INTERNATIONAL STANDARD

ISO/IEC 13650-2

First edition
1997-06-15

Information technology — Open Systems Interconnection — Conformance test suite for the OSI TP protocol —

Part 2:
Test management protocol specification

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — Suite d'essais de conformité pour le protocole OSI TP —

Partie 2: Spécification du protocole de gestion des essais

Reference number
ISO/IEC 13650-2:1997(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75% of the national bodies casting a vote.

International Standard ISO/IEC 13650-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open systems interconnection, data management and open distributed processing*.

ISO/IEC 13650 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — Conformance test suite for the OSI TP protocol:*

— *Part 1: Test suite structure and test purposes*

— *Part 2: Test management protocol specification*

Annexes A and B of this part of ISO/IEC 13650 are for information only.

Introduction

ISO/IEC 10026, Distributed Transaction Processing (OSI TP), is one of a set of standards produced to facilitate the interconnection of computer systems. It is related to other international standards in the set as defined by the Basic Reference Model for Open Systems Interconnection (ISO/IEC 7498-1). The Basic Reference Model subdivides the area of standardization for interconnection into a series of layers of specification, each of manageable size.

In order to ensure that implementations of ISO/IEC 10026-3 are conformant, testing standards have been defined using the methodology described in ISO/IEC 9646. The Coordinated test method has been chosen for the Abstract Test Suite. This particular test method requires the use of a Test Management Protocol, and this part of ISO/IEC 13650 defines the protocol.

This page intentionally left blank

Information technology — Open Systems Interconnection — Conformance test suite for the OSI TP protocol — Part 2: Test management protocol specification

1 Scope

This part of ISO/IEC 13650 provides:

A specification of a Test Management Protocol for use in conformance testing of Distributed Transaction Processing systems when the Coordinated Test Method of ISO/IEC 9646 is employed.

This Test Management Protocol does not rely on any specified Application Programming Interface (API) within the System Under Test, but rather assumes the existence of a mapping from the abstract service primitives issued by the upper tester to the functionality of the implementation under test realizing ISO/IEC 10026-3.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 13650. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 13650 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 7498-1:1994, *Information technology — Open Systems Interconnection — Basic Reference Model: The Basic Model.*

ISO/IEC 7498-3:1997, *Information technology — Open Systems Interconnection — Basic Reference Model: Naming and addressing.*

ISO/IEC 8649:1996, *Information technology — Open Systems Interconnection — Service definition for the Association Control Service Element.*

ISO/IEC 8822:1994, *Information technology — Open Systems Interconnection — Presentation service definition.*

ISO/IEC 8824-1:1995, *Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation.*

ISO/IEC 8825-1:1995, *Information technology — ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*

ISO/IEC 9545:1994, *Information technology — Open Systems Interconnection — Application Layer structure .*

ISO/IEC 9646-1:1994, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 1: General concepts.*

ISO/IEC 9646-2:1994, *Information technology — Open Systems Interconnection — Conformance testing methodology and framework — Part 2: Abstract Test Suite specification.*

ISO/IEC 10026-1:1992[1], *Information technology — Open Systems Interconnection — Distributed Transaction Processing — Part 1: OSI TP Model.*

ISO/IEC 10026-2:1996[1], *Information technology — Open Systems Interconnection — Distributed Transaction Processing — Part 2: OSI TP Service.*

ISO/IEC 10026-3:1996[1], *Information technology — Open Systems Interconnection — Distributed Transaction Processing — Part 3: Protocol specification.*

ISO/IEC 10026-4:1995, *Information technology — Open Systems Interconnection — Distributed Transaction Processing: Protocol Implementation Conformance Statement (PICS) proforma.*

ISO/IEC 10731:1994, *Information technology — Open Systems Interconnection — Basic Reference Model — Conventions for the definition of OSI services.*

1) Currently under revision.

3 Definitions

3.1 Terms defined in this International Standard

For the purposes of this part of ISO/IEC 13650, the following definitions apply.

a) Action-list: a sequence of actions to be performed by the upper tester in support of a dialogue used in a test case.

b) Done-needed: one or more of the following events has occurred during the current transaction and no subsequent TP-DONE req has been issued:

- a TP-ROLLBACK req;
- a TP-ROLLBACK ind;
- a TP-COMMIT ind;
- a TP-BEGIN-DIALOGUE(rejected, rollback = “true”) ind;
- a TP-P-ABORT(rollback = “true”) ind;
- a TP-U-ABORT(rollback = ”true”) ind;
- a TP-P-ABORT(rollback = “false”) ind on a coordinated dialogue during the termination phase of the current transaction; or
- a TP-U-ABORT(rollback = ”false”) ind on a coordinated dialogue during the termination phase of the current transaction.

c) Rollback-next-needed: on a chaining dialogue, a TP-P-ABORT ind was received after receipt of a TP-COMMIT ind, and there has been no subsequent TP-ROLLBACK ind received.

d) SYNC group: a set of Action-lists participating in synchronization.

e) Test session: the execution of a sequence of test cases from a test campaign.

3.2 Terms defined in other International Standards

3.2.1 Terms defined in ISO/IEC 7498-1

This part of ISO/IEC 13650 uses the following terms defined in ISO/IEC 7498-1:

a) application-protocol-data-unit;

b) open system.

3.2.2 Terms defined in ISO/IEC 9646

This part of ISO/IEC 13650 uses the following terms defined in ISO/IEC 9646-1:

a) Abstract Test Suite;

b) Coordinated Single Test Method;

c) Implementation Conformance Statement;

d) Implementation Under Test;

e) Multi-party testing context;

f) Single-party testing context;

g) System Under Test;

h) Test Management Protocol;

i) Test Management Protocol Data Unit;

j) Upper Tester.

3.2.3 Terms defined in ISO/IEC 10731

This part of ISO/IEC 13650 uses the following terms defined in ISO/IEC 10731:

a) request;

b) indication;

c) response;

d) confirm;

e) OSI-service primitive; primitive;

f) OSI-service-provider;

g) OSI-service-user.

4 Abbreviations

For the purposes of this part of ISO/IEC 13650, the following abbreviations apply.

APDU	Application Protocol Data Unit
ASN.1	Abstract Syntax Notation One
FU	Functional Unit
IUT	Implementation Under Test
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
SUT	System Under Test
TMP	Test Management Protocol
TMPDU	Test Management Protocol Data Unit
TP	(Distributed) Transaction Processing
U-ASE	User-Application Service Elements

5 Conventions

This part of ISO/IEC 13650 is guided by the conventions defined in ISO/IEC 10731, as they apply to the OSI TP service.

6 Specification of the TP test management protocol

The Transaction Processing Test Management Protocol (TMP) is designed for cooperative conformance testing between one or more lower testers and an upper tester. At any particular time, one unique lower tester shall be responsible for the transfer of TMP. This part of ISO/IEC 13650 defines the format the PDUs sent between this specific lower tester and upper tester, and the log records sent from the upper tester to the lower tester during a TP test campaign.

Information is exchanged between one of the lower testers and the upper tester in the form of TMPDUs. The TMPDUs are carried as parameters of U-ASE APDUs on TP dialogues. There are five TMPDUs which may be sent during a test case, namely:

- TM-TEST-PROCEDURE
- TM-START-TEST
- TM-REQUEST-LOG
- TM-TEST-LOG
- TM-END-TEST-SESSION

6.1 Test management phases

Each test case is structured in three phases, characterized by separate dialogues:

- Test Procedure Establishment Phase
- Test Execution Phase
- Test Result Collection Phase

The lower tester and the upper tester transmit the TMPDUs appropriate to each phase. Figure 1 depicts the overall control flow between phases, for the cases of one and more than one test in a test session, respectively. Each phase is described below. Subclause 6.2 describes dialogue management for the Test Procedure Establishment and Test Results Collection Phases.

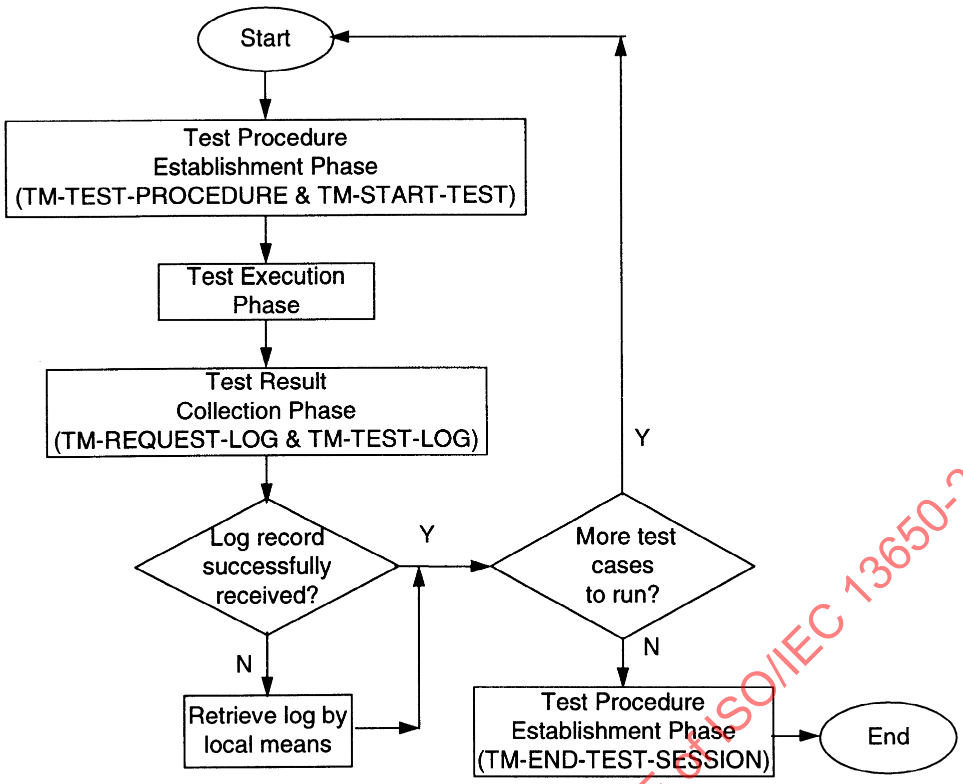
Start
Test Procedure Establishment Phase (TM-TEST-PROCEDURE & TM-START-TEST)
Test Execution Phase
Test Result Collection Phase (TM-REQUEST-LOG & TM-TEST-LOG)
Log record successfully received?
Y
N
Retrieve log by local means
More test cases to run?
Y
N
Test Procedure Establishment Phase (TM-END-TEST-SESSION)
End

Figure 1 — Control flow for a test session

6.1.1 Test procedure establishment phase

The Test Procedure Establishment Phase always starts with the establishment of a dialogue between the lower tester and the IUT, as described in 6.2. After that there are two different ways to proceed depending on whether the Test Procedure Establishment Phase is used to either initiate execution of a test case or to terminate the test session.

If the Test Procedure Establishment Phase is used to initiate execution of a test case, the lower tester uses the dialogue established at the beginning to send a TM-TEST-PROCEDURE PDU which contains complete information on how the upper tester is to behave during this test case. After that the lower tester sends a TM-START-TEST PDU on the dialogue established at the beginning, thus informing the upper tester that execution of the test case is to be started.

If the Test Procedure Establishment Phase is used to terminate the test session, the lower tester uses the dialogue established at the beginning to send an TM-END-TEST-SESSION PDU, thus indicating that no further test cases are to be executed.

In both cases, the Test Procedure Establishment Phase concludes with termination of the dialogue established at the beginning, as described in 6.2. If the Test Procedure Establishment Phase was used to terminate the test session, the upper tester terminates together with the Test Procedure Establishment Phase.

6.1.2 Test execution phase

During the Test Execution Phase, a single test case is executed. The behaviour of the upper tester within this test case is determined by the contents of the TM-TEST-PROCEDURE PDU which was sent during the preceding Test Procedure Establishment Phase.

The protocol exchanges required by the test case occur only on dialogues which are established between the lower testers and the IUT, as part of the test case itself. No TMPDUs are exchanged during this phase.

During the Test Execution Phase, the upper tester records the service primitives that it sends and receives in a log file. The Test Execution Phase ends when all Action-lists have been executed, or when the default procedure is invoked.

6.1.3 Test results collection phase

The Test Results Collection Phase starts with establishment of a dialogue between the lower tester and the IUT, as described in 6.2. After that, the lower tester uses this dialogue to send a TM-REQUEST-LOG PDU in order to ask the upper tester to send the log file recorded during the preceding Test Execution Phase. The upper tester responds by sending a TM-TEST-LOG PDU containing the requested log file to the lower tester using the same dialogue.

The Test Results Collection Phase concludes with termination of the dialogue established at the beginning, as described in 6.2. See figures A.1 and A.2 for examples of the Test Results Collection Phase.

6.2 Dialogue management

During both the Test Procedure Establishment and Test Results Collection phases, a dialogue is required for TMP transfer. Establishment and termination of these dialogues is subject to the following rules.

It is a local matter how the IUT and UT cooperate to achieve dialogue management, provided the resultant behaviour satisfies the following requirements.

When establishing or terminating a dialogue for transfer of TMP PDUs, the IUT shall:

a) behave in a manner consistent with the capabilities given in the PICS and PIXIT documents;

b) act as a dialogue acceptor, if it has this capability; otherwise as a dialogue initiator;

c) after the TMP has been transferred, the IUT shall:

- receive a TP-DEFER(end-dialogue)-RI and commit the current transaction, if the coordination level is commitment and the IUT is an acceptor of the dialogue;
- issue a TP-DEFER(end-dialogue)-RI and commit the current transaction, if the coordination level is commitment and the IUT is an initiator of the dialogue;
- receive a TP-END-DIALOGUE-RI, if the coordination level is none.

6.3 Protocol data unit descriptions and effects

6.3.1 TM-TEST-PROCEDURE PDU

The TM-TEST-PROCEDURE PDU contains one or more lists of actions to be executed by the upper tester. Each such list is called an Action-list. The upper tester is intended to support multi-party testing so a TM-TEST-PROCEDURE PDU will always contain one Action-list for each lower tester involved in a test case. Each Action-list is composed of a sequence of actions. An action is one of the following:

SEND

RECEIVE

MATCH

SYNC

TIMER

END

which are described in clause 7.

All of the actions associated with one particular dialogue, channel or association are contained in a single Action-list. it is possible that an Action-list contains the actions associated with two or more dialogues (channel, or associations) if the lifetimes of these dialogues (channels, or associations) do not overlap. However, it is not allowed that, within one Action-list, two actions associated with the same dialogue are separated by an action associated with a different dialogue.

Actions for sending or receiving service primitives which are not associated with one particular dialogue may be contained in any Action-list.

On receipt of a TM-TEST-PROCEDURE PDU, the upper tester erases the log file relating to the previously executed test, if any.

The TM-TEST-PROCEDURE has two parameters:

Normal-default-behaviour is a mandatory parameter, the value of which is of type BOOLEAN. If its value is set to true, the normal default behaviour is performed if the default procedure is invoked (see 7.3.1); otherwise , the test execution phase terminates and the IUT is returned to the idle state.

Action-lists is a mandatory parameter which contains a sequence of Action-lists corresponding to the dialogues, channels and associations used in the test case. An individual Action-list contains all of the actions associated with a particular lower tester.

6.3.2 TM-START-TEST PDU

The TM-START-TEST PDU is a command to the upper tester to begin execution of all currently stored Action-lists. It has no parameters.

6.3.3 TM-REQUEST-LOG PDU

The TM-REQUEST-LOG PDU is a command to the upper tester to issue the TM-TEST-LOG PDU containing the accumulated log information. It has no parameters.

6.3.4 TM-TEST-LOG PDU

The TM-TEST-LOG PDU contains a sequence of log records developed by the upper tester during the course of a test case. It has two parameters:

Unexpected-event-occurred is a mandatory parameter, the value of which is of type BOOLEAN. If its value is set to true, the UT has detected an error, otherwise no error was detected by the UT.

Log-records is an optional parameter, the value of which is a sequential log of the events (if any) that occurred at the boundary between the upper tester and the IUT during the execution of a test case. For each event, the time of occurrence, the service primitive involved, an indication of whether it was expected or not, and optional extra information are recorded.

6.3.5 TM-END-TEST-SESSION PDU

The TM-END-TEST PDU is a command to the upper tester to terminate operation at the end of a test session. It is a local matter whether to keep or destroy any outstanding log files. It has no parameters.

7 Upper tester execution rules

The Test Management Protocol specified in clauses 6 and 10 describes the TMPDUs and exchanges which occur between an upper tester and one or more lower testers. The TMPDUs are transferred in the Test Procedure Establishment and Test Results Collection phases of the process. The Upper Tester's actions during the Test Execution phase are governed by commands received on a TM-TEST-PROCEDURE PDU. These commands are organized as one or more Action-lists. If there is only one Action-list, the test case is an instance of single-party testing, otherwise it is an instance of multi-party testing.

Each Action-list is a sequence of actions. In the following, the semantics of the different types of actions are given and the order in which the actions from the Action-lists are executed is described both for the case of single-party testing and for the case of multi-party testing.

7.1 Semantics of TM-TEST-PROCEDURE actions in the single-party testing context

In the case of single party testing, the TM-TEST-PROCEDURE PDU contains a single Action-list.

7.1.1 SEND action

Parameters:	(M) A request or response TP service primitive, with its necessary parameters.
Effect:	An instruction to the upper tester to stimulate the specified outgoing (Request or Response) TP Service Primitive.

The upper tester issues the service primitive specified in the parameter of the action to the IUT and logs this service primitive.

7.1.2 RECEIVE action

Parameters:	None.
Effect:	An instruction to the upper tester to suspend execution pending receipt of an incoming (Indication or Confirm) TP Service Primitive.

Instructs the upper tester to suspend execution of the Action-list pending the next incoming TP primitive. On receipt of an incoming service primitive, the service primitive is logged. Since RECEIVE and MATCH are always used together, the next event in an Action-list shall always be a MATCH. If no primitive at all is received then the upper tester will remain suspended indefinitely. Therefore it is the responsibility of the Abstract Test Case writer to ensure that such situations are handled correctly by the lower tester, e.g. by the use of a drop-dead timer.

7.1.3 MATCH action

Parameters:	(M) Expected primitive, with its parameters,
	(O) Upper tester action if the MATCH is true (match-case),
	(O) Upper tester action if the MATCH is false (unmatch-case).
Effect:	An instruction to the upper tester to determine whether the incoming primitive from the previous RECEIVE action matches the primitive expected and the definition of the subsequent action to be taken depending on the result of MATCH.

The MATCH action may only occur immediately after a RECEIVE action, or as the very first of the actions for the unmatch-case of an enclosing MATCH action.

The unmatch-case of a MATCH action has to contain either no explicit actions at all or another MATCH action as its very first action.

The effect of the MATCH action is that the service primitive obtained by the RECEIVE action is compared with that specified in the MATCH action and subsequent actions are then performed depending on the result of the

MATCH action. The actions may either be stated explicitly as part of the MATCH action, or they may be the implicit actions as described below, and as summarised in Table 1.

Table 1 — MATCH

MATCH Result	Explicit Actions Specified	Explicit Actions Not Specified
True	Perform the actions specified in the match-case and continue with the next action following the MATCH action.	Continue with the next action following the MATCH action.
False	Perform the actions specified in the unmatch-case and continue with the next action following the MATCH action.	Perform the default procedure (see 7.3.1).

7.1.4 SYNC action

SYNC has no effect in the single-party test context. Consequently, ATS specifiers should not use SYNC when writing single-party test cases.

7.1.5 TIMER action

Parameters: (M) Integer specifying the number of seconds.

Effect: An instruction to the upper tester to delay execution of all Action-lists for the specified amount of time.

The upper tester suspends execution of all Action-lists for the duration specified by the parameter. When the timer expires, execution continues with the next action after the TIMER action.

7.1.6 END action

Parameters: None.

Effect: An instruction to the upper tester to terminate execution of the current Action-list.

The upper tester terminates execution of the current Action-list. If any actions follow the END, they will not be executed.

7.2 Semantics of TM-TEST-PROCEDURE actions in the multi-party testing context

Corresponding to each lower tester, there exists an Action-list which is stored in the upper tester. Action-lists are executed in the manner described in this subclause. As the effect of executing an individual action in a multi-party testing context is for all types of actions, except the SYNC action, the same as in a single-party testing context, emphasis will be on the order in which actions from different Action-lists are executed.

All of the Action-lists are identified by the upper tester in a numbered sequence, corresponding to the order in which they were sent to the upper tester on the TM-TEST-PROCEDURE PDU. The upper tester executes each Action-list in this sequence (the first one follows the last one, to create a continuous cycle), as described below for each action.

7.2.1 SEND action

Subclause 7.1.1 also applies in the multi-party test context. After executing the SEND action, processing continues with the next action in the same Action-list.

7.2.2 RECEIVE action

The UT shall queue service primitives received from the IUT. These service primitives shall be processed in the order in which they are received.

When a RECEIVE action is first encountered, the current Action-list is suspended, and execution passes to the next Action-list. When all Action-lists have been suspended the queue is polled.

If the service primitive at the head of the queue relates to a single dialogue or channel, and its Action-list is suspended at a RECEIVE action, then execution shall resume with that Action-list. Otherwise the default procedure shall be invoked.

If the service primitive does not relate to a single dialogue or channel, then execution shall resume with the Action-list which is suspended at a RECEIVE action, and whose subsequent MATCH action expects this service primitive. If there is no such Action-list, or if there is more than one, the default procedure shall be invoked.

7.2.3 MATCH action

Subclause 7.1.3 also applies in the multi-party test context

7.2.4 SYNC action

Parameters: (O) Groupno: identifies the Action-lists which are suspended pending synchronization. The parameter is a bitstring denoting which Action-lists are to be synchronized, where '1' denotes "synchronize" and '0' denotes "don't synchronize". The zeroth bit corresponds to Action-list 1, the next bit corresponds to Action-list 2, etc. Absence of the parameter indicates that all Action-lists are to be synchronized. This is the equivalent of synchronizing all Action-lists by setting all the bits corresponding to all existing Action-lists.

Effect: An instruction to the upper tester to suspend execution of the current Action-list until all specified Action-lists are synchronized with this one.

When a SYNC action is encountered, execution of the current Action-list (X) is suspended, pending synchronization with all Action-lists identified in the 'Groupno' parameter.

Other Action-lists continue to execute until they encounter a SYNC. When all Action-lists involved in the SYNC have suspended or terminated (and therefore implicitly "synchronized") execution continues.

When the last SYNC action of a SYNC group is encountered, all suspended Action-lists of this SYNC group are reactivated. Execution continues with the lowest sequence numbered Action-list of that group.

Values of bits for Action-lists that have terminated shall be ignored. So, if an Action-list executes a SYNC action and any of the Action-lists that it is synchronising with have already terminated, then the terminated Action-list shall be removed from the SYNC group. If this results in the current Action-list being the only member of the SYNC group, then the SYNC action has no effect, i.e. the next action in the Action-list is executed immediately.

Two or more SYNC actions contained in different Action-lists are synchronized only if their SYNC groups are identical, i.e. if their Groupno parameters have the same value.

NOTE — Overlapping SYNC groups may result in a deadlock situation. ATS specifiers should take care to avoid such situations.

7.2.5 TIMER action

Subclause 7.1.5 also applies in the multi-party test context. In addition, after execution of a TIMER action in an Action-list (X), execution continues with the current Action-list (X).

7.2.6 END action

The semantics of the END action in the multi-party testing context are the same as those for the single-party testing context which are described in 7.1.6.

When an Action-list is terminated by executing the last action or the END action, a SYNC group waiting only for the corresponding SYNC action of the terminating Action-list will complete synchronization. If the END action has the effect of releasing more than one SYNC group, then this is a test case error.

7.3 Upper tester default procedure

When the default procedure is called by any Action-list, execution of all Action-lists is terminated.

The unexpected-event-occurred field of the TM-TEST-LOG PDU is set to true. If an unexpected service primitive caused the default procedure to occur, then the unexpected-event field of its log record shall be set to true.

If the normal-default-behaviour parameter of the TM-TEST-PROCEDURE PDU was set to true, the upper tester shall perform the normal default behaviour described in 7.3.1. After that, or if the normal-default-behaviour parameter was set to false, the test execution phase terminates and the IUT is returned to the idle state.

NOTE — It is then a local matter what actions are taken to return the IUT to the idle state.

7.3.1 Normal default behaviour

The normal default behaviour is defined as follows.

The upper tester shall issue a TP-U-ABORT req on all dialogues for which the coordination level is none.

If the IUT is not within a transaction, then the normal default behaviour terminates.

If the IUT is within a transaction, then

NOTE — An IUT is within a transaction without any coordinated dialogues when a TP-BEGIN-DIALOGUE(rejected, rollback = “false”) cnf has been received for each coordinated dialogue.

- if the transaction is in the active phase, the UT issues a TP-ROLLBACK req
- if the transaction is rolling back, the UT issues a TP-U-ABORT req on all dialogues with a coordination level of commitment
- if done-needed is true, the upper tester issues a TP-DONE req immediately
- otherwise, it is suspended until done-needed becomes true or either a TP-COMMIT-COMPLETE ind or TP-ROLLBACK-COMPLETE ind is received. While suspended, various service indications will be received. The upper tester need only note their effect on the done-needed and rollback-next-needed

conditions — apart from this they are ignored. Each time done-needed becomes true, it issues a TP-DONE req and becomes suspended once again.

- If a TP-ROLLBACK-COMPLETE ind is received, the normal default behaviour terminates.
- If a TP-COMMIT-COMPLETE ind is received, and rollback-next-needed is false, the normal default behaviour terminates.
- If the TP-COMMIT-COMPLETE ind is received, and rollback-next-needed is true, the upper tester waits for a TP-ROLLBACK ind, and then repeats the behaviour from “if the transaction is rolling back” above.

8 Interface between the IUT and the upper tester

The upper tester acts as a TP service user. This clause describes the use of TP services for the purposes of testing.

8.1 Values of the In-service and Out-service ASN.1 types

The MATCH and SEND actions take as parameters values of the In-service and Out-service ASN.1 types, as defined in clause 10. These types contain values which correspond to particular TP service primitives.

8.2 Relationship to TP services

Values corresponding to the TP services of the same name have corresponding parameters.

Values TP-DATA req and TP-DATA ind have, compared to their "placeholder" counterparts in ISO/IEC 10026-2, an additional parameter for carrying user data. The encoding of the value of the user data parameter is derived from the application context in the TP-DIALOGUE req and TP-DIALOGUE ind.

9 TMPDU errors

If the upper tester encounters an invalid TMPDU, either in its syntax or its semantics, it shall notify the test case operator of a test case error. It is a local matter how this notification is achieved.

10 Structure and encoding of test management protocol data units

This part of ISO/IEC 13650 assigns the following value for use with the abstract syntax defined as ISOIEC13650-TP-TMP

```
ISOIEC13650-TP-TM DEFINITIONS ::=
BEGIN
  tp-tmp OBJECT IDENTIFIER ::= { iso standard 13650}
END
```

The abstract syntax for the abstract syntax is defined below

```
ISOIEC13650-TP-TMP DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORTS
-- all definitions --

IMPORTS
  AE-qualifier, AP-title, AE-invocation-identifier,
  AP-invocation-identifier, Application-context-name
  FROM acse-1
    { joint-iso-ITU-T association control(2)
      modules(0) apdus(0) version1(1)
    }
    -- Note: the latest version of the ACSE ASN.1 definitions
    -- uses ASO-context-name instead of Application-context-name

  TPSU-title, FU-list, Confirmation-urgency
  FROM Transaction-Processing-APDUs
    { joint-iso-ITU-T transaction processing(10)
      modules (1) apdus-abstract-syntax(1) version1(0)
    }

TP-TMP-apdu ::= CHOICE
{ tm-test-procedure                      [1] TM-Test-Procedure,
  tm-start-test                          [2] NULL,
  tm-request-log                         [3] NULL,
  tm-test-log                            [4] TM-Test-Log,
  tm-end-test-session                       [5] NULL
}

TM-Test-Procedure ::= SEQUENCE
{ normal-default-behaviour                      BOOLEAN,
  SEQUENCE OF                                   Action-list
}

TM-Test-Log ::= SEQUENCE
{ unexpected-event-occurred                     BOOLEAN,
  SEQUENCE OF                                   Log-Record OPTIONAL
}

Log-Record ::= SEQUENCE
{ time                                          GeneralizedTime,
  serviceprim                                   ServicePrimitive,
  unexpected-event                              BOOLEAN DEFAULT FALSE,
  extra-info                                    OCTET STRING OPTIONAL
}

ServicePrimitive ::= CHOICE
{ inprim                                    [1] In-service,
  outprim                                   [2] Out-service
}
Action-list ::= SEQUENCE OF Actions

Actions::= CHOICE
{ send                                      [1] Send,
  receive                                   [2] NULL,
  match                                     [3] Match,
  sync                                      [4] Sync,
  timer                                     [5] Timer,
  end                                       [6] NULL
}

Send ::= SEQUENCE
{ send-data                                     Out-service
}

Match ::= SEQUENCE
{ recv-data                                     In-service,
  match-case                                [1] Action-list OPTIONAL,
  unmatch-case                              [2] Action-list OPTIONAL
}

Sync ::= SEQUENCE
{ groupno                                       BIT STRING OPTIONAL
}

Timer ::= SEQUENCE
{ time                                          INTEGER
}

-- Output TP Service code definition
Out-service ::= CHOICE
{ tp-begin-dialogue-req                     [1] Tp-begin-dialogue-req,
  tp-begin-dialogue-rsp                     [2] Tp-begin-dialogue-rsp,
  tp-end-dialogue-req                       [3] Tp-end-dialogue-req,
  tp-end-dialogue-rsp                       [4] Tp-end-dialogue-rsp,
  tp-data-req                               [5] Tp-data-req,
  tp-u-error-req                            [6] Tp-u-error-req,
  tp-u-abort-req                            [7] Tp-u-abort-req,
  tp-grant-control-req                      [8] Tp-grant-control-req,
  tp-request-control-req                    [9] Tp-request-control-req,
  tp-handshake-req                         [10] Tp-handshake-req,
  tp-handshake-rsp                         [11] Tp-handshake-rsp,
  tp-handshake-and-grant-control-req       [12] Tp-handshake-and-grant-control-req,
  tp-handshake-and-grant-control-rsp       [13] Tp-handshake-and-grant-control-rsp,
  tp-deferred-end-dialogue-req       [14] Tp-deferred-end-dialogue-req,
  tp-deferred-grant-control-req      [15] Tp-deferred-grant-control-req,
  tp-commit-req                      [16] Tp-commit-req,
  tp-done-req                        [17] Tp-done-req,
  tp-prepare-req                     [18] Tp-prepare-req,
  tp-rollback-req                    [19] Tp-rollback-req,
  tp-begin-transaction-req           [20] Tp-begin-transaction-req
}

-- Input TP Service code definition
In-service ::= CHOICE
{ tp-begin-dialogue-ind               [1] Tp-begin-dialogue-ind,
  tp-begin-dialogue-cnf               [2] Tp-begin-dialogue-cnf,
  tp-end-dialogue-ind                 [3] Tp-end-dialogue-ind,
  tp-end-dialogue-cnf                 [4] Tp-end-dialogue-cnf,
  tp-data-ind                         [5] Tp-data-ind,
  tp-u-error-ind                      [6] Tp-u-error-ind,
  tp-u-abort-ind                      [7] Tp-u-abort-ind,
  tp-p-abort-ind                      [8] Tp-p-abort-ind,
  tp-grant-control-ind                [9] Tp-grant-control-ind,
  tp-request-control-ind             [10] Tp-request-control-ind,
  tp-handshake-ind                   [11] Tp-handshake-ind,
  tp-handshake-cnf                   [12] Tp-handshake-cnf,
  tp-handshake-and-grant-control-ind [13] Tp-handshake-and-grant-control-ind,
  tp-handshake-and-grant-control-cnf [14] Tp-handshake-and-grant-control-cnf,
  tp-deferred-end-dialogue-ind       [15] Tp-deferred-end-dialogue-ind,
  tp-deferred-grant-control-ind      [16] Tp-deferred-grant-control-ind,
  tp-prepare-ind                     [17] Tp-prepare-ind,
  tp-ready-ind                       [18] Tp-ready-ind,
  tp-commit-ind                      [19] Tp-commit-ind,
  tp-commit-complete-ind             [20] Tp-commit-complete-ind,
  tp-rollback-ind                    [21] Tp-rollback-ind,
  tp-rollback-complete-ind           [22] Tp-rollback-complete-ind,
  tp-heuristic-report-ind            [23] Tp-heuristic-report-ind,
  tp-begin-transaction-ind           [24] Tp-begin-transaction-ind
}

-- Output TP Service definition
Tp-begin-dialogue-req ::= SEQUENCE
{ ini-tpsu-title                      [1] TPSU-title OPTIONAL,
  rep-ap-title                            AP-title,
  rep-api-id                          [2] AP-invocation-identifier OPTIONAL,
  rep-ae-qualifier                    [3] AE-qualifier OPTIONAL,
  rep-aei-id                          [4] AE-invocation-identifier OPTIONAL,
  rep-tpsu-t                          [5] TPSU-title OPTIONAL,
  func-unit                               FU-list,
  qos                                 [6] OCTET STRING OPTIONAL,
  ap-context                              Application-context-name,
  bgn-trans                           [7] BOOLEAN OPTIONAL,
  confirmation                            ENUMERATED,
                                          { always   (1),
                                            negative (2)
                                          },
  user-data                               OCTET STRING OPTIONAL
}

Tp-begin-dialogue-rsp ::= SEQUENCE
{ result                                  ENUMERATED,
                                          { accepted          (1),
                                            rejected-provider (2),
                                            rejected-user     (3)
                                          }
  user-data                               OCTET STRING OPTIONAL
}

Tp-end-dialogue-req ::= SEQUENCE
{ confirm                                 BOOLEAN
}
```

```
Tp-end-dialogue-rsp ::= SEQUENCE
{
}

Tp-data-req ::= SEQUENCE
{ user-data                             OCTET STRING
}

Tp-u-error-req ::= SEQUENCE
{
}

Tp-u-abort-req ::= SEQUENCE
{ user-data                             OCTET STRING OPTIONAL
}

Tp-grant-control-req ::= SEQUENCE
{
}

Tp-request-control-req ::= SEQUENCE
{
}

Tp-handshake-req ::= SEQUENCE
{ urgency                               Confirmation-urgency
}

Tp-handshake-rsp ::= SEQUENCE
{
}

Tp-handshake-and-grant-control-req ::= SEQUENCE
{ urgency                               Confirmation-urgency
}

Tp-handshake-and-grant-control-rsp ::= SEQUENCE
{
}

Tp-deferred-end-dialogue-req ::= SEQUENCE
{
}

Tp-deferred-grant-control-req ::= SEQUENCE
{
}

Tp-commit-req ::= SEQUENCE
{
}

Tp-done-req ::= SEQUENCE
{ heuristic-report                      ENUMERATED
                                        { heurictic-mix    (1),
                                          heuristic-hazard (2)
                                        } OPTIONAL
}

Tp-prepare-req ::= SEQUENCE
{ data-perm                             BOOLEAN OPTIONAL
}

Tp-rollback-req ::= SEQUENCE
{
}
```

```
Tp-begin-transaction-req ::= SEQUENCE
{
}

-- Input TP Service definition

Tp-begin-dialogue-ind ::= SEQUENCE
{ ini-ap-title                  [1] AP-title OPTIONAL,
  ini-api-id                    [2] AP-invocation-identifier OPTIONAL,
  ini-ae-quali                  [3] AE-qualifier OPTIONAL,
  ini-aei-id                    [4] AE-invocation-identifier OPTIONAL,
  ini-tpsu-t                    [5] TPSU-title OPTIONAL,
  func-unit                         FU-list,
  bgn-trans                         BOOLEAN OPTIONAL,
  confirmation                      ENUMERATED
                                    { always   (1),
                                      negative (2)
                                    },
  user-data                         OCTET STRING OPTIONAL
}

Tp-begin-dialogue-cnf ::= SEQUENCE
{ func-unit                         FU-list OPTIONAL,
  result                            ENUMERATED
                                    { accepted          (1),
                                      rejected-provider (2),
                                      rejected-user     (3)
                                    },
  diagnostic                        ENUMERATED
                                    { recipient-tpsu-title-unknown  (1),
                                      tpsu-not-available-permanent  (2),
                                      tpsu-not-available-transient  (3),
                                      recipient-tpsu-title-required (4),
                                      fu-not-supported              (5),
                                      fu-combination-not-supported  (6),
                                      association-reserved          (7),
                                      no-reason-given               (8)
                                    } OPTIONAL,
  rollback                          BOOLEAN,
  user-data                         OCTET STRING OPTIONAL
}

Tp-end-dialogue-ind ::= SEQUENCE
{ confirm                           BOOLEAN
}

Tp-end-dialogue-cnf ::= SEQUENCE
{
}

Tp-data-ind ::= SEQUENCE
{ user-data                         OCTET STRING
}

Tp-u-error-ind ::= SEQUENCE
{
}

Tp-u-abort-ind ::= SEQUENCE
{ rollback                          BOOLEAN,
  user-data                         OCTET STRING OPTIONAL
}

Tp-p-abort-ind ::= SEQUENCE
{ diagnostic                        ENUMERATED
                                    { permanent-failure         (1),
                                      begin-transaction-reject  (2),
                                      transient-failure         (3),
                                                   protocol-error          (4)
                                                 }
  rollback                                       BOOLEAN
}

Tp-grant-control-ind ::= SEQUENCE
{
}

Tp-request-control-ind ::= SEQUENCE
{
}

Tp-handshake-ind ::= SEQUENCE
{
}

Tp-handshake-cnf ::= SEQUENCE
{
}

Tp-handshake-and-grant-control-ind ::= SEQUENCE
{
}

Tp-handshake-and-grant-control-cnf ::= SEQUENCE
{
}

Tp-deferred-end-dialogue-ind ::= SEQUENCE
{
}

Tp-deferred-grant-control-ind ::= SEQUENCE
{
}

Tp-prepare-ind ::= SEQUENCE
{ data-perm                                      BOOLEAN OPTIONAL
}

Tp-ready-ind ::= SEQUENCE
{
}

Tp-commit-ind ::= SEQUENCE
{
}

Tp-commit-complete-ind ::= SEQUENCE
{
}

Tp-rollback-ind ::= SEQUENCE
{
}

Tp-rollback-complete-ind ::= SEQUENCE
{
}

Tp-heuristic-report-ind ::= SEQUENCE
{ heuristic-report                               ENUMERATED
                                                 { heuristic-mix    (1),
                                                   heuristic-hazard (2)
                                                 }
}
```

```
Tp-begin-transaction-ind ::= SEQUENCE
{
}

END
```

11 Conformance

A system claiming conformance to this part of ISO/IEC 13650 shall exhibit external behaviour consistent with having implemented

a) a protocol machine that processes TMPDUs according to the semantics defined in clauses 6 to 9;

b) an encoding and a decoding of TMPDUs as defined in clause 10.

Annex A
(informative)
Examples

A.1 Examples of protocol flow in the results collection phase

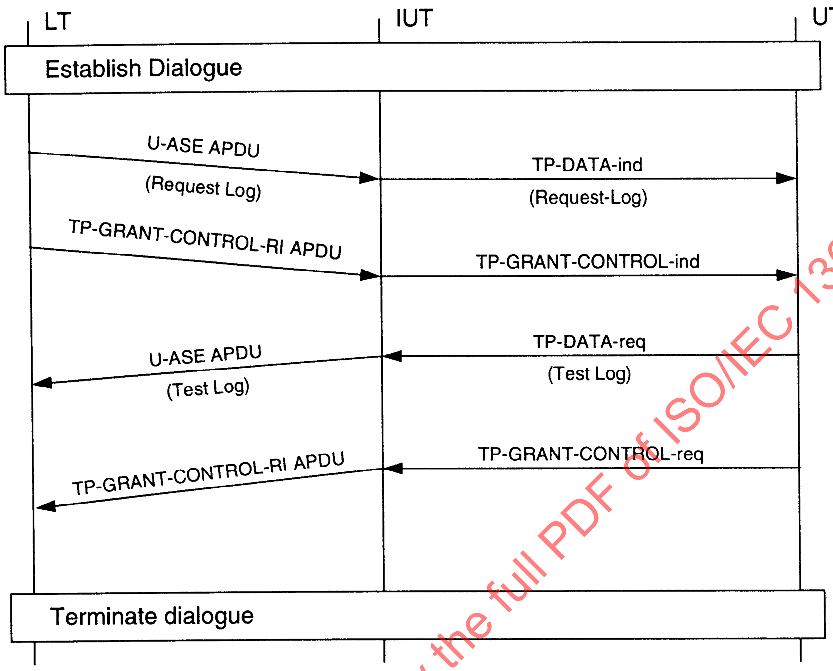

Figure A.1 — Polarized control functional unit was selected

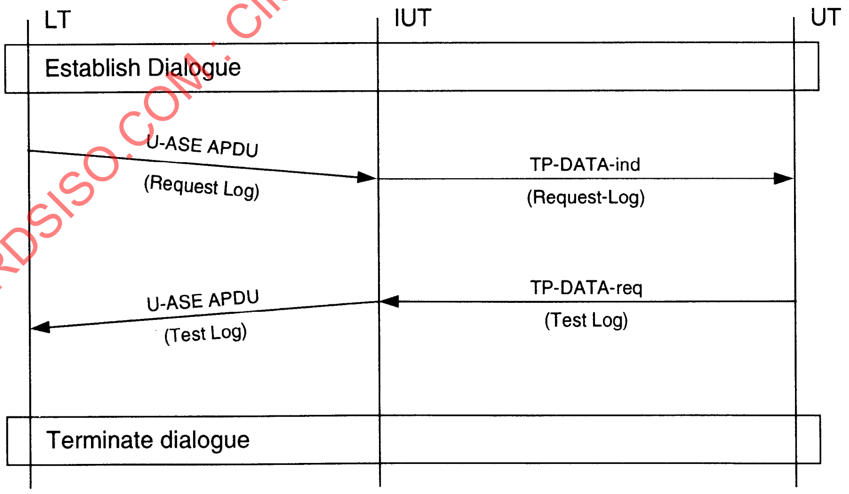

Figure A.2 — Shared control functional unit was selected

A.2 Examples of the use of actions

A.2.1 Sequence of Actions

This example shows two alternative ways of writing TMP to perform the following sequence of events:

Table A. 1 — Sequence of actions

TTCN	TMP equivalent - alternative 1	TMP equivalent - alternative 2
UT?TP-REQUEST-CONTROL-ind	RECEIVE	RECEIVE
UT!TP-GRANT-CONTROL-req	MATCH(TP-REQUEST-CONTROL-ind)	MATCH(TP-REQUEST-CONTROL-ind)
UT?TP-HANDSHAKE-ind	SEND(TP-GRANT-CONTROL-req)	match-case {
UT!TP-HANDSHAKE-rsp	RECEIVE	SEND(TP-GRANT-CONTROL-req)
	MATCH(TP-HANDSHAKE-ind)	RECEIVE
	SEND(TP-HANDSHAKE-rsp)	MATCH(TP-HANDSHAKE-ind)
		match-case {
		SEND(TP-HANDSHAKE-rsp)
		}
		}

A.2.2 Branching

This example shows a number of alternative service primitives that may be received once the first TP-DATA-req has been sent. It illustrates the use of MATCH actions within both match-case and unmatch-case.

Table A. 2 — Branching

TTCN Style	TMP Equivalent
UT!TP-DATA-req	SEND(TP-DATA-req)
UT?TP-HANDSHAKE-ind	RECEIVE
UT!TP-HANDSHAKE-rsp	MATCH(TP-HANDSHAKE-ind)
UT?TP-DATA-ind	match-case {
UT!TP-END-DIALOGUE-req	SEND(TP-HANDSHAKE-rsp)
UT?TP-U-ERROR-ind	}
UT!TP-U-ABORT-req	unmatch-case {
	MATCH(TP-DATA-ind)
	match-case {
	SEND(TP-END-DIALOGUE-req)
	}
	unmatch-case {
	MATCH(TP-U-ERROR-ind)
	match-case {
	SEND(TP-U-ABORT-req)
	}
	}
	}

A.2.3 Multi-Party Testing

A.2.3.1 Initial Situation

When the following test is executed, Action-list 1 is first executed, sending TP-DATA-req(1) and TP-DATA-req(2). The RECEIVE action places Action-list 1 in the suspended state. Action-list 2 is executed, sending TP-DATA-req(3) and TP-DATA-req(4). Action-list 2 enters the suspended state. The same process occurs in Action-list 3. After these actions have been executed, the upper tester suspends pending input from the IUT.

NOTE — In the table, the character ">" is used to denote control positions in an Action-list.

Table A. 3 — Multi-party testing

	Action-list 1	Action-list 2	Action-list 3

Initial Situation	SEND(TP-DATA-req(1))
	SEND(TP-DATA-req(2))
	RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SEND(TP-DATA-req(3))
		SEND(TP-DATA-req(4))
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
			SEND(TP-DATA-req(5))
			SEND(TP-DATA-req(6))
			RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			MATCH(TP-END-DIALOGUE-ind)

First Progression	SEND(TP-DATA-req(1))
	SEND(TP-DATA-req(2))
	>RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SEND(TP-DATA-req(3))
		SEND(TP-DATA-req(4))
		>RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
			SEND(TP-DATA-req(5))
			SEND(TP-DATA-req(6))
			>RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

Second Progression	SEND(TP-DATA-req(1))
	SEND(TP-DATA-req(2))
	>RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SEND(TP-DATA-req(3))
		SEND(TP-DATA-req(4))
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		>RECEIVE
		MATCH(TP-U-ABORT-ind)
			SEND(TP-DATA-req(5))
			SEND(TP-DATA-req(6))
			>RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

Third Progression	SEND(TP-DATA-req(1))
	SEND(TP-DATA-req(2))
	>RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SEND(TP-DATA-req(3))
		SEND(TP-DATA-req(4))
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
		>
			SEND(TP-DATA-req(5))
			SEND(TP-DATA-req(6))
			>RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

Fourth Progression	SEND(TP-DATA-req(1))
	SEND(TP-DATA-req(2))
	RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
	>
		SEND(TP-DATA-req(3))
		SEND(TP-DATA-req(4))
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
		>
			SEND(TP-DATA-req(5))
			SEND(TP-DATA-req(6))
			>RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

A.2.3.2 First Progression

If we assume that TP-HANDSHAKE-ind is received from the leaf node 1, Action-list 2 is executed, sending TP-HANDSHAKE-rsp. Action-list 2 then suspends pending receipt of the TP-U-ABORT-ind.

A.2.3.3 Second Progression

If we assume that TP-U-ABORT-ind is received from the leaf node 1, Action-list 2 is executed and then terminates.

A.2.3.3 Third Progression

If we assume that TP-END-DIALOGUE-ind is received from the root node, Action-list 1 is executed and ends. Subsequent actions are executed in Action-list 3.

A.2.3.5 Fourth Progression

After all actions on all Action-lists are executed, the test execution phase terminates.

A.2.4 Alternative Styles

This example provides 3 possible alternative ways of writing TMP for the following RECEIVE and SEND actions.

Table A. 4 — Alternative styles

TTCN	Alternative 1	Alternative 2	Alternative 3

UT?TP-HANDSHAKE-ind
 UT!TP-HANDSHAKE-rsp
  UT!TP-DATA-req
	RECEIVE
	MATCH(TP-HANDSHAKE-ind)
	SEND(TP-HANDSHAKE-rsp)
	SEND(TP-DATA-req)
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		 match-case {
		  SEND(TP-HANDSHAKE-rsp)
		  SEND(TP-DATA-req)
		 }
			RECEIVE
			MATCH(TP-HANDSHAKE-ind)
			 match-case {
			  SEND(TP-HANDSHAKE-rsp)
			 }
			SEND(TP-DATA-req)

The TMP alternatives produce identical results. If the MATCH is successful, then both service primitives are sent. If the MATCH fails, then the default behaviour is performed.

A.2.5 SYNC

Table A. 5 — Operation of SYNC

	Action-list 1	Action-list 2	Action-list 3

Stage 1	SEND(TP-DATA-req)
	SEND(TP-DATA-req)
	RECEIVE
	MATCH(TP-DATA-ind)
	SYNC
	RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SYNC
		SEND(TP-DATA-req)
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
			SYNC
			SEND(TP-DATA-req)
			RECEIVE
			MATCH(TP-U-ERROR-ind
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

Stage 2	SEND(TP-DATA-req)
	SEND(TP-DATA-req)
	>RECEIVE
	MATCH(TP-DATA-ind)
	SYNC
	RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		>SYNC
		SEND(TP-Data-req)
		RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
			>SYNC
			SEND(TP-DATA-req)
			RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

Stage 3	SEND(TP-DATA-req)
	SEND(TP-DATA-req)
	RECEIVE
	MATCH(TP-DATA-ind)
	SYNC
	>RECEIVE
	MATCH(TP-END-DIALOGUE-ind)
		SYNC
		SEND(TP-DATA-req)
		>RECEIVE
		MATCH(TP-HANDSHAKE-ind)
		SEND(TP-HANDSHAKE-rsp)
		RECEIVE
		MATCH(TP-U-ABORT-ind)
			SYNC
			SEND(TP-DATA-req)
			>RECEIVE
			MATCH(TP-U-ERROR-ind)
			:
			:
			RECEIVE
			MATCH(TP-END-DIALOGUE-ind)

When the above test is executed, in Stage 1, Action-list 1 is first executed, sending the TP-DATA-req and the next TP-DATA-req. The RECEIVE action cause Action-list 1 to suspended. Thereafter, Action-list 2 is executed and suspends due to the SYNC action. Action-list 3 is also executed and suspends. After these actions have been executed, the IUT waits to receive the TP-DATA-ind.

In Stage 2, Action-list 1 is executed when the TP-DATA-ind is received and it then executes the SYNC action. All Action-lists are now synchronized, and the Action-lists become available for execution.

Action-lists 1, 2 and 3 are then executed in that order.

A.2.6 Overlapping SYNC

If sets of Action-lists which contain SYNC actions with different SYNC groups are considered, it may become difficult to understand in what order the upper tester executes the actions from the different Action-lists. The following example demonstrates how the upper tester behaves in the presence of overlapping SYNC groups.

Table A. 6 — Overlapping SYNC

Action-list 1	Action-list 2	Action-list 3

SYNC(2)
SEND(service-prmA)
RECEIVE
MATCH(service-prmB)
	SYNC(3)
	SEND(service-prmC)
	SYNC(1)
	SEND(service-prmD)
		SYNC(2)
		SEND(service-prmE)

The Action-lists contain two different instances of synchronization: one between Action-lists 2 and 3, and the other one between Action-lists 1 and 2. In this case, the upper tester executes Action-lists in the following order.

Initially, execution of Action-list 1 is suspended after executing its first SYNC action, which requests synchronization with Action-list 2. Then, execution of Action-list 2 is suspended by its first SYNC action, which requests synchronization with Action-list 3. After that, the SYNC action in Action-list 3 completes synchronization between Action-lists 2 and 3.