INTERNATIONAL STANDARD

ISO/IEC 10026-1

Second edition
1998-10-15

Information technology — Open Systems Interconnection — Distributed Transaction Processing —

Part 1: OSI TP Model

Technologies de l'information — Interconnexion de systèmes ouverts (OSI) — Traitement transactionnel réparti —

Partie 1: Modèle OSI TP

Reference number
ISO/IEC 10026-1:1998(E)

Contents

Annexes

Tables

Figures

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 10026-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 21, *Open systems interconnection*, *data management and open distributed processing.*

This second edition cancels and replaces the first edition (ISO/IEC 10026-1:1992), which has been technically revised. It also incorporates Technical Corrigendum 1:1996.

This part of ISO/IEC 10026 is technically aligned with ITU-T Recommendation X.860, but is not published as identical text.

ISO/IEC 10026 consists of the following parts, under the general title *Information technology — Open Systems Interconnection — Distributed Transaction Processing*:

— *Part 1: OSI TP Model*

— *Part 2: OSI TP Service*

— *Part 3: Protocol specification*

— *Part 4: Protocol Implementation Conformance Statement (PICS) proforma*

— *Part 5: Application context proforma and guidelines when using OSI TP*

— *Part 6: Unstructured Data Transfer*

Annex A forms an integral part of this part of ISO/IEC 10026. Annexes B to E are for information only.

Introduction

ISO/IEC 10026 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 Reference Model for Open Systems Interconnection (ISO/IEC 7498-1). The Reference Model subdivides the area of standardization for interconnection into a series of layers of specification, each of manageable size.

The aim of Open Systems Interconnection is to allow, with a minimum of technical agreement outside the interconnection standards, the interconnection of computer systems

a) from different manufacturers;

b) under different management;

c) of different levels of complexity; and,

d) of different technologies.

ISO/IEC 10026 defines an OSI TP Model, an OSI TP Service and specifies an OSI TP Protocol available within the Application Layer of the OSI Reference Model.

The OSI TP Service is an Application Layer service. It is concerned with information which can be related as distributed transactions, which involve two or more open systems.

ISO/IEC 10026 provides sufficient facilities to support transaction processing, and establishes a framework for coordination across multiple OSI TP resources in separate open systems.

ISO/IEC 10026 does not specify the interface to local resources or access facilities that are provided within the local system. However, future enhancement of the standard may deal with these issues.

Information technology — Open Systems Interconnection — Distributed Transaction Processing —

Part 1:
OSI TP Model

1 Scope

This part of ISO/IEC 10026:

a) provides a general introduction to the concepts and mechanisms defined in ISO/IEC 10026;

b) defines a model of distributed transaction processing;

c) defines the requirements to be met by the OSI TP Service; and

d) takes into consideration the need to coexist with other Application Service Elements, e.g. RDA (Remote Database Access), ROSE (Remote Operations Service Element), and non-ROSE based applications.

This part of ISO/IEC 10026 makes sufficient provisions to allow the specification of transaction-mode communications services and protocols that meet the properties of: atomicity, consistency, isolation, and durability (the ACID properties), as defined in ISO/IEC 9804.

This part of ISO/IEC 10026 does not specify individual implementations or products, nor does it constrain the implementation of entities or interfaces within a computer system.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 10026. 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 10026 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 7498-2:1989, *Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture.*

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

ISO/IEC 8326:1996, *Information technology - Open Systems Interconnection - Session service definition.*

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 9545:1989, *Information technology - Open Systems Interconnection - Application Layer structure.*

NOTE - this edition of ISO/IEC 10026 uses the terminology and modelling mechanisms of the first (1989) edition of the Application Layer Structure (ISO/IEC 9545:1989).

ISO/IEC 9579-1:1993, *Information technology - Open Systems Interconnection - Remote Database Access - Part 1: Generic Model, Service, and Protocol.*

ISO/IEC 9594-2:1995, *Information technology - Open Systems Interconnection - The Directory: Models.*

ISO/IEC 9804:1997, *Information technology - Open Systems Interconnection - Service definition for the commitment, concurrency and recovery service element.*

ISO/IEC 10026-2:1998, *Information technology - Open Systems Interconnection - Distributed Transaction Processing - Part 2: OSI TP Service.*

ISO/IEC 10026-3:1998, *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.*

ISO/IEC 13712-1:1995, *Information technology - Remote Operations: Concepts, model and notation.*

3 Definitions

For the purposes of ISO/IEC 10026, the following definitions apply.

3.1 Terms defined in other International Standards

3.1.1 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 7498-1:

a) application-entity;
b) application-process;
c) application-protocol-data-unit;
d) concatenation;
e) open system;
f) presentation-service;
g) presentation-service-access-point;
h) presentation-service-data-unit;
i) real open system; and
j) separation.

3.1.2 ISO/IEC 10026 makes use of the following terms defined in ISO 7498-2:

a) access control;
b) audit;
c) authentication;
d) confidentiality;
e) integrity; and
f) non-repudiation.

3.1.3 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 7498-3:

a) application-process-invocation-identifier;
b) application-process-title;
c) application-entity-invocation-identifier;

d) application-entity-qualifier; and
e) application-entity-title.

3.1.4 ISO/IEC 10026 makes use of the following term defined in ISO/IEC 8326:

quality-of-service

3.1.5 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 10731:

a) request;
b) indication;
c) response;
d) confirm;
e) service primitive; primitive;
f) service-provider; and
g) service-user.

3.1.6 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 9545:

a) application-association; association;
b) application-context;
c) application-context-name;
d) application-entity-invocation;
e) application-process-invocation;
f) application-service-element;
g) association control service element;
h) multiple association control function;
i) single association control function; and
j) single association object.

3.1.7 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 9594-2:

a) Directory Information Tree;
b) Directory entry; entry;
c) distinguished name;
d) object class; and
e) relative distinguished name.

3.1.8 ISO/IEC 10026 makes use of the following terms defined in ISO/IEC 9804:

a) atomic action data;
b) atomicity;
c) bound data;
d) consistency;
e) durability;
f) final state;
g) heuristic decision;
h) initial state; and
i) isolation.

3.2 Terms defined in ISO/IEC 10026

3.2.1 application-supported distributed transaction: A transaction where the user of the OSI TP Service is responsible for the maintenance of the ACID properties.

3.2.2 chained sequence: A sequence of related contiguous (provider-supported) transaction branches, on the same dialogue, that are aimed at achieving a common goal.

3.2.3 Channel Protocol Machine; CPM: The part of an AEI involved in OSI TP that establishes and terminates TP channels.

3.2.4 channel; **Transaction Processing channel**: A relationship over an association between two AEIs to facilitate Transaction Processing Service Provider (TPSP) recovery activity. Channels are not visible to the TPSUIs.

3.2.5 commit master: The neighbour to which a node has sent a ready signal.

NOTE - with the static commitment procedures, the commit master will be the dialogue superior.

3.2.6 commit slave: A neighbour from which a ready signal has been received.

NOTE - CCR uses the term "commit subordinate"; TP uses the term "commit slave" to avoid confusion with dialogue subordinate.

NOTE - with the static commitment procedures, a commit slave will be a dialogue subordinate.

NOTE - the terms commit master and commit slave do not apply when a read-only signal or early-exit signal or one-phase signal is sent.

3.2.7 commitment; transaction commitment: Completion of a transaction with the release of transaction data in the final state.

NOTE - commitment requires two-phase commitment procedures if bound data are affected; one-phase commitment procedures may be used if bound data are not affected; see section 8.6.1 for two-phase commitment procedures and 8.6.4 for one-phase commitment procedures.

NOTE - the terms "commitment" and "rollback" have a different scope from that defined in ISO/IEC 9804. ISO/IEC 10026 is concerned with the commitment and rollback of a complete transaction, whereas ISO/IEC 9804 refers to the commitment and rollback of a single atomic action branch.

3.2.8 commitment coordinator: A TPPM involved in a distributed transaction that arbitrates the final outcome of the transaction.

NOTE - With the static two-phase commitment procedures, the commitment coordinator will be at the root of the transaction tree. If the static one-phase commitment procedures are in use in the transaction tree, the commitment coordinator will either be a leaf node or an intermediate node. With the dynamic two-phase commitment procedures, the position of the commitment coordinator may be predetermined or may be determined dynamically.

3.2.9 commitment hinterland: A node's current commitment hinterland is the set of nodes in the transaction tree which include:

a) the neighbouring nodes from which ready signals have been received; and

b) the commitment hinterlands of those neighbouring nodes, and so on recursively.

NOTE - the commitment hinterland excludes those nodes which signal read-only or one-phase or early-exit.

NOTE - with the static two-phase commitment procedures and no use of either read-only or one-phase commitment or early-exit, the commitment hinterland of a node will be identical to the transaction subtree of the node.

3.2.10 commitment order: A statement from a node to a neighbour that has signalled ready, that the transaction shall be committed.

3.2.11 control: The permission, on a particular dialogue, for a TPSUI to communicate with its partner.

3.2.12 coordination level: An agreement between two TPSUIs on what mechanism will be used to guarantee the four properties of a transaction; the coordination level may be "commitment", "one-phase commitment" or "none".

3.2.13 coordinated dialogue; dialogue is coordinated: A dialogue currently having a coordination level of "commitment" or "one-phase commitment".

NOTE - a dialogue supporting chained transaction branches is always coordinated and a dialogue supporting unchained transaction branches is coordinated only when it supports a transaction branch.

3.2.14 dialogue: The relationship between two TPSUIs that communicate with each other. The initiator of the dialogue is the superior and the recipient is the subordinate.

3.2.15 dialogue tree: A tree consisting of TPSUIs as the entities with dialogues as the relationships between them.

3.2.16 distributed transaction: A transaction, parts of which may be carried out in more than one open system.

3.2.17 dynamic commitment procedures: The two-phase commit procedures without the constraints of the static commitment procedures; subject to optional controls, the commitment coordinator may be a predetermined node in the transaction tree (not necessarily the root) or may be dynamically determined.

3.2.18 early-exit signal: A statement from a node to a superior that this node and its subtree can make no contribution to the work of the transaction and so it withdraws from participation in the transaction; conditions are that the bound data of this node have not been altered by the transaction, that read-only or early-exit signals have been received from all the node's subordinates in the transaction tree, if there are any, and that reporting of the transaction outcome is not required.

3.2.19 heuristic-hazard: The condition that arises when, as a result of communication failure with a subordinate, the bound data of the subordinate's subtree are in an unknown state.

3.2.20 heuristic-mix: The condition that arises when, as a result of one or more heuristic decisions having been taken, the bound data of the transaction are in an inconsistent state.

3.2.21 intermediate: An entity in a tree which has one superior and one or more subordinates.

3.2.22 leaf: An entity in a tree which has one superior and no subordinates.

3.2.23 local resource: A resource that is resident on the same real open system as the requester of the resource, or a resource that is managed by an entity residing in the same real open system as the requester of the resource.

3.2.24 log-commit record: A record written to the recovery log that reflects the transaction's decision to commit.

3.2.25 log-damage record: A record written to the recovery log that reflects the current inconsistent state of bound data in the subtree.

3.2.26 log-heuristic record: A record written to the recovery log that reflects the node's heuristic decision.

3.2.27 log-ready record: A record written to the recovery log that records information required for recovery and that the bound data of this node is ready-to-commit and, if there is more than one neighbour in the transaction tree, that one of ready signal, one-phase signal or read-only signal or early-exit signal has been received from all but one of the neighbours in the transaction tree.

3.2.28 long lived data: Data which are accessed and manipulated by the TPSUI within the scope of either a provider supported transaction or an application supported transaction but for which the TPSUI takes responsibility for recovery in the event of failures.

NOTE - "long lived data" are not "bound data", and vice versa.

3.2.29 neighbour: An entity in a tree which has a direct relationship with another entity.

NOTE - thus a subordinate and its superior are neighbours, each with the other.

3.2.30 node: A TPSUI together with its TPPM.

3.2.31 node crash: A failure of the node (i.e. TPPM and TPSUI) or of the local environment supporting the node such that dialogues are aborted and all data not recorded in secure storage may be lost.

3.2.32 one-phase signal: A statement from a node to a neighbour that this node has no bound data (in the strict sense defined by CCR) and that either read-only or early-exit or one-phase signals have been received from all other neighbours in the transaction tree, if there are any.

3.2.33 polarized control mode: A mode of communication over a dialogue where only one TPSUI involved in the dialogue is allowed to have control at a time.

3.2.34 Protocol Machine; PM: A generic term to denote either a Transaction Processing Protocol Machine or a Channel Protocol Machine.

3.2.35 provider-supported distributed transaction: A transaction where the provider of the OSI TP Service is responsible for the maintenance of the ACID properties.

3.2.36 read-only signal: A statement from a node to a superior that the bound data of this node have not been altered by the transaction, that read-only or early-exit signals have been received from all the node's subordinates in the transaction tree, if there are any, and that reporting of the transaction outcome is not required.

3.2.37 ready signal: A statement from a node (to a neighbour) that a log-ready record has been written. The neighbour to whom the signal is sent is the one neighbour (if there is more than one) that had not sent a ready signal or one-phase signal or read-only signal or early-exit signal when the log-ready record was written.

NOTE - thus ready signal excludes read-only signal or one-phase signal or early-exit signal.

3.2.38 ready-to-commit state: A state of bound data in which, until the transaction has been terminated by commitment or rollback, the bound data can be released in either their initial or their final state.

3.2.39 recovery: Action taken after a failure to remove undesired consequences of the failure.

3.2.40 recovery log: A repository in secure storage used to record data and state information for the purposes of restart and recovery.

3.2.41 remote resource: A resource that is resident on a different real open system than the real open system making the request for resources.

3.2.42 resource: Data and processing capabilities necessary for a TPSUI to carry out the part of a transaction for which it is responsible.

3.2.43 rollback: **transaction rollback**: Completion of a transaction with the release of bound data in the initial state.

NOTE - the terms "commitment" and "rollback" have a different scope from that defined in ISO/IEC 9804. ISO/IEC 10026 is concerned with the commitment and rollback of a complete transaction, whereas ISO/IEC 9804 refers to the commitment and rollback of a single atomic action branch.

3.2.44 root: The single entity in a tree which has no superior and has one or more subordinates.

3.2.45 secure storage: A reliable non-volatile place where stored information survives any type of recoverable failure within the real open system.

3.2.46 shared control mode: A mode of communication over a dialogue where both TPSUIs involved in the dialogue have control.

3.2.47 static commitment procedures: The two-phase commitment procedures constrained such that the commit decision is made at the root of the transaction tree and is propagated down the tree.

NOTE - this is equivalent to the commitment procedures of 10026:1992 and 10026:1995.

3.2.48 subordinate: The entity which accepts a relationship (from a superior).

3.2.49 subordinate subtree: The subtree of a subordinate node.

3.2.50 subtree: A subset of a tree. The subtree of a particular node contains

a) the node itself, called the root node of the subtree; and

b) the subtrees of each subordinate node of the root node of the subtree, recursively.

A leaf node is its own subtree.

3.2.51 superior: The entity which initiates a relationship.

3.2.52 transaction: A set of related operations characterized by four properties: atomicity, consistency, isolation, and durability. A transaction is uniquely identified by a transaction identifier.

NOTE - For reasons of brevity, the term "transaction" is used as a synonym of the term "provider-supported distributed transaction", from 7.8 onwards.

3.2.53 transaction branch: The portion of a distributed transaction performed by a pair of TPSUIs sharing a dialogue.

NOTE - For reasons of brevity, the term "transaction branch" is used as a synonym of the phrase "branch of provider-supported distributed transaction", from 7.8 onwards.

3.2.54 transaction branch identifier: An unambiguous identifier for a specific branch of a specific transaction.

3.2.55 transaction data: Data which are accessed and manipulated by the TPSUI within the scope of a transaction (either a provider-supported transaction or an application-supported transaction); transaction data is either "bound data" or "long-lived data".

3.2.56 transaction hinterland: The transaction hinterland of node B as viewed from node A is the node B together with the transaction hinterland (as viewed from node B) of all B's neighbouring nodes except A which are participating in or have participated in the current transaction on a transaction branch with B.

NOTE - nodes which are no longer participating in the transaction because they have signalled read-only or early-exit, continue to be part of the transaction hinterland until the transaction is terminated.

3.2.57 transaction identifier: A globally unambiguous identifier for a specific transaction.

3.2.58 transaction logging: The recording of node state information and data in a recovery log.

3.2.59 Transaction Processing Application Service Element; TPASE: That part of a Transaction Processing Protocol Machine (TPPM) which handles the OSI TP Protocol on a single application-association.

3.2.60 Transaction Processing Protocol Machine; TPPM: The provider of the OSI TP Service for exactly one TPSUI. A TPPM handles the OSI TP Protocol on all associations that are used for its TPSUI's activity.

3.2.61 Transaction Processing Service Provider; TPSP: The provider of the OSI TP Service. The TPSP provides the OSI TP Service to all the TPSUIs involved in a particular dialogue tree. The TPSP spans several application-process-invocations (APIs) and is the conceptual view of the OSI TP Service as a whole.

3.2.62 Transaction Processing Service User; TPSU: A user of the OSI TP Service: it refers to a specific set of processing capabilities within an application-process.

3.2.63 TPSU Invocation; TPSUI: A particular instance of a TPSU performing functions for a specific occasion of information processing.

3.2.64 TPSU-title: A name, unambiguous within the scope of the application-process containing the TPSU, which denotes a particular TPSU. The TPSU-title implies the type of processing (capabilities) of this TPSU.

3.2.65 transaction recovery: Action taken after a failure in order to put all the bound data of that transaction into a consistent state.

3.2.66 transaction tree: A tree with nodes as the entities, and transaction branches as the relationship between them.

3.2.67 tree: A set of linked entities arranged in a hierarchical structure and connected by relationships.

3.2.68 unchained sequence: A sequence of non-contiguous (provider-supported) transaction branches, on the same dialogue, that are aimed at achieving a common goal.

3.2.69 uncoordinated dialogue; dialogue is not coordinated: A dialogue currently having a coordination level of "none".

3.2.70 user-ASE: An application-specific ASE.

4 Abbreviations

For the purposes of ISO/IEC 10026, the following abbreviations apply:

ACID	Atomicity, Consistency, Isolation, and Durability
ACSE	Association Control Service Element
AE	Application-Entity
AEI	Application-Entity Invocation
ALS	Application Layer Structure
AP	Application-Process
APDU	Application-Protocol-Data-Unit
API	Application-Process Invocation
ASE	Application Service Element
CCR	Commitment, Concurrency, and Recovery
CPM	Channel Protocol Machine
MACF	Multiple Association Control Function
OSI	Open Systems Interconnection
OSIE	Open Systems Interconnection Environment
PICS	Protocol Implementation Conformance Statement
PM	Protocol Machine (either a TPPM or a CPM)
PSAP	Presentation Service Access Point
PSDU	Presentation-Service-Data-Unit
RDA	Remote Database Access
ROSE	Remote Operations Service Element
SACF	Single Association Control Function
SAO	Single Association Object
TP	Transaction Processing
TPASE	Transaction Processing Application Service Element
TPPM	Transaction Processing Protocol Machine
TPSP	Transaction Processing Service Provider
TPSU	Transaction Processing Service User
TPSUI	Transaction Processing Service User Invocation
U-ASE	User-Application Service Element

5 Conventions

ISO/IEC 10026 is guided by the conventions discussed in ISO/IEC 10731 as they apply to the OSI TP Service.

6 Requirements

6.1 Introduction

This clause summarizes the requirements for OSI TP. It includes both requirements which are addressed by ISO/IEC 10026, and also requirements which are not addressed and which require further study; these additional requirements are candidates for further standardization as amendments and/or additional parts to ISO/IEC 10026.

6.2 User requirements

In order to satisfy user needs, ISO/IEC 10026

a) defines procedures which support distributed transactions, as discussed in 7.2. These procedures
   1) allow a distributed transaction to be organized into a transaction tree;
   2) provide multi-party coordination (part of which is multi-party commitment), including local resources;
   3) allow restoration to a consistent state, following failure, of the state/context of a distributed transaction and of bound data;
   4) allow the detection of a distributed transaction's failure to achieve ACID properties;
   5) allow a distributed transaction to be restarted following successful state restoration; and
   6) indicate the completion status of a transaction;

b) provides for the delimitation of a sequence of logically related transactions;

c) allows the grouping of TPSUs within an application-process;

d) allows for one, or more, of the following security requirements:

   NOTE - The provision for security is for further standardization as an amendment.

   1) access control: it must be possible to support multiple access control policies. At least those types described in ISO 7498-2 (administration imposed and dynamically selectable, rule-based and identity-based) should be included;
   2) access control granularity: it should be possible to classify OSI TP objects into groups in order to simplify the specification of access control and allow for distribution of the authorization database. Such classification should be for optimization, not a substitute for individual auditing;
   3) authentication between:
      i) corresponding TPSUIs;
      ii) TPPMs;
      iii) AEIs; and
      iv) TPSUIs and TPPMs. However, this is considered to be a local matter;
   4) non-repudiation: prevent denial of having participated in a specific transaction or dialogue;
   5) confidentiality: to prevent unauthorized reception of part, or all of the information exchanged within a dialogue tree;
   6) integrity: to detect unauthorized changes to part, or all of the information exchanged within a dialogue tree; and
   7) audit: to record significant security events occurring within a dialogue tree;

e) allows conformance testing of the protocol defined by ISO/IEC 10026-3 and delineate clearly the static conformance requirements (through the PICS defined in ISO/IEC 10026-4).

6.3 Modelling requirements

The OSI TP Model provides a model of distributed transaction processing and the communications mechanisms to support it which are consistent with the OSI architecture defined in ISO 7498-1 and ISO/IEC 9545, and that addresses the following requirements:

a) definition of mechanisms for partitioning into transactions the interactions between application-processes of two or more open systems. In particular, these mechanisms provide for
   1) indication of the completion status of a transaction;
   2) support of transactions which do not require the full distributed commitment mechanisms to ensure the ACID properties: the application is responsible for ensuring the ACID properties; and
   3) flexibility in order to match the choice of data transfer method to the semantics of the transaction;

b) specification of mechanisms to use the services of the Presentation Layer;

c) procedures that have acceptable performance and efficiency; and

d) procedures that cover a wide variety of needs (short or long, simple or complex transactions).

NOTE - Some of these procedures are candidates for further standardization.

6.4 OSI TP Service and Protocol requirements

The OSI TP Service and Protocol provide for

a) flexibility to handle changing load conditions;

b) efficient support of operations under high, low or burst conditions;

c) efficient handling of short APDUs;

d) acceptable response time for users;

e) resilience from failure, including the means to recover and restart processing after faults have been corrected or circumvented;

f) optimal resource usage; and

g) minimization of the dependence of local resource control upon communications.

In order to meet these requirements, the OSI TP Protocol

a) optimizes the use of the Presentation Layer Service;

b) minimizes the communication overhead required for each transaction - in particular, the OSI TP Protocol limits the number of round trips required by the communication protocols to be no greater than the number of round trips required by the semantics of the application;

c) optimizes operations to the needs of high volume transaction processing; and,

d) optimizes operations to the needs of the normal case rather than to those of exception cases.

7 Concepts of distributed TP

7.1 Transaction

A transaction is a set of related operations characterized by four properties: atomicity, consistency, isolation, and durability.

7.2 Distributed transaction

A transaction that spans more than one open system is called a distributed transaction.

A distributed transaction is composed of at least as many parts as there are open systems involved in this distributed transaction. Within each open system, a part of the distributed transaction relates to an entity called a TP Service User (TPSU).

The TPSU is the user of the OSI TP Service. It refers to a specific set of processing capabilities within an application-process. There may be zero, one, or more TPSUs within any given application-process.

NOTE - A TPSU may in turn be distributed within an application-process. ISO/IEC 10026 does not preclude such a refinement, but does not discuss it, since distribution within an open system lies beyond the scope of OSI.

A TPSU invocation (TPSUI) models, from the perspective of the OSIE, a particular instance of a TPSU, within an application-process-invocation, performing functions for a specific occasion of information processing.

To maintain the four properties of transactions, coordination is required among the TPSUIs performing a distributed transaction. Such coordination requires communication among TPSUIs.

7.3 Transaction data and coordination level

A TPSUI may manipulate data within the scope of a transaction and place that data into their final state or their initial state depending on whether the transaction commits or rolls back. Such data are called transaction data.

The mechanism which is used to coordinate the outcome of a transaction is determined by the coordination level. Three coordination levels are supported for use by the TPSUI:

a) "commitment" when the TPSP is responsible for the demarcation of transactions and the reporting of the transaction outcome, including when failures occur during transaction termination; the TPSP uses a two-phase commit mechanism to support this coordination level;

b) "one-phase commitment" when the TPSP is responsible for the demarcation of transactions and the reporting of the transaction outcome, except when failures occur during transaction termination; it is then the responsibility of the TPSUI to determine the outcome and any necessary recovery by means outside of mechanisms provided by TP; or

c) "none" when the TPSUI is responsible for the demarcation of transactions and any necessary recovery.

During a transaction, the TPSUI may manipulate transaction data. Transaction data which is protected by the use of the "commitment" coordination level is called bound data (as defined in ISO/IEC 9804). Transaction data which is protected by application means is called "long lived data". Table 1 shows the permitted combinations of transaction data and coordination levels.

Table 1 - Permitted combinations of transaction data and coordination levels

Transaction data	Coordination level		
	commitment	one-phase commitment	none
bound data	YES	NO	NO
long lived data	YES	YES	YES

NOTE - The mechanisms, if any, required to maintain the ACID properties for long lived data are beyond the scope of ISO/IEC 10026.

7.4 Tree relationships

In this specification, a tree is a set of linked entities arranged in a hierarchical structure and connected by relationships. Two entities which are linked by a relationship are neighbours. An individual relationship defines roles for the two neighbours:

- the superior of the relationship is the entity which initiated it; and
- the subordinate of the relationship is the entity which accepted it.

Each entity can only have one superior; an entity which is already in one tree can not join in a further tree. Thus a tree does not contain any loops.

7.5 Dialogue

TPSUIs communicate among themselves in a peer-to-peer relationship; this peer-to-peer relationship between two TPSUIs is called a dialogue.

In a dialogue, TPSUIs may communicate for the following purposes:

a) transfer of data;

b) error notification;

c) initiation and termination of a transaction;

d) orderly or abrupt termination of their dialogue; and

e) handshake activities.

Dialogues may be controlled in two modes:

a) polarized control, when only one TPSUI has control of the dialogue at a time; and

b) shared control, when both TPSUIs have control of the dialogue simultaneously.

In polarized control mode, a TPSUI needs to have control of the dialogue to initiate a request other than

a) error notification;
b) rollback of a transaction;
c) early exit from a transaction;
d) abrupt termination of the dialogue; and
e) request control.

7.6 Dialogue tree

A dialogue tree is a tree with TPSUIs as the entities, and dialogues as relationships between the entities. The purpose of a dialogue tree is to support a sequence of one or more transactions.

Within the dialogue tree, the TPSUI that establishes the dialogue is referred to as the direct superior of the TPSUI with which the dialogue is established. The TPSUI with which the dialogue is established is referred to as the direct subordinate of the adjacent superior TPSUI.

The TPSUI in the dialogue tree that has no superior is called the root TPSUI. A TPSUI that has no subordinate is called a leaf TPSUI. A TPSUI that has both a superior and at least one subordinate is called an intermediate TPSUI.

7.7 Transaction branch

When requested, the TPSP provides the TPSUIs with a commitment service for use on a given dialogue. The value of the coordination level determines which commitment service if any is used on that dialogue by the TPSUIs:

a) "commitment" when the two-phase commitment service is used by the TPSUIs; or
b) "one-phase commitment" when the one-phase commitment service is used by the TPSUIs; or
c) "none", otherwise when no commitment service is used by the TPSUIs.

The portion of a distributed transaction performed by a pair of TPSUIs sharing a dialogue is called a transaction branch.

There are two basic kinds of transaction branches with respect to the division of responsibility between the TPSP and the TPSUIs:

a) application-supported transaction branches: transaction branches operating on a dialogue with coordination level equal to "none".

   For application-supported transaction branches, the TPSUI is responsible for maintenance of the ACID properties, recovery, and delineation of transaction branches.

   The TPSP provides only access to data transfer, error notification and dialogue control services, and is not aware of the beginning or completion of the application-supported transaction branches; and

b) provider-supported transaction branches: transaction branches operating on a dialogue with coordination level equal to "commitment" or "one-phase commitment".

   For provider-supported transaction branches with coordination level equal to "commitment", the TPSP is responsible for coordinating the maintenance of the ACID properties (therefore making use of globally unambiguous transaction identifiers, commitment, etc.), recovery, and delineation of transaction branches, as well as providing access to the remaining services.

   For provider-supported transaction branches with coordination level equal to "one-phase commitment", the superior TPSUI declares that it will have no bound data and does not require reliable reporting of the transaction outcome. The TPSP is responsible for delineation of transaction branches and reporting the transaction outcome to the superior TPSUI in the absence of failures.

Hereafter, for reasons of brevity, the term "provider-supported transaction branch" is referred to by the short term "transaction branch". When needed, the term "application-supported transaction branch" is used explicitly.

7.8 Transaction tree

A transaction tree is a tree with nodes (TPSUIs and their TPPMs) as the entities, and transaction branches as relationships between the entities. The purpose of a transaction tree is to support one transaction.

A transaction tree is formed over an existing dialogue tree. That is, the nodes of a transaction tree are those of an existing dialogue tree. A transaction tree extends over a connected part of the dialogue tree. Within a transaction tree, the TPSUI that initiates the transaction branch is referred to as the direct superior of the TPSUI with which the transaction branch is being established. The TPSUI with which the transaction branch is being established is referred to as the direct subordinate of the adjacent superior TPSUI.

The TPSUI in the transaction tree that has no superior is called the root TPSUI. A TPSUI that has no subordinate is called a leaf TPSUI. A TPSUI that has both a superior and at least one subordinate is called an intermediate TPSUI.

If a commit decision is taken in a transaction tree, the TPSP guarantees that all services related to transfer of data between the TPSUIs, error notification, and handshake activities, have been successfully completed on all transaction branches with polarized control mode selected.

7.9 Channel

During recovery there is a requirement for the AEIs to communicate directly with each other, without the involvement of any TPSUIs. This requirement is realized by channels; a channel is modelled as a relationship over an association; its purpose is to recover one or more transaction branches.

A channel is established between two AEIs over an existing association or one which has been established specifically for the purpose. Channels are established and terminated by a Channel Protocol Machine (CPM). The CPMs in two peer systems may establish one or more channels between them for the purpose of recovery.

A channel has the following properties:

a) it is not directly visible to the TPSUIs. There are, therefore, no OSI TP primitives referring to channels in the OSI TP Service; and

b) a channel is assigned by a CPM to a TPPM for the purpose of recovery.

For the purpose of recovery, channels are modelled as being used to recover one transaction branch at a time.

7.10 Handshake

TPSUIs may have to synchronize their activities, in order to reach a mutually agreed processing point. The semantics of such a processing point are application dependent.

When requested, the TPSP provides the TPSUIs with a handshake service, available for the duration of the dialogue, as a tool for application structuring, independently from the mode in which dialogues may be controlled.

7.11 Hinterland

7.11.1 Transaction hinterland

A transaction tree is constructed as described in clause 7.6. A transaction hinterland is a region in the transaction tree viewed from the perspective of a particular node in a particular direction.

If figure 1 represents a transaction tree where node F has exited the transaction prior to completion of the transaction (e.g. node F signalled read-only or early-exit to node E), then the transaction hinterland of node A viewed from node B consists of nodes A, E and F.

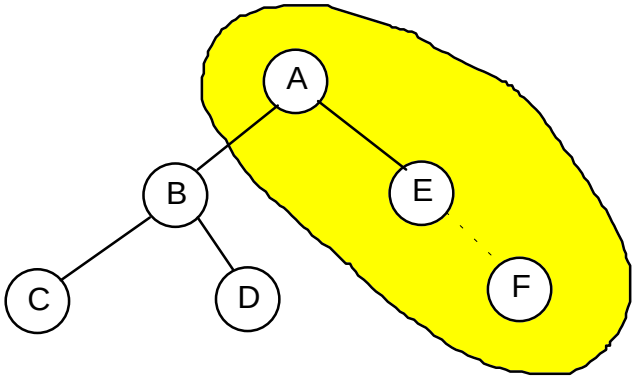


Figure 1 - Transaction hinterland of node A viewed from node B

7.11.2 Commitment hinterland

The commitment hinterland of a node consists of the set of nodes in the transaction tree which includes the neighbouring nodes from which ready signals have been received; and the commitment hinterland of those neighbouring nodes, and so on recursively. Thus in the previous example, if:

1) node F signals read-only to node E;
2) node E signals ready to node A; and
3) node A signals ready to node B;

then the commitment hinterland of node B consists of nodes A and E but not node F.

NOTE - with the static two-phase commitment procedures and no use of either read-only or early-exit or one-phase commitment, the commitment hinterland of a node will be identical to the transaction subtree of the node.

8 Model of the OSI TP Service

8.1 Nature of the OSI TP Service

The term OSI TP Service pertains to the service provided by the TPSP and used by the TPSUIs.

The following functions are associated with the OSI TP Service:

a) establishment, maintenance, and termination of the dialogue between two TPSUIs. The OSI TP Service

   1) provides for the selection of a TPSU from a set of TPSUs. The TPSU-title serves that purpose;

   2) ensures that the attributes requested by the initiating TPSUI are compatible with those of the recipient TPSUI. If so, the dialogue is established between a new invocation of the requested TPSU and the initiating TPSUI; and

   NOTE - From the OSIE perspective, a "new invocation" means a TPSU invocation which is not currently in the OSIE. It is a local matter as to whether the "new invocation" is mapped, in a real open system, to a new instance of the TPSU, or to an old instance that is being reused.

   3) provides means to both TPSUIs to interact, access remote resources, and possibly include them in a transaction.

b) according to the selected coordination level, overall coordination of resources, in a reliable fashion, to either successfully or unsuccessfully terminate a transaction. This achieves consistent state of resources, except possibly when heuristic decisions are taken. The ACID properties apply to the whole transaction, in particular to both remote and local resources.

   In order to allow control and management of local resources by either the TPSP, the TPSUI, or both, the coordination of resources may be fully located within the TPSP, or may be shared between the TPSP and the TPSUI. In the latter case, the TPSUI gathers the related information

from part or all of its local resources and controls the subsequent commitment or rollback of these local resources upon decision of the TPSP.

The OSI TP Service

1) includes the necessary provisions to coordinate all remote resources in order to ensure the application of the ACID properties: at termination of a transaction, the TPSP is responsible for coordinating the correct commitment or rollback of the entire set of remote resources; and

2) provides the ability to include local resources in the termination of the transaction. Depending on the sharing between the TPSP and the TPSUIs:

   i) the TPSP includes the local resources together with the remote resources in the termination of the transaction; or

   ii) the TPSP provides all the information required by the TPSUIs to correctly include (other) local resources such that the ACID rules can be applied to resources.

The TPSP guarantees, by the execution of the appropriate protocol, that all resources obey the ACID properties. In particular, the TPSP includes appropriate recovery mechanisms to re-establish a consistent state of all resources after failure and to resume transaction processing after re-establishment of a consistent state of all resources, when possible.

8.2 Rules on dialogue trees

8.2.1 Growth of dialogue trees

A TPSUI may activate remote TPSUIs in order to execute parts of a distributed transaction; this is done by having the remote open system invoke a new TPSUI and then establishing a dialogue with it (see also 8.2.3 and 8.4.1). It is in this way that a new dialogue is attached to the dialogue tree.

NOTE - From the OSIE perspective, a "new invocation" means a TPSU invocation which is not currently in the OSIE. It is a local matter as to whether the "new invocation" is mapped, in a real open system, to a new instance of the TPSU, or to an old instance that is being reused.

Attributes of the dialogue indicating the type of transaction processing to be performed are specified at establishment of the dialogue. These attributes determine the subset of communication facilities to be selected on that dialogue. These may include

a) the polarized control mode or the shared control mode;

b) the handshake service; and

c) the two-phase commitment service or the one-phase commitment service.

An uncoordinated dialogue (with an initial coordination level of "none") may be added to a dialogue tree at any time. A coordinated dialogue (with a coordination level of "commitment" or "one-phase commitment") may only be added when it is permitted to start a transaction, or to add a transaction branch to the current transaction.

A TPSUI may establish dialogues with one or more subordinate TPSUIs. However, two TPSUIs share at most a single dialogue. Communication may take place on some or on all dialogues of a TPSUI at the same time. All the dialogues of a TPSUI belong to the same dialogue tree.

8.2.2 Pruning of dialogue trees

Two TPSUIs that no longer need to communicate with each other may terminate their dialogue. They may do so at any time, provided that they ensure that the four ACID properties are still maintained.

A dialogue can terminate normally if and only if there is no transaction branch in progress on that dialogue. Dialogue termination is possible when

a) the coordination level is "none"; or

b) the current transaction branch is terminated, and the next one has not yet been started.

Dialogue termination may also occur upon communication failure or node crash. In this event, the corresponding transaction branch may be terminated with the dialogue.

When a dialogue between two TPSUIs is terminated, the dialogues in the subtree of the subordinate TPSUI do not necessarily need to be terminated. Hence, a new dialogue tree, previously part of an already established dialogue tree may be created. The new dialogue tree is independent from the dialogue tree from

which it originated. The intermediate node, for which the dialogue with the superior has been terminated, becomes the root of the new dialogue tree.

As dialogues are established and terminated, the dialogue tree changes.

8.2.3 Support of dialogue trees

A dialogue between two TPSUIs is supported by a single application-association.

When a dialogue is related to an application-association, there is a one-to-one correspondence between them at any given time. However, the lifetime of a dialogue and that of an application-association may be distinguished in that the lifetime of an application-association may span the lifetime of one or more dialogues.

The OSI TP Service does not constrain the establishment or existence of application-associations. In particular, they are not constrained to a tree or other topological structure between AEIs. Hence, they are considered to form a graph of interconnected open systems.

To be able to support a dialogue, an application-association must have been established

a) between the AEIs supporting the communications requirements of the TPSUs related to the requested dialogue;

b) with an application-context that supports the communication requirements of the TPSUs related to the requested dialogue;

c) with Presentation and Session Service support compatible with the requirements of the requested dialogue; and

d) with quality-of-service compatible with the requirements of the requested dialogue.

8.2.4 Initiative of activities and tree structure

The roles of superior node and subordinate node of a TP dialogue or of a transaction branch are clearly asymmetrical with respect to the TP protocol. This asymmetry corresponds to the fundamental assumption of this model that, at the application level, the superior node typically has the role of an initiator of activities while the subordinate contributes to these activities by reacting to requests which it receives from its superior. A subordinate recursively takes the role of an initiator of activities towards its subordinates, and so forth.

Sometimes there may be an application requirement to transfer the initiative and topmost responsibility for the further execution of an application task from one node to a neighbouring node, e.g. from a client to a server. The node now interested in giving up the initiative, may have originally created the task being processed in the current transaction but either wants to withdraw to an observer position (in TP terms becoming a subordinate) or it wants to disconnect totally from the transaction - at least for the time being with the intent to discover the outcome and its detailed results at a later time and outside of the current transaction.

To allow a root node of a transaction tree to exchange roles with a neighbouring subordinate node would be an extremely difficult operation; important parts of the protocol flow would have to be redirected. Thus a node should be enabled to spontaneously initiate the establishment of a dialogue tree and subsequently a transaction tree and nevertheless to take over a subordinate role in the tree from the very beginning. It could then define the essence of an application task and transfer its execution to the root node.

When the establishment of a TP dialogue is solicited at a remote system, an already existing association is offered that is appropriate for the requested dialogue; and it must be used for this purpose by the dialogue establishing system.

The TP protocol assumes that the soliciting entity representing a specific information processing context is a (TP) node. The incoming dialogue needs a node as subordinate; whether this is the mentioned soliciting node or a newly created one (which then takes over the given information processing context) is a local matter.

8.3 Rules on transaction trees

8.3.1 Growth of transaction trees

A new transaction branch may only be added to a transaction tree prior to the commencement of the transaction termination procedures (see 8.6).

There are two ways to grow a transaction tree:

a) a new transaction branch is added to the transaction tree, as perceived by the TPSP, by establishing a new coordinated dialogue (i.e. a dialogue with coordination level of "commitment" or "one-phase commitment"); and

b) where the coordination level is permitted to change (see also "unchained sequences" in 8.3.3), a new transaction branch is added to the transaction tree when the coordination level changes from "none" to either of "commitment" or "one-phase commitment". Only a superior node of the dialogue tree is allowed to modify the coordination level.

8.3.2 Lifetime of transaction trees

A transaction tree lasts only for the duration of a single transaction.

Where the coordination level is permitted to change (see also "unchained sequences" in 8.3.3), the coordination level can only change to "none" at the termination of a transaction branch.

The growth and termination of a transaction tree are not immediate. Both actions require multiple elementary exchanges that must be propagated throughout the transaction tree.

8.3.3 Support of transaction trees

The existence of a dialogue between two TPSUIs is a prerequisite for a transaction branch to be established between the two TPSUIs.

Where the coordination level is "commitment" or "one-phase commitment", there is normally a one-to-one correspondence between a dialogue and a transaction branch at any given time. The TPSP is aware of the relationship between dialogues in a dialogue tree and the branches in the corresponding transaction tree(s), and coordinates their combined operations, for example, to achieve consistent commitment semantics across all of the open systems involved in a transaction.

The root of a transaction tree is not necessarily placed at the root of the dialogue tree. Within the bounds of the transaction tree, and with respect to the superior to subordinate relationships, there is a one-to-one correspondence between the nodes of the transaction tree and nodes of its supporting dialogue tree. The transaction tree and its supporting dialogue tree have the same orientation.

A dialogue whose coordination level is "none" does not support a transaction branch of a transaction tree.

The same dialogue tree may be used to support a sequence of distinct transactions. The relationship between dialogues in the dialogue tree persists across these distinct transactions. Within the bounds of a dialogue, a sequence of one or more transaction branches may take place. Two types of sequences are permitted:

a) chained sequences: these are uninterrupted sequences of one or more transaction branches on the same dialogue that operate at the same coordination level (either "commitment" or "one-phase commitment"). Each transaction branch is initiated directly by the superior TPPM; and

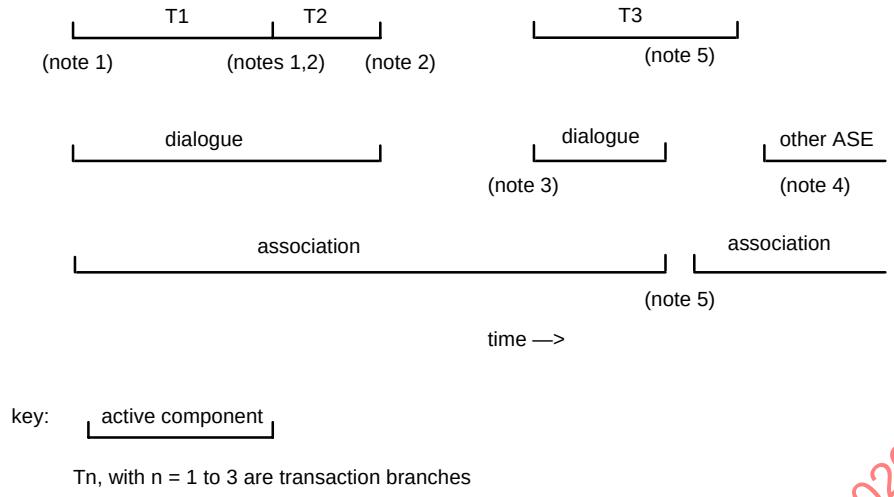
b) unchained sequences: these are sequences of transaction branches on the same dialogue such that there is a dialogue coordination level transition to "none" between each transaction branch. At dialogue establishment time as well as at dialogue termination time, the coordination level may be one of "none" or "commitment" or "one-phase commitment". Each transaction branch is initiated by the superior TPSUI.

If a part of a dialogue tree exists which has no transaction in progress (i.e., the dialogues have a coordination level of "none"), then a TPSUI in this part of the dialogue tree may initiate a new transaction. This can lead to there being zero, one or more transaction trees in a single dialogue tree, at the same time.

At any given time, transaction trees are disjoint among themselves. Between two transaction trees, disjunction shall be guaranteed by at least one dialogue with coordination level of "none".

After dialogue termination between an intermediate node and its superior, the intermediate node becomes the root of a new dialogue tree, and may become the root of a transaction tree.

Figure 2 shows the correspondence, over time, between transactions branches, dialogues and associations. This set of correspondences is depicted between two adjacent open systems.



NOTES

1 The beginning of a transaction branch occurs either at the beginning of a dialogue or during a dialogue.

2 The end of a dialogue implies the end of the current transaction branch. The end of a transaction branch occurs either during a dialogue or at the end of a dialogue.

3 A dialogue may follow another dialogue within the bounds of the same application-association.

4 Another ASE can use the application-association when the dialogue terminates.

5 If the application association fails, the dialogue is immediately terminated. However if the transaction is in the READY or DECIDED (commit) state, transaction recovery will take place on a further association; the transaction branch will continue to exist until transaction recovery is completed.

Figure 2 - Transaction branches, dialogues, and application-associations

8.4 Naming

In addition to the naming facilities already established for OSI in ISO 7498-3, OSI TP requires titles for TPSUs, and identifiers for transactions and transaction branches. Definitions for these names and identifiers are given in clause 3.

8.4.1 TPSU-title

The TPSU-title is used during dialogue establishment to select a TPSU within a designated application-process with which the dialogue is to be established. The dialogue is established between the initiating TPSUI and a recipient TPSUI of the TPSU specified by the TPSU-title. The dialogue is established over an application-association (either pre-existing or newly established) between the two application-entity-invocations supporting the respective TPSUIs.

By denoting the target TPSU for dialogue establishment, the TPSU-title indicates the processing capabilities of the TPSU.

In the case where the dialogue is established over a pre-existing application-association, the TPSU-title may be used to derive information necessary to enable the initiating TPPM to select a suitable application-association from among those which may be available. An example of this information is the application-context.

In the case where no pre-existing association is available, the TPSU-title may be used to derive information necessary to enable the TPPM to establish the required association.

The TPSU-title is unambiguous within the scope of an application-process.

8.4.2 Transaction identifier

A transaction is denoted unambiguously within the OSIE by a transaction identifier. The transaction identifier consists of

a) the application-entity-title of the application-entity that supports the root node of the transaction; and

b) the transaction suffix, the value of which is unambiguous within the scope of the application-entity that supports the root node of the transaction. For example, the transaction suffix may be an integer which is incremented by one for each new transaction instantiated.

NOTE - The transaction identifier should also be globally unambiguous over time, with respect to recovery and business auditing requirements.

8.4.3 Transaction branch identifier

A transaction branch is denoted unambiguously within the scope of a transaction by a transaction branch identifier. The transaction branch identifier consists of

a) the application-entity-title of the application-entity that supports the superior node of the transaction branch; and

b) the transaction branch suffix, the value of which is unambiguous within the scope of the application-entity that supports the superior node of the transaction branch.

8.5 Data transfer

8.5.1 Requirements and objectives

To meet the requirements of TPSUIs involved in a distributed transaction to exchange data, the OSI TP Service allows data transfer to meet the following objectives:

a) the OSI TP Service allows the TPSUI to convey data according to its own semantics;

b) data transfer always relates to a single dialogue;

c) the TPSUI is free to organize the style of its semantic exchange using one or more specific user-ASEs. In particular, its semantic exchanges may be based on different disciplines; and

d) the definition of user-ASEs may be the same whether they work with or without OSI TP.

8.5.2 Coordination of data transfer

User-ASEs generate the data transfer APDUs which are mapped onto underlying services, coordinated by the SACF.

The TPPM handles the protocol for dialogue management; it does not itself directly generate data transfer APDUs.

The TPPM determines the temporal ordering of the use of the underlying application-association for Transaction Processing.

Hence, within OSI TP, data transfer

a) may occur only within the bounds of a dialogue;

b) is subject to control modes. In particular, in the polarized control mode, data may only be sent if the TPSUI has control of the dialogue. The selection of the control mode depends on the particular requirements expressed by the user-ASEs; and

c) is subject to the states of the TPPM.

The TPPM ensures that data transfer is coordinated with the commitment phases during the termination of the transaction.

8.6 Coordination of resources

8.6.1 Two-phase commitment

The termination phase of a distributed transaction is entered upon request from one or more TPSUIs of the transaction tree. Within the transaction tree, the TPSP coordinates the termination phase among the TPSUIs to ensure that the transaction's bound data will be released in a consistent state. Coordination of the termination phase occurs in two steps:

a) commitment phase 1; and

b) commitment phase 2.

In commitment phase 1, the participating nodes attempt to place all bound data in the transaction tree in the ready-to-commit state. Bound data are in the ready-to-commit state if, until the transaction has been

terminated by commitment or rollback, they can be released in either their initial or their final state. If all bound data in the transaction tree are placed in the ready-to-commit state, then commitment phase 2 is entered; if this can not be achieved, then the transaction is rolled back.

In commitment phase 2, within the transaction tree, the transaction may be committed whenever:

a) all the bound data are in the ready-to-commit state; and

b) there are no ongoing operations changing the transaction tree, e.g. establishment of new transaction branches, or affecting communication between its nodes.

If the transaction is to be committed, the TPSP propagates the commit decision throughout the transaction tree and coordinates the completion of the transaction.

After completion of these two steps, commitment is complete. A new transaction may or may not begin.

NOTE - these procedures may be modified if read-only, early-exit or one-phase commitment procedures are used; see 8.6.2, 8.6.3 and 8.6.4.

8.6.1.1 Static commitment procedures

In phase 1, each node is informed by its superior that the termination phase has been entered; in particular, that no more data will be received from the superior, and that bound data shall be put in the ready-to-commit state.

If the node agrees to proceed, it attempts to place the bound data, within its subtree, in the ready-to-commit state. The node attempts to place its local bound data in the ready-to-commit state. For remote resources, it informs its subordinates; etcetera, recursively.

Whenever, within its subtree, all its bound data are in the ready-to-commit state, the node notifies its superior (if any) by sending a ready signal, and waits for the final outcome of the transaction; etcetera, recursively; if the node has no superior in the transaction tree, phase 2 of commitment is entered. A node which sends a ready signal to its superior becomes a commit slave and the superior becomes its commit master. The set of subordinates (including their subtrees) from which a node receives ready signals becomes its commitment hinterland.

If application data were to be received after a ready signal or a read-only or one-phase signal was sent, bound data may be affected, thus invalidating the 'readiness' of the node. Such 'ready/data' collisions are to be avoided; they are only possible for some combinations of functional units, and applications which use these combinations of functional units must ensure that such collisions do not happen. This collision is only possible on a branch if polarized control mode is not selected.

NOTE - 10026-3 contains scenarios which illustrate these collisions.

If the node is unable to place the bound data in the ready-to-commit state, it initiates rollback of the transaction.

In phase 2 after the commit decision is taken, each node is ordered by its superior to release the bound data within its subtree in the final state. The node commits its local bound data. For remote resources, it orders its subordinates to commit; and so on, recursively.

Whenever, within its subtree, all its bound data have been released in the final state, the node informs its superior; and so on, recursively. The transaction is complete.

8.6.1.2 Implicit prepare

The right to initiate the termination phase can be restricted to the root node and the signal that the termination phase has been entered (prepare signal) is then carried from the superior node to the subordinate node by the TPSP.

Alternatively, the signal that the termination phase has been entered can be carried implicitly in the application semantics - this is Implicit Prepare. The signal may be carried in a particular application message, may be an agreed inference from a sequence of messages, or may be implied by the beginning of the transaction.

NOTE - The use of Implicit Prepare can allow the ready state to be reached earlier in a transaction subtree and so reduce the overall time taken for completion of the transaction.

The passage of the implicit prepare signal thus is not visible to the TPSP. Consequently, from the perspective of the TP service provider, the use of an implicit signal can not be distinguished from granting the right to

initiate the termination phase to any node. If the implicit prepare mechanism is in use between a superior and subordinate, the subordinate may enter commitment phase 1, and attempt to bring its subtree to the ready state at any time.

An intermediate node that does not use the implicit prepare mechanism with its superior, but does with one or more subordinates does not enter commitment phase 1 until permitted by a signal from the superior. However, the TPSP is unable to police the entry into commitment by the subordinates.

NOTE - Attempting to provide such a tree-wide policing would require considerable complexity in protocol exchanges.

8.6.1.3 Dynamic commitment procedures

In phase 1, each node can apply the termination procedure locally when it has permission to initiate termination (because it is the root, or implicit prepare is in use) or proceed with termination (because it has received a signal from its superior that termination has begun), provided that it can determine that no more data will be received from any neighbour.

If a node agrees to proceed with commitment, it attempts to place its local bound data in the ready-to-commit state. If the node is unable to place its bound data in the ready-to-commit state, it initiates rollback of the transaction.

Whenever all its bound data are in the ready-to-commit state and ready signals or read-only or early-exit or one-phase signals have been received from all but one neighbour, the node sends a ready signal to the one remaining neighbour from which a ready signal has not been received, and waits for the final outcome of the transaction; etcetera, recursively. The nodes from which ready signals are received are the commit slaves of the node; the node to which it sends a ready signal (if any) is the node's commit master.

If a node using the dynamic commitment procedures sends a ready signal to a subordinate, and the prepare signal has not previously been sent, the ready signal itself is the signal to the subordinate that the termination phase has been entered.

NOTE - If the implicit prepare mechanism is in use, the subordinate may already know this.

A node's current commitment hinterland is the set of nodes in the transaction tree which include:

a) the neighbouring nodes from which ready signals have been received; and

b) the commitment hinterland of those neighbouring nodes, and so on recursively.

If application data were to be received after a ready signal was sent, bound data may be affected, thus invalidating the 'readiness' of the node. Such 'ready/data' collisions are to be avoided; they are only possible for some combinations of functional units and applications which use these combinations of functional units must ensure that such collisions do not happen. This collision is only possible on a branch if polarized control mode is not selected.

NOTE - 10026-3 contains scenarios which illustrate these collisions.

Whenever all its bound data are in the ready-to-commit state and ready signals or read-only or early-exit or one-phase signals have been received from all neighbours, a node will enter phase 2 of commitment and the commit decision may be taken. A node may receive a ready signal from a node to which it sent a ready signal (a ready/ready collision). In this case, a tie-break mechanism is used within the TP protocol specification to determine which node will make the commit decision and so become the commitment coordinator.

NOTE - an alternative design would have been to allow both nodes which received a ready signal having sent a ready signal, to be commitment coordinators and to propagate commitment. However to do so would have introduced further complexity into the protocol machines to cope with the commit-log record failing to be written in one or both nodes; for simplicity a tie-break mechanism has been adopted.

In phase 2 after the commit decision is taken, each node is ordered by its commit master to release the bound data within its commitment hinterland into the final state. The node commits its local bound data. For remote resources, it orders its commit slaves to commit; and so on, recursively.

Whenever, within its commitment hinterland, all its bound data have been released in the final state, the node informs its commit master; and so on, recursively. The transaction is complete.

Mechanisms exist to allow the direction of ready signals to be controlled. It would be possible for a transaction tree to be created such that commitment would never be possible; such a tree is called a "deadlocked transaction tree". Checks are included to guarantee that such tree can not be created; however this can only be done by erring on the side of safety and inhibiting certain tree structures that are in fact

viable. TP provides an optional facility to allow some checks to be inhibited, in which case the application is responsible for ensuring that a deadlocked transaction tree is not constructed.

8.6.2 Read-only

A node which has processed all requests from its superior and has not changed any transaction data (either bound data or long lived data) from its initial state may attempt to withdraw from the two-phase commitment procedures as its transaction data will not be affected by whether the transaction is committed or rolled back. If such a node receives read-only or early-exit signals from all of its subordinates, it can send a read-only signal to its superior. Once a node has issued a read-only signal then it does not participate further in the commitment procedures. Such a node will be unaware of the transaction outcome (whether it committed or rolled back) unless either the chained transactions functional unit is selected for the dialogue or a deferred action is outstanding on it, and the transaction rolls back. A node which signals read-only has no obligation to write any log records to secure storage.

With unchained transactions, a node which signals read-only ceases to be part of the transaction tree once it receives a completion indication; it is then free to initiate other actions even though the transaction of which it was once a participant is still in progress. The superior of such a node could initiate a further transaction branch with the node for the same transaction.

NOTE - a node which issues a read-only signal is not a commit slave and does not form part of the commitment hinterland.

8.6.3 Early-exit

A node which can not contribute to the work of a transaction and which has received read-only signals or early-exit signals from all subordinates in the transaction tree, if there are any, may withdraw from the transaction by issuing an early-exit signal to its superior. Additional conditions on issuing an early-exit signal are that the transaction data of this node have not been altered by the transaction and that reporting of the transaction outcome is not required. Any further requests received from the superior related to the current transaction branch are discarded by the TPSP.

NOTE - for example, a node may determine from a request received from its superior that it does not have access to data required for the fulfilment of the request.

With unchained transactions, a node which signals early-exit ceases to be part of the transaction tree once it receives a completion indication; it is then free to initiate other actions even though the transaction of which it was once a participant is still in progress. The superior of such a node could initiate a further transaction branch with the node for the same transaction.

NOTE - a node which issues an early-exit signal is not a commit slave and does not form part of the commitment hinterland.

8.6.4 One-phase commitment

A node which has no bound data but which desires to discover the transaction outcome can use the one-phase commit procedures. If such a node receives one-phase commit signals or read-only signals or early-exit signals from all but one neighbour, it can send a one-phase signal to its last remaining neighbour. Once a node has issued a one-phase signal then it will receive notification of the transaction outcome as long as a dialogue or node failure between it and the commitment coordinator does not prevent the decision from being transmitted. A node which signals one-phase has no obligation to write any log records to secure storage.

NOTE - a node which uses one-phase commitment may have transaction data (ie data which is manipulated during the transaction) but which is not bound data in the strict sense of the definition given by CCR; this long lived data may be released in an updated state at the termination of the transaction but if certain failures occur, such release will not be coordinated with the transaction outcome (for example, the data could be released in an updated state yet the transaction may roll back). Coordination may then be attempted later by other means than the TP protocol.

NOTE - a node which issues a one-phase signal is not a commit slave and does not form part of the commitment hinterland.

8.6.5 Rollback

Rollback of a transaction may be initiated by any node of the transaction tree that has not previously issued a ready signal or a one-phase signal or a read-only signal or early-exit signal. Rollback returns the transaction's bound data to their initial state.

Issuing rollback does not, by itself, make the underlying dialogue terminate. If a TPSUI desires to terminate the dialogue, it may abort it. If a dialogue is aborted before the commencement of the transaction termination procedures, the transaction is rolled back.

After rollback has been completed, a new transaction may (but need not necessarily) be initiated.

8.6.6 Heuristic decisions

After it has entered commitment phase 1, a node may decide to release part or all of its bound data, which has reached the READY state, in the final state or initial state even though it has not been notified by its commit master of the final outcome of the transaction. Such a decision is called a heuristic decision.

Heuristic decisions may be taken by individual nodes as the result of a communication failure, or as a result of system specific local conditions. The decision whether or not to take one or more heuristic decisions and what decisions to take is a local matter. Within the scope of OSI TP, whenever a node takes a heuristic decision, no propagation of the decision occurs to other nodes.

A node that has taken a heuristic decision is required to record that decision, using a log-heuristic record, in secure storage. If the state of the node's bound data and the outcome of the transaction prove to be consistent, then the log-heuristic record is erased, and normal termination of the transaction proceeds.

8.6.7 Detection of heuristic inconsistency

A node that has taken a heuristic decision determines that a heuristic inconsistency exists if the state of its local bound data is inconsistent with respect to the outcome of the transaction. The node can make this determination as soon as it is informed of the final outcome of the transaction by its commit master. If the state of the node's bound data is inconsistent with the outcome of the transaction, then the ACID properties have been violated. This is a heuristic-mix condition.

A heuristic-hazard condition exists when a node is unable to determine the exact state of the bound data for its subordinate nodes within their subtree. This would result if communication were lost with one or more subordinates. If the final outcome of the transaction was to rollback, the state of the bound data of the subtree cannot be reported to the direct superior. This is due to presumed rollback (see 8.7.2 and annex C). This is a heuristic-hazard condition, as the state of the bound data within the subtree is potentially a heuristic-mix.

A heuristic-hazard condition also exists if a TPSUI is unable to determine whether the state of the local bound data is consistent with the outcome of the transaction. This would come about as a result of a local loss of communications.

8.6.8 Reporting

A node may apply a policy of heuristic containment such that heuristic inconsistency occurring within its subtree is contained and rectified within the subtree. Such a node will never report heuristic damage to a superior, but will otherwise behave as described here.

Unless one or more nodes contain the heuristic information of their subtree, each node acquires knowledge of the state of bound data within its subtree, and thus the root node will acquire knowledge of the state of bound data within the whole transaction tree.

Within the TPSP, each TPPM collects reports on the state of the bound data within its subtree, as a result of

a) the state of the node's local bound data compared to the final outcome of the transaction; and

b) the report, from each subordinate, on the state of the bound data in the subordinate's subtree.

If the node determines that the state of the bound data in its subtree is consistent with the final outcome of the transaction, the TPPM reports to its superior that all bound data in its subtree is in a consistent state.

If the node determines that the state of the bound data within its subtree is inconsistent with the final outcome of the transaction, and is unable to compensate for the inconsistency, then the TPPM

a) as reports are collected, retains knowledge of bound data inconsistency within the node's subtree by means of the log-damage record. Refer to table 2 for resulting values of the log-damage record according to reported inconsistency;

b) reports the inconsistency to its TPSUI;

c) reports to the superior node, if any, whenever a complete report of the state of the bound data within its subtree is available; and

d) reports the inconsistency to some local entity, e.g. a system operator.

Table 2 - Update of log-damage record

Previous state of log-damage record	Reported inconsistency		
	No inconsistency	**Heuristic hazard**	**Heuristic mix**
no log-damage record	no report	heuristic-hazard	heuristic-mix
heuristic-hazard	heuristic-hazard	heuristic-hazard	heuristic-mix
heuristic-mix	heuristic-mix	heuristic-mix	heuristic-mix

The log-damage record is kept after the propagation of the final outcome of the transaction, until assurance has been received that its superior has received appropriate reporting.

NOTES

1 This does not imply that the information about the inconsistency may not be kept until damage has been repaired.

2 The mechanism by which the subordinate node is assured that the superior is aware of the heuristic damage is outside the scope of OSI TP.

3 Optionally, heuristic reporting can be suppressed; the above clause describes the case where it is not suppressed.

4 Heuristic reports are not reliably transmitted to nodes which use the one-phase commitment procedures.

The collection of information on the state of bound data and its transmission towards the superior can impose delays on the completion of the transaction. These delays are not imposed if a dialogue uses the Heuristic Containment Required facility. If the Heuristic Containment Required facility is not employed, a node that, nevertheless, applies heuristic containment, will send TP messages as if it were reporting, but the reports will always be empty or absent.

Where a node is using the Implicit Prepare facility on the dialogue to the superior, but the superior is not using Implicit Prepare or Dynamic Commit to its superior, it is possible for the lowest node to make a heuristic decision before the top-most node has given the intermediate node permission to enter commitment phase 1. In this case, the intermediate node must apply heuristic containment, to avoid sending a heuristic report to the top-most node.

8.7 Recovery

8.7.1 Types of failure

8.7.1.1 Introduction

Table 3 identifies the potential causes of failures, the types of failures which may occur during a transaction, and the actions which should be taken to return the transaction to a manageable state.

Table 3 - Types of failures

Possible causes	Failure type	Action by TPPM
application error	locally recoverable	none
transaction abort; or dialogue abort	recoverable before node is ready to commit	rollback
dialogue abort	recoverable after node is ready to commit	recovery procedures
node crash; TPPM failure; or AEI failure	atomic action data unavailable	recovery of atomic action data; dialogue abort; possibly, association abort; and recovery procedures
storage media failure	atomic action data destruction	beyond scope of ISO/IEC 10026

8.7.1.2 Locally recoverable failures

If a failure occurs, the TPSUI may be able to recover by its own means such that the transaction can continue to commit. If the TPSUI or TPPM does so, there is no external manifestation (other than possibly delays) of the incident. This case is a local matter.

8.7.1.3 Failures recoverable before the node is ready to commit

Any failure occurring before the node is ready to commit causes a rollback. These failures may originate from either

a) transaction abort, due to the following:
   1) the inability of the TPSUI to operate for the current transaction such that a rollback is explicitly requested;
   2) a distributed deadlock, where a transaction is part of a waiting cycle with other transactions;
   3) a storage media failure where the current value of the bound data is no longer accessible, but the bound data in their initial state are available; or
   4) a storage media failure where the bound data are destroyed, but the state of the transaction is known and local intervention is required to re-construct the bound data; or

b) dialogue abort, due to the following:
   1) a failure in the application-association supporting the dialogue. This could occur as a result of a failure in ACSE, the Presentation service, or a supporting service (e.g. the Session service);
   2) an application protocol error in any of the following protocols on the dialogue:
      i) user-ASEs;
      ii) OSI TP; or
      iii) CCR.
   3) a user-ASE failure; or
   4) a failure of the TPSUI and/or the TPPM such that they are unable to continue communication on the dialogue (e.g. node crash).

8.7.1.4 Failures recoverable after the node is ready to commit

Recoverable failures occurring after the node is ready to commit are those failures that cause dialogue aborts. Upon failure, a recovery procedure is initiated to complete the transaction. See 8.7.1.3(b) for possible causes of dialogue abort.

8.7.1.5 Atomic action data unavailability

A failure has occurred on an open system such that the working copy of the atomic action data for the current transaction is unavailable. The working copy of the atomic action data may have become unavailable (i.e. lost) due to failures, for example a TPPM failure, an AEI failure, or a node crash.

The recovery log must be read to restore the working copy of the atomic action data for the transaction. All dialogues and/or underlying associations that pertain to the current transaction are aborted.

8.7.1.6 Atomic action data destruction

The atomic action data have been lost due possibly to a storage media failure. Recovery from this type of failure is beyond the scope of ISO/IEC 10026.

8.7.2 Support for recovery of transactions

The TPSP includes facilities to recover after communication failure or node crash. Recovery support is limited to recovery of transactions. Recovery of a transaction means that, after occurrence of a failure, all bound data that have been involved in the transaction will be re-instated to their final state or their initial state. It is the responsibility of the TPSP to ensure that all resources are re-instated to the same consistent state, i.e. either the final or the initial state.

Transaction recovery is achieved within the bounds of the transaction tree by the TPSP. Outside of the transaction tree, recovery is the responsibility of the TPSUIs.

Provision for transaction recovery requires that key steps in the progress of transaction branches (atomic action data) have been appropriately logged in every open system involved in the transaction tree. The presumed rollback two-phase commit protocol is used.

The following information must not be lost, and thus must be saved in the recovery log:

1) atomic action data: these data are not subject to the commit/rollback procedure, but are used during the recovery process; and
2) bound data: the data of the objects on which the transaction operates. These data are subject to the commit/rollback procedure, are invisible from outside of the transaction during the execution of the transaction, and will only be available to any other transaction after the transaction has terminated.

8.7.3 Node states

When a failure occurs, the transaction may either be active or in the termination process. In the latter case, timing is an important factor in the recovery mechanism. The termination of a transaction is not immediate since several steps and exchanges are needed in the whole transaction tree between the moment where a TPSUI asks for the termination and the moment it is informed of the completion of its request.

When a failure occurs during transaction termination, the branches of the transaction may be in different states. Therefore recovery may require different types of action depending on the states of the nodes within the transaction tree.

The node can be in one of the following states:

a) **ACTIVE**: processing of the transaction is going on. The node can choose to order rollback of the transaction and release bound data under its responsibility in their initial state without threatening their consistency.

   According to the presumed rollback approach, the node is not required to log in the recovery log the creation of any transaction branch.

b) **READY**: either

   1) the node is able to put the bound data under its own responsibility into their final state (committed) or into their initial state (rolled back). The node has received ready signals or read-only or early-exit or one-phase signals from all neighbours but one; i.e. there is precisely one adjacent node from which it has not received such a signal. Thus the bound data of this node and all nodes in its commitment hinterland are in the ready state.

Before sending a ready signal for the complete commitment hinterland to the adjacent node from which no ready or read-only or early-exit or one-phase signal has been received (i.e. indicating that the commitment hinterland can be committed), the node shall write a log-ready record in the recovery log. The log-ready record includes the transaction identifier, the ready vote, the list of the adjacent nodes that have signalled ready and the identification of the adjacent node to which the ready signal will be sent; this latter node will be the commit master. After writing this log-ready record, the node is in the READY state; or

2) the node had previously entered the READY state, receives a ready signal from the node to which it had sent a ready signal and determines it is to be commit slave to the node.

c) **READ-ONLY**: the transaction data of the node have not been modified during the current transaction, and read-only or early-exit signals have been received from all subordinates. The node has no requirement to be informed of the outcome of the transaction.

No logging is required for a node to enter the READ-ONLY state.

d) **EARLY-EXIT**: the node has exited from the transaction before the termination phase; the transaction data of the node have not been modified during the current transaction, and read-only or early-exit signals have been received from all subordinates. The node has no requirement to be informed of the outcome of the transaction.

No logging is required for a node to enter the EARLY-EXIT state.

e) **ONE-PHASE**: the node has accessed no bound data during the current transaction, and read-only or early-exit or one-phase signals have been received from all neighbours but one. The node is to be informed of the outcome of the transaction if no failure occurs, but recovery is not required at this node.

No logging is required for a node to enter the ONE-PHASE state.

f) **DECIDED (commit)**: A node enters the DECIDED (commit) state when the node is able to put the bound data under its own responsibility into the final state, and either

1) all the adjacent neighbours have signalled ready or read-only or early-exit or one-phase and the node has determined it is the commitment coordinator; or

2) the node had previously entered the READY state and has since been ordered to commit by its commit master.

If the node has determined it is the commitment coordinator and decides to commit the transaction, it shall write a log-commit record in the recovery log before it propagates the decision to the commit slaves. After deciding to commit the transaction, the node will also propagate the commit decision to any neighbour from which a one-phase signal was received if the dialogue still exists. The log-commit record includes the transaction identifier, the commit decision, and the list of the adjacent nodes that signalled ready. After writing the log-commit record, the node is in the DECIDED (commit) state. Such a node, that writes a log-commit record without receiving an order to commit, is called the commitment coordinator for the transaction.

A collision of ready signals may occur, i.e. two neighbouring nodes may send ready signals to each other which collide on the dialogue. Then, by means of a tie break mechanism (described in ISO/IEC 10026-3, under Collisions of Ready Signals) one of the ready signals is ignored by the two nodes in a consistent manner; after this, the normal procedures are applied.

NOTE - the objective for this way of handling the collision of ready signals is to limit protocol complexity by returning, as fast as possible, to a state that is typically reached when no collision has occurred. The advantage of shortening the average time needed to inform the nodes of the commit decision by letting both nodes become autonomous sources of the commit decision is given up.

If the node was in the READY state and then receives an order to commit, it immediately enters the DECIDED (commit) state. The node propagates the decision to its commit slaves;

the node will also propagate the commit decision to any neighbour from which a one-phase signal was received if the dialogue still exists. The node optionally can write a log-commit record in the recovery log. The log-commit record includes the transaction identifier, the commit decision, and the list of the commit slaves. There is no requirement for a non-commitment coordinator to log the commit decision in the recovery log; however, in doing so, it may gain performance in the recovery procedure.

In all cases, after commitment has been completed at the node (bound data released) and all adjacent nodes that signalled ready have reported that commitment has either been completed or logged, it may remove the log-ready or log-commit record. If it is not the commitment coordinator, it then reports that commitment is completed to the commit master.

If the node was not a commitment coordinator but writes the optional log-commit record, it may optionally report commitment completion when it has written the log-commit record (but see below on heuristic reporting).

A node that ordered commitment to a neighbour may optionally remove information about that neighbour from the log-commit or log-ready records when that neighbour reports completion of commitment.

g) **DECIDED (rollback)**: The node decided to rollback the transaction or received an order to rollback while it was in the ACTIVE state, or it was in the READY state or READ-ONLY or ONE-PHASE states and either

1) received an order to rollback from the neighbour to which it sent a ready signal or read-only or one-phase; or

2) determined that it was the commitment coordinator and decided to rollback the transaction.

No logging is required for the rollback decision as such. If a node was in the READY state and receives a rollback order or determines it is the commitment coordinator and decides to rollback the transaction, it may delete its log-ready record.

h) **DECIDED (unknown)**: The node which was in the READ-ONLY or EARLY-EXIT state received an indication that it will not be informed of the transaction outcome.

No logging is required for a node entering the DECIDED (unknown) state.

The state of a node determines whether a heuristic decision is possible. A heuristic decision is only possible if the bound data are in the ready-to-commit state. A node in the ACTIVE state may have some of its bound data already in the ready-to-commit state, provided it has permission to enter commitment phase 1, as described in 8.6.1.2. A node in the READY state will have its bound data in the ready-to-commit state.

Thus heuristic decisions are possible for a node in the ACTIVE state, if it has permission to enter commitment phase 1 or in the READY state. In either case, the node stays in the same state.

Depending on constraints chosen by the superior, the state of bound data at completion of the transaction as affected by heuristic decisions may be reported to the superior, and eventually to the root of the transaction. If a heuristic decision is made (i.e. if a node releases bound data in the initial or final state before it has received the final outcome of the transaction), the node writes a log-heuristic record in the recovery log before releasing the data. The log-heuristic record includes the transaction identifier and the decision.

NOTE – the direction of heuristic reporting is always to the root of the transaction tree, and thus may be opposite or in the same direction to the commit-orders.

At a node where a heuristic decision was made, if the final outcome of the transaction is different from the heuristic decision and the node is not able to repair the damage, a log-damage record is written.

The following paragraphs apply if heuristic reporting is required and the final outcome of the transaction was to commit.

If the node received the commitment order from its superior, it will not report completion of commitment until it has been informed of the state of bound data from its transaction subtree.

If the node received the commitment order from a subordinate, the subordinate will inform the node of the state of bound data in the subordinate's subtree when this is known. If this report has not been received when

the superior node is otherwise ready to report completion of commitment, the superior will wait for the report; this ensures that the state of bound data will be reliably determined.

8.7.4 Phases of recovery

8.7.4.1 Overview

Recovery in OSI TP can be divided into three distinct steps called recovery phases:

a) failure detection and containment;

b) transaction recovery; and

c) application or user recovery.

The objective of each phase is independent of the state of the node, but the type of recovery action taken within each recovery phase is dependent on the state of the node. The sequence of recovery phases is illustrated by figure 3.

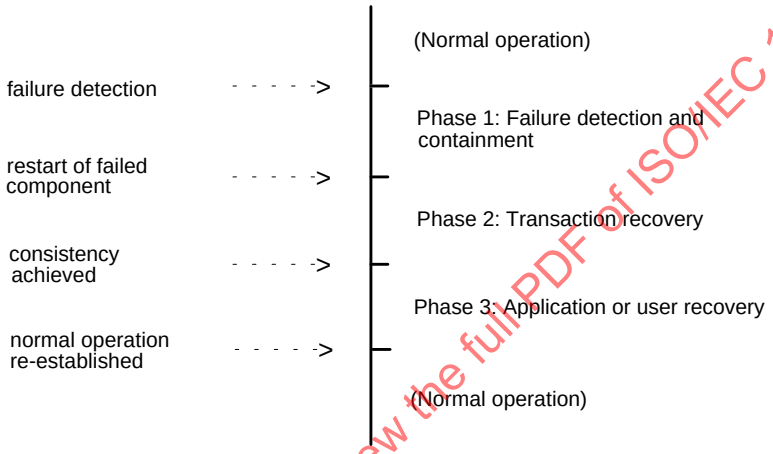


Figure 3 – Phases of recovery

Phase 1 is entered upon failure detection; recovery is initiated. From the TPSP perspective, communication on some branch of the transaction tree is impossible. This phase attempts to limit the cost of failure, i.e. the time that scarce resources are tied up unproductively.

Phase 2 is entered to recover or restart failed components of the transaction. Once the failed components are restarted, activities are initiated to determine whether bound data are in a consistent state and, if not, restore them to a consistent state; the type of recovery activity initiated is dependent on the state of the node.

Phase 3 may only be entered after either:

a) recovery phase 2 has been completed successfully; or

b) a communication failure while the node was in the ACTIVE state. Presumed rollback ensures that the state of the bound data is consistent.

OSI TP provides no specific features for phase 3 recovery which is the responsibility of the application or user.

8.7.4.2 Phase 1: Failure detection and containment

This phase is entered as a result of the failures types listed in table 3 above.

For all failure types, except for atomic action data unavailability, the current state of the node determines what recovery action, if any, should be taken.

If atomic action data becomes unavailable, the state of the node is recovered from the log records; then, the state of the node determines what recovery action, if any, should be taken.

The state of a node which crashed is restored as follows:

a) READY if a log-ready record is available for this transaction; or

b) DECIDED (commit) if a log-commit record is available for this transaction. In this case, the outcome of the transaction is to commit; or

c) transaction forgotten if no log record is found. Table 4 below summarizes the restoration of the node state after atomic action data has become unavailable.

NOTE - Presence of a log-heuristic record does not affect restoration of the node state.

Table 4 - Restoration of node state after atomic action data unavailability

Type of log record	**No log record**	**Log-ready record**	**Log-commit record**
Node state	transaction forgotten	READY	DECIDED (commit)

NOTE - "transaction forgotten" is not a node state; "transaction forgotten" relates to actions taken by the CPM when information regarding the transaction no longer exists, except possibly for a log-heuristic or log-damage record.

Recovery actions:

a) ACTIVE state: the node brings its bound data to the initial state and propagates rollback to all other nodes with which it is in communication, if any.

When rollback is complete, the node forgets the transaction, and recovery phase 1 terminates. Then TP recovery terminates.

b) READY state: the node may have taken a heuristic decision before the failure occurs. In this case, there is no particular action to take: a log-heuristic record has already been written.

Alternatively, the node may take a heuristic decision, in which case, the node writes a log-heuristic record.

Recovery phase 2 is entered.

c) READ-ONLY or EARLY-EXIT state: if the dialogue with the superior still exists, the node remains in the same state. Otherwise the node informs any subordinates with which it is in communication that the transaction has terminated, forgets the transaction, and recovery phase 1 terminates; then TP recovery terminates.

d) ONE-PHASE state: if the dialogue with the neighbour to which one-phase was signalled still exists, the node remains in the ONE-PHASE state. Otherwise the node informs any neighbours with which it is in communication that the transaction outcome is unknown, forgets the transaction, and recovery phase 1 terminates; then TP recovery terminates.

e) DECIDED (commit) or DECIDED (rollback) states: for the part of the transaction tree that has not been affected by the failure, the node behaves normally, as discussed in 8.6.

For the part of the transaction tree that has been affected by the failure, recovery phase 1 terminates, and recovery phase 2 is entered.

8.7.4.3 Phase 2: Transaction recovery

This phase is entered after recovery phase 1 has terminated. Recovery phase 2 is entered when communication with an adjacent node has been disrupted, and the final outcome of the transaction needs to be communicated. There are only two situations that cause the node to re-establish communication for recovery purposes:

a) with the commit master if the node is in the READY state; or

b) with a commit slave if the node is in the DECIDED (commit) state, and communication was disrupted with the neighbour before reporting of the state of the bound data within the neighbour's commitment hinterland is complete.

Communication is re-established by means of a channel.

The current state of the node determines what recovery action, if any, should be taken.

Recovery actions:

a) ACTIVE state: the node never enters recovery phase 2 while in the ACTIVE state: according to the presumed rollback paradigm, there is no need to communicate the outcome of the transaction.