データベースシステム 119−60
（1999. 7. 23）

Javaのスレッドを用いた分散入れ子トランザクションの実現

井川 智崇† 宮崎 純‡ 横田 治夫†

組織の中で複数の部門が協調して作業を行う場合に、ネットワークで接続されたコンピュータによって部門間の処理の流れをサポートするものとしてワークフローがあげられる。我々は既存のデータベースシステムとネットワークプログラミング環境によって独立アクティブデータベース間の入れ子トランザクション構造を実現する方法を検討してきた。今回、補償処理を扱う枠組みも含めて入れ子トランザクションの構成方法をさらに具体化したのでその内容に関して報告を行う。本稿ではまず、ワークフローの例として病院のモデルを示し、そのモデルを基に独立したアクティブデータベース間での協調を実現する実験システムの構成を示す。次に、その中での入れ子トランザクションの構造とその処理動作について述べる。

Implementation of Distributed Nested Transactions Using Java Threads

Tomotaka Ikawa,† Jun Miyazaki ‡ and Haruo Yokota†

Computers connected by a network can be used for supporting processes among sections of an organization. Such a system is called a workflow system. We proposed a mechanism for the workflow system using a nested transaction structure among distributed independent active databases which are implemented ordinary database management systems on PCs and network programming environment of Java. In this paper, we report a construction of the distributed nested transaction containing a compensation mechanism. We use a workflow example for some hospital to explain the behavior of our experimental system.

1 はじめに

組織の中で複数の部門が協調して作業を行う場合に、ネットワークで接続されたコンピュータによって部門間の処理の流れをサポートするものとしてワークフローがあげられる。ワークフローを実現する手段として、メールを用いたエージェントシステムや集中サーバーを用いたシステムなど多数の提案がされている[1]が、ここではデータベース処理を対象に、分散した独立アクティブデータベースシステム間のメッセージ通信によってワークフローを実現する方法を考える。

アクティブデータベースとは従来の受動的なデータベースに対し、イベントをトリガーとしてデータベースが能動的に動作する事を可能にしたデータベースモデルである[2]。アクティブデータベースでは、あるデータベース操作をトリガーに与えられたルールに従って別のデータベース操作を起動し、その操作がまた別の操作を起動するといったように再帰的に処理が進む。この様な処理の流れはトランザクションの入れ子を構成する事から、入れ子トランザクションとして管理する方法が提案されている[2]。

ここで取り扱うワークフローモデルでは、ネットワークによって繋げられた複数の部門によって処理が進む事が前提となっている。このため、ネットワーク上の独立した複数のシステムにまたがって入れ子トランザクションが存在するような機構を用意する必要がある。この場合の入れ子トランザクションのサブトランザクションはネットワーク上の別のアクティブデータベースのトランザクションとして生成される事になる。なお、ここでは簡単化のため、サブトランザクション間のインターラクションは考えない。

我々は、既存のデータベースシステムとネットワークプログラミング環境によってこの様な独立アクティブデータベース間の入れ子トランザクション構造を実現する方法、 DIADEM (Distributed Indepen-

†東京工業大学 情報理工学研究科 計算工学専攻
Department of Electrical and Electronic Engineering, Tokyo Institute of Technology
‡北陸先端科学技術大学院大学 情報科学研究科
School of Information Science, Japan Advanced Institute of Science and Technology

dent Active Database with subtransaction Exporting Mechanism)、を検討している[3]。入れ子構造を管理するモジュールを分散した独立アクティブデータベース上に用意する事で、上述した機構の実現を試みている。現在我々は、ネットワークに接続されたPC上のOracle8 SQL ServerとJavaを用いて実験を行っている。CORBAの利用を前提とすることもできるが、我々は単純なメッセージ通信のみを用いている。

今回、これまでの提案を基に、補償処理を扱う枠組みも含めて、入れ子トランザクションの構成方法をさらに具体化したのでその内容に関して報告を行う。補償トランザクションを含めたワークフローの扱いも既に提案されている[4]が、我々のアプローチは入れ子トランザクションをネットワーク上に展開する点を特徴とする。本稿ではまず、ワークフローの例として病院のモデルを示し、そのモデルを基に独立したアクティブデータベース間での協調を実現する実験システムの構成を示す。次に、その中での入れ子トランザクションの構造とその処理動作について述べる。

2 対象とするワークフロー

ここでは病院をモデルとするワークフローの例を示す。前提とする病院には、受付、薬局、医局(内科・外科など)、検査部、会計といった部署があり、それぞれが独立のデータベースサーバーを動かしているとする。

患者の流れに沿ったワークフローを見てみよう。まず患者は受付を訪れ、診察を受ける部署に対して予約を行う。複数の部署に行く場合には、可能ならば、待ち時間の少ない方から先に診察を行うようにスケジュールを組む。患者は次に予約を行った医局にて診察を行う。この時患者へ投薬を行う必要があるならば薬局と連携して投薬を行い、血液検査などが必要場合は検査部に予約を行って必要な検査を行う。この際の診察や投薬にかかる費用は会計のデータベースに登録される。全ての診察を終えた患者は会計で支払いを済ませ、薬局にて処方箋を受け取る。

アクティブルールの例を示すために、薬品の投与を考えてみよう。今、患者が内科にいて、内科に来る前に外科で薬品Aが投薬されており、ここで診察の結果、薬品Bを投薬することになったとする。内科は薬局の投薬データベースにこの患者へ薬品Bを投薬するという情報を挿入する。薬局のデータベースでは投薬データベースへの挿入をトリガとして、投薬される薬品が今までに投薬した薬品との相性問題を起こさないかどうかを調べる処理を開始する。薬局には副作用を生じる薬の組み合わせを示すデータベースがあり、このデータベースを用いて、以前投薬された薬品(薬品A)と今投薬しようとしている薬品(薬品B)との間の副作用を調べる。その結果、問題が無い事が分かった場合は投薬された薬品Bの費用を会計のデータベースに挿入する。副作用がある場合は薬品Bの投薬を行う操作をアボートし、内科に副作用のメッセージを示す。

この場合の医局におけるルールは以下の様に記述される。ここで、処理が実行されるサイトの情報は、**on** に続く値(例えば、 else 節の account(会計)) で示される。

```
when insert(medicine(:Name, :Charge,
                     :Patient, :Department))
if (immediate)
   select name into :Conflict
   from medicine
   where patient = :Patient
   and name in (
      select name1
      from combination
      where name2 = :Name);
then (immediate)
   inform('副作用から投薬できません: %s %s',
          :Name, :Conflict)
   on :Department;
   rollback insert(medicine(:Name, :Charge,
                            :Patient, :Department))
else (immediate)
   insert into payment
   on account
   values (:Patient, :Charge, :Department);
```

3 実験システムの概要

上記の様な状況を取り扱うワークフロー処理を行う実験システムを実際に構築し、有用性の検証を行

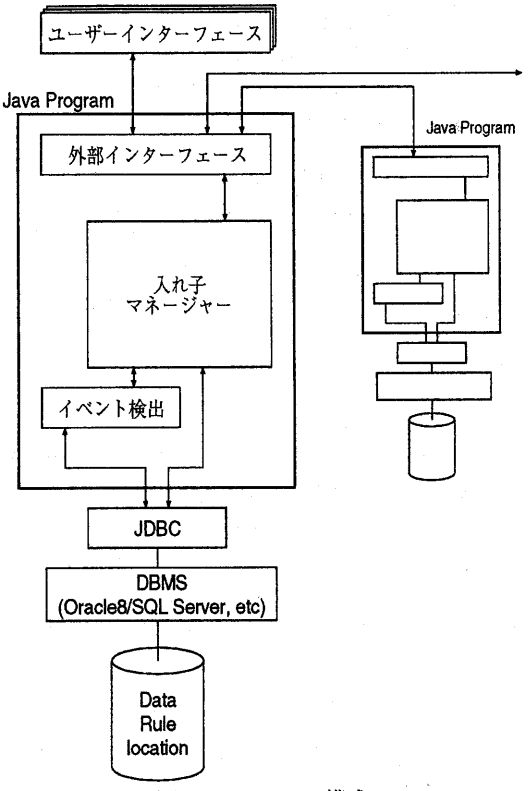

図 1: システムの構成

う。実験システムは安価で入手しやすいコンポーネント、すなわち PC/AT 、市販のリレーショナルデータベース を用い、 Java を用いてアクティブデータベースの機能とそれぞれのサイト間の通信機能を実現する。アクティブデータベースを実現するために、トリガーを検出する機構と入れ子トランザクションの管理を行う機構が必要となる。また、ソケットを用いて、互いに通信を行い、外部データベース上にサブトランザクションの生成を行ったりコミットもしくはアボートを行うための機構も必要となる。

以上をふまえ、システムは次のようなモジュールにより構成されている。ここに述べるシステムが、前述した、受付、薬局、医局、検査部、会計などにそれぞれ1つづつあり、お互いに通信をしあいながらワークフローを処理していく。ただし、データベース管理システムは同一である必要はない。

・ユーザーインターフェース部：

ユーザーからのデータベース操作を受け付けたり検索結果を表示する等、ユーザーに対するインターフェースを提供するクライアントとして働く。

・外部インターフェース部：

ユーザーインターフェース部や他のアクティブデータベースとの間のデータの受け渡し等、外部とのインターフェースを担っている。外部のアクティブデータベースは1つとは限らず、複数のデータベースとの交信も可能である。

・イベント検出部：

データベースへの操作を管理し、アクティブルールのイベント検出部分をキャッシュし、イベントが検出された場合にはデータベースからルールの実体を読み取り、イベントによって発火するルールを特定する。発火したルールは入れ子マネージャーのトランザクションを表す木構造内でサブトランザクションとして処理が行われる。

イベント検出にデータベース管理システムのトリガー検出機能を使う事も考えられるが、、複数種類のリレーショナルデータベースシステムを利用することを前提とした場合、全てのデータベース管理システムのトリガー検出機能を期待することが出来ないため、本システムでは入力コマンドのレベルでトリガを検出することとしている。このため、一部の例外を除きデータベースに対する操作は全てここに操作を通知しなければならない[1]。本システムではここに通知されないデータベースへの操作が行われた場合、イベントを検出することができない。

・入れ子マネージャー：

入れ子トランザクションの木構造の管理を行う。新たにユーザーインターフェース部もしくは外部のデータベースからアクセス要求が来た場合は、入れ子トランザクションの根を生成する。同時実行制御はこの根単位にデータベース側で管理する。

外部のデータベースに輸出する必要のあるデータベース操作は入れ子マネージャーの中から外部インターフェース部を通じて外部のデータベースに送られる。外部データベースの名前とアドレスの対応はデータベースに格納されており、それを読み出して利用する。入れ子マネージャーの構造と動作の詳細は次節 にて述べる。

[1]例外として、ルールの読み出しと外部データベースの所在地の読み出しはトリガーの検出を行う必要がないので直接データベースにアクセスする。

・**DBMS**：

SQL インターフェースを前提とする他は特にDBMS の種類は想定しないが、実験では ORACLE 社 の Personal Oracle8 を使用している。データベースには、各部局のデータやルール、外部データベースの情報を格納する。データベースへのアクセスは、Java の標準 API である JDBC を介して行う。

3.1 入れ子トランザクションの構成

入れ子マネージャーの中では、アクティブルールによって生成されるデータベース処理を入れ子トランザクションとして管理する。このため、そのデータベース処理に対応させた Java のオブジェクトを階層的に構成して、オブジェクト間の入れ子構造により入れ子トランザクションを実現する。

データベース処理は、それ自身の SQL 実行のためのオブジェクトとロールバックのための補償処理用のオブジェクト、およびそのデータベース処理によって発火させられるルールを管理するためのオブジェクトの3つ組みとして実現される。このため、その3つ組みをまとめるオブジェクトも用意する。

以下では、SQL を実行するオブジェクトを Action オブジェクト(a)、補償トランザクション用のオブジェクトを Compensation オブジェクト(c)、ルール管理のためのオブジェクトを Rule オブジェクト(r)、3つ組みを管理するオブジェクトを Triplet オブジェクト(t) と呼ぶ。さらにサブトランザクションを他のノード上で展開するためのオブジェクト Export オブジェクト(e) と、全体の根となる Root オブジェクト(R) を用意する。以下に各オブジェクトについて述べる。

・**Action オブジェクト(a)**：

JDBC を介してデータベース処理の発行を行う。図2では a と書かれた円で表されている。

・**Compensation オブジェクト(c)**：

Action オブジェクトで行ったデータベース処理の補償処理を行う。図2では c と書かれた破線の円で表されている。

・**Rule オブジェクト(r)**：

ルールを管理するためのオブジェクトで、ルールに従って Triplet オブジェクト もしくは Export オブジェクト を持つ。図2では r と書かれた四角形で表されている。

・**Triplet オブジェクト(t)**：

Action オブジェクト、 Compensation オブジェクト、 Rule オブジェクト の3つ組みを持つ。図2では t と書かれた楕円で表されている。

・**Export オブジェクト(e)**：

ネットワークで繋がれた別のアクティブデータベース上で実行される場合に用いられる。 SQL を実行するデータベースへのソケットを保持しており、トランザクションはこのソケットを介して実行が行われる。図2ではソケットを表す四角形の付いた、 e と書かれた楕円で表されている。

・**Root オブジェクト(R)**：

1ユーザーに対し、データベースごとに1つづつ存在している。ユーザーインターフェースとのやりとりはこれをここを介して行われ、JDBC の Connection を保持している。基本となる振る舞いは Rule オブジェクト と同じなので、これは Rule オブジェクト を継承して作られている。図2では R と書かれた2重四角で表されている。

3.2 マルチスレッドによるオブジェクトの実行

上述したオブジェクトのうちのいくつか (Root, Rule, Action, Export オブジェクト) は Java の Runnable インターフェースを実装しており、各々が別々のスレッドとして同時に実行する事が可能である。こうする事により、トランザクション内のサブトランザクションの同時実行のスケジューリングを、オブジェクトのスレッドの実行のスケジューリングとして取り扱うことができる。特に、外部のデータベースとの非同期の動作を実現するうえで、スレッドは有用である。

図2のようなトランザクションの木構造を考える。最初にデータベース処理要求を受け付けると、 Root オブジェクトおよびその下の t0, a0, c0, r0 のオブジェクトが作成される。 c0 は a0 の逆のデータベース処理の SQL 命令を持つ。 a0 でデータベース処理が実行されるのと同時に、 r0 はイベント検出部にイベント検出を依頼する。 r0 は、発火するルールの SQL 命令がイベント検出部から戻されると、例えば図のように、 t1, a1, c1, r1 と e2 および t3, a3, c3, r3 と e4 のオブジェクトを作成し、 a0 の終了を待つ。 a0

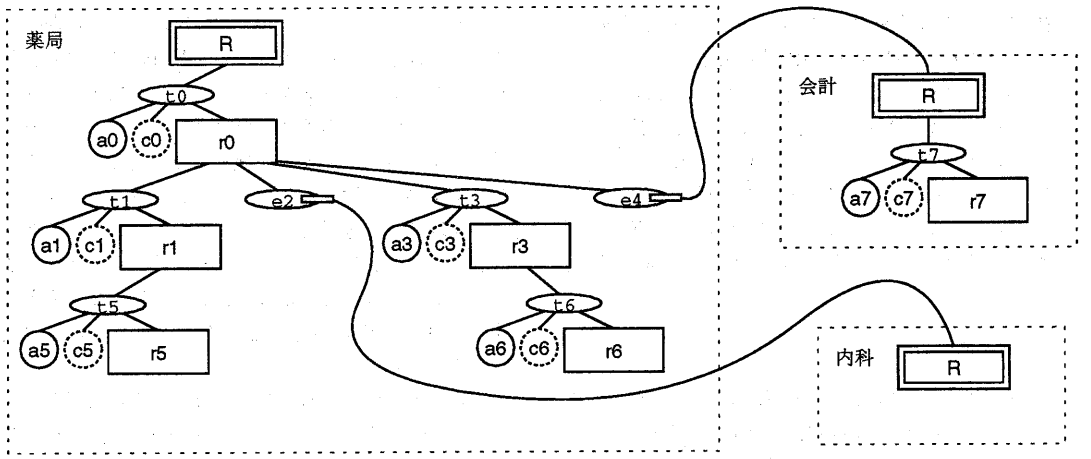

図 2: トランザクションの木構造

が終了すると、 immediate モードの場合には、すぐに a1, r1 の実行を開始するよう t1 にメッセージを送る。 a1, r1 の実行は、 a0, r0 と同様に再帰的に行われる。r1 以下の処理が終了すると、その結果に従って e2 を介して外部のアクティブデータベースに処理を依頼すると同時に a3, r3 を開始するように t3 にメッセージを送るか、 e4 を介して t7, a7, c7, r7 のオブジェクトを別のサイト上に生成する。もし deferred モードの場合には、 a0 が終了しても、全体のトランザクションが終了する間際まで、 t1 に開始メッセージは送られない。

具体的に2章で例示したルールにあてはめてみると、 insert コマンドが薬局に届くと、 ルート R と t0, a0, c0, r0 が生成される。 c0 は insert の補償処理として delete コマンドが用意される。実際の insert コマンドは a0 で実行される。r0 は相性データベースの検索トランザクション t1, a1, c1, r1 と医局へのメッセージを送る e2 、ロールバックのための t3, a3, c3, r3 および会計に費用を挿入する e4 を生成する。c1 は検索に対する補償の必要が無いので空であり、r1 もこの場合は発火するルールが無いので空である[2]。2章のルールは immediate モードなので、 a0 の insert コマンドの終了後すぐ t1 に開始メッセージが送られ、 a1 が実行される。a1 の検索の結果が空でない場合には e2 と t3, a3, c3, r3 の処理が行われ、空の場合には e4 を介して会計のサイトで t7, a7, c7, r7 のオブジェクトを生成する。

[2]2章の例では t5 以下および t6 以下は存在しない

コミットは Rule オブジェクトのレベルで階層的に2相コミットに従って行われる。r0 では t1, e2, t3 もしくは t1, e4 から ready-to-commit を受け取ると、コミット命令をそれぞれに送り返し、全てのコミット終了を待って自分をコミットする。ただし、データベースのコネクションはルートのレベルで行っているので、ルートまでコミットが行く前にはデータベース側のコミットも補償オブジェクトの消去も出来ない。

3.3 補償トランザクション

補償処理は実際に行われたデータベース処理の実行に対して逆順で行われなければならない。これはサイトをまたがっていても同様である。全てのサブトランザクションが immediate モードで実行されているならば補償処理はオブジェクトの木構造から実行順序を特定することができる。Export オブジェクトにより別サイトへのリンクも含み、関連するサイト全体でネットワークを介した木構造を構成するため、サイトにまたがる逆順除の実行を実現できる。

しかし1つでも defered モードの実行が行われる場合は状況が異なる。deferred モードのサブトランザクションの実行順序は一意ではないので木構造から SQL の実行順序を特定することができない。これは、そのまま補償オブジェクトの実行順序が特定できない事を意味している。一般にデータベース管理システム側でコネクション単位のロールバック機能を備えているため、1つでも deferred モードで実行が行わ

れた場合にはコネクションでロールバックを行う。

補償処理自体は論理的なコマンド対で実現する。delete の補償は delete 条件に沿って削除対象のタプルを一時的なリレーションとして Compensation オブジェクトで管理させ、補償時にはそれを再挿入することで実現している。また、insert の補償処理も、やはり Compensation オブジェクトで挿入対象のタプルを一時的なリレーションとして管理し、補償時にはそれらを元のリレーションから削除させる様にする。

4 考察

このシステムが対象とするワークフローモデルでは、トランザクションの処理に人間が関係することがある。よってトランザクションは長時間に渡って処理されることがあり、この場合データベースの利便性を損なう可能性について考慮しなければならない。

今回データベースへのロックはタプル単位で行うことにした。ここで前提としているワークフローのモデルに限れば、タプル単位のロックを用いれば他の利用者に与える影響はなく、長時間トランザクションが存在していても利便性は低下しないと考えている。しかし場合によっては Saga の様に途中でロックを開放する様な機構が必要となるかもしれない。

また、長時間にわたって処理されたトランザクションをすべてロールバックし、もう一度そのほとんどが前回行ったのと同じ処理を行う必要は無い場合が多い。この様なトランザクションの部分的なロールバックについては、全てのルール実行が immediate モードならば、深さ優先でサブトランザクションの実行が行われるので、部分的なロールバックは問題なく行える。しかし1つでも deferred モードのルールが実行されると、ロールバックを行いたい部分のみの補償が困難となる (補償は SQL の実行の逆順で行う必要があるため)。この場合の部分的なロールバック処理は複雑なものとなるので、今回は、1つでも deferred モードのルールが実行された場合にはトランザクション全体をロールバックすることにした。これに関してはまだまだ議論する余地がある。

5 おわりに

本稿では、分散した独立アクティブデータベースシステム間のメッセージ通信によるワークフロー処理の実現手法を示した。一般的な PC/AT と市販のリレーショナルデータベースシステムを用いて実験システムを構成し、その上に Java を用いて機能の実現を試みた。アクティブルールのイベント検出、入れ子トランザクションの管理、複数サイト間のメッセージ通信による外部でのルール実行などを Java オブジェクトを用いて実現した。特に、入れ子トランザクションの管理では、分散した Java オブジェクトの階層構造を入れ子トランザクションの木構造にマッピングさせる機構を実現した。さらに、その分散した木構造では、補償処理をその対象となるデータベース処理の逆順で行う事が出来る。

現在、実験システムの基本部分の実装は終了し、木構造に沿ったデータベース処理の実行とその補償処理は確認している。しかし、ルール検出は不完全であり、また補償処理も単純な insert と delete 命令のみしか対応していない。今後、実験システムを充実させ、病院をモデルとするワークフロー処理の様々なケースに対して動作を検証する予定である。

謝辞

本研究の一部は、文部省科学研究費補助金特定領域研究「高度データベース」の助成により行われた。

参考文献

[1] 速水 治夫 , 坂口 俊昭 , 渋谷 亮一. "ここまで来たワークフロー管理システム: (3) ワークフロー製品の実際", 情報処理 39巻 12号, pages 1258-1263, Dec 1998.

[2] J. Widom and S. Ceri. "Active Database Systems: Triggers and Rules For Advanced Database Processing", Morgan Kaufmann Publishers,Inc., San Francisco, CA, 1996.

[3] 井川 智崇 , 宮崎 純 , 横田 治夫. "独立アクティブデータベース間の入れ子トランザクションの実現", 第10回 データ工学ワークショップ, May 1999.

[4] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, C.Mohan. "Advanced Transaction Models in Workflow Contexts", 12th ICDE, 1996.