2005－SE－147 (9)
2005／3／18

Webサービスの対話型トランザクションアーキテクチャの提案と評価

大谷 洋子　坊田 美寿珠　青山 幹雄

南山大学 数理情報学部 情報通信学科

Web サービスにおいて柔軟性の高いトランザクション処理を行う対話型トランザクションアーキテクチャを提案する．現在，策定が進められている Web サービストランザクションは長期の処理に対応するため，トランザクション処理が完了する前に個々の処理を確定し，結果を公開する．そのため個々の結果と最終的な結果が一致しない可能性がある．これに対し，本論文では，状況の変化に対応し，ビジネスプロセス間の対話によって，状況に即したトランザクション処理を実現する．対話型トランザクションは，WS-Transaction に基づいて，対話モデルを統合し処理を協調させ，期限や優先度を考慮した処理を可能とする．アーキテクチャ，挙動，設計方法を提案し，対話型トランザクションの有用性について旅行代理店予約システムを用いて評価した．

Conversation Transaction Processing Architecture for Web Services and its Evaluation

Yoko Otani　Misuzu Boda　Mikio Aoyama

Department of Information and Telecommunication Engineering,
Faculty of Mathematical Sciences and Information Engineering, Nanzan University

We propose a *conversation transaction processing architecture* in order to provide transaction processing of high flexibility for Web services. In the transaction specifications under standardization of Web services, it is required to complete the individual task before completion of the entire transaction processing. However the specifications are too inflexible to process multiple transactions efficiently. We proposed the conversation transaction processing architecture by integrating WS-Transaction substructure and the conversation model. It enables conversation transaction to process under the time constraints and priority of business processes. We will propose the software architecture, behavior model, and the design method of conversation transaction and evaluate the effectiveness by examples.

1. はじめに

トランザクション処理[5]は，関連する複数の処理を一連の処理として管理し，データの整合性を保つ技術である．従来はイントラネット内のデータベースへの書き込みなど，信頼性の高い通信状態で一括管理する場合に用いてきた．しかし，ネットワーク化が進み，異なる複数のシステムにまたがってトランザクション処理を行う必要が生じている．既存のトランザクション処理技術ではビジネスプロセス連携に対応できないため，Web サービスのためのトランザクション処理や，ビジネスプロセスの連携を実現する仕様の策定が進められている[1, 2]．

Web サービスに対応したトランザクション仕様として BTP (Business Transaction Processing)[8]，WS-T (Web Services Transaction)[4]，WS-CAF (Web Services Composite Application Framework)[3]の 3 つの仕様が提案されている．

これらの仕様の問題点として，トランザクション処理としての成功と，それが果たすべき目的とは必ずしも一致しないことが挙げられる．従来のトランザクション処理では，ACID 特性の原子性を保証するために，全ての処理の成功がトランザクション処理としての成功の必要条件である．しかし，ビジネスプロセス連携のトランザクション処理では，全ての処理が完了することより，部分的な処理を迅速に完了する方が望ましい場合がある．そのため，一定の条件の下では原子性を緩和させたトランザクション処理が必要となる．

また，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，異なる企業ドメイン内のリソースを扱う．上記のトランザクション仕様では長期のリソースロックを避けるため，トランザクション全体の処理とは独立に個々のサービスの処理を確定する方法を用いる．しかし，この確定は，全体の処理結果次第で取り消される可能性が高く中間結果といえる．そのため ACID 特性における独立性が保証できない．独立性を保証するために，中間結果と最終結果を区別して公開する必要がある．

本稿では，上記の Web サービスのためのトランザクション処理の問題点を解決するために原子性を緩和し，独立性を保証したトランザクション処理のアーキテクチャを提案する[9]．そして，旅行代理店予約システムを例として，提案するアーキテクチャの評価を示す．

2. 関連研究

Web サービスのトランザクション仕様として，BTP，WS-T，WS-CAF の策定が進められている．本章では，それぞれの仕様の特徴と問題点を議論する．また，ビ

ジネスプロセスの連携方法としては，コレオグラフィ記述言語[6]でワークフローを記述する方法と，対話による方法がある．ここでは，処理の間に起きる状況の変化を通知し，全体の処理に反映する仕組みが必要である．したがって，個別の通知とそれに対応し柔軟に処理できる，対話モデルを検討する．

2.1. Web サービスのためのトランザクション仕様

(1) BTP

BTP では，ビジネスプロセス間の通信メッセージ仕様と，その通信に関連してトランザクション処理の参加者が最低限果たすべき義務を定義している[8]．この仕様によって，処理に長時間要する長期トランザクションに対応できる．しかし，ACID 特性は保証されない．

(2) WS-T

WS-T は，WS-C (WS-Coordination)，WS-AT (WS-Atomic Transaction)，WS-BA (WS-Business Activity)により構成される[4]．WS-C は，図 1に示すようにアプリケーションごとに Coordinator を持つことによって，分散アプリケーションの振る舞いを調整するプロトコルを提供する拡張可能なフレームワークである．WS-AT は WS-C で使用する 3 つの調整プロトコルを提供する．この調整プロトコルは，ACID 特性を保証したトランザクション処理を実現する．WS-BA は，ビジネスプロセス連携等の異なる実行環境で相互運用するための調整プロトコルを定義している．複数のサービスを任意の処理単位としてまとめ，一つのビジネスタスクとして処理できる「スコープ」機能を持つ．個々のサービスの処理を直ちに確定し，全体の処理結果に応じて決定するか，補償処理を行う．補償処理とは，確定した処理を元の状態へ戻すことである．しかし，WS-T では，補償処理の仕組みや実装方法が明確に定められていない．

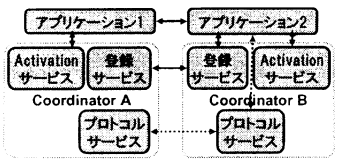

図 1 Coordinator によるアクティビティへの参加

(3) WS-CAF

WS-CAF はトランザクション処理を含む複合的な Web サービスの仕様群であり，WS-CTX(Web Services Context), WS-CF(Web Services Coordination Frameworks)，WS-TXM(Web Services Transaction Management)から構成される[3]．これらによって複数の Web サービスから構成されるアプリケーション群が同一のコンテキスト情報を共有し，ビジネスプロセス連携の自動化を図る．しかし，この連携方法は，単一組織によって定義される方法をとっており，トランザクションへの参加者の意向は反映されない．

2.2. 対話モデルによるビジネスプロセス連携

対話とは，メッセージ交換によるビジネスプロセス間の協調を行うことである．対話によって異なる企業ドメイン内に分散するビジネスプロセスを統合できる．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 つで相互運用性を実現する．ビジネスプロセスと相互運用性を分離することで，企業独自のビジネスロジックを隠蔽できる．

対話モデルの要素技術として，CP (Conversation Policy)，忍者ゲートウェイ(Ninja Gateway)，CA (Conversation Adapter)がある[7]．

(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，その挙動は，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を実現する技術は，図 2 に示す忍者ゲートウェイによりカプセル化される．

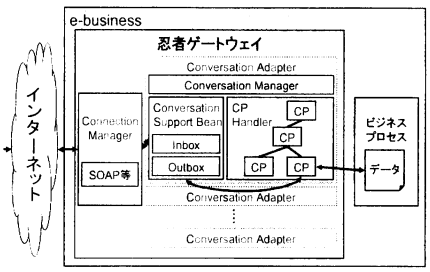

図 2 忍者ゲートウェイの構成

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 が一つ存在し，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．

ここで示した対話モデルの技術を用いて，複数の企業間にまたがる Web サービスのトランザクション処理を柔軟に連携できる．

3. Web サービスのトランザクション処理の解決策

3.1. 原子性の緩和

ACID 特性の原子性を緩和させるために，WS-T のスコープの概念を拡張する．スコープによって，必ず成功しなくてはならないものを指定範囲と定める．一方で，同一の結果が得られるシステムを集め選択的に必要な処理だけ成功すれば良いものを子スコープとして管理する．こうして全体のトランザクション処理としては原子性を保証し，子スコープ内の原子性を緩和させるトランザクション処理を実現する．

3.2. 独立性の保証

外部組織からの長時間のリソースロックは，ビジネスプロセス連携において重要な問題である．そこで，トランザクション全体の処理結果に関わらず子スコープ内のビジネスプロセスを早い段階で確定する．しかし，この確定は他の子スコープの処理結果によっては，補償処理によって取り消される可能性が高く，結果的に中間結果となる．これによってビジネスチャンスを失う危険が発生する．そこで,外部のビジネスプロセスから，公開中の結果が中間結果であるのか，それとも最終的な確定結果なのか区別して公開するトランザクション処理を提案する．

3.3. 柔軟性の向上

Web サービスのトランザクションでは，処理が長時間に及ぶ．そのため，必要な結果が，必要な期限内に得られなければ，トランザクション処理として成功しても，ビジネスとしての目的を達成できない場合がある．そこで，トランザクション処理の結果を確定する期限や，ビジネスプロセス間の優先度を設定できるシステムが必要となる．期限内に処理の結果が出せない場合は，そのビジネスプロセスをキャンセルし，クライアントが第 2，第 3 に希望するビジネスプロセスへの予約処理を起動し，柔軟なトランザクション処理を可能とする．

4. 対話型トランザクションの提案

Web サービストランザクションの問題点を解決し，状況に応じた柔軟なトランザクション処理を実現するため，3章で述べた解決策を用い，ビジネスプロセスの連携に対応したトランザクション処理を提案する．基礎となるトランザクションの仕様は具体的な構成を提案している WS-T を用いる．

コレオグラフィの利点は複数のサービスの複雑な連携を，流れとしてまたは事前条件と事後条件によって制御できる点である．対話の利点は複数のサービスを個々の独立した対話処理によって制御できる点である．参加者が全体の状況に応じて振舞うか，自身に最適な処理を交渉するかという違いがある．

WS-T では，Coordinator が全ての参加者の処理状況を把握し，結果に関係する参加者を状況に応じて自由に選び，最終的な決定を行う．参加者はトランザクション全体の進行状況を知る必要がない．よって，独立した個々の対話によって複雑な連携を効率よく処理するため，対話方式を WS-T に適用した対話型トランザクションを提案する．

4.1. 対話型トランザクションのアーキテクチャ

対話型トランザクションのアーキテクチャを図 3に示す．トランザクション全体の仕組みは WS-T，対話機能は対話モデルを用いる．Coordinator 内に対話の機能を付加することで，参加者との間で対話を行い，期限などの交渉やトランザクションが失敗した時の補償処理の内容を決定する．処理の進行によって状況が変化した場合も対話によって調整を図り，ビジネスプロセスへの対応が可能である．Coordinator は，図 2 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．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，識別子を用い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．

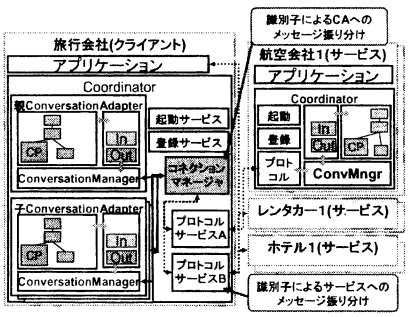

図 3 対話型トランザクションアーキテクチャ

4.2. 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．図 4に示すように，同じ結果を得られるシステムごとにスコープとしてまとめて扱い，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，少なくとも一つコミットすると，トランザクション全体の指定範囲を満たすように定める．全体の結果を確認する前にスコープごとにコミットを行い，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．個々のタスクは全体の状況とは独立に処理を完了

し，長時間リソースをロックするリスクを回避できる．

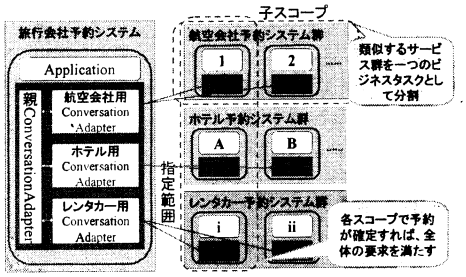

図 4 スコープの分割

4.3. CA の階層化

対話モデルではメッセージを常にアクティブな CP に渡す．CA は ActiveCP を一つしか持たないので，図 5 に示すようにメッセージを渡したい CP まで到達するために，他ノードから目的の CP のあるノードに渡していく必要がある．並行処理する各スコープの状況を一つの CA で同時に管理すると，アクティブな CP とメッセージを渡したいCPが合致しない可能性が高く効率が悪い．その上，対話モデルで複数の CA を利用できる利点を失う．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，全体の決定は親が発するコミットで確定する．

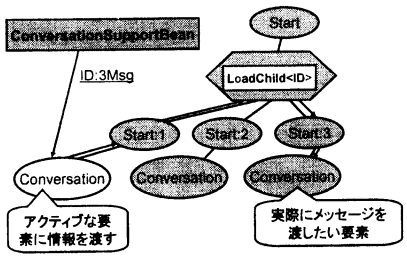

図 5 CA の分割の必要性

4.4. 親 CA と子 CA の役割

2.2節で説明した対話モデルの CP は，対話をする相手と同じCPを呼び出すことでメッセージの解析をする．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 の役割はサービスとの 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ことである．そのため，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 の役割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ことである．そのため子 CA ごとに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の制御方法

対話の利点は，処理の進行状況に応じてリアルタイムに条件の交渉を行えることである．従って，対話によって期限や補償処理の内容を処理の状況に応じて最適化できる．サービスの期限などの条件変更を要求すると，子 CA はそれを親 CA に伝える．親 CA はアプリケーションから期限や優先度などの情報を得て，調整を行う．親 CA が対話の内容を常に把握し調整をすることでトランザクション全体の一貫性を保つ．

5.1. 適応型トランザクション制御

対話を用いたトランザクションでは，WS-T と同様に，対話で調整中の中間結果を外部に公開する．これは，長期間のリソースのロックを避けるために必要である．しかし，公開することで独立性が保証できないという問題点がある．そこで中間結果と最終結果を区別して公開する．処理が完了した時に中間結果と最終結果が異なっていても，トランザクションの進行状況と他のシステムから見た進行状況は一致するので，独立性が保たれる．中間結果までの処理を確定するコミットを「一次コミット」，最終的な決定を「二次コミット」，中間結果に対して入れる予約を「キャンセル待ち」と定義し，以下に説明する．

(1) 一次コミット

キャンセル可能な予約処理で，スコープごとに独立して実行可能である．一次コミットで決定することは，二次コミットで確定する課金やキャンセル時の補償など，ビジネスプロセスに関わる契約の内容である．中間結果の公開後も期限などのトランザクション処理に関わる情報は変更でき，順次更新する．一次コミットの後，サービス側 Coordinator はリソースを解放し，コミットされた条件を保持しておく．一貫性を保つため，一次コミット後は契約の内容を変更しない．子スコープで一次コミットすると，そのスコープの CA は一次コミット完了を親 CA に伝達する．

(2) 二次コミット

一次コミットで決定した契約内容を確定する．確定後のキャンセルは補償処理に則って行う．親 CA はすべての子スコープが一次コミットを完了したことを知ると，二次コミットを発する．二次コミットを受けた子 CA はサービス側にそれぞれコミットを発し，一次コミットで決めた契約を確定させる．サービスは確定処理を行い，トランザクションを終了する．

(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る．親 CA によって通知された期限内に，サービス側がキャンセルによる空きを通知した場合，クライアントはそのままトランザクションを再開する．

キャンセル待ちを含む処理の流れを図 6に示す．中間結果を「一次コミット中」とし，最終結果の「満席」と区別して公開することで，実際の状態と外から見た状態が一致し，独立性が保たれる．

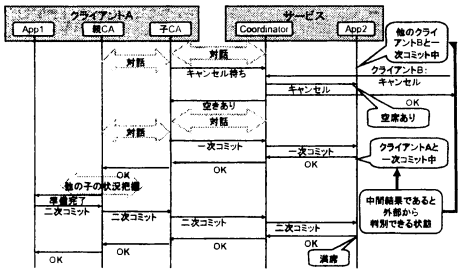

図 6 キャンセル待ちの利用

独立性を保証することは，キャンセルによるリスクが高い時に有効である．たとえば，画像変換処理などのサービスは要求の数だけサービスを提供することができるので，キャンセルが重大な損失をもたらす可能性は低い．しかし，チケットや商品販売サービスなどでは提供するサービスに限りがあるため，キャンセルによるリスクが高い．

サービス提供側は，課金を伴わない予約によってリスクを負うが，リアルタイムの正確な状況を公開することで，クライアントを一つに束縛されないという利点がある．クライアント側は，複雑な組み合わせのサービスを利用したいときに，対話によって期限などの交渉を行うことで効率よくビジネスの変化に対応できる．また，キャンセル待ちによってサービスはクライアントの確保が容易になり，クライアントが望むサービスを受けられる可能性が広がる．よって，これらの機能は実際の商品を扱うビジネスにおいて特に有用である．

5.2. トランザクションの直列処理

WS-T 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の場合，各 CA は一度に複数の相手と対話できないので，各スコープの処理は1対1になる．スコープ内の処理が直列になる場合，スコープ内の全てのサービスとトランザクションを起動すると全てのトランザクションの完了までの時間が長くなる．そこで，優先度をつけた直列処理によって，クライアントにとって満足度の高い結果を得る方法を提案する．

クライアントがサービスに対して最終結果に含むことを希望する順位を優先度とする．まず，アプリケーションがクライアントからどのサービスを優先的に予約するべきか情報を取得し，親 CA に伝える．親 CA はその情報を元にスコープ同士の優先順位を決定し，さらにスコープ内でも優先度の高い順に交渉を行う．図 7に示すように，優先度を用いれば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，並行処理と変わらない優先度のサービスを得られる可能性が高い．優先度の低いサービスは中間結果の公開によって束縛されず，ビジネスチャンスを失わずにすむ．さらに，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る処理が実現できる．

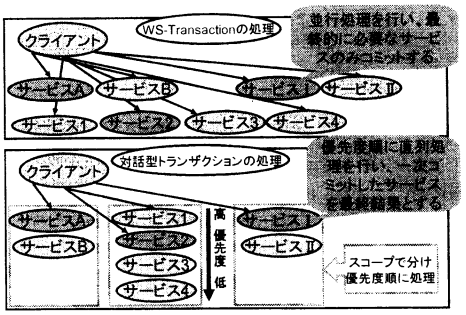

図 7 トランザクションの直列処理

6. CP の設計

対話型トランザクションを評価するため，対話モデルの CP 作成例[7]から状態遷移図の表現規則(表 1)を作成し，CP を設計した．トランザクション処理の一貫性を保証するため，WS-T の BACC プロトコル(Business Agreement with Coordinator Completion protocol)のフローを基礎とし，状態ごとに対話を行えるようにした．状態遷移図をもとに CP を作成し，Coordinator の CA 内で呼び出すことで，対話型トランザクションが実現する．

表 1 状態遷移図の表現規則

表記	説明
<CP-name>	CP名
Role	参加者の役割
⟶	遷移
<state>	<state>状態を表す
<state> LoadChild "<CP-name>"	<CP-name>という子CPを呼び出す <state>状態を表す
<state> Return "<returnname>"	<returnname>という通知を返す<state>状態を表す エンドポイントを兼ねる
<Conversation-ID>:"<notification>"	<Conversation-ID>というRoleからの<notification>通知を表す
ChildReturn:"<notification>"	呼び出された子CPからの<notification>通知を表す

6.1. 子 Conversation Adapter の CP

子 CA の役割は外部との対話と，全体のトランザクションとの情報交換である．対話の機能は参加者用と親用に分かれており，識別子で使用する CP を区別する．

大まかな状態遷移の流れを図 8に示す.

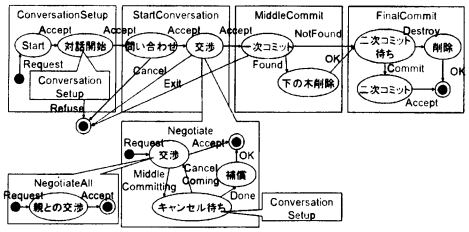

図 8 子 CA の状態遷移図

各 CP とその役割を以下に列挙する.

(1) ConversationSetupCP

ConversationSetupCP はトランザクションで対話を開始するために最初に作成される CP である. Coordinator から対話の開始を通知し, サービスから承諾を得ると StartConversationCP を呼び出す. スコープ内で一次コミットが取れるまで, LoadNextService によって次のサービスをアプリケーションが登録し, 交渉を続ける. 二次コミットが完了すると子 CA から"Done"が返り, Conversation Manager によって木が破壊される. "Destroy"が返った場合は, Conversation Manager が同じ ID のサービスとの対話によってできた木構造をすべて破壊し, アクティブな CP に"Destroyed"を返す.

(2) StartConversationCP

対話全体の流れを表す CP(図 9). クライアントのリクエストに対してサービスが承諾を返すと交渉の CP を呼び出し, 交渉が成立すると一次コミットの CP を呼び出す. 一次コミット CP は内部で二次コミット CP を呼び出すので, "Done"が返るとトランザクションが完了する.

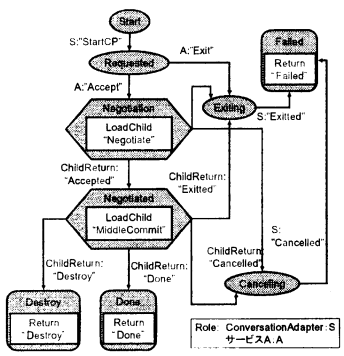

図 9 StartConversationCP の状態遷移図

(3) NegotiateCP

対話による交渉を行う CP. クライアントは交渉したい内容についてサービスに要求を出す. もしサービス側が他のクライアントとの一次コミット中だった場合, クライアントはキャンセル待ちに入り, 次のサービスとの交渉を開始する. 変更要求が来た場合は, その内容について親 CA との対話を行う NegotiateAllCP を呼び出し, そこで得られた回答を再びサービス側に返す. 交渉が成立すると上のノードに"Accept"が返される.

(4) MiddleCommitCP

交渉された契約内容を一次コミットする CP(図 10). クライアントの発した一次コミットをサービス側が受け取り, 一次コミットを返すと, クライアントは下に木がないかを検索する. もし下に木が存在するのなら Conversation Manager によって下の木を破棄したのち, FinalCommitCP を呼び出し二次コミット待ちに入る. 二次コミットが完了した場合は"Done"が返り, トランザクションが完了する.

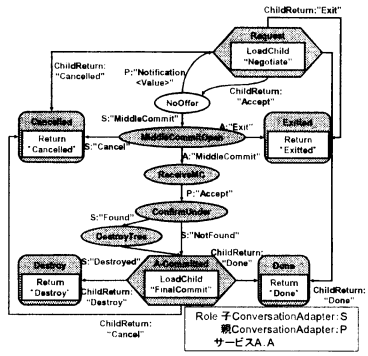

図 10 MiddleCommitCP の状態遷移図

(5) FinalCommitCP

二次コミットを行う CP. この CP に到達した時点で契約内容は確定する. サービス側からはキャンセルが返ることはない. 親 CA からの"Commit"を受け取るとそれをサービス側に伝え, トランザクション全体を完了する.

6.2. 親 Conversation Adapter の CP

親 CA の役割は主にトランザクション全体の管理である. そして, 親 CA は内部との対話のみ行う. トランザクションの開始時にコネクションマネージャから子 CA の識別子を受け取り, 状況に応じてメッセージを発する. 大まかな状態遷移の流れを図 11で示す.

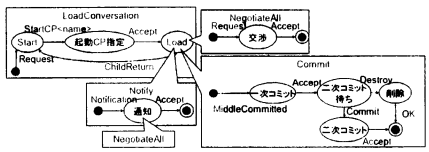

図 11 親 CA の状態遷移図

各 CP とその役割を以下に列挙する.

(1) LoadConversationCP

LoadConversationCP は, 親 CA からメッセージを送信したり, 子 CA からのメッセージを受け取ったりすると開始される. 交渉したい内容の CP を対話で決め呼び出す. ノードから通知が返ると Start 状態に戻り, 次の呼

び出しに備え待機する.

(2) NegotiateAllCP

親 CA と子 CA の間で行う対話のための CP. 子 CA はサービス側から要求の変更を通知されるとこの CP を呼び出し, 親 CA に情報を通知する. 親 CA は各スコープの CA から通知された情報を管理し, 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 CP. 親 CA から与えられた条件を子 CA が承認すると終了する.

(4) CommitCP

CommitCP は一次コミットから二次コミットまでを管理する CP(図 12). 子 CA から一次コミット完了の通知が届くと, 親 CA はその子の CP を作成する. アプリケーション(クライアント X)からのコミットを受けて各 CP でコミットを発し, クライアントから"Done"を受け取るとアプリケーションに"Committed"を返し終了する. 一次コミットが全て揃ったところでコミットを発するので, 全ての子 CA にコミットが通知されることになる.

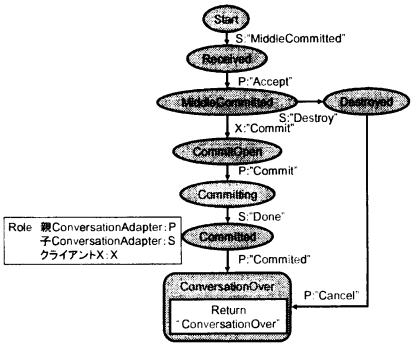

図 12 CommitCP の状態遷移図

7. 対話型トランザクションの評価

WS-T の BACC プロトコルと, 対話型トランザクションを利用した場合の, (1)原子性の緩和, (2)独立性の保証, (3)期限と優先度による最適化の 3 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.

評価には, 図 4に示す旅行代理店予約システムのモデルを用いる. これは, クライアントの要求に従って, 旅行の予約処理を開始する. 全体の予約処理がトランザクション処理であり, 各子スコープ内の予約処理が処理を成功させるためのタスクである. 比較のために表 2に示すシナリオを用いる.

表 2 旅行代理店予約シナリオ

スコープ間の優先度は, 航空会社＞ホテル＞レンタカーである. スコープ内の優先度は, 航空会社は 1>2, ホテルは A>B, レンタカーは I＞IIとする. 予約処理を確定させる期限は 3 日とする. 予約処理を実行した時点で, 航空会社 1 の予約に失敗する. ホテル A は一次コミット状態を 1 日しか保持できない.

このシナリオに沿って WS-T の BACC プロトコルでトランザクション処理を実行すると, 図 13 に示す状態遷移を行う. 処理結果の組み合わせとしては, 航空会社 2, ホテル A, レンタカー I となる.

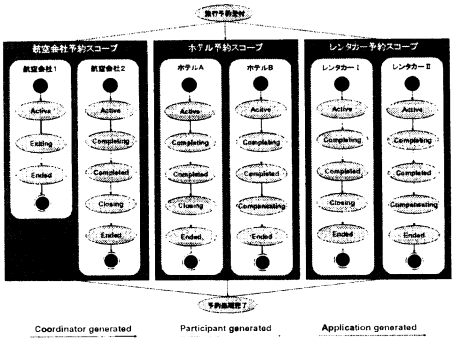

図 13 WS-T(BACC プロトコル)の予約フロー

一方, 対話型トランザクションで, このシナリオを適用すると, 表 3に示す処理の組み合わせが得られる. 図 14にパターン 4 の状態遷移図を示す.

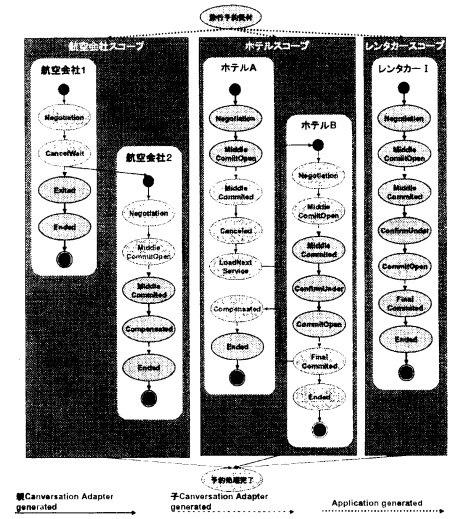

図 14 対話型トランザクションの予約フロー

表 3 対話型トランザクションの処理結果

	キャンセル待ち	期限延長	処理結果
1	OK	OK	航空会社 I, ホテル A, レンタカー I
2	OK	NG	航空会社 I, ホテル B, レンタカー I
3	NG	OK	航空会社2, ホテル A, レンタカー I
4	NG	NG	航空会社2, ホテル B, レンタカー I

7.1. 原子性の緩和

対話型トランザクションでは，WS-T のスコープ制御の概念を導入しているため，対話型トランザクションも WS-T も原子性を緩和する処理ができる．しかし，スコープ内のビジネスプロセスの制御方法に違いがある．

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．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し，必要がないと判断された場合，補償処理によって取り消す．

一方，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順次実行し，予約できなかった場合は，図 14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．

このように，対話型トランザクションでは状況に応じて原子性の緩和の範囲を変化できる柔軟な処理を実現できた．

7.2. 独立性の保証

BACC プロトコルでは，図 13 で示すように中間結果と最終結果の区別がなく，外部のビジネスプロセスに処理結果を公開している．このため，一度 Completed 状態と公開されてから，補償処理によって取り消される場合が多々発生する．

一方，対話型トランザクションでは，図 14で示すようにトランザクション全体の処理結果が確定するまでは，MiddleCommit(一次コミット)状態として外部から参照され，中間結果と最終結果を明確に区別できる．この結果から，独立性を保証できたといえる．

7.3. 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，処理結果を確定させる期限と，スコープ間とスコープ内のビジネスプロセスの優先度を設定できる．優先度の高いスコープの予約に失敗した場合は，設定した期限の間はキャンセル待ち状態を保ち，他のビジネスプロセスの一次コミット状態保持期限を考慮しながら処理を進める．その結果，表 3に示すようにトランザクション処理参加者の要求も処理結果に組み込みながら，状況に応じた様々な処理結果を得ることができる．

ただし，状況によっては，パターン 4 のように，BACC プロトコル以下の結果で処理が確定する場合がある．

8. 今後の課題

今後の課題は，提案した対話型トランザクションでは期限に関する交渉にしか対応していないことが挙げられる．CP の機能を拡張し，ビジネスにおいて期限と共に重要となる金銭取引や，補償処理の内容を対話によって交渉できるようにする必要がある．優先度についても，アプリケーションから与えられる順位だけでは常に最適な結果を得られないことが評価によって判明した．キャンセル待ちの成功確率と優先度を組み合わせ，より最適化された予約処理を実行できるトランザクション処理を構築することが望ましい．

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と考える．

9. まとめ

Web サービスにおけるトランザクション処理の目指すものは，all-or-nothing の画一的な処理ではなく，ビジネスプロセス連携に対応した柔軟性の高い処理である．しかし，Web サービスのトランザクション仕様は，ビジネスプロセス連携上の処理に対応するだけの柔軟さを持った処理ができない．

そこで本論文では，Web サービスに対応したトランザクション仕様である WS-T 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，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を用いることで，参加サービスがクライアントと条件に関する対話を行うことができ，変化するビジネスの状況に対応できる．また，優先度や期限を考慮することで，クライアントにとって望ましいサービスを選択でき，不必要なサービスの拘束を行わずに済む．従って，クライアントにとってもサービスにとっても満足度の高い処理を提供することが可能となった．

参考文献

[1] 青山 幹雄，ソフトウェアサービス技術へのいざない，情報処理，Vol.42, No. 9, Sep. 2001, pp. 857-862.

[2] D. Kaye, Loosely Coupled: The Missing Pieces of Web Services, RDS Press, 2003.

[3] D. Bunting, et al., WS-CAF, 2003, http://developers.sun.com/techtopics/webservices/wscaf/.

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

[5] J. Gray and A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993 [喜連川 優 (監訳)，トランザクション処理 (上・下)，日経 BP, 2001] .

[6] N. Kavantzas, et al., Web Services Choreography Description Language Version 1.0, W3C, 2004, http://www.w3.org/TR/ws-cdl-10/.

[7] J. E. Hanson, et al., Conversation Support, 2003, http://www.research.ibm.com/convsupport/.

[8] 小山 和也，BTP によるトランザクション管理，2002 ， http://www.xmlconsortium.org/Websv/kaisetsu/C16/main.html.

[9] 大谷 洋子，坊田 美寿珠，Web サービスのためのトランザクション処理に関する研究，南山大学数理情報学部 情報通信学科 卒業論文，Jan. 2005.