知能と複雑系 109－1
（1997．8．30）

タスクグラフを用いたトランザクションの効率的な処理方法

鈴木幸一郎 渡辺豊英

名古屋大学大学院工学研究科
〒464-01 愛知県名古屋市千種区不老町
Phone: 052-789-2764
E-mail: {ksuzuki,watanabe}@watanabe.nuie.nagoya-u.ac.jp

概要

今日，計算機システムの利用用途は単一的な処理から複合的な処理へと拡大し，それぞれの処理要求間の連携的，統一的な実行制御機構が重要になっている．データベースでは，このような長期的に渡って計算機システムに存在し，相互に関連する複数の処理要求を効率的，効果的に実現する課題として長時間トランザクション処理がある．本稿では，長時間トランザクションを効率的に処理するための枠組みとして三階層トランザクション・モデルを提案し，関連するが個々に独立なトランザクション (処理要求) を迅速，かつ的確に実行制御する方法について検討する．

キーワード

長時間トランザクション，三階層トランザクション・モデル，エージェント，タスク・グラフ，ワーク・フロー

Effective Transaction Management Using Task Graph

Koichiro SUZUKI and Toyohide WATANABE

Graduate School of Engineering, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-01, JAPAN
Phone: 052-789-2764
E-mail: {ksuzuki, watanabe}@watanabe.nuie.nagoya-u.ac.jp

Abstract

Recently, the utilizaions of computer systems have been expanded day by day and also the processing requests in a wide range have been shifted from the management of single tasks to that of complex tasks. In such a situation, it is important to investigate cooperatively integrated execution mechanism among interrelated tasks. From a viewpoint of database management, the investigation of long-lived transaction management is one solution for such a subject. In this paper, we address an experimental approach to manage long-lived transactions effectively in the three-layers transaction model and organize the task procesing with respect to the agent-oriented paradigm.

Keywords

long-lived transaction, three-layers transaction model, agent, task graph, work flow

1 はじめに

今日，計算機システムの使用は単一的な処理から複合的な処理へと拡大し，それぞれの処理要求間の連携的，統一的な実行制御機構が重要になっている．データベースでは，このように長期的に渡って計算機システムに存在し，相互に関連する複数の処理要求を効率的，効果的に実現する課題として長時間トランザクション (long-lived transaction) 処理がある．

長時間トランザクションの処理のためのモデルとして，入れ子トランザクション・モデル [1, 2] が提案されている．これは長時間トランザクションをコミット可能なトランザクションの集合とみなして二分割し，個々の分割されたトランザクションを制御することにより長時間トランザクションの処理を実現する方法を採っている．入れ子トランザクション・モデルは二階層で構成され，下位レベルのトランザクションの並行処理，個々のトランザクションの独立性という点で特徴があり，ロールバック制御することが簡単となっている．しかし，入れ子トランザクション・モデルでは，下位のトランザクションの処理が上位のトランザクションの処理状況に影響を及ぼすため，一般に効率的な実行を実現することが困難である．

本稿では，長時間トランザクションの効率的な処理に関して，三階層において実行制御を実現する枠組みを提案する．この三階層で構成されるトランザクション・モデルを三階層トランザクション・モデルと呼ぶ．三階層トランザクション・モデルでは，入れ子トランザクション・モデルで問題視される「下位層の処理遅延などによる上位層への影響」に対して，各層の従属関係的な制御機構だけではなく，対等関係の制御機構を導入することにより，解決するという方法を採る．すなわち，それぞれの層に自律動作可能なエージェント [3, 4, 5, 6] を配置し，エージェントが対応する層の役割に応じた処理を実行するとともに，相互の通信を可能とすることにより実行制御権移行を柔軟に扱うという立場で，実現を図る．

2 三階層トランザクション・モデル

本三階層トランザクション・モデルでは，長時間トランザクションそのものを管理・処理する長時間トランザクション層，長時間トランザクションを構成し，独立に実行可能なトランザクションを管理，処理するトランザクション層，そしてトランザクションを構成し，個々に実行可能な最小の処理単位であるタスクを実行するアクション層で実現される．従って，二階層の入れ子トランザクション・モデルと同様に，このような構成の三階層トランザクション・モデルでも上位の処理が下位の処理に依存し，効率的な実行を実現することが困難である．これを解決するために，本三階層トランザクション・モデルではそれぞれの層に自律動作可能なエージェントを割り当て，各エージェントが層の役割に応じた機能を果たすと共に，層内の実行制御に束縛されることなく，層間に渡って相互に情報を交換できる枠組みを構成する．すなわち，三階層トランザクション・モデルではそれぞれの層における処理要求，処理単位をエージェントによって管理，処理する．長時間トランザクション層には関連した複数のトランザクションの系列を manager と呼ぶエージェントで管理し，トランザクション層での実行を制御する．トランザクション層には個々の独立なトランザクションを，そのトランザクションに割り当てられた executor と呼ぶエージェントが管理，制御し，アクション層での実行を制御する．そして，アクション層には最小の処理単位を実際に実行する actor と呼ぶエージェントが割り当てられ，並行的に個々の処理単位を実行する．図 1 に三階層トランザクション・モデルの構成を示す．

3 ワーク・フローとタスク・グラフ

関連する複数の処理要求，処理単位を効率良く制御するためにワーク・フロー，タスク・グラフを導入する．ワーク・フローは長時間トランザクション層の manager が個々のトランザクションをワークとして管理し，その実行の手順を制御するために用いられる．例えば，旅行代理店における旅行業務は

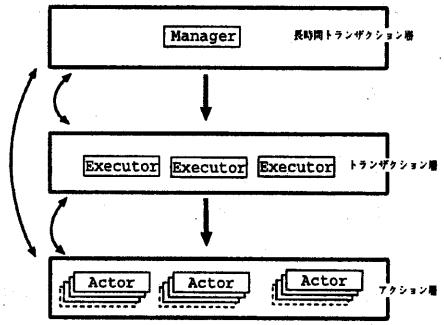

図 1: 三階層トランザクション・モデル

図 2 のように表すことができる.

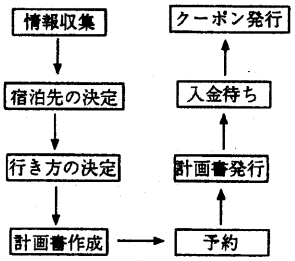

図 2: ワーク・フローの例

図 2 では, 四角形が一つ一つのワークを表し, 矢印がワーク間の順序を表している. 図 2 は最も簡単な手順を表し, 一つ一つの処理手続きが逐次的に連鎖している. もちろん, 旅行業務はこのように単純ではない. 一方, タスク・グラフは一つのワークを最小の処理単位としてどのように実行可能かを表し, トランザクション層の executor が actor の実行を制御するために用いられる. 例えば, 予約手続きの一例を図 3 に示す. 図 3 で, 円は最小の処理単位であるタスクを表し, 矢印は制御関係を表している.

図中の四角形で囲まれた, "航空予約", "列車予約" 等の各タスクは, それぞれが独立なタスクであるため並行に実行可能なことを表している.

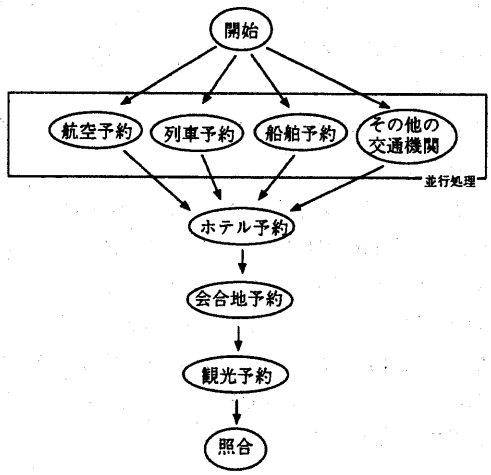

図 3: タスク・フローの例 : 予約手続き

4 manager と executor

manager はワーク・フローの下に関連する複合的なトランザクション (ワーク) を管理, 制御し, 個々のワークをその適切な executor に割り付ける. 一方, executor はそれが実行可能な状態となったとき, ワークに基づいてタスク・グラフを展開し, タスク・グラフのタスクに対応する actor を生成し, 実行する. manager は必要に応じて executor を生成, 消滅させ, executor は actor を生成, 消滅させる. もちろん, これらの操作はワーク・フロー, タスク・グラフで完全に制御されるのではなく, ワーク・フロー, タスク・グラフは executor の生成時点, actor の生成時点を基本的に指示する. 例えば, 総ての予約が完全に O.K. とならずに, 次の手続きに移行する場合があり, このような場合には actor であれば executor が, また executor であれば manager が適当な時に強制的にそれらを実行終了させ, それらを消滅させる.

このような manager, executor, actor 間の制御時点を判断できるために, データ・シートと呼ぶ, この一連の処理要求で達成すべき処理内容を書き込むデータ・オブジェクトを導入する. データ・シートは一連の業務処理について共通に利用可能なデータを保持し, 各 executor がその手続きで生成すべ

きデータ部を書き込み, 他の executor がそれを利用し, 処理内容を決定できるように, manager によって管理されている. すなわち, ワーク・フローが一連の各業務処理の内容に対する指示書に対応すると考えられる.

従って, 正規の手順に従った manager の executor に対する制御はワーク・フローに依存するが, 再実行, 不定期な実行制御に対しては, データ・シートの情報に従って manager が executor の実行を一時的に強制可能とする. 例えば, 旅行代理店業務であれば, データ・シートは各種の申込み書 (航空, 宿泊, 鉄道等) に相応する.

5 executor と actor

executor は, トランザクションをタスク・グラフとして表し, トランザクションを実行制御する. あるタスクの処理を開始するとき, executor はそのタスクに対応する actor をパラメータなどを与えて呼び出し (生成し), 実行する. actor はタスクの実際の実行主体である. actor はアクションと呼ぶ複数の関連し合う処理要求の原子的な処理単位を実行する.

C.Hewitt が提案したアクタ・モデル [7] とは異なり, actor は非同期な通信手段を有する. すなわち, actor は共有メモリに書き込むことにより, executor との非同期な通信を可能にする. 例えば, actor はその処理を開始するとき共有メモリに対し "起動" を報告するメッセージを, どのタスクを処理しているのかということとともに書き込む. この共有メモリはタスク・グラフとともにトランザクションの実行制御に利用される.

6 タスク間の関係とタスク・グ ラフ

タスク・グラフは executor が処理するトランザクションを表現する有向グラフで, そのノードとエッジはそれぞれタスクとタスクの関係を表している. executor は, このタスク・グラフを参照することにより, トランザクションの処理を進めていく.

処理関係を制御するために, タスク・グラフでは個々の並行可能なタスク間の関係を的確に表すことが必要である. 現在, 想定しているタスク間の実行制御に関する関係は次のようである.

- 順序関係
  A は B の実行後に実行
- 制御依存関係
  A は B によって制御
- データ依存関係
  A の実行は B によって得られるデータに依存
- 排他関係
  A, B のどちらかが成功すればよい
- 代替関係
  A が失敗したとき, B を実行
- 強制実行関係
  A は B を同時に実行させる
- 強制終了関係
  A の処理終了は B の実行を停止

順序関係では, タスク B の実行はタスク A の実行後に行なわれることを示している. 例えば, ユーザの希望と予約内容が合っているかどうかを判断する "照合" タスクは観光地予約を含むすべてのタスクが終了している必要がある.

制御依存関係は, タスク B が行なわれるかどうかがタスク A に依存していることを表している. 例えば, 予約手続きが終了したとき, 予約手続きで行なった予約がユーザの条件に合致するかどうかを確認するタスクは, 次にすべきタスクが予約の変更のためのデータをユーザから収集するタスクか, それとも予約に関する情報をデータ・シートに書き込むタスクかを決定する.

データ依存関係では, タスク A はタスク B の成果物を使用するということを表している. 例えば, 予約変更タスクはユーザからの変更データに依存している.

排他関係は, タスク A またはタスク B のどちら一方のみがこのトランザクションで必要であるということを表している. 例として, 名古屋から大阪までの鉄道での移動の予約を考える. タスク A として, "新幹線: 名古屋→新大阪" タスク B として, "近鉄特急: 近鉄名古屋→難波" を考えたとき, これらのどちらか

が成立すればよい場合を排他関係と呼ぶ.

代替関係は, タスクAの実行が失敗したとき, タスクBを実行するという関係を表している. 排他関係での例で考えると, タスクAが失敗したとき, タスクBを実行するという関係を表している.

強制実行関係では, 代替関係にあるタスク同時に実行させることを表す. そして, 強制終了関係にあるタスクが終了したとき, 強制実行させていたタスクを終了させる. 先程の新幹線と近鉄の例を使用し, タスクAとタスクBは強制実行, 強制終了関係にあるとする. タスクAの実行と同時にタスクBを実行し, もしタスクAが終了したならタスクBの実行を終了させる. また, タスクBの実行が終了し, タスクAの実行が失敗, もしくは一定時間経過しても終了しない場合は, タスクBの結果が使われる.

これらは並行的に実行可能なタスクをタスク相互間の性質に基づいて, 効率良い処理を遂行するためにタスク・グラフの制御エッジとして導入される. 例えば, 旅行代理店の予約業務をタスク・グラフを用いて表現すると, 図4のようになる.

このタスク・グラフでは, タスク "予約表の整理" においてデータ・シートから予約のためのデータを得て, そのデータで予約を行なってよいかどうかをタスク "確認" で確認する. 予約表は数個の独立した旅行プランから構成されているかもしれない. タスク "予約開始" では旅行プランのデータをパラメータ化して, actorを生成する.

タスク "ホテル予約" の実行前に, executorは同期をとり, タスクの処理が正しく行なわれたかどうかをチェックする. 正しく行なわれていない場合は失敗とみなし, 次の旅行プランの予約を開始する. そうでないなら, タスク "userの条件との一致" において予約したプランがユーザの希望を満たしているかどうかを調べる. もし満たしてないなら, 変更点をユーザから得て, 予約しなおし, 満たしているのであればタスク "予約情報作成" で, データ・シートに対してexecutorが書き込む情報を作成して処理を終了する.

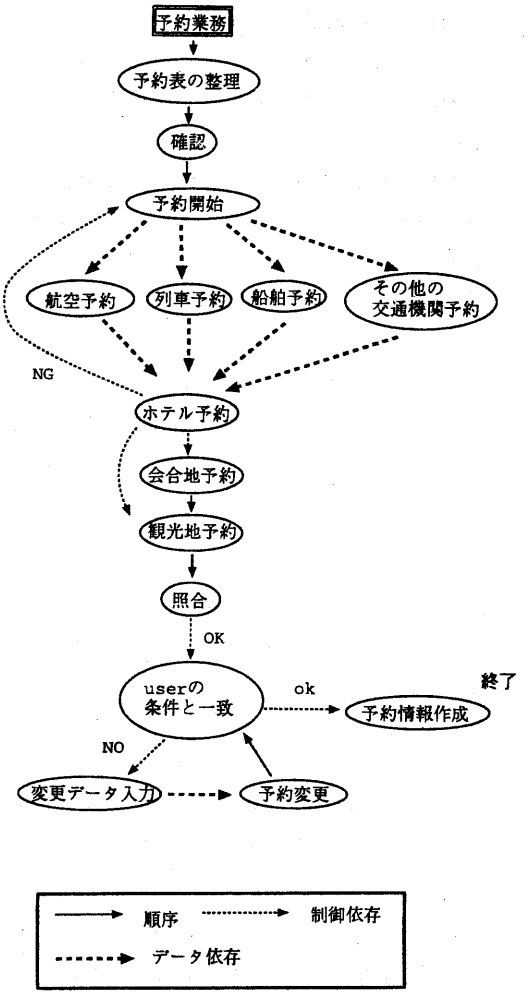

図 4: 旅行代理店の予約業務でのタスク・グラフの例

7 おわりに

本稿では，互いに関係し合う複合的な処理要求を扱うために三階層トランザクション・モデルを提案し，処理要求間の関係を表すワーク・フロー，タスク・グラフを導入することにより効率良い実行制御を行なうことを試みた．

今後は，失敗回復等も視野に入れ，より柔軟な制御が可能となるように検討しなければならない．

謝辞

日頃，熱心に討論していただいている佐川雄二講師，朝倉宏一助手をはじめ，渡邊研究室の皆様に感謝致します．

参考文献

[1] A. K. Elmagarmid ed.: DATABASE TRANSACTION MODEL for Advanced Applications, *Morgan Kaufmann Publishers.*

[2] J. E. B. Moss: Nested Transactions: An Approach to Reliable Distributed Computing, *MIT Press*, 1985.

[3] 木下哲男: “エージェント指向コンピューティング”, SRC, 1995.

[4] 石田亨, 他: “特集エージェントの基礎と応用”, 人工知能学会誌, Vol.10, No.5, 1995.

[5] T.Nishida and H.Takeda: Towards the Knowledgeable Facility, *Proc. of KB & KS '93, pp.157-166*, 1993.

[6] 木下哲男: “エージェントテクノロジの応用とその課題”, 信学技報, Vol.95, No.364, pp.41-47, 1995.

[7] C.Hewitt: Viewing Control Structures as Patterns of Passing Messages, *Artificial Intelligence*, Vol.8, pp.323-364, 1977.