1H−2

階層トランザクションを利用したアプリケーションの開発

(株)東芝　関口幸一　関川容子　真鍋俊彦

1　はじめに

近年、オンライントランザクション処理のニーズが高まり、その処理が高度化されてきている。また、トランザクションが実行される環境も、従来は集中型で行なわれていたものが、分散型へと変化してきている。

これにより、以前と比べ、

- トランザクションが複雑化
- 複数の計算機システムにトランザクションがひろがる

というような特徴が見られ、トランザクションの記述は困難になってきた。

そこで我々は、階層トランザクションという考え方を導入し、従来記述の困難だった、分散環境にトランザクションが広がるアプリケーションを試作した。

ここでは、階層トランザクションを有効活用したアプリケーションの例について説明する。

2　階層トランザクション

階層トランザクションは、トランザクションを階層化して管理する方式で、トランザクションが図1のように、親子関係をもったトランザクションとして表現される。

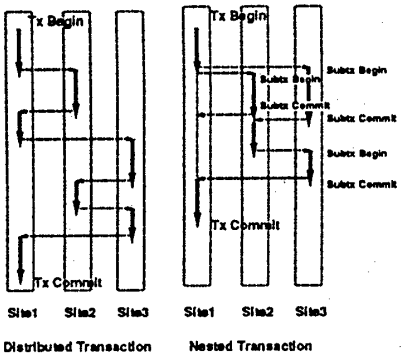

図 1: 階層トランザクション

この階層トランザクションを利用すると、

- トランザクションの一部がアボートされたときには、そのサブトランザクションだけをロールバックすれば良い。
- トランザクションの中の処理は、サブトランザクション間で並列実行できる。

という特徴がある。

3　アプリケーションの例

3.1　スケジュール予約システム

今回、Unix WS をネットワークでつないだ環境下で、スケジュール予約システムを構築した。

スケジュール予約システムは、それぞれの部所のスケジュールを管理しているシステムである。

それぞれの部所 (Dept.A,Dept.B,Dept.C) に一つづつ、合計3つのシステムからなり、それぞれの部長 (Director A、Director B、Director C) のスケジュールをローカルに管理している。

予約プログラム及びデータベースは個々に存在しており、全体の構成は、図2のようになっている。

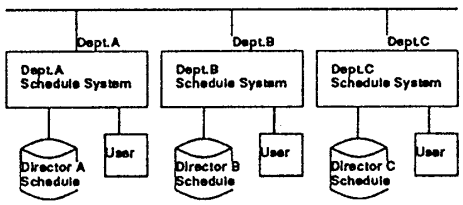

図 2: スケジュールシステム

従来、Dept.A,Dept.B,Dept.C は独立したシステムで、システム間にまたがって処理を行ないたい場合、例えば Director A と Director B のスケジュール予約をしたい時には、電話を使って会話しながら2つのシステムを動かしていた。

今回試作したシステムは、独立に稼働するスケジュールシステムを統合したもので、一つのスケジュール予約システムから、他のスケジュール予約システムを呼び出せるような機能がサポートされている。

3.2　階層トランザクションの利用

ここで、3つのシステムに広がるスケジュール予約トランザクションを考えてみる。例として、Director A が、『Director B、Director C の両方か、または、いずれか一人と会議を開きたい。』という例についてトランザクションを記述してみる。

ここで、階層トランザクションを利用すると、図3のように記述できる。

この場合、Dept.A で発生した3人の部長のスケジュールを取るというトランザクションがトップトランザクションとなり、ここから、一人一人の部長のスケジュールを取る Director A 予約、Director B 予約、Director

---

Development of Nested Transaction Application.
K.Sekiguchi, Y.Sekigawa and T.Manabe.
Toshiba Corporation.

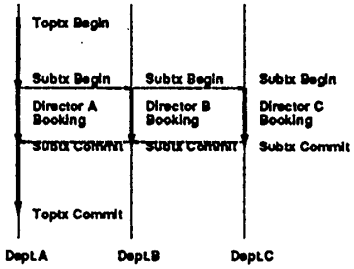

図 3: トランザクションの記述

C予約という3つのサブトランザクションを発生し、これらがそれぞれ独立して処理を行なうことになる。

このようにすると、処理の並列度を上げることができる。

また、Director B か Director C の予約がとれなかった時でも、予約自体はどちらかの予約が取れれば良いので、予約の取れなかったサブトランザクションだけをアボートし、その他のサブトランザクションをコミットするということも図4のように可能となる。

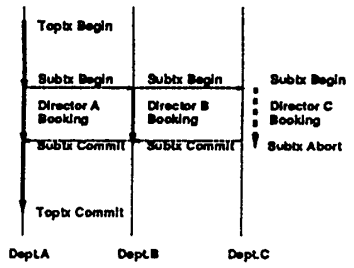

図 4: サブトランザクションのアボート

3.3 システム構成

我々は、以上のことより、階層トランザクションを利用してスケジュール予約システムを試作した。

この時に、先に東芝で開発した、トランザクション・モニタ [1] を利用することにより、分散階層トランザクションを実現した。

試作したアプリケーションの構成は、サーバクライアントモデルに基づいている。アプリケーションはC++を用いて記述して、各サーバを C++のクラスとして実現した。サーバとして実現されているのは、各Dept. の予約管理モジュールである。サーバの提供するサービスはメソッドとして記述している。一方クライアントは、ウインドウシステムを通してユーザと会話し、適切なサービスの依頼を各サーバに送るようになっている。

これにより、クライアントから 3 人の予約を取るトップトランザクションが発生し、ここから Dept.A, Dept.B, Dept.C の予約管理サーバを呼び出して、これがサブトランザクションとして実行されることになる。

クライアントサーバ間のインタフェースは、トランザクションモニタ用のインタフェース記述言語で記述している。

この記述言語は、モニタ用 RPC スタブジェネレータにより C++に変換される。この RPC ジェネレータにより、クライアント側では通信プロトコルを意識することなく、C++のメソッド起動でサーバ側との通信を記述できるようになっている。

このシステム構成は図5のようになる。

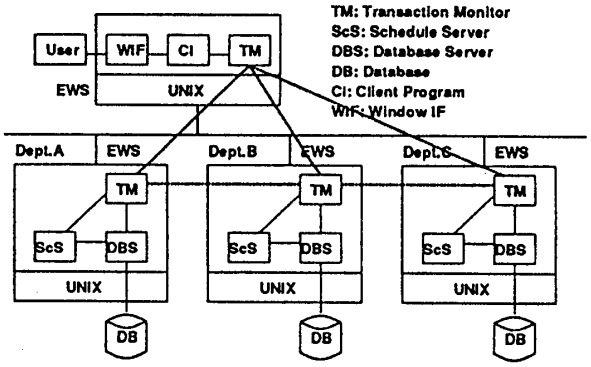

図 5: システム構成

4 結論

トランザクション処理のアプリケーションを構築する際、階層トランザクションとオブジェクト指向の考え方を導入すると、従来複雑になり過ぎて記述の困難であった分散環境に広がるトランザクションも、容易に記述できるようなり、高い機能のサービスを提供できるようになった。

他に階層トランザクションを適用できる例としては

- 複数の帳票を提出し、それぞれの帳票が確実に処理される必要のある、転入/結婚時などにおける帳票処理
- 飛行機の予約と、ホテルの予約などのように、複数のシステムに処理が広がる旅行予約業務

などが考えられる。この様に様々な分野で階層トランザクションは有効であると思われる。

5 最後に

以上のように、階層トランザクションという考え方を導入することにより、今後ますます複雑化分散化していくトランザクションに対処していくことができるものと思われる。

ただ、複数のプロセスが同時に走ることになるので、デバッグが非常に難しいなどの問題は今後の課題として残っている。

参考文献

[1] 金井, 白木原: 階層トランザクション機構による UNIX 上の高信頼分散処理環境, 情報処理学会第 84 回計算機アーキテクチャ研究会 92-8, 1992.