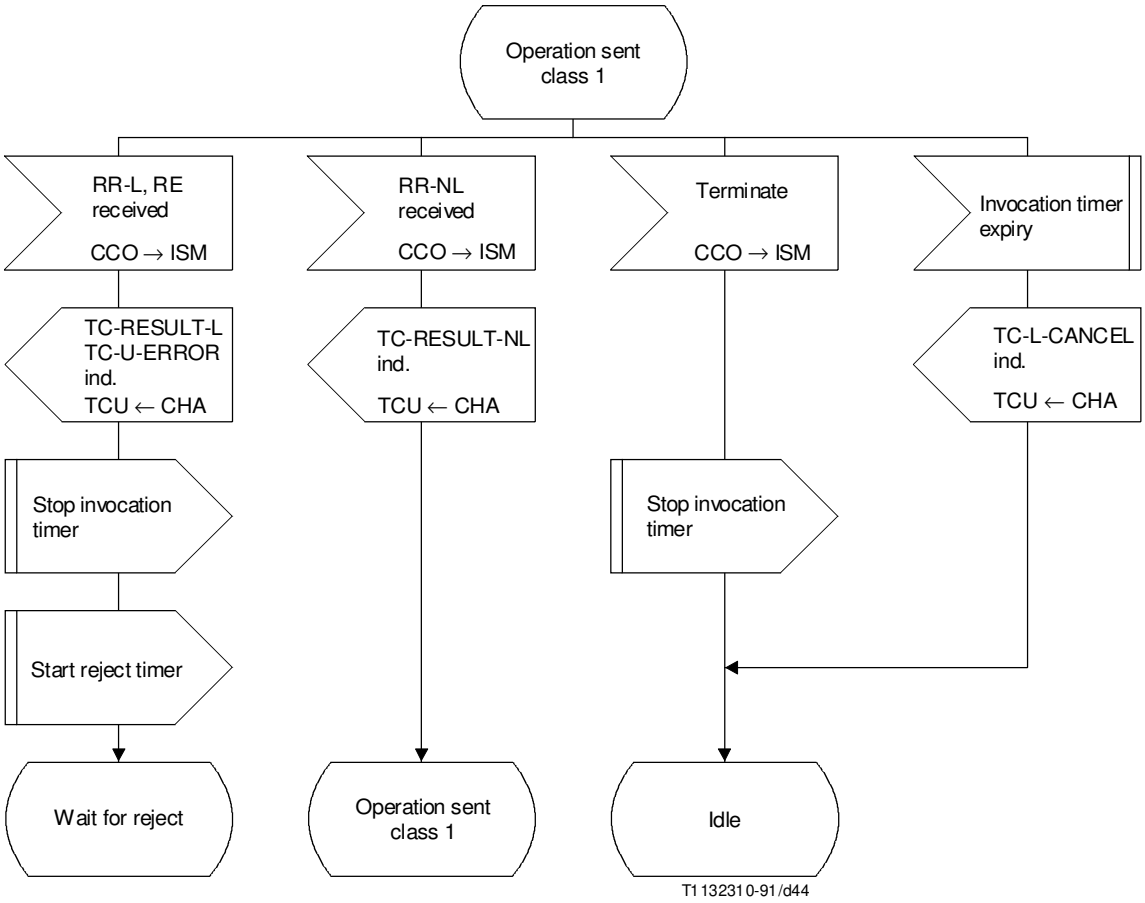


FIGURE A.7/Q.774 (sheet 2 of 6)

Invocation state machine

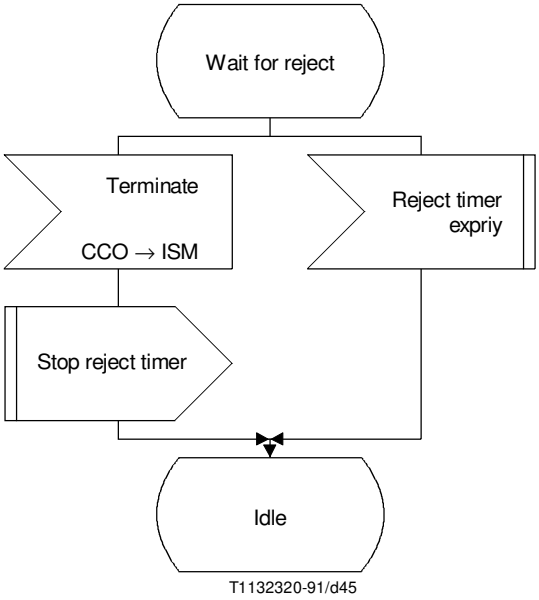


FIGURE A.7/Q.774 (sheet 3 of 6)

Invocation state machine

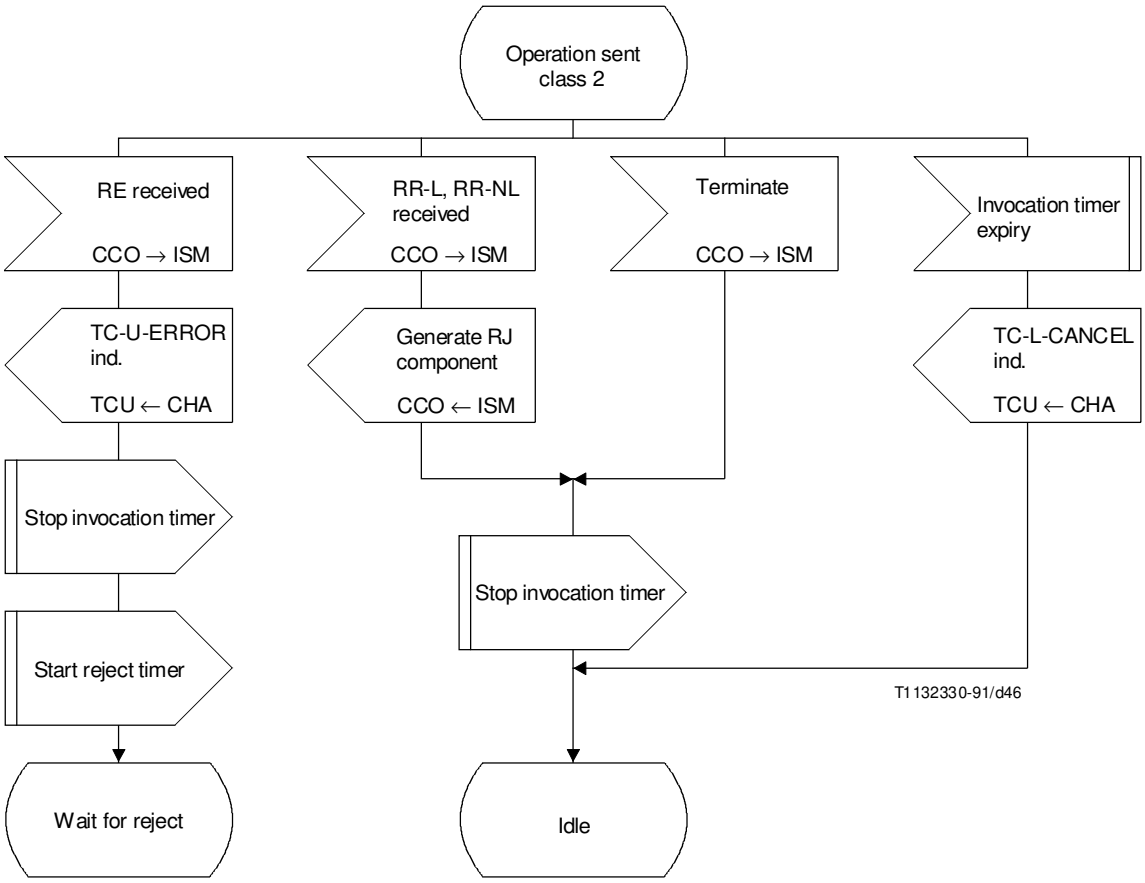


FIGURE A.7/Q.774 (sheet 4 of 6)
Invocation state machine

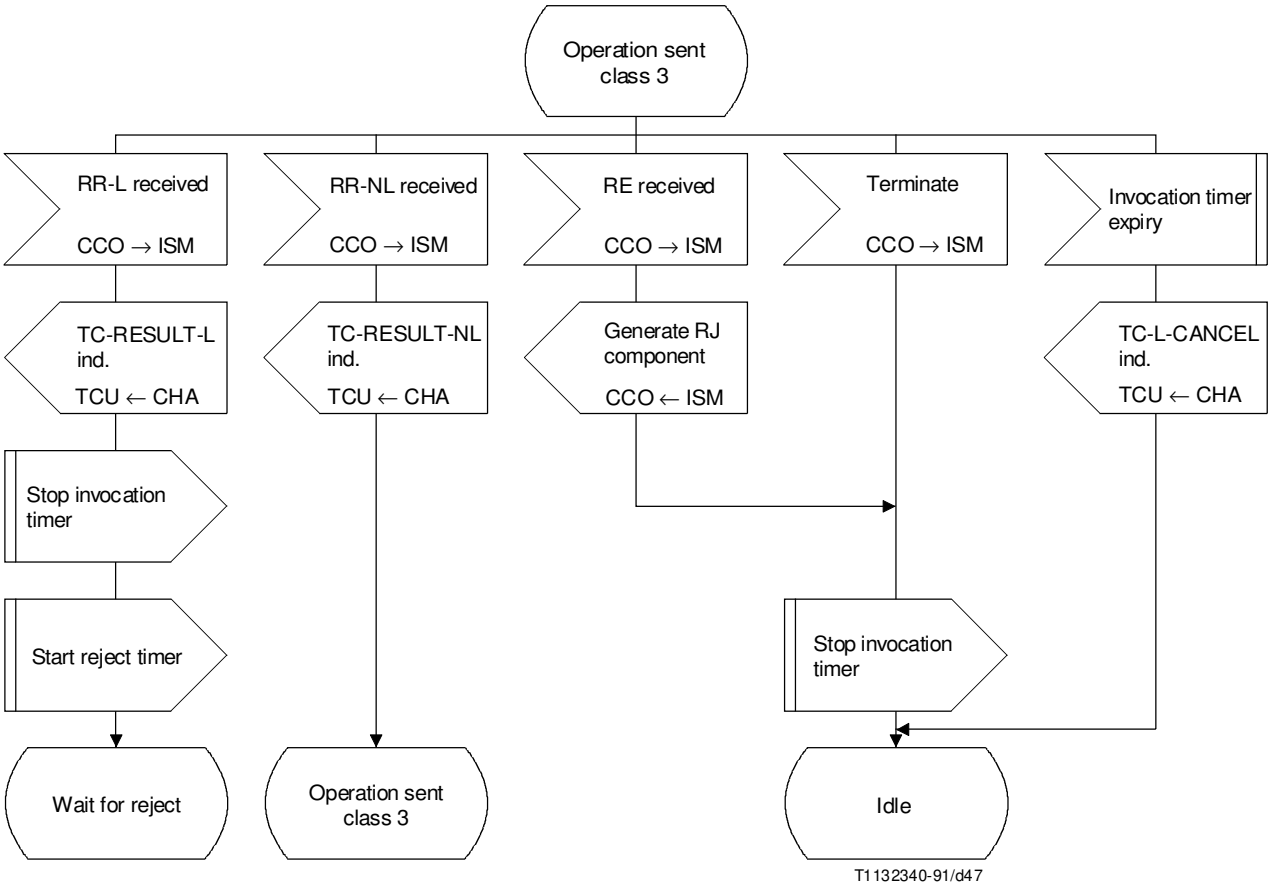


FIGURE A.7/Q.774 (sheet 5 of 6)

Invocation state machine

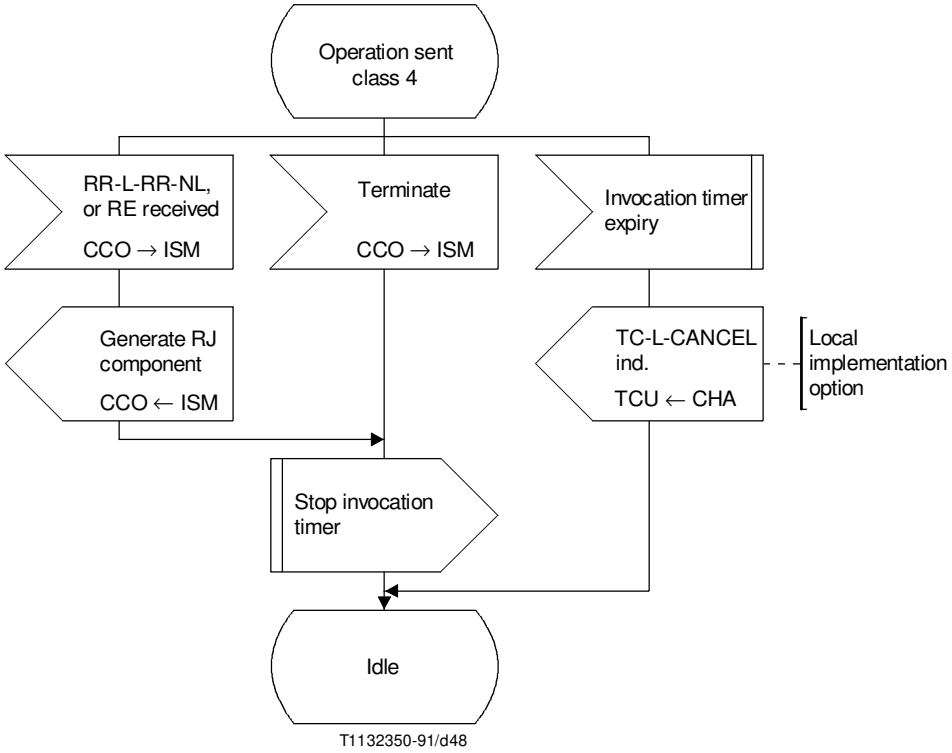


FIGURE A.7/Q.774 (sheet 6 of 6)

Invocation state machine

A.3 Abbreviations used in the SDL diagrams

CHA	Component handling
CCO	Component coordinator
CSL	Component sub-layer
DHA	Dialogue handling (component sub-layer)
INV	Invoke
IR	Initiation received state
IS	Initiation sent state
ISM	Invocation state machine
L	Last component
NL	Not last component
RE	Return error
REJ	Reject
RR	Return result
RR-L	Return Result Last
RR-NL	Return Result Not Last
SCCP	Signalling connection control part
TC	Transaction capabilities
TCAP	Transaction capabilities application part
TCO	Transaction Coordinator
TCU	TC-user
TSL	Transaction sub-layer
TSM	Transaction state machine
UNI	Unidirectional