OSI Distributed Transaction Processing
Commitment Optimizations

Richard Banks[1], Peter Furniss[2], Klaus Heien[3], H. Rüdiger Wiehle[3]
[1]R.Banks@x400.icl.co.uk, ICL
[2]P.Furniss@mailbox.ulcc.ac.uk, Peter Furniss Consultants
[3] {heien|wiehle}@informatik.unibw-muenchen.de, Federal Armed Forces University Munich

Abstract: This paper briefly summarizes the work towards the final version of 'Distributed Transaction Processing' (OSI TP). Several well-known optimizations of the presumed abort protocol are introduced: dynamic flow of READY-messages, a one-phase protocol, a read-only extension. Moreover, some useful extensions such as containment of heuristic decisions and reporting of the completion status of a transaction are presented. The requirements and the functionality are discussed especially from the user's point of view. Finally, the evolving requirements for distributed transaction processing are discussed given the increasing number of applications on the Internet (Electronic Commerce).

1 Introduction

Towards the end of 1986 the standardisation project 'Distributed Transaction Processing' (abbreviated to OSI TP) was started by ISO (International Organization for Standardization). The objective of the project was the development of a communication standard - based on the OSI (Open Systems Interconnection) architecture and existing OSI standards - to support transaction-oriented cooperation between several open systems, i.e. to support the cooperation of transaction processing systems.

In 1992, the so-called **base standard** [4] was published comprising the core of a transaction processing protocol. Extensions and generalizations that had already been discussed during the development of the base standard finally led to an enhanced version which is scheduled to be published as a revised international standard in 1998 [3]. The following text discusses and introduces the corresponding extensions and generalizations especially from the user's point of view. The technical implementation of the protocol only plays a minor role in this paper.

The OSI TP standard is divided into three parts: (1) the presentation of models used in the protocol, (2) the definition of the OSI service boundary including all details about the abstract service primitives, and (3) the communication protocol itself in the form of an abstract machine which (in cooperation with several others) is able to perform the service described in the second part. The three parts only define the behaviour on the connections between transaction processing systems. In the OSI-philosophy sense - which may today be regarded as somewhat obsolete - no internal interfaces of these systems are defined by the standard although some features of these systems - such as non-volatile storage - are an explicit precondition.

Chapter 2 outlines fundamental concepts, requirements and the terminology used in OSI TP. A short introduction to the base standard is given in chapter 3. Chapter 4 discusses the modified preconditions regarding the application background, which are necessary to understand some of the major extensions and the resulting technical solutions which are part of the enhanced version. However, the enhanced version does not cover all the possible and useful mechanisms for distributed transaction processing. An example of a remaining, non-trivial application-specific deficiency is given in chapter 5. This article ends with a conclusion about the use of transaction processing in the changing environment of computer networks such as the Internet and its new services (e.g. Electronic Commerce).

2 Fundamental concepts, requirements and some elements of OSI TP

This section provides an introduction to some basic concepts for the reader who is not acquainted with transaction processing in general and introduces some of the terminology used in the OSI TP specifications.

According to textbook literature [2] a **transaction** is a unit of work that is atomic, consistent, isolated, and durable (**ACID properties**):

- **atomic**: either all actions being part of a transaction happen or none happen;

- **consistent**: means simply ”correct”; the effects of a transaction do not contradict the consistency constraints given in the specifications of the involved programs or data;
- **isolated**: designates the property that two transactions running in parallel have the illusion that there is no concurrency, i.e. a transaction is isolated from the uncommitted (= intermediate) results of other transactions;
- **durable**: means that the long-lived data objects of a transaction based system survive failures; in other words: these data objects are only changed as the result of a committed transaction.

A **distributed transaction** is a transaction in which several systems participate. A **transaction processing system** (TPS) is a computer system that runs transaction processing applications.

The principles of two-phase commit are applicable to any distributed cooperation. To ensure that a combined action involving several autonomous, geographically distributed partners is applied atomically, a number of stages are required. Each partner, when it has been informed of what its contribution to the action is, and if it will be able to perform that, supplies a "failure-resistant" promise (**READY-message**) that it is prepared to perform that contribution and that it is waiting (in READY-state) for a GO-message or NO-GO-message (or, to use the OSI TP terms, a **COMMIT-message** or a **ROLLBACK-message**). If the GO-message is received, that partner's part of the action is performed, if a NO-GO message its part is not performed and there is no effect of unsuccessful action. The GO-messages are sent out from a **commitment coordinator** when and if it knows that all participants are in the READY-state. If one or more partners determine that they are not able to perform their part in the action, they do not supply the READY-message, GO-messages are never sent and partners that have entered the READY-state eventually receive NO-GO-messages (exactly who sends these is one of the features that varies between two-phase commit protocols). The participants may relay READY-, GO-, NO-GO-messages. To ensure robustness and durability, a two-phase commit protocol must allow for the possibility that the participants and the means for conveying messages may malfunction or fail at any time during the procedure.

The **presumed abort protocol** is a variant of the two-phase commit protocol where a partner A requesting another partner B to signal its READY-state (for a transaction T) has no administrative transaction data regarding T in secure storage. In case of a crash at A all remembrance of T may be lost at the site of A. When B asks at the site of A to be informed about the outcome of T, it may receive the answer: ”T is unknown”. B will conclude that T has been aborted, hence: presumed-abort. The presumed abort protocol is, counted by secondary storage accesses, cheaper than other variants, but has certain drawbacks, e.g. it cannot precisely report on heuristic mixes (cf. section 3.3).

Fundamental requirements

The purpose of OSI TP is to support the formation of distributed transactions in a group of transaction processing systems

- even and especially if the emergence of the group was ad-hoc e.g. by looking up dictionaries (of potential trade partners);
- even and especially in the presence of certain types of failures in the communication infrastructure and in the transaction processing systems.

OSI TP primarily and directly ensures atomicity of a set of actions distributed in a group of systems if these systems follow the (partly informal) prescriptions of the standard, e.g. place ready-to-commit data in secure storage.

Atomicity allows globally consistent state transitions if the participating systems work correctly. It allows isolation of transactions, if appropriate techniques e.g. holding locks until the end of a transaction, are applied. Durability of results is anyway a purely local property of the involved systems and as such not explicitly dealt with in OSI TP, which is primarily a protocol standard.

Means of communication

OSI TP is based on connection-oriented communications where a connection (here used as a general, not as an OSI term) is understood as a communication link for order-preserving two-way message transfer between TPSs or between two nodes in two TPSs.

Remark: A node is an addressable component of a TPS that represents a distributed, transaction-oriented cooperation in this TPS; some more details are given below.

With OSI TP we use the term **association**, designating a connection between nodes; an association has a somewhat enriched functionality, e.g. special message types etc.

Remark: While the OSI TP standard was created as part of the OSI communications family of specifications, assuming an underlying OSI network, it has been successfully implemented and exploited over a TCP/IP network using the Internet Society RFC1006 specification.

Failures and recovery

The standard does not deal with all conceivable types of failures, but does expect implementations to be able to maintain the ACID properties despite **communication failure** or **node failure.**

The standard's concept of communication failure is the failure of the association between two TPS with the following properties:

- a communication failure will be signalled to the two adjacent nodes – but not necessarily at the same time;
- as long as the failure is not signalled to a node:
    - the node may send messages which are liable to be lost in an upcoming failure;
    - all incoming messages are correct; they arrive in the order as sent, none missing;
- after a failure, a new supporting association can be established between the two nodes in a reasonable time.

Remark: The standard does not define precisely what a reasonable time is.

The types of node failure that are in the scope of OSI TP can be characterized as follows:

- all associations of the node are lost; connected nodes will eventually perceive a communications failure;
- a node fails fast, i.e. before it stops operating because of a failure, it has (during its malfunction period) not done damage to the contents of the system's secure storage;
- if necessary, a successor node can be established (in a reasonable time) that is able to complete an unfinished transaction correctly.

Other failures, such as a long-lasting breakdown of major, non-redundant components of the underlying computer system or damage to the contents of secure storage are considered to be outside of the scope of OSI TP.

A **system failure** (crash) should not be more severe than a simultaneous failure of many or all of the existing nodes in the system.

Availability

In addition to the restrictions on types of failures there is an assumption on the availability of the TPSs; it roughly states that failures, no matter of which type, occur rarely and, moreover, occur never in dense or large groups so that the system can always do substantial productive work within any randomly chosen time interval of length L where L depends on the type of application the system supports.

Trees of nodes

Next, some ideas on the structure of distributed, business-based cooperations are outlined though not all of them are explicitly found in the OSI TP model text.

A potentially long-term business relationship between systems results from time to time in **activity periods** involving several systems which exchange messages as an essential part of the ongoing activity period; such periods are separated from each other by pauses (periods of quiescence) in which no messages related to the business relationship are exchanged. During an activity period part of the long-lived data representing the business relationship are moved from the databases to a network of nodes. A **node** is the representative of a certain business relationship in one of the involved transaction processing systems during an activity period.

Remark: Often implementations establish the messages accepting active components only for the shortest possible time, e.g. as threads, when a message must be dealt with. In the remaining time, the state is stored as specific system data. This technique does not invalidate the idea of a node as a continuously existing small subsystem, maintained as long as needed.

According to OSI TP such a network of nodes is built from one node, the **root node**, by soliciting at the remote system the creation of a cooperating, **subordinate node** which is specialized on the business relationship in question. A subordinate node may become a **superior node** of other nodes, and so on.

Such a network of nodes and associations has no loops; in OSI TP it is called a **dialogue tree**, which may grow and shrink during its lifetime. The understanding is that in a dialogue tree one or a series of transactions are executed. A transaction need not always spread over the whole dialogue tree, and different transactions may involve different groups of nodes. The area (a subtree) of a dialogue tree where the nodes are involved in a particular transaction is called a **transaction tree**.

Structure of a node

A node is decomposed into exactly two parts or components:

- the **TP Service User Invocation** (**TPSUI**);
- the **TP Protocol Machine** (**TPPM**).

These two components cooperate via the so-called **TP service boundary** (**TPSB**). The usage and the functioning of the TPSB shows that it is a kind of local protocol between two highly autonomous components.

Remark on terminology: Traditionally the TPSUI is said to be above the TPSB, and the TPPM to be below the TPSB.

As the name indicates, the TPPM handles the OSI TP protocol on all associations that are used for its TPSUI's activities. The TPSUI may interoperate with a neighbouring TPSUI (in the dialogue tree) only via the two corresponding TPPMs.

One advantage of the TPSUI/TPPM decomposition is that the TPPM is largely independent of the application oriented details of a concrete TPS, and thus there may be prefabricated TPPM constructions for otherwise very different TPSs. The standard describes the TPPM in a very detailed and precise manner (state tables), so that this description may be used to generate large parts of the TPPMs for specific system environments automatically.

The TPSUI does not usually correspond exactly to a single component in a concrete TPS, but models all the application-oriented specifics in such. A TPSUI must correctly use the TPSB when doing transaction processing; that is all the standard demands of TPSUIs.

A very strong reason for the considerable complexity of the TPSB is the fact that accessing long-lived data objects, e.g. in (one of) the database system(s) of a TPS, does not pass through the TPSB. The inherent autonomy of the TPPM and the TPSUI of a node is stressed and illustrated by the fact that they fail and do a first step of recovery independently of each other. Recovery work done on each side of the TPSB (or on one side only) may first lead to an inconsistent node state which must be made consistent by cooperative recovery actions.

A simple application programming interface (API), i.e. a transaction-oriented execution environment for application processes, hiding as much as possible the sophisticated techniques needed for distributed transaction processing is not defined in OSI TP. An example for such an API is given in [5].

Measures of protocol efficiency

A standard for distributed transaction processing should propose an efficient protocol to do the job – efficiency being measured in units like writing records to secondary storage, message exchanges, totally elapsed time of completing distributed transactions, or ease of use of the service boundary for the application programmers which is certainly more difficult to compare.

Specializations of OSI TP together with their requirements will be reported on in chapters 3 to 5 (e.g. heuristic decision and heuristic reporting, one-phase commit, early-exit, conditional prepare).

Definition of an often used OSI TP term:

Bound data of a transaction and of a node are the long-lived data objects used in a transaction at this node not only for 'snapshot-reading'.

Remark: In most cases bound data will be changed if the outcome of the transaction is COMMIT.

3 The base standard - The presumed abort protocol

3.1 The protocol without failures

Transaction termination is based on **the (collective) READY-state** of the nodes in a transaction tree. The READY-state is built by the uptree flow of READY-messages starting at the leaf nodes of the tree. A node in READY-state is not allowed to initiate rollback of the transaction and it guarantees that bound data related operations at the node are only performed depending on the outcome of the transaction received from the superior node. Moreover, it is guaranteed that these operations are possible even after failures. This can only be ensured if the knowledge about bound data operations and knowledge about adjacent nodes in the transaction tree are stored in stable storage. The latter[1] information is called the **log ready record** which has to be written to stable storage (forced log write). A **READY-message** is sent to the superior to indicate that the node is in the READY-state. The root node is the only node in the transaction tree to be able to determine whether the (collective) READY-state is reached and the only node to be able to determine that the transaction can be committed.

Commitment termination is a part of the protocol flow of a transaction that takes place after the set of participants has been fixed and the participants have reached agreement on the result of the transaction (i.e. on the contents, not on the outcome which remains to be seen) and has two major purposes:

(1) find out whether all participants are able to take part in the planned combined action, i.e. to contribute their part of the collective results and

(2) inform the participants that they shall do their part of the combined action (if the condition expressed in (1) is fulfilled).

Commitment termination in the base standard is initiated by a recursive downtree flow of **PREPARE-messages** starting at the root node. The PREPARE-message is used to indicate that the application oriented processing is complete. After completion of application-oriented processing, all the intermediate nodes having received a PREPARE-message will send a PREPARE-message to their subordinates. Forwarding the prepare flow may be delayed and does not affect all subordinate nodes at the same time.

A leaf node enters the READY-state after receiving a PREPARE-message from the superior and a READY-message from the local TPSUI and after a log ready record has been written. Intermediate nodes do not issue a READY-message, unless, in addition to the above conditions, a READY-message is received from all subordinate nodes.

[1]The set of related operations on the bound data is not part of the log ready record. A common recording of this information is possible but not stipulated.

The root node takes the commit decision and enters the COMMIT-state if a READY-message is received from all subordinates and a log commit record (forced log write) is written. (*Remark: The root node does not enter the READY-state.)*

The root node issues a **COMMIT-message** to its subordinate nodes immediately after having taken the commit decision. The subordinate nodes immediately pass this message on to all their subordinate nodes and (within the node) to the local TPSUI. As the number of forced log writes is to be minimised, intermediate and leaf nodes are not obliged to replace the existing log ready record by a log commit record (this precondition is quite problematical; for further information see section 3.2).

The downtree flow of COMMIT-messages is followed by an uptree flow of **COMMIT-CONFIRM-messages** starting at the leaf nodes. Prior to confirming the receipt of the COMMIT-message, the local work related to the combined action is done at the nodes, i.e. the bound data operations are completely performed and a COMMIT-CONFIRM-message is to be received from the local TPSUI. This work includes deletion of the log ready record as a forced log write. The TPSUI is informed about the local completion of the transaction and a (next) new transaction may therefore be initiated.

The **rollback termination** is used to propagate that no planned combined action shall take place. Each node which has not yet entered the READY-state and which is not in the ROLLBACK- or COMMIT-state, is allowed to initiate rollback for the current transaction and thus enters the ROLLBACK-state (the reasons for initiating a rollback are numerous, e.g. distributed deadlock, storage media failure, inability to take part in the planned combined action).

The propagation of rollback starts at an initiating node with the downtree flow of **ROLLBACK-messages** to each subordinate not having issued a ROLLBACK-message. A node having received a ROLLBACK-message from the superior issues a **ROLLBACK-CONFIRM-message** to the superior only when the rollback is completed in the subtree. If no ROLLBACK-message is received from the superior, a ROLLBACK-message will be issued to the superior, when the rollback is completed in the subtree.[2] With these procedures, each node finally enters the ROLLBACK-state or disappears (rollback is presumed) and thus the rollback decision becomes known by each node in the transaction tree.

3.2 Recovery

A recovery manager **CPM** (**channel protocol machine**, see [6]) is required in each TPS to support recovery after a failure. The CPM is either totally failure-resistent as long as there are operating nodes or, in case of a total system crash (all nodes crash), it will be back in operation before any recovered node.

In a node, only the TPPM can directly access the CPM. Nodes can access a (local) stable storage and can thus save information, the so-called log records, for the continuation of processing after failure. If a communication failure occurs, recovery actions (reestablishment of the association to the neighbour) are only initiated by a node,

- if the node is in the READY-state and the communication failure occurs on the superior dialogue;
- if the node is in the COMMIT-state and the communication failure occurs on a subordinate dialogue and no COMMIT-CONFIRM-message has been received from the subordinate.

No recovery actions are performed for the propagation of rollback (presumed abort is used!).

After detecting a node crash, the CPM reestablishes a node either in the READY-state or in the COMMIT-state depending on the existing log record. It is important to note that, after a node crash, a node in the COMMIT-state is replaced by a node in the READY-state if no log commit record was written after receiving the COMMIT-message (see above). The new node is established without any associations to neighbouring nodes.

The recovery protocol of OSI TP must be able to handle more complex situations which are not described in detail in this paper; some reasons for the vast complexity are:

- a node crash may also occur on a newly created node;

[2] The delayed uptree propagation of the rollback is justified in order to combine the rollback propagation with the messages needed for reporting (see: heuristic decisions and reports).

– communication failures and node crashes may also occur during recovery;
– the acceptable delay for the local restart and the reestablishment of associations is not stipulated in the standard but recovery actions must be completed after a tolerable delay.

3.3 Heuristic decisions and heuristic reporting

It is not unrealistic to imagine application areas in which the average elapsed time of a distributed transaction is a few seconds and where recovery from a severe system crash may take ten to twenty minutes. In a situation where a node has to stay in READY-state several ten times longer than usual and than expected, it should have a legitimate way to 'leave the transaction'. In OSI TP this autonomous and isolated escape takes the form of a so-called **heuristic decision**, i.e., the node decides either to put the bound data into their initial state or to perform the operations on the bound data which are agreed upon for the outcome COMMIT. In both cases the node releases the bound data so that they can become immediately bound data of other transactions.

Taking a heuristic decision clearly risks violating the consistency of the databases - excessive use of heuristic decisions will make the use of transaction processing, and especially the recovery mechanisms pointless. However, practical business considerations make it preferable to release database locks and risk violating the ACID properties, rather than prevent any access by other transactions. The circumstances that make a heuristic decision appropriate will depend on the business requirements of the application and thus cannot be standardised in OSI TP. The standard merely allows the escape mechanism as an exceptional case.

The node having made a heuristic decision is not allowed to deviate from the normal protocol behavior in any way. The only modification of the normal protocol flow (after the outcome of the transaction has become known at the node) is for **heuristic reporting** which means adding certain parameter values to regular protocol data units.

A heuristic decision contrary to the outcome of the transaction is called a **heuristic mix** at the node in question. A node sends a heuristic mix report to its superior node if and only if it has produced a heuristic mix itself or has received a heuristic mix report from at least one of its subordinates or if both conditions are fulfilled. Some additional logging is needed for heuristic reporting.

Remark: In the OSI TP commitment optimizations protocol, a node which is able to repair the damage resulting from heuristic mixes in its subtree (including itself) is allowed to suppress the heuristic mix report to its superior (for more details cf. 4.5.3).

As there are no recovery actions in the rollback procedures of OSI TP (presumed abort protocol is used!), heuristic mix reporting in all parts of the transaction tree is only guaranteed, if the outcome is COMMIT.

Remark: To have reliable mix reporting in all cases, a 'presumed nothing protocol' is needed [8].

Heuristic mix reporting is only a first step to limit the damage resulting from heuristic mixes. Further protocols – not included in OSI TP – are needed to reduce or even to get rid of the data inconsistencies introduced by contrary heuristic decisions.

4 Commitment Optimizations of OSI TP - an introduction

Certain objectives of the optimizations can be well understood in the current application-specific background, e.g. runtime reduction within commitment termination by generalizing the flow of READY-messages (see section 4.1) or the accelerated propagation of rollback (see section 4.5.1). Other objectives and measures can only be considered as useful if some new preconditions about the application background are taken into account.

4.1 Choosing the commitment coordinator

4.1.1 Objectives

There are very different reasons for transferring the role of the commitment coordinator from the root node of the transaction tree to another node and therefore to liberalise the flow of READY-messages: (1) reduction of runtime of a transaction [8],[9], and (2) allowance for different degrees of availability of the participating systems (see section 4.1.2 for a detailed discussion).

Firstly: The transfer of the commitment coordinator role is possible in any tree structure by giving up the strict uptree-oriented flow of READY-messages. By this, the condition for sending a READY-message is generalised to: ”A node having received READY-messages from all neighbours but one writes a log ready record, enters the READY-state and issues a READY-message to the one neighbour, from which no READY-message has been received.” This generalised rule is the basis for the well known optimization called ‘Last Subordinate Optimization’ (LSO) [8].

Secondly: The initiator of a transaction may reside in a TPS with a lower degree of availability while the subordinate resides in a highly-available system. The subordinate is not willing to enter the READY-state and to wait for the decision taken by the superior because the bound data of the subordinate are locked while waiting for the outcome of the transaction. Therefore, the subordinate expects that the superior is willing to enter the READY-state and thus the subordinate becomes the commitment coordinator.

With the concept of an alternate commitment coordinator alone, it is also possible to include a node in a transaction tree (as a subordinate) which has to become the commitment coordinator because of technical reasons, i.e. nodes which are not able to perform local processing dependent on a decision taken by another node. But only one such node is allowed in a transaction tree.

4.1.2 Admitting partners with unpredictable availability patterns (‘client-only systems’)

In this section, arguments are given for admitting small end-user systems (PC’s: computer systems with a single user interface and a person, the end-user, as the predominant i/o-partner), as partners in a distributed transaction.

In the distant past, terminals were usually ‘stateless’, i.e. they did not accommodate long-lived data. This has changed. We call a small end-user system that maintains long-lived data for its user(s) or owner(s) a client system. A WWW-browser is an increasingly common example for such a small end-user system. The server may leave some long-lived data in the client (a ‘cookie’); the server has to deal with the cookie disappearing when it next tries to access it.

If a client system is used as the basic equipment for calling a normal TPS (=server system), the user of the client system may very well be interested in making updates of her/his local long-lived data as part of a distributed transaction executed with the called server or with several servers pulled into the transaction by one directly called server.

The decision to admit a typical client system as a partner in a distributed transaction is not straightforward since these small systems normally have other failure characteristics and quite a different level of availability than the normal TPS.

In and around a client system, failures of nodes and associations tend to be more frequent than in or between normal TPSs and the repair times (e.g. reestablishment of a switched line) may be considerably longer. Moreover, the end-user – especially when being in a customer position – is in complete and autonomous control of his client system and may decide at any time and at any occasion to take the system out of operation for some hours or some days. This results in a completely different degree and pattern of availability than found with a normal TPS. One reason for this end-user autonomy is that their client systems never offer any kind of service via the network to anybody, neither to an open group of requestors nor to a predetermined authority as it may be observed in a command and control environment. Anyway, client-only (=customer-only) computer systems seem to be a sufficiently large and interesting category of transaction partners to justify specialized distributed transaction protocols.

If some long-lived data residing in a client-only system become blocked in the READY-state (= in the in-doubt-phase of a transaction), e.g., because of a communication failure, and remains blocked and inaccessible for a long time, the damage may be small or limited, if there is only one user or owner interested in the state of these data. This evaluation of locking times is, in general, completely different when long-lived data of a normal TPS are concerned, since many of these data objects are randomly and frequently accessed from many sources in the network. The pressure to inform the participants of a distributed transaction about its outcome is usually high for normal, especially very active TPSs and may be much lower for client-only systems.

From all observations made on client-only systems it should be plausible that – in a distributed transaction – a client-only system transfers the role of the commitment coordinator to the called server. Then the group of participating highly-available servers can assure the propagation of the outcome of the transaction among themselves with ‘normal-TPS-speed’, occasionally leaving the client-only system temporarily aside.

Remark: Since a client-only system cannot be pulled (unpredictable availability!) into a dialogue tree (cf. chapter 2) by another system, it must contain the root node of the dialogue tree it is part of, i.e. it must be the initiator of the dialogue tree.

4.1.3 Realization: Dynamic Two Phase Commitment

The complete generalisation of the ready flow (uptree and downtree in the whole transaction tree) is possible but not always required. The possible flow of READY-messages is determined during dialogue establishment. By this method, the set of possible commitment coordinator nodes is fixed by static constraints.

The determination of possible flow directions for READY-messages may lead to a transaction tree in which no collective READY-state can be built. (e.g. (1) a node is not able to receive a READY-message from two neighbours, (2) there are two nodes in the transaction tree which are not able to issue a READY-message). Such transaction trees are useless for distributed transaction processing. A set of rules for the useful extension of a transaction tree is given within the protocol (**tree checking**).

With the generalisation of the READY-message flow, the collision of two READY-messages on one dialogue might be possible. Both nodes are in the READY-state waiting for the outcome of the transaction. The role of the commitment coordination is assigned to exactly one of these nodes without any additional message exchange being required. This rule is based on an asymmetrical property of the association. Thus, there is exactly one commitment coordinator in a transaction tree (if any).

From the sequence of messages, two commitment coordinators would be acceptable. However, there can be practical complications with changing the log-record from a ready-record to a commit-record. If a would-be coordinator is unable to write the commit-record without deleting or damaging the existing ready-record, it cannot become a commitment coordinator, and must rely on the node at the other end of the dialogue. Such inability to convert the record is inevitable for some logging mechanisms; for others it is a possibility that has to be dealt with. Coping with this would be possible, at the expense of some additional complexity; the collision resolution rule was chosen for simplicity.

4.2 Several commitment initiators

4.2.1 Objective: Initiation of commitment procedure with application message

The receipt of a PREPARE-message from the superior node stipulated in the base standard must have occurred prior to sending other PREPARE-messages to subordinates and prior to sending a READY-message to the superior. This precondition is no longer required if the subordinate is able to determine the end of application oriented processing, by using application semantics (by messages exchanged in the application protocol). The subordinate may then enter the termination phase of the transaction and issue PREPARE-messages as well as a READY-message without requiring any explicit request from the superior.

By optionally giving up **explicit PREPARE-messages** it is possible to use **conditional PREPARE-messages** in the application protocol (with the meaning: If the order X is acceptable enter READY-state, otherwise a modified order may be issued). The recipient then enters the READY-state if the order is acceptable, otherwise an appropriate message will be sent to the initiator. If the order is accepted by the subordinate, a READY-message may follow immediately and one round-trip of messages is saved.

As far as application level PREPARE-messages are concerned, it is interesting to find out whether it is useful to send **early PREPARE-messages** during the active phase of a transaction. The total runtime of the transaction might be reduced by the early-requested READY-message. If an intermediate node Y uses early PREPARE-messages for preparing a subordinate node Z, this subordinate will then enter the READY-state. If the active processing between the superior node X and node Y is not yet completed, it might be necessary for node Y to send another application message to node Z. But node Z is in the READY-state and therefore not able to perform any action (the reversible READY-state is an unknown concept in the OSI TP protocol) and a rollback of the whole transaction follows. If this risk is correctly assessed in most of the cases, a runtime reduction can be obtained for a sequence of transactions.

4.2.2 Realization: Implied Prepare

By giving up explicit PREPARE-messages, the optimization ‘Unsolicited Ready’ [8] (named 'Unprompted Ready' in the OSI TP protocol) is achieved. From the point of view of a correct application, a READY-message will never be received unexpectedly. The READY-message was either preceded by an explicit PREPARE-message or by an application message (**implicit PREPARE-message**). The complexity of the OSI TP protocol is improved to a large extent by shifting PREPARE-messages to the application protocol.

4.3 Avoidance of protocol based recovery

4.3.1 Objective: Searching for the outcome after completion of the transaction (application based recovery)

The following simplification of the two-phase commit protocol to be discussed here is specially based on application semantics.

It is assumed that the information about the outcome of a transaction is often available in application data. By analysing those data, a node might be able to find out if the outcome was COMMIT or ROLLBACK, e.g. information about contracts, orders or settled payments are used for this purpose. Such an inquiry which might not be allowed due to security requirements is not discussed in this paper. A typical application for this kind of interworking is given in section 4.3.2.

If the application data modified in a transaction will be accessible after a disruptive failure and the outcome of the transaction can be deduced from that data, a node need not write a log record in advance. To allow such a simplified protocol to be used, the following requirements have to be met:

1. The costs involved for saving data to support investigations by a neighbour at a later time shall be kept to a minimum. No additional information is required if the result of a transaction is represented in application data as given in the following example: ”I have just (date, time) ordered X DM to be transferred from my bank account to supplier Y”.
2. The failures and thus the investigations involved occur only rarely and the costs for the investigation are low. If this is not the case, the optimized recovery procedures of the OSI TP protocol should be used.
3. An agreement for not performing any preliminary recovery actions (write a log record) at both nodes is required.

The OSI TP protocol without recovery actions is shorter than the two-phase commit protocol: there are no COMMIT-CONFIRM-messages. These messages are needed for deleting the log records that are not required in this protocol.

There are a few problems regarding an application protocol for investigating the outcome of a transaction at a later time. This investigation cannot be performed if the transaction is not yet completed at the neighbouring node. The outcome of the transaction must be known at the neighbouring node and the state of the application data has to reflect this outcome. The investigating node is blocked if exclusive locks are used and the operations on the bound data are not yet performed. It might be helpful to know the maximum runtime of a transaction at the neighbouring node. An investigating node may be confused by heuristic decisions or by the effects of a contrary heuristic decision which are not yet compensated. It may also be confused if strict isolation is not supported, i.e. locks are released before the outcome of the transaction is known. This discussion clarifies that a special representation of some application data is required and that e.g. special tags shall be used to indicate that the current state of these application data does not reflect the outcome of the transaction.

In the enhanced OSI TP protocol version a node is allowed to state at the end of active processing whether or not recovery actions are to be used. There are two reasons for this (late) decision: (1) it might not be known a priori if the requirements for giving up recovery actions are met in the current transaction, (2) the need for preliminary recovery actions may depend on the expected damage caused by a longer delay of being informed about the outcome of the transaction and/or on the costs for keeping locks on application data (there might be a longer lock-time, if investigations are necessary).

4.3.2 *Realization: One-Phase[3] Commitment*

A node may relinquish protocol-based recovery by using **the one-phase protocol** variant which may be selected at dialogue establishment time or during active processing. The protocol machine of the node giving up protocol-based recovery and its neighbour will not perform any recovery action in case of a failure and therefore no log records are required on stable storage after the **ONE-PHASE-message** is sent or received.

If no failure occurs after sending or receiving the ONE-PHASE-message, a message about the outcome (COMMIT or ROLLBACK) will be issued and no confirmation is required. The protocol machine provides no information about the outcome of the transaction if a communication failure occurs while waiting for the outcome of the transaction, i.e. the neighbour forgets the waiting node. The transaction is completed with the outcome **UNKNOWN** at the separated node. The outcome UNKNOWN is compatible with the outcome ROLLBACK and the outcome COMMIT of the distributed transaction.

A typical application for this optimization is given together with the transfer of the commitment coordinator role to the subordinate (see Fig. 1). The initiator of the transaction (the root node) which gives up protocol based recovery will not perform any recovery actions. If, in addition, the subordinate is an intermediate node in a transaction tree and this node and its subordinate are using the recovery capabilities of the protocol machines, we have a client-only system accessing a highly available distributed database system (the intermediate and leaf node). No PREPARE-messages are required: the ONE-PHASE-message and the READY-message is a substitute for the PREPARE-message (these messages are carrying prepare semantics).

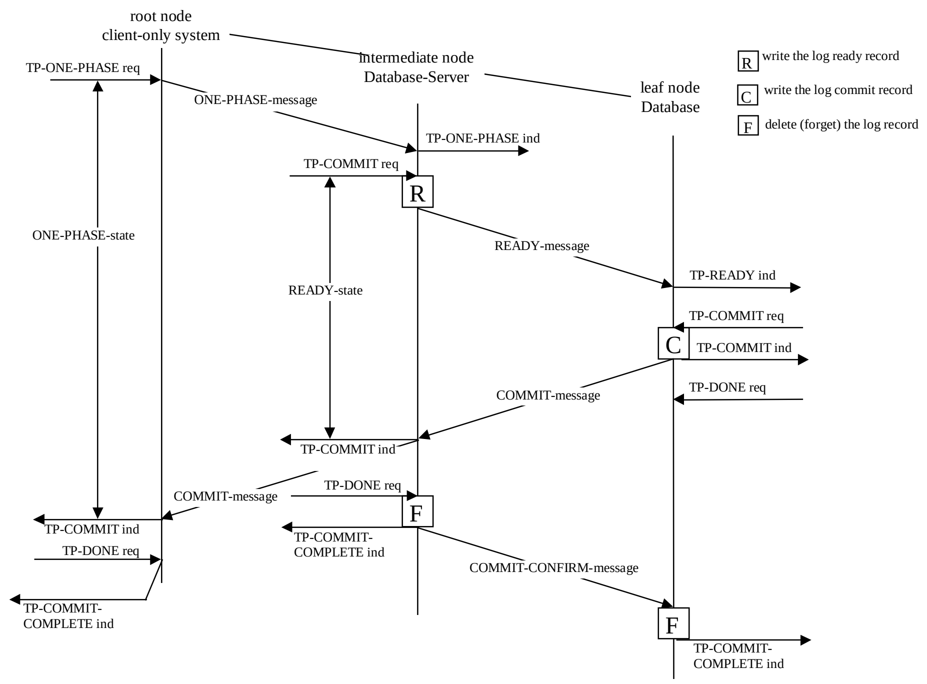

Figure 1 : One-phase and two-phase commitment

Once active processing is completed, the root node enters the **ONE-PHASE-state** and issues a ONE-PHASE-message. The protocol machine in the root node does not write a log ready record. The intermediate node writes a log ready record when entering the READY-state, but this log ready record does not contain any information about the root node because the record is only used for recovery with the leaf node. The leaf node is the commitment coordinator and a log commit record is written with the commit decision. The intermediate node and the leaf node are able to perform recovery actions until the second phase of commitment (successful flow of COMMIT-CONFIRM-message) is completed. The root node gives up recovery actions by sending the ONE-PHASE-message. The outcome of the transaction will not be known at the root node if a communication

[3]This term is illogical and was probably derived by back-construction from the Two-Phase protocol as viewed from the initiating application.

failure occurs prior receiving the COMMIT-message; the transaction is neither rolled back nor committed at the root node; further investigations are necessary for the root node to determine the outcome of the transaction.

4.4 Leaving inactive partners out of the current transaction

4.4.1 Objective: Nodes which can not make a contribution can be excluded or replaced

The negotiation between two nodes during active processing may end without any useful result if the subordinate node is not able to provide any contribution to the transaction. If such a subordinate knows or even assumes that there might be another node which is able to contribute, a rollback should not be initiated for the (whole) transaction.

An important prerequisite for continuation without rollback is that the node which can not make a useful contribution and all nodes in its subtree are able to set all bound data to their initial state and that the outcome must not be known to any of these nodes. Moreover, it is not necessary for nodes (and subtrees) without a useful contribution to take part in the two-phase commit protocol. Therefore, the request to exit from the transaction is confirmed.

There might be an agreement between the node which can not make a useful contribution and its direct superior. As far as this is concerned, two variants are defined in the protocol: Read-Only[4] (if an agreement exists) and Early-Exit (otherwise). Therefore only Read-Only calls for a PREPARE-message from the superior (if explicit PREPARE-messages are used on the dialogue).

From the point of view of the superior node the two approaches to leave the transaction differ as far as the spontaneity of the occurrence is concerned.

4.4.2 Realization: Read-Only and Early-Exit

A subordinate node without any contribution to the current transaction may be asked by the superior to leave the transaction. With this agreement, the subordinate is allowed to issue a **READ-ONLY-message** towards the superior and thus to indicate that the node no longer requires to participate in the actual protocol flow (especially the node has no preference as to whether the transaction commits or rolls back). The detailed preconditions for issuing a READ-ONLY-message are as follows:

1. the bound data of the node are not modified;
2. no additional local instance (especially no temporarily involved local resource manager) needs to know the outcome and the time of completion of the transaction;
3. the preconditions (1) and (2) are valid at every node in the node’s subtree, i.e. a read-only or an EARLY-EXIT-message is received from each subordinate node.

The other variant is a spontaneous attempt of a subordinate node to leave the transaction without any necessary agreement: this is the early-exit. Here the detailed preconditions are the same as for the read-only variant, but a PREPARE-message need not necessarily have been received. The subordinate is left out of the transaction by the superior, which initiates rollback if the early exit of the subordinate is not acceptable. The attempt to leave the transaction early is successful (i.e. the other nodes continue with normal processing) only if no rollback is initiated.

4.5 Further issues

4.5.1 Accelerated propagation of rollback - cancel

The base standard protocol flow for the propagation of rollback was not changed during the enhancements of the protocol, although the transportation of heuristic reports embedded in the uptree flow of ROLLBACK-messages may cause a delay of the rollback propagation in the transaction tree (see section 3.1).

The rollback protocol is extended by a new message - the **CANCEL-message** - which is introduced to speed-up propagation of rollback up the transaction tree. This message is issued by the subordinate towards the superior on entering the ROLLBACK-state if the ROLLBACK-message is not received from the superior. The receiving node initiates rollback in the remainder of the transaction tree as the CANCEL-message is a warning for the

[4]We agree with the authors of [2], that ’READ-ONLY’ is misleading term for this state of a node (for further discussion see [2]).

upcoming rollback. ROLLBACK-messages (and, if needed, a CANCEL-message) can then be sent earlier than in the normal rollback protocol by nodes and a speed-up is thus attained.

4.5.2 Additional information about the completion of the transaction - completion reporting

The initiation of a rollback by a node may be caused by protocol-based reasons (protocol error or a communication failure, called diagnostic) or by application based reasons (e.g. deadlock in a database, fault of the application program, time-out, called completion report). The initiator of the transaction - the root node of the transaction tree - may be able to find out (by analysing the reason for rollback) if a restart of the transaction might be successful or if this attempt will again lead to the rollback of the transaction. To make this analysis possible, the enhanced protocol introduces completion reports. The transportation of the reports is directed towards the root node of the transaction tree. An intermediate node merges all receipt reports, adds some local information, if necessary, and issues the report towards the superior. The reporting values and the rules for merging incoming reports are part of the application semantics.

Although diagnostics are not needed, completion reporting is also relevant in the commitment termination. With such reports, information about the completion status, the technical reason for a heuristic decision and others may be sent to the root node during commitment termination.

No additional messages are defined for the transportation of reports. They are embedded into the ROLLBACK-, the ROLLBACK-CONFIRM- and COMMIT-CONFIRM-messages as are heuristic reports. All reports received are given to the TPSUI in the termination phase of the related transaction.

4.5.3 Containment of the transportation of heuristic reports - heuristic report containment

A report about a contrary heuristic decision indicates a violation of the atomicity in the subtree. In the base standard, all reports must be transported towards the root node of the transaction tree. The damage caused by such a decision might be compensated within the subtree using measures that are not defined in the protocol (e.g. a compensating action at the node and its subordinates). Therefore, the enhanced version of OSI TP allows the ’compensation’ of a heuristic report. The node receiving the report decides if the damage can be compensated and thus the report does not need to be issued towards the superior. The standard does not provide any information about how to take such a decision.

Where it is known that this containment of heuristic reports will take place, the protocol can be enhanced to avoid waiting for them (static containment). A superior node may define at dialogue establishment time that no heuristic report is required from the subordinate. With this, the superior node either states that no heuristic decisions are allowed in the subtree or that these decisions are allowed with the damage to be compensated during the transaction.

5 Open issues: intra-transactional conflicts while accessing data

The OSI TP protocol only guarantees that the same outcome of a transaction is finally known at all participating nodes. Mechanisms for deadlock detection and avoidance while accessing data (Concurrency Control) are not defined within the standard. Only the tree wide usage of a transaction identifier is defined for this purpose [4, OSI TP model, annex b]. If it is taken into account that the communication between nodes is restricted to message exchanges and that bound data sharing is not intended, the usage of the transaction identifier for the purpose of concurrency control may cause major problems. To give an example: two nodes of a transaction in the same TPS have to access the same data; the TPS (or the resource manager) uses exclusive locks for isolation. Therefore, only the first accessing node is able to modify the data (i.e. these data are then called bound data). The transaction is inevitably rolled back due to the transactional deadlock (**the data clash**) and each attempt to restart the transaction - with the same arrangement of the nodes - will lead to a rollback. Moreover, it may be realized that such a data clash can generally not be foreseen by the initiator of a transaction and thus such a scenario is not unusual. More of these examples may be found for electronic payment applications.

The concept of detecting and immediately merging nodes which are instances for the same transaction in a single transaction processing is not contradictive, but this may lead to an difficult (local and distributed) application design.

If all the data is managed by a local resource manager, such a resource manager must be able to realise that the accessing nodes are parts of the same distributed transaction (this is possible by using the transaction identifier). Moreover, the resource manager must be able to provide a semantic based access control, i.e. the ordered and overlapping access to the same data item must be allowed for more than one node of the same transaction.

It is necessary to notice, that the relaxation of the isolation is not requested to increase performance. The strong locking scheme must be given up to make such a cooperation possible at all.

The deliberate relaxation of the isolation between two different transactions is not new [1] and the isolation in one transaction is not demanded at all. But that is not all of the problem when defining a common local working set for different application processes, which are ’unknown’ to each other although they are part of the same distributed transaction. Finally, an ordering instance (i.e. a global access coordinator) cannot easily be created due to the distribution of the application.

A general solution for the data clash problem requires support within the database resource managers.

6 Future work

The requirements for distributed transaction processing were recently stressed and changed due to the increasing number of applications on the Internet (Electronic Commerce). The type of cooperation between systems as well as the requirements for user interaction and security are concerned.

6.1 Authenticated documents on a commit-termination

With electronic commerce applications it may be a normal situation that a contract or a set of contracts is made if and only if a certain transaction has the outcome COMMIT. As a very simple example we assume that two TPSs, e.g. a customer TPS and a merchant TPS, want to make a contract as the result of a distributed transaction; they may proceed as follows: they negotiate the contents of the intended contract, finally they exchange signed, i.e. authenticated and non-repudiable, documents

- describing the contents of the contract to be made and
- containing an agreement that the contract will become valid, i.e. legally binding, if and only if the transaction with the universal identifier xyz (this is the transaction they are just engaged in) ends with the outcome COMMIT.

After the successful exchange of these signed documents the termination phase of the transaction xyz will be executed. If the outcome is COMMIT, both partners will create judicially usable documents proving that xyz with the partners a and b had the outcome COMMIT and that the other partner has been informed of this fact.

It seems that the judicially usable documents must be OSI TP protocol units (READY-message, COMMIT-message) signed by the originating systems, i.e. it is not feasible to produce the required documentation by combining ‘secure associations’ with normal OSI TP protocol flow.

To make several two-party contracts at a time under the all-or-nothing rule (as a customer may want to when tying several last minute offers together) the contracts must first be prepared with respect to their contents as outlined in the simple example above. In the termination phase an unsymmetric documentation may be built up in the sense that only the commitment coordinator creates a complete set of documents on the outcome COMMIT which identifies all participants and the fact that all participants have been reliably informed. As a consequence, the commitment coordinator must be trusted that it is capable and willing to collect and maintain these documents securely for some time and that, on request, it will give a copy to any entitled party.

The current OSI TP version does not support the generation of authenticated documentation on a commit termination of a transaction. Preparatory design work on this topic [7] was formally accepted by ISO but was finally refused due to lack of contributions.

6.2 Distributed Commitment in Electronic Commerce

In the electronic commerce area there are currently different approaches available starting from the traditional manufacturer of TPSs (IBM, Digital Equipment, SNI, ICL et.al.) and new companies like Netscape and OpenMarket (see [10] for an overview of known problems and solutions). First discussions about a general structure of electronic commerce applications resulted in a three-tiered architecture. The user first selects his products and the basket is filled. This is done in cooperation with one or several merchant servers. Finally, the whole basket - including all the information needed for payment and delivery - is delivered within one transaction from the user system (this is the root system) to a trusted TPS (the broker, this is the second-level system). The broker reliably sorts the contents of the basket into a set of orders and payment instructions. This decomposition is done in a distributed transaction with TPSs of the contractors and credit card institutions (these are the third-level systems). The need for a standardised TPS is given, if the user really wants to contact more than one merchant server and if either all of the ordered products or nothing should be delivered. The atomicity of delivery is especially desired and required if the cancellation of an order is not always possible without costs being involved.

References:

[1] Elmargarmid, A.K. (ed): **Database Transaction Models – For Advanced Applications**, The Morgan Kaufmann Series in Data Management Systems, San Mateo, California, 1995

[2] Gray, J., Reuter, A.,**Transaction Processing: Concepts and Techniques,** Morgan Kaufmann Publishers, San Mateo, California, 1993

[3] International Standard ISO/IEC 10026 – **1998**: Information technology – Open Systems Interconnection **- Distributed Transaction Processing** - Part 1: OSI TP Model (Edition 2), Part 2: OSI TP Service (Edition 3), Part 3: Protocol Specification (Edition 3) [currently in publication]

[4] International Standard ISO/IEC 10026 **- 1992**: Information technology – Open Systems Interconnection **- Distributed Transaction Processing** - Part 1: OSI TP Model, Part 2: OSI TP Service, Part 3: Protocol Specification

[5] X/Open Company Limited, Distributed Transaction Processing: The XATMI Specification, November 1995, X/Open CAE Specification C506

[6] Lindsay, B. G., Haas, L. M., Mohan, C., Wilms, P. F., Yost, R. A., **Computation and Communication in R*: A Distributed Database Manager,** ACM Transactions on Computer Systems, Vol. 2 No. 1, S. 24-38, 1984

[7] ISO/IEC JTC 1/SC21 N10913, National Body Contribution, Working Draft text for ISO/IEC 14447, OSI TP Security, Source: UK, July 1997

[8] Samaras, G., Britton, K., Citron, A., Mohan, C., **Two-Phase Commit Optimizations and Tradeoffs in the Commercial Environment**, 9th Intern. Conf. on Data Engineering, IEEE Comp. Soc. Press, S. 520-529, 1993

[9] Traverson, B., **Optimization Strategies and Performance Evaluation for Two-Phase Commit Protocols,** Doctoral Thésis - University of Paris 6, 1991

[10] Tygar, J.D., **Atomicity in Electronic Commerce**, Report No. CMU-CS-96-112, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996