南山大学 数理情報学部 情報通信学科 2004 年度卒業論文要旨集

Web サービスのトランザクション処理

2001MT009 坊田 美寿珠　2001MT083 大谷 洋子

指導教員 青山 幹雄

1. はじめに

現在, Web サービスを B2B で利用するために, 複数のシステムを連携する様々な仕様の策定が進められている. Web サービスに対応したトランザクション仕様では長期のトランザクション処理[1]を実現している. しかし, トランザクション処理としての成功と, ビジネスとしての成功は必ずしも一致しない. また, トランザクション処理に参加する全ての企業が自らのポリシーを表現できる仕組みが必要である.

本研究では, ビジネスとしての成功を優先する処理を実現するために, 必要な機能を考察し, より柔軟なトランザクション処理として対話型トランザクションを提案する.

2. トランザクション処理の問題点と解決策

2.1. Web サービスのトランザクション処理の問題点

Web サービスのためのトランザクション仕様として BTP（Business Transaction Protocol）, WS-Transaction（Web Services Transaction）[2], WS-CAF（Web Services Composite Application Framework）の策定が進められている. これらは, 複数の企業間をまたがる処理を 1 つのトランザクションとして処理することを実現する. しかし, ビジネスプロセスの連携を実現するために, 以下の問題を解決する必要がある.

(1) ビジネスプロセス連携におけるトランザクション処理では, all-or-nothing の必要性がなく, 状況に即した柔軟性のあるトランザクション処理が求められる.

(2) Web サービスでは, 長時間のリソースロックを避けるために, トランザクション全体の結果が判明しないまま, 子タスクの処理を確定させる. これでは ACID 特性における独立性が保証できない.

(3) Web サービスでは, 処理を終了するまでに長時間かかることがある. 刻々と状況が変化するビジネスにおいて, 処理の遅延がデータの価値自体を失わせる可能性がある. 期限や連携するシステムの優先度を判断し, 状況に応じ最適な予約処理を行う必要がある.

2.2. 問題点に対する解決策

Web サービスにおけるトランザクション処理の問題点を解決するために必要な条件を以下に示す.

(1) 原子性の緩和
ACID 特性の原子性を緩和させたトランザクション処理を構築する. 必ずしもすべてのタスクが成功する必要はなく, 指定範囲を満たせばトランザクション処理として成功と判断する仕組みが必要である.

(2) 独立性の保証
中間結果と, 最終結果を区別できる必要がある.

(3) 長期トランザクションへの対応
Web サービスにおけるトランザクション処理では, サーバアプリケーションの不具合や通信障害等で処理が長時間中断されることがある. トランザクション処理が, いつ終了するか判断できない状態で放置されないように, 処理を終える期限を設定する. 期限内に処理結果が出せない場合はキャンセルを可能とする.

3. 対話モデルによるビジネスプロセス統合

前述した Web サービスのトランザクション処理における問題点を解決するために, ビジネスプロセスの統合方法として対話モデル[3]を導入する. トランザクションの参加者は全体の進行状況を理解する必要がなく, 参加者と要求元の独立した対話によって複雑なビジネスプロセス連携を行う.

3.1. 対話モデルとは

対話モデルでは, メッセージを送受信するための「メッセージング」と, 受信メッセージの構文解析や送信メッセージの形式を定義する「対話支援」の 2 つで相互運用性技術を実現する. ビジネスプロセスと相互運用性技術を分離することで, 企業独自のビジネスロジックを隠蔽できる.

3.2. 対話モデルの要素技術

(1) CP (Conversation Policy)
CP とはメッセージ交換仕様である. これはメッセージスキーマ, シーケンス, タイミング情報によって構成され, 状態遷移図による表現が可能である.

(2) 忍者ゲートウェイ(Ninja Gateway)
対話モデルにおける相互運用性技術は, 図 1で示す忍者ゲートウェイによってカプセル化される.

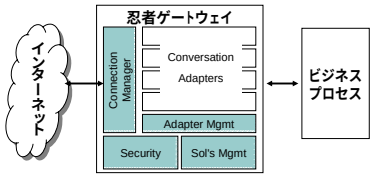

図 1 忍者ゲートウェイの構成

相互運用性技術の「メッセージング」は Connection Manager が,「対話支援」は CA(Conversation Adapter)が役割を担っている. また, 個々の CA は他のビジネスプロセスと対話を行う.

(3) CA(Conversation Adapter)

CA は CSB(Conversation Support Bean), CPH(CP Handler), Conversation Manager で構成される. CSB は対話のためのコンテキスト情報を, CPH は CP を木構造で管理している. Conversation Manager は対話中にサブ対話が開始された場合に, 適切な型の CP インスタンスを生成し, AcitveCP の子として CPH にインストールする. CP の木には必ず ActiveCP が 1 つ存在し, 受信したメッセージはすべてこれに渡される. もし受信したメッセージの形式が, その CP に許可されたものと異なる場合はメッセージを親 CP へ渡す.

4. 対話型トランザクションの提案

図 2は前述の対話モデルと WS-Transaction を組み合わせたトランザクションの構成図である. Coordinator は, 図 1 の忍者ゲートウェイの構成と同様に, 内部に対話のための CA を複数持つ. プロトコルサービスは登録されるプロトコルの数だけ作成される. コネクションマネージャは各プロトコルサービスと接続しており, 渡されたメッセージの振り分けを行う. プロトコルサービスによって外部のメッセージの送信者の判定を行い, コネクションマネージャによって内部のメッセージ受信者の判定を行うことで, 内部の CA 同士の会話も実現可能である.

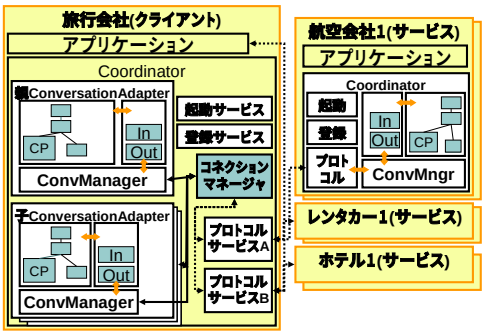

図 2 対話型トランザクションの構成

(1) スコープの定義

スコープとは複数のサービスをまとめたビジネスタスクである. スコープを同じ結果を得られるシステムごとに分割し, それぞれを CA で管理する. 指定範囲とは, トランザクション処理を成功と判断するために最低限満たすべきサービスの集合である. 各スコープは, 少なくとも 1 つコミットするとトランザクション全体の指定範囲を満たすように設定する. スコープ全体の結果を確認する前にコミットを行い, 全体の処理が失敗した場合は補償処理によって元の状態に戻す. この結果, 個々のタスクは全体の処理状況とは独立に処理を完了でき, 長時間のリソースロックを回避できる.

(2) CA の階層化

CA は ActiveCP を 1 つしか持たないので, 全体の状況と子スコープの状況を一つの CA で同時に管理するのは効率が悪い(図 3). そのため子スコープごとに子 CA を用意し, 全体の状態を把握する親 CA をその上に置く. 親 CA と子 CA はコネクションマネージャを通じて対話を行い, 全体の決定は親 CA が発するコミットで確定する.

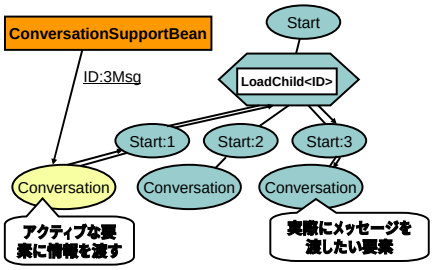

図 3 CA の分割の必要性

(3) 親と子の役割

親CA はトランザクション全体の流れを把握し, 対話の条件を調整する. 子CA はサービスとの 1 対 1 の対話と, スコープ内のトランザクションの管理を行う. 3章で説明した対話モデルの CP は, 対話をする相手と同じ CP を呼び出すことでスムーズにメッセージの解析を行う. 親 CA は子 CA の処理状況を把握する必要があるが, 同じ CP を使い複数の子 CA を管理することは, 上記の木構造と同様に効率が悪い.

そこで対話型トランザクションでは親 CA と子 CA が, それぞれの役割に合った CP を使用する. 子CA は自身のスコープ内の状況を把握する木構造の CP を管理する. 親 CA との対話時には, そのつど親 CA との対話用 CP を別に呼び出し, 対話が終わるたびに破棄する. 親 CA は, 各子 CA と対話した情報を保持し, 全体のトランザクション処理に沿った木構造の CP を管理する.

CP の分担によってそれぞれの役割に特化した CP を利用でき, 管理の効率が向上する.

5. 対話型トランザクション処理の制御方法

対話を用いたトランザクションでは, Web サービストランザクション同様, 中間結果を外部に公開する. 公開することでビジネスチャンスを失うという問題を解消するために, 以下に 2 つの方法を提案する.

(1) 中間結果と最終結果の区別

図 4はキャンセル待ちを利用したときの処理の流れを示した例である. キャンセル待ちによってサービス側システムはクライアントの確保が容易になり, クライアント側は望ましいサービスを受けられる可能性が広がる. 以下に 3 つの機能について説明する.

1) 一次コミット

中間結果を公開するコミットで, 全体の処理とは独立にスコープごとに行われる. 一次コミットまでに, 対話によって

二次コミットで確定するビジネス的な契約内容を決定する. 子スコープで一次コミットすると, そのスコープの CA はコミット完了を親 CA に伝達する.

2) 二次コミット

最終結果を確定する. 親 CA は, すべての子スコープが一次コミットを完了すると, 二次コミットを発する. 二次コミットを受けた子 CA はサービスにコミットを発し, 一次コミットで決めた契約を確定する. サービスは課金を含めた確定処理を行い, トランザクション処理を終了する.

3) キャンセル待ち

サービスの要求先が既に一次コミットを終えて中間結果を公開している時に, キャンセルを見越して予約を入れられる. 親 CA によって決められた期限内に, サービス側がキャンセルによってできた空きを通知してきた場合, クライアントはそのままトランザクション処理を再開する.

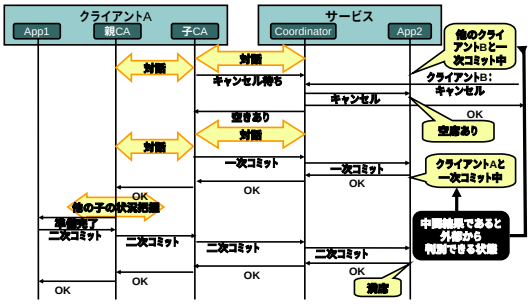

図 4 キャンセル待ちの利用

(2) トランザクションの直列処理

WS-Transaction では並列処理を用い, 全てのトランザクションが完了した後に最終結果に含めるサービスを指定し, トランザクションを完了する. 対話型トランザクションでは, 各 CA は一度に複数の相手と対話できず直列処理になる. そこで, 優先度を利用し, 最終結果に含めるサービスをトランザクション開始時に予測する. アプリケーションが取得した優先度を親 CA に伝え, 親 CA がその情報を元に優先度の高い順にサービスを並べ, 交渉を開始する. すると, 全てのサービスとトランザクションを起動することなく, 並行処理で最終結果に含めるサービスと同じサービスに一次コミットできる可能性が高い. 優先度によってキャンセル待ちの猶予期間を変化させれば, クライアントにとってより望ましい結果を得られる処理が実現できる

6. CP の設計

前述のシステムを実現するために設計した CP を示す. CP は参加者用 CP と親 CA との対話用 CP に分かれており, 木のノードとして呼び出すことで対話を実現する.

6.1. 子 CA の CP

子 CA の役割は外部との対話と, 親 CA との情報交換である. CP は参加者用と親 CA 用に分かれており, 識別子で使用する CP を区別する. 忍者ゲートウェイでは対話を行う際, まず 2 者間でどの CP を用いるかを交渉し, そこで決定した CP を呼び出して対話を開始する. しかし, トランザクション処理に適用する場合, 一貫性を保証するため, CP 作成の流れは図 5に示すトランザクションに沿ったものとした.

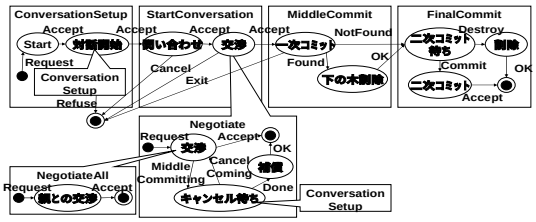

図 5 子 CA の状態遷移図

各 CP とその役割を以下に列挙する.

(1) ConversationSetupCP：対話の準備を行い, スコープ内で一次コミットが取れるまで次のサービスを呼び出し, 交渉を続ける.

(2) StartConversationCP：対話全体の流れを表す.

(3) NegotiateCP：対話による交渉を行う. サービス側が一次コミット中の場合はキャンセル待ちを行い, 次のサービスを呼び出す.

(4) MiddleCommitCP：交渉された契約内容を一次コミットする. 下に木があれば, 破棄した後, 二次コミット待ちに入る.

(5) FinalCommitCP：二次コミットを行う. 親からコミットを受け取るとそれをサービス側に伝え, トランザクションが完了する.

6.2. 親 CA の CP

親 Coordinator の役割はトランザクション全体の管理である. トランザクションの開始時に子 CA の識別子を把握し, 状況に応じてメッセージを発する(図 6).

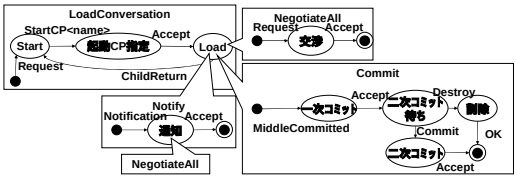

図 6 親 CA の状態遷移図

各 CP とその役割を以下に列挙する.

(1) LoadConversationCP：交渉したい内容の CP を対話で決定し, 呼び出す.

(2) NegotiateAllCP：親 CA は各スコープの CA から伝えられた情報を管理し, 調整して返す.

(3) NotifyCP：対話のためにアプリケーションから渡された期限や優先度などの前提条件を子 CA に通知する.

(4) CommitCP：トランザクションの一次コミットからの流れを管理する. 二次コミットを全ての子 CA に伝え, トランザクションを完了する.

7. 対話型トランザクションの評価

WS-Transaction の BACC プロトコル(Business Agreement with Coordinator Completion Protocol)と，対話型トランザクションを利用した場合の，(1)独立性の保証，(2)期限と優先度による最適化の 2 項目について比較し，対話型トランザクションの有用性を評価する．

7.1. 独立性の保証

BACC プロトコルでは，すべてのビジネスプロセスに対して並列処理を行う．その後，予約処理が成功したものの中から，顧客の希望順位の高い順に処理を確定させ，必要がないと判断されたものは，補償トランザクションによって取り消す．

対話型トランザクションでは，直列に処理が進行する．顧客の希望順位の高いものから逐次実行し，予約ができなかった場合は，図 7で示すようにキャンセル待ち状態を保持し，下位の予約処理を実行する．処理を終了させるまでは，一次コミット状態として外部から参照されるために，中間結果と最終結果を明確に区別できる．

7.2. 期限と優先度による最適化

BACC プロトコルでは期限や優先度を考慮することができない．しかし，対話型トランザクションでは，処理を終える期限とスコープ間の優先度を設定できる．優先度の高いスコープの予約をとるため，図 7で示すように優先度の低いスコープの一次コミット中の予約をキャンセルする．その結果，新たに顧客の希望順位の低い予約を実行する処理を動的に行うことが可能となった．

上記の結果から，対話型トランザクションは BACC プロトコルと比較して，独立性の保証が向上している．また対話を用いることで，状況に即した柔軟なトランザクション処理を実現していると評価できる．

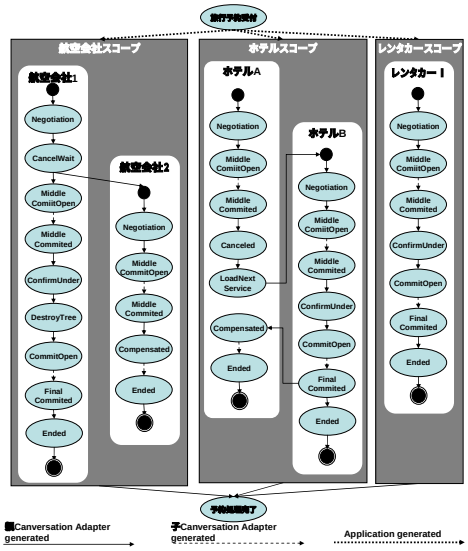

図 7 対話型トランザクションの予約処理フロー

8. 考察と今後の課題

対話型トランザクションでは ACID 特性の原子性を緩和し，トランザクション全体の成功ではなくビジネスとしての成功を優先させた．このため必要条件となるサービスを指定範囲として定義し，この中に含まれるサービスを種類ごとにスコープで分割し，子 CA で管理する．

中間結果の公開による独立性の喪失に対し，「一次コミット」と「キャンセル待ち」によって中間結果と最終結果を区別すること，直列処理にすることを提案した．複数のサービスを同時にコミットし，後で補償処理を行うよりも，最終結果に残るサービスのみを選別してコミットする方がトランザクション参加者の満足度が向上すると考えられる．これはキャンセルによるリスクが高いサービスに向いている．

処理の長期化によるデータの価値自体が失われる問題は，対話による期限の設定により解決する．期限内に処理が終わらないサービスは切り捨て，他のサービスとの交渉を開始することでビジネスの成功を優先している．

今回の提案では期限に関する交渉のみ評価した．今後の課題として，CP を拡張し，ビジネスにおいて期限と共に重要となる課金処理や，補償処理の内容を対話によって交渉できるようにしたい．優先度についても，条件によっては最適な結果を得られないことが評価によって判明した．それを改善するために．キャンセル待ちの成功確率を組み合わせ，より最適な予約処理を実行できるトランザクション処理を構築したい．

また，本研究では，対話型トランザクション仕様の提案という段階であるため，実装レベルの詳細な検証を行う必要がある．

9. まとめ

本研究では，ビジネスプロセス連携に対応した，柔軟性の高いトランザクション処理の実現を目的とした．そこで，Web サービスに対応したトランザクション仕様である WS-Transaction と，ビジネスプロセスの連携方法である対話モデルを組み合わせた，対話型トランザクションを提案した．対話型トランザクションは，トランザクション参加者との交渉や，期限や優先度の設定によって処理の最適化を図る．そして，参加者にとって満足度の高い処理を提供した．

参考文献

[1] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993.

[2] F. Cabrera, et al., WS-Transaction, 2004, http://www-106.ibm.com/developerworks/library/specification/ws-tx/

[3] J. E. Hanson, et al., Conversation Support for Business Process Integration, Proc. of IEEE EDOC2002, 2002, pp. 65-74.