トランザクション管理オブジェクトによる一貫性保証方式

広津 登志夫[†] 所 真理雄[†,††]

本稿では，分散環境を考慮した共有資源・情報の一貫性維持手法として，トランザクション管理オブジェクト（TMO）を導入する方法について述べる．TMOは一つのトランザクションに対して一つずつ生成され，トランザクション中に各オブジェクト上で発生したトランザクション間の依存関係を回収する．そしてこの依存関係を元に，衝突しているトランザクションのTMOと協調し全体の一貫性を維持する．処理が競合して一貫性が崩れた場合には，TMO同士の交渉によって犠牲が決定されその競合が解消される．TMOによる一貫性保証手法は，競合検出の機能と一貫性維持の機構をオブジェクトでの処理の中から取り出して，TMOの協調の形に分離して実現している．このためトランザクションやオブジェクトの適当な性質を選んでTMOの挙動に反映することが可能となり，アプリケーションや状況に応じた一貫性の基準を実現できる．したがってTMOによる手法は，柔軟性の高い一貫性保証を実現する基礎技術になり得ると考えられる．

A Distributed Consistency Management Mechanism using Transaction Management Objects

TOSHIO HIROTSU[†] and MARIO TOKORO[†,††]

We propose a scheme for guaranteeing the consistency of objects based on Transaction Management Objects (TMOs). A TMO is created for each transaction, collects the dependency relations that arise on each object during the transaction, and exchanges those dependency relations with the other TMOs to detect the inconsistency among transactions. When the inconsistency is detected, it is resolved by selecting the victims through negotiation among TMOs. We separated the mechanism for detecting and resolving inconsistency from objects' usual processing and build those mechanisms by cooperation of TMOs. Since TMO can reflect the characteristics of transactions and the objects that involved in the transactions to the consistency management, we can construct some policies for managing consistency which are specific to each application or the environment. Thus the scheme based on TMO is considered a basic technology for flexible consistency management mechanisms.

1. はじめに

計算機やネットワークの高速化や低価格化によって分散システムが現実のものとして広まりつつある．このような環境の変遷を見ると，ユーザ同士の情報や資源の共有を基礎としたユーザ間の協調に対する支援が今後重要になると考えられる．このような場面では，近年一般的になってきたオブジェクト指向による情報や資源の抽象化の技術が有効であると考えられる．オブジェクトは情報や資源の一塊とそれに対する操作（メソッド）から成るモジュールで，システムやアプリケーションを構成する単位となる．オブジェクト指向によると資源のモジュール化を進めることができ，また，メソッドという明確に定義されたインタフェースを通じて処理を要求することで，システムやアプリケーションの構造を単純なものにすることができる．

このオブジェクトは，マルチユーザ環境においては情報・資源の共有の際の共有の単位となり，複数のユーザによって利用される．各ユーザの要求する処理はこれらのオブジェクトの複数を利用する形になり，複数のオブジェクトにわたる一連の処理列として捉えることができる．ここでは，各オブジェクト上でのメソッドの実行順序から，処理列間に実行の順序関係が発生する．ところが，分散した複数のオブジェクト上ではこの順序関係の一貫性が崩れることがある．この場合，処理列間に一意の実行順序を決定できず，競合が発生していることになる．分散システム中では分散した複数のユーザが非同期的に処理を行い，また一貫性に関

† 慶應義塾大学大学院計算機科学専攻
Department of Computer Science, Keio University
†† ソニーコンピュータサイエンス研究所
Sony Computer Science Laboratory Inc.

わる情報が分散して存在するために，この一貫性の問題は顕著になって表れると考えられる．オブジェクト指向に基づくことにより，隠蔽されたオブジェクト内部からの入れ子型の呼び出しが発生するために，この競合の検出・回避はより困難な問題になる．

このような，複数ユーザ間の競合を取り扱った技術としてはトランザクション処理がある[1]．一般的にトランザクションは一連の基本的な処理のまとまりで，原子的に実行されることが保証されている．原子的な実行とは「トランザクション中の処理は他のトランザクションにより割り込まれることがなく」また「その終了時にはトランザクションのすべての処理結果が反映された状態（コミット）か全く反映されない状態（アボート）のどちらかになる」ことを保証する実行をいう．トランザクションは古くからデータベースにおいて応用されており，原子性を保証するためにいくつかの並行制御手法と回復手法が利用されている[1]．中でも二相ロックの手法は，主に単純な構造のデータと read/write のような単純な操作を対象として広く利用されている．

分散したオブジェクトに対してトランザクション処理を導入することに関しては，局所原子性[2]や協調原子性[3]といった研究においてその一端を見ることができる．局所原子性では，各オブジェクトが局所的に局所原子性と呼ばれる規則を守ることにより全体の一貫性を保証する．この規則としては，二相ロックと同等の動的原子性やタイムスタンプに基づく静的原子性が提案されている．分散システム中でのユーザの作業をトランザクションとして扱うことを考慮に入れると一般的にトランザクションは長期化する．これに対して，ロックによる動的原子性では封鎖が長期化することによって生じる効率の低下が問題となり，タイムスタンプによる静的原子性では不要なアボートの増加などの点が問題となる．協調原子性では，各オブジェクトが能動的にトランザクション間の順序関係を交換することにより全体の一貫性を維持する．この手法では，順序関係を収集してトランザクションの間の競合を検査している間に，そのトランザクションに係わる全オブジェクトの処理を停止する必要がある．また，競合が検出された場合には双方をアボートすることにより正常化する以外の手段は提供されていない点に問題がある．

ここで，簡単な旅行予約システムを例として考えてみよう．ここでは，旅行の予約には旅行代理店を通じて予約する飛行機・ホテル・観光ツアーといったものと，電車や旅館のように直接予約をとるものがあるとしよう．この場合，ある顧客の旅程に関する予約は，個々の予約を管理するオブジェクトや旅行代理店のオブジェクトにまたがる一つのトランザクションと考えられ，複数の顧客が同じ飛行機や宿を予約しようとした時に競合が起こり得る．そして，一つの旅程に関する予約がすべて獲得できて旅程を確定する場合がコミット，どれかが獲得できなくて予約を止める場合がアボートに相当する．ここで一方の顧客は希望の飛行機は予約できたが希望の宿が取れない，他の顧客はその逆という場合を考えてみると，一方のトランザクションをアボートすることで他方がコミット可能になる．ここで，どちらが放棄をするかを決定する基準には複数考えられ，状況により適当な規準を利用できることが望ましい．また双方が妥協することでともにコミットに持ち込むような解決も考えられる．このためには，トランザクションに関する情報のモジュール化と，その情報に基づく協調による競合解消機構が必要となる．

本稿では，トランザクション管理オブジェクト（TMO: Transaction Management Object）と呼ばれる管理レベルのオブジェクトを導入し，その協調によりトランザクション間の一貫性を保証するオブジェクト共有モデルを提案する．これにより，能動的なトランザクション管理が通常のオブジェクトの処理から独立して動作し，競合検出や解消の間にオブジェクトを封鎖する必要がなくなる．また，競合が起こった場合には状況に応じた柔軟な処理を選択することが可能になる．

まず次章でオブジェクトやトランザクションのモデルを述べ，3章で一貫性保証方式について説明する．4章で本稿の手法について議論し，5章で結論を述べる．

2. トランザクション処理

2.1 オブジェクト

オブジェクトは，その内部にデータやそれを操作する手続き（メソッド）が隠蔽された計算のモジュールで，メッセージを受けてそれに対応するメソッドを実行することで計算が進む．メソッド実行時のオブジェクトの動作には，内部状態を変更・参照する**内部動作**と他のオブジェクトへメッセージを送信する**外部動作**がある．

オブジェクトには，その内部に並行なメソッド実行を許すモデルと許さないモデルが考えられる．並行性を許すモデルでは，並行度の高いオブジェクトを構築することが可能であるが，内部状態の一貫性の保証はプログラマに任され，通常ロック・セマフォ等の機

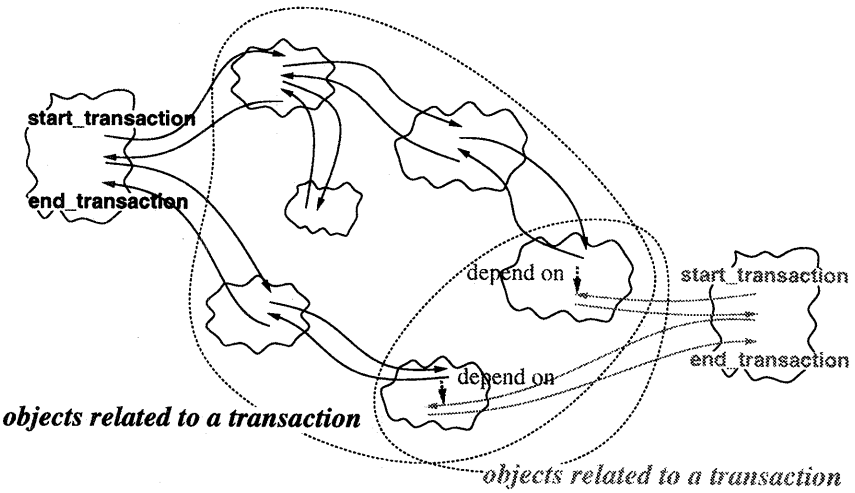


図1 トランザクションの実行と衝突
Fig. 1 Execution and confliction of transactions.

構が用いられる．並行性を許さないモデルでは，オブジェクトがメッセージを受信時に直列化し，それを順に実行するために，内部の一貫性は自動的に保証される．このため，オブジェクトの動作が単純になり，大規模で複雑なシステムの構築においては有用であると考えられる．本稿では，オブジェクト内部の並行性に関してはこの後者のモデルをとる．この内部並行性に関する制限は，線形化可能性[4]，MCO (Multiversion Concurrent Object)[5]などにより取り除くことができる．

オブジェクト上でのメソッド実行にはその結果が他のメソッド実行の有無に依存するものがある．これを**メソッド実行の衝突**と呼び，このような依存関係が生じるようなメソッドの実行の関係を衝突関係という．これは，メソッド実行間に順序関係を発生させる．メソッドの可換性[6]の考え方に基づくと，依存関係のない可換なメソッドは実行順序に関係しないので衝突しないと考えられ，衝突の頻度を減らすことが可能である．

2.2 トランザクション

オブジェクトの外部動作であるメッセージ交信により，処理は複数のオブジェクトに跨って実行される．オブジェクトにおける**トランザクション処理**とは，入れ子呼び出しを含むこの一連の処理を原子的に実行することと考えられる（**図1**）．原子的な処理はその処理を単独で実行した場合と等価な結果を生成し，複数の処理を順番に実行したのと等価な結果を得られるので一貫性の保証された処理と考えられる．

オブジェクト上で，異なるトランザクションのメソッド実行間に衝突関係がある場合には，トランザクションの間にも依存関係が発生し，処理の原子性を阻害する要因になる．これを**トランザクションの衝突**と呼び，その依存関係に応じた順序関係を引き起こす．ここで，一方向の衝突関係しか発生しなければ，その衝突関係の順序にトランザクションを実行したのと等価な直列化実行列を生成することが可能であるから問題ないが，二つ以上のトランザクションの間に双方向の衝突関係が生じた場合には，二つの処理間の順序が決定できない事になり問題となる．これを**トランザクションの競合**と呼ぶ．

そこで，一貫性管理機構は原子的な処理結果を生成するように実行を制御しなければならない．このためには，直列化可能になるように各オブジェクト上での処理をスケジュールする必要がある．

定義1 直列化可能なスケジュール：複数のトランザクション処理を，直列に実行したのと等価な結果を生成するように制御することを直列化可能なスケジュールと言う．これは，コミットする前に双方向の依存関係を検出することに相当する． □

このように，依存関係を元にトランザクションの正当性を考えると，依存しているトランザクションがアボートしてなくなる場合にも正当な実行結果を生成しなくてはならない．このためには，次の回復可能なスケジュールを実現する必要がある．

定義2 回復可能なスケジュール：トランザクションがアボートしても，既にコミットしたトランザクションがその影響でアボートされないように処理を制御することを，回復可能なスケジュールという．これは，

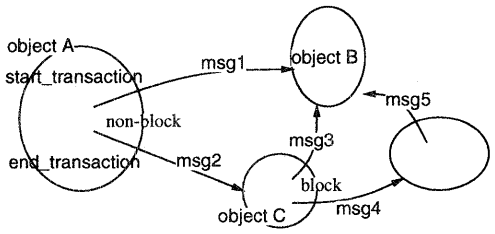


図2 トランザクション内の並行性
Fig. 2 Internal concurrency of a transaction.

依存するトランザクションがすべてコミットするまでトランザクションをコミットしないことに相当する. □

ここで，オブジェクト内の並行性と同様に，トランザクション内部の並行性に関しても考えておく．トランザクション内に並行性を許すモデルとは，複数のメッセージを同時に送信できるもので，この場合トランザクション内部でのメッセージの正しい受信順序がわからなくなるという問題が生じる（図2のobject B）．そこで，本稿におけるトランザクションのモデルでは，トランザクション内部に並行性はないとする．この制限は，内部の並行なメソッド呼び出しをサブトランザクションと考え，入れ子型トランザクション[7),8)]を導入することにより取り除くことができる．

2.3 トランザクション集合

オブジェクトが共有されるということは，複数の他のオブジェクトからメッセージを受け得るということである．ところが，実際に共有されて処理の衝突を起こすオブジェクトをあらかじめ静的に知ることは困難であり，この衝突を起こしているオブジェクトの集合は実行時に決定する以外に方法はない．トランザクション同士の衝突は，

- トランザクション処理中にメッセージを受信するオブジェクトの集合に重なりがあり，
- その重なりの部分にあるオブジェクト上での各オブジェクトのメソッド実行に衝突関係が生じる

場合にのみ起こる．このことから，次の三つの集合が一貫性の保証では重要になる．

定義3 トランザクション集合：あるトランザクションの処理を実行しているオブジェクトの集合．start_transaction から end_transaction の間にメッセージを送られたすべてのオブジェクトの集合となる．□

定義4 共有集合：複数のトランザクション集合に属するオブジェクトの集合．トランザクション $T_1, T_2, \cdots, T_n$ のトランザクション集合をそれぞれ $TS_1, TS_2, \ldots, TS_n$ とするとき，トランザクション T_i の共有集合 S_i は

$$S_i = \bigcup_{j}^{n} (TS_i \cap TS_j) \qquad (\text{for } i \neq j)$$

となる． □

定義5 衝突集合：共有集合 S_i のうちで実際にメソッドの衝突が起こっているオブジェクトの集合 □

ここで，メソッドの可換性を考慮せずすべてのメソッド実行が衝突すると考えると，共有集合は衝突集合と同じになる．以上の各集合の情報は，各オブジェクトで検出されたトランザクションの衝突情報から構築される．そして，衝突集合中の各オブジェクト上で，トランザクションの衝突の順序が同じになるようにメソッドの実行順序を制御することによりトランザクション間の衝突関係の一貫性が保証される．

2.4 トランザクション管理オブジェクト

ここで，分散システム中のトランザクション管理機構に対する要求について考えてみると次のようなものが挙げられる．

- システム全体を管理する大域的なトランザクションマネジャやデータマネジャ等を仮定できない．
- トランザクションの性質やシステムの状態に応じた，きめ細かく柔軟性の高いトランザクション管理機構の実現が望まれる．
- あるオブジェクト上で生じた衝突関係に起因して一貫性検査を行う際に，その影響を被るオブジェクトが少ないことが好ましい．

そこで，上述の一貫性管理に本質的に必要となるトランザクション集合や衝突集合の情報および，コストやタイムスタンプといったトランザクション制御に必要とされる情報を管理するオブジェクトを導入する．分散したオブジェクト間のトランザクション管理機構は，通常のオブジェクトの処理から分離され管理オブジェクトの協調の形で実現される．この管理オブジェクトを**トランザクション管理オブジェクト（TMO）**と呼び，一つのトランザクションに対して一つずつ用意される．TMOはそのトランザクションに関する情報（衝突集合，実行コスト等）を収集するとともに，他のTMOとの交渉を通じてトランザクション間の競合の検出・回避・解決を行う（図3）．一貫性維持のためには，各TMOは以下の集合を管理する．

Transaction set (TS) 管理しているトランザクションの処理を実行したオブジェクトの識別子の集合．

Conflict set (CS) トランザクション集合のうち，他のトランザクションの処理と衝突が生じているオブジェクトの識別子の集合．

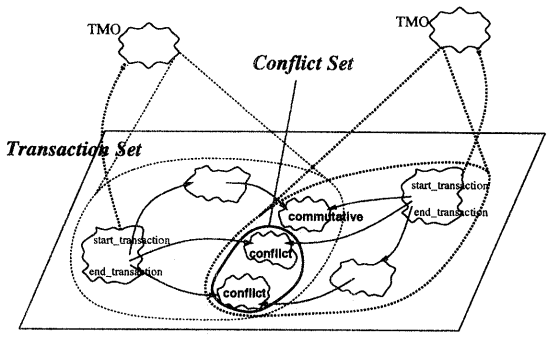


図3 トランザクション集合とTMO
Fig. 3 Transaction set and TMO.

Precede set (PS) このトランザクションが衝突（依存）するトランザクションのTMOの識別子の集合．PSに含まれるTMOの管理するトランザクションは，このTMOが管理しているトランザクションより前に順序付けられる．

Follow set (FS) このトランザクションに衝突（依存）するトランザクションのTMOの識別子の集合．FSに含まれるTMOの管理するトランザクションは，このTMOが管理しているトランザクションより後に順序付けられる．

3. TMOによる原子性保証機構

ここでは，TMOにより複数のトランザクション間の一貫性を保証する手法について述べる．この手法の基本となっているのは，逐次化グラフテスト（SGT）の手法[1]である．前述した通り，オブジェクト内の並行性とトランザクション内の並行性はないと仮定する．また，システムの障害もないことを仮定する．このため，回復に関してはアボートしたトランザクションの影響を消去するだけになる．

トランザクション間の一貫性を保証するには，衝突集合中のすべてのオブジェクトにおいて依存関係の一貫性を保証すればよい．ここで，一貫性の確認処理と各オブジェクトの処理の同期の取り方により，悲観的手法と楽観的手法が考えられる．悲観的手法では，衝突が発見された時点でそのトランザクションに関する依存関係を調べ，オブジェクトの処理を続行する前に異常を検出する．これに対して，楽観的手法では衝突が発見された時点では衝突情報をTMOに通知するのみで，一貫性の保証はオブジェクトでの処理に並行してコミットまでの間にTMOにより行われる．楽観的手法は，コミットの前に必ず一貫性の確認が行われる点を除くと手順としてはほぼ同じなので，本稿では悲観的手法についてのみ述べる．

3.1 競合の検出

トランザクションの起動時にTMOが一つ生成される．このTMOの識別子は，オブジェクト間のメッセージに埋め込まれ伝播される．また，トランザクション処理中のメッセージには，そのトランザクション処理を実行したオブジェクトの履歴（アクセスヒストリ）が付加され，TMOはこれを受けとることにより，トランザクション集合を構成する．

3.1.1 オブジェクトの処理

(1) トランザクション（ここではTとする）中のメッセージを受けたオブジェクトは，それに対応するメソッドを実行する．この際に外部にメッセージを送る場合には，自分の識別子をアクセスヒストリに付加してメッセージに添付する．実行の結果，まだコミットしていない他のトランザクションの実行と衝突しなければ，アクセスヒストリに自分の識別子を付加し処理結果を返戻する．

(2) トランザクション$T_1,\ldots,T_n$と処理が衝突した場合は，以下のように処理を進める．

(a) Tを管理するTMOにconflictメッセージを，$T_1,\ldots,T_n$を管理するTMOにfollowメッセージを送る．conflictには，$T_1,\ldots,T_n$を管理するTMOの識別子の集合とアクセスヒストリが含まれている．また，followではTを管理するTMOの識別子が伝えられる．

(b) 送信したすべてのconflictとfollowに対する返答を待つ．すべてのTMOからcontinueが返ってきたらトランザクション処理を再開する．followを送ったTMOからcontinueより先にcommitが届いた場合には，continueを受けとったものとして処理する．

(c) 一つでもabortが返ってきたら，アボート処理に移行する．

(3) トランザクションの開始オブジェクトまで処理が戻ったら，アクセスヒストリをTを管理するTMOに伝播する．衝突を検出したり，開始オブジェクトまで処理が戻った際にはアクセスヒストリがTMOに伝播されるので，一旦その中身を空にすることができる．以上の処理をend_transactionに達するまで続ける．

3.1.2 TMOの処理

(1) トランザクションの起動者から送られてくるアクセスヒストリは，その要素をTSに加える．

(2) conflictを受けたTを管理するTMOは，アクセスヒストリの要素をTSに，衝突を通知してきたオブジェクトをCSに加える．また，衝突している

トランザクション $T_1,\ldots,T_n$ を管理する TMO の識別子を PS に加える．この結果，

(a) PS ∩ FS = ϕ（空集合）ならば競合が発生していないので，conflict を送信してきたオブジェクトに，continue を通知する．

(b) PS ∩ FS ≠ ϕ ならば競合が発生しているので，競合しているトランザクションの TMO との間での交渉の調停者となる．

(3) follow を受けた $T_1,\ldots,T_n$ を管理する TMO は，衝突を通知してきたオブジェクトを CS に加え，衝突を起こしたトランザクション Tを管理する TMO を FS に加える．この結果，

(a) PS∩FS = ϕ ならば競合が発生していないので，follow を送信してきたオブジェクトに，continue を通知する．

(b) PS ∩ FS ≠ ϕ ならば競合が発生しているので，競合しているトランザクション Tの TMO に negotiate を送信し，競合の調停を任せる．この調停者からの指示があるまで，オブジェクトへの返答は行わない．

(4) 交渉の調停者は自己の情報や他の TMO から集めた情報で，トランザクションの競合を解消するのに必要な犠牲（victim）を選出する．犠牲になるトランザクションの TMO に abort を，negotiate を送信してきた TMO のうちアボートされないものに continue を送信する．

この犠牲の選出には，次の幾つかの指針が考えられる．

- 既存の順序に従う方法（trivial rule）
  調停者の TMO が，競合している他のトランザクションにアボートを指示する．これは，先に発生した依存関係に逆らわないことを意味する．
- 起動の順序に従う方法（timestamp rule）
  調停者の TMO が，競合している他のトランザクションの TMO から起動時のタイムスタンプを収集する☆．ここで，調停者の起動が一番早かった場合は他のトランザクションを，そうでなければ調停者のトランザクションをアボートする．
- コストによる方法
  調停者の TMO が，競合している他のトランザクションの TMO からアボートのコストを収集する．アボートのコストは，トランザクションの処理を実行したオブジェクトの数，そのトランザクションに依存しているトランザクションの数，起動からの時間，重要度等の情報から生成する．ここで，調停者のコストが一番大きかった場合は他のトランザクションを，そうでなければ調停者のトランザクションをアボートする．

先に挙げた旅行予約システムの場合は，顧客の予約の競合が検出された後の処理が，この TMO 間の交渉の形にモデル化される．ここで，顧客情報や顧客間の交渉などを元に競合の解消を図ることになる．

3.2 コミット処理

オブジェクト上の処理が end_transaction に到達したら，そのトランザクションの TMO に start_commit メッセージを送信してコミット処理を開始する．

(1) 最初に，コミットできるかどうかの確認を行う．コミットするトランザクションの TMO は PS が空かどうかでコミット可能かどうかを決定する．

(a) PS = ϕ ならば，TS の各オブジェクトに commit を送り，各オブジェクトでそのトランザクションの処理結果をコミットする☆☆．すべてのオブジェクト上でコミット処理が完了すれば，FS 中の各 TMO に対して committed を送りトランザクションの終了を通知する．

(b) PS ≠ ϕ ならば，依存するトランザクションの完了を待ち，すべての依存するトランザクションがコミットした段階つまり PS が空になった時点でコミット処理に移る．

(2) PS ≠ ϕ の状態では，PS に登録されている TMO から committed が来ると，その TMO の識別子を PS から外す．その結果 PS = ϕ になったら，1 (a) の処理を行う．

また，PS ≠ ϕ の場合には二者以上での依存関係の輪が発生している可能性があるので，それを検出するために自分の識別子を付加した check_loop メッセージを，PS の各 TMO に対して送信する．check_loop メッセージを受けた TMO はそのメッセージが保持しているリストに自分の識別子が含まれていないかどうか確認する．含まれていれば，そのリストのメンバと交渉して，犠牲を定める．自分が含まれていなければ，リストに自分の識別子を加えて check_loop メッセージを PS のすべての TMO に送信する．

3.3 アボート処理

競合の解消のための犠牲になった場合に，トランザクションをアボートする際の処理を示す．ここでアボートはトランザクション処理の途中で生じる可能性があ

☆ これ以前の処理でタイムスタンプを piggy-back しておくと，この収集のフェイズは省略できる．

☆☆ システム障害がないと仮定しているので，1 フェーズのコミットで良い．

るために，その段階でTMOの保持しているTSの情報が，実際のトランザクションの処理を実行したオブジェクトの集合に満たないことがある点に注意が必要である．

(1) abortを受ける，もしくは自らの決定によりアボートすることに決まったトランザクションのTMOは，TSの各オブジェクトに対してabortを送りアボートを指示する．

(2) TMOや他のオブジェクトからabortメッセージを受けた各オブジェクトは以下のように処理する．
- まだアボート処理を行っていなければ，アボートするトランザクションの影響を消去する．アボートするトランザクションの処理中に他のオブジェクトへメッセージを送信していれば，その送信先オブジェクトに対してもabortを送り（アボート伝播），すべてのアボート伝播に対する完了通知を待つ．
- アボート処理中にabortメッセージが届いたら，アボート完了後abortedを返す．既にアボートした後ならすぐにabortedを返す．

(3) アボート伝播の必要がない場合や，すべてのアボート伝播から完了通知が戻ってきた場合は，アボート指示の送信者にabortedを送ることによりアボートの完了を通知する．

(4) アボートするトランザクションのTMOが，すべてのabortメッセージに対する返戻を獲得したらアボート処理は完了する．

4. 議　　論

最初に，本稿の手法が直列化可能かつ回復可能なスケジュールであるかどうかについて検討する．トランザクション間の衝突関係は，衝突するメソッドの実行により生じる．コミットはトランザクションの最後にあるので，あるトランザクションでの衝突はコミットより前に発生する．本稿の手法では，衝突の後，次の処理の前にTMOにその衝突関係が通知される．したがって，コミットより前にそのトランザクションに関する衝突関係はTMOに通知されている．衝突により他のトランザクションに依存しているトランザクションは，コミット処理に入っても，依存しているトランザクションがすべてコミットするまでコミットされない．つまり，二つ以上のトランザクションが互いに依存し合い，依存関係の輪を形成しているような場合には，トランザクションがコミットされることはない．また，依存関係の輪の中で最後のトランザクションがコミット処理に入った際に，check_loopメッセージにより依存関係の輪が検出される．したがって，TMOによる一貫性管理手法は直列化可能なスケジュールといえる．また，コミット時の処理を考えると，あるトランザクションがコミットする時には，それが依存するコミットされていないトランザクションは存在しない．したがって，コミットされたトランザクションは他のコミットされていないトランザクションの処理結果に依存しないので，回復可能なスケジュールともいえる．

次に本稿の手法について，他の原子性の手法と比較する．動的原子性ではロックを用いていたために封鎖の長期化の問題があった．TMOでは衝突関係の回収により一貫性の監視を行い，封鎖しない手法を提供している．また，静的原子性では大域時計の情報を必要とした．これに対して，TMOは衝突するトランザクションがアクセスしているオブジェクトの集合のうち実際に衝突を起こしている最小限のオブジェクトの集合に対してトランザクションの一貫性に関わる情報を回収する．これにより，大域情報は不要となる．

協調原子性では，衝突が発生した時に現在処理を実行しているトランザクションに係わる順序関係のみを調べる．また，このためにトランザクションのアクセスしたすべてのオブジェクトの順序関係を調べる．しかし，TMOでは既に実行の済んだ衝突関係にあるトランザクションを管理するTMOに能動的に衝突情報が伝播され，先のトランザクションと現在処理しているトランザクションの双方の順序関係を調べることができる．これにより，トランザクションに参加するすべてのオブジェクトから情報を回収する必要がなく，TMOは衝突を起こしている必要最小限のオブジェクトだけから衝突情報を回収すれば良いので，衝突情報を集めるコストは協調原子性より小さくて済む．

また，他の原子性保証の手法では，衝突が生じて一貫性がとれなくなった際に，アボートされるトランザクションは静的に決定されている．しかし，この犠牲となるトランザクションがアプリケーションや状況に応じた適切なものであるとは限らない．TMOの場合は，アボートの際に各TMOごとに情報を収集し独自の基準（例えば，長く動いているトランザクションが生き延びる等）で判断して適切な犠牲を選ぶことができる．

最後に，衝突が生じた際に必要な封鎖時間について検討する．ここでは，オブジェクトの協調によりロックを用いずに原子性保証を提供する点で似た手法である協調原子性と比較する．衝突を発見したオブジェクトとトランザクション集合中のすべてのオブジェク

表1 衝突発生時の封鎖時間の比較
Table 1 Comparison of the blocking time.

	トランザクション
協調原子性	$4 \times max(p_0, \cdots, p_n)$
TMO	$2 \times max(t_0, \cdots, t_n)$
	トランザクション集合中のオブジェクト
協調原子性	$2 \times max(p_0, \cdots, p_n)$
TMO	0

トの間の通信時間を $\{p_0, \cdots, p_n\}$，衝突を発見したオブジェクトと衝突を起こしているトランザクションのTMOとの通信時間を $\{t_0, \cdots, t_m\}$ として，トランザクション処理の封鎖時間とトランザクション集合中の各オブジェクトの封鎖時間をまとめたものが**表1**である．ここで，TMOがトランザクションの起動者の近くに生成され，衝突している二つのトランザクションのトランザクション集合が同じという仮定を置くと，$max(t_0, \cdots, t_n)$ は最悪でも $max(p_0, \cdots, p_n)$ の二倍程度に押えることができる☆．したがって，競合の検出のための中断は同程度，競合検出中の他のオブジェクトの動作は封鎖されないと考えられ，全体として応答時間は向上する．

TMOの手法では，一貫性の崩れを検出した際に，ネゴシエーションのためのメッセージが交換される．最も単純なネゴシエーションルールでは，トランザクションの順序のみから，アボートの影響が少なくなるように犠牲が選ばれる．この場合は，ネゴシエーションのためにそれ以上情報が交換されることはなく，大きな問題とはならない．

5. 結　論

本稿では，大規模分散環境を考慮した共有資源の一貫性維持手法について，トランザクション管理オブジェクト（TMO）を導入する方法について述べた．一つのTMOは一つのトランザクションを管理し，オブジェクトが検出したトランザクション間の衝突情報を収集する．この情報を元にTMO間で協調し，逐次化グラフテストの手法に基づいて一貫性を保証している．競合の解消自体はTMO同士の交渉によって行われる．また，ロックや大域的な実体といった分散システムで問題になる要素を仮定していないので，大規模分散システムにも適すると考えられる．さらに，TMOによる一貫性保証手法は，トランザクションやオブジェクトの適当な性質を選んでTMOの挙動に反映することが可能で，柔軟性の高い一貫性保証のための基礎技術になり得る．本稿では起動時間やコストを例に上げたが，他にも重要度や競合解消時のユーザの指示等を採り入れることも可能であろう．ロックなどに基づく悲観的手法との融合や状況による動的な管理手法の変更等は今後の応用としては興味深い．

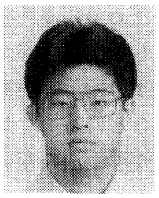
謝辞 本稿の査読者の方々からは，多数のコメントをいただいた．ここに感謝の意を表する．

☆ あるオブジェクトを介して通信した場合に相当．

参考文献

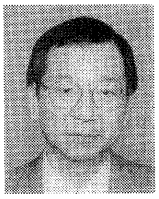
1) Bernstein, P., Hadzilacos, V. and Goodman, N.: *Concurrency Control and Recovery in Database Systems*, Addison Wesley (1987).
2) Weihl, W.E.: Local Atomicity Properties: Modular Concurrency Control for Abstract Data Types, *ACM Trans. Prog. Lang. Syst.*, Vol. 11, No. 2, pp. 249–283 (1989).
3) 中島達夫，所真理雄：原子オブジェクトにおける協調原子性，コンピュータソフトウエア，Vol. 8, No. 3, pp. 9–23 (1991).
4) Herlihy, M.P. and Wing, J.M.: Linearizability: A Correctness Condition for Concurrent Objects, *ACM Trans. Prog. Lang. Syst.*, Vol. 12, No. 3, pp. 463–492 (1990).
5) Hirotsu, T., Fujii, H. and Tokoro, M.: A Multiversion Concurrent Object Model for Distributed and Multiuser Environments, *Proceedings of the 15th International Conference on Distributed Computing Systems*, IEEE, pp. 271–278 (1995).
6) Weihl, W.E.: *Specification and implementation of atomic data types*, PhD Thesis, Massachusetts Institute of Technology (1984).
7) Moss, J.E.B.: *Nested Transaction: An approach to reliable distributed computing*, MIT Press (1985).
8) 広津登志夫，藤井寛子，所真理雄：オブジェクト指向共有環境での入れ子型トランザクションの支援，オブジェクト指向コンピューティングIII—日本ソフトウェア科学会WOOC'93—(大堀 淳(編))，近代科学社（印刷中）．

(平成7年3月17日受付)
(平成7年9月 6 日採録)



広津登志夫

1967年生．1990年慶應義塾大学理工学部電気工学科卒業．1992年同大学大学院理工学研究科計算機科学専攻修士課程，1995年博士課程修了．工学博士．現在，(株) 日本電信電話 基礎研究所勤務．分散システム，トランザクション処理，オペレーティングシステムなどに興味を持つ．ソフトウェア科学会，ACM各会員．



所　真理雄（正会員）

1947年生．1970年慶應義塾大学工学部電気工学科卒業．1972年同大学大学院修士課程（管理工学専攻），1975年博士課程（電気工学専攻）修了．工学博士．同大学電気工学科助手，専任講師，助教授を経て現在教授．その間カナダウォータールー大学（1979年1月～1980年6月），米国カーネギーメロン大学（1980年7月～12月）訪問助教授，1988年4月よりソニーコンピュータサイエンス研究所副所長を兼務．計算モデル，プログラミング言語，分散・開放型システム，人工知能などに興味を持っている．著書に「計算システム入門」(岩波講座ソフトウェア科学1)，「システム構成技術」(岩波講座マイクロエレクトロニクス10，共著)，編著に「Object Oriented Concurrent Programming」(MIT Press)，「Concepts and Characteristics of Knowledgebased Systems」(North Holland)，「Object-Based Concurrent Computing」(LNCS, Springer) などがある．ソフトウェア科学会，電子情報通信学会，人工知能学会，ACM，IEEE，AAAI各会員．