ビジネス・プロセス・エンジンを用いたサービス連携における ビジネス・トランザクション設計の実践的アプローチ

北尾 寧宏　亀田 綾

A Business Transaction Design Approach for Service Coordination Using Business Process Engines

Yasuhiro Kitao and Aya Kameda

疎結合なサービス連携の実現を目的としてビジネス・プロセス・エンジンを採用した場合，そのビジネス・トランザクション設計において，①ビジネス・プロセスと呼び出されるサービスの間のトランザクション，②ビジネス・プロセス・エンジンがフロー制御のために内部処理で使用するリソース，③ビジネス・プロセス・エンジン独自実装による挙動の考慮を見落としやすいという課題がある．本論文は，その課題を解決するために，サービス呼び出しのトランザクションやビジネス・プロセス・エンジンの内部処理に着目した実践的な設計アプローチを提案する．さらに，WebSphere® Process Server（WPS）を使用した実機検証によって，アプローチの有用性を示す．

When business process engines have been chosen for service coordination, the following important items for designing Business Transactions are commonly missed. 1) transactions between business processes and services, 2) internal resources for business process engines, and 3) business process engines' unique behavior. To solve these problems in designing "Business Transactions", we have suggested a practical approach focusing on "transactions between processes and services" and "internal resources for business process engines". Additionally, we have verified this approach using the WebSphere Process Server (WPS) and so can demonstrate its potency.

Key Words & Phrases：ビジネス・トランザクション，WS-BPEL，トランザクション設計，SOA，WebSphere Process Server
Business transaction, WS-BPEL, Transaction design, SOA, WebSphere Process Server

1. はじめに

Service Oriented Architecture（SOA）の普及に伴い，サービスの連携を「ビジネス・プロセス」として自動化させる「ビジネス・プロセス・エンジン」の採用を検討する事例が増加している．こうした中で，密結合のサービス連携を主対象としてデータ整合性を確保する従来の「ACIDトランザクション」（2.1 にて後述）に加えて，疎結合のサービス連携のデータ整合性を確保する「ビジネス・トランザクション」の実現が新たに求められている［1］．

ビジネス・トランザクション実現のための現実解としてWebSphere Process Server（WPS）などのビジネス・プロセス・エンジンを取り上げ，そのプロセス・フローのトランザクション（図 1 の①）に着目した議論はなされてきている［2］［3］が，その設計過程において以下の 3 つの考慮点が見落とされやすい．

a) ビジネス・プロセスと，そこから呼び出されるサービスの間のトランザクション（図 1 の②）

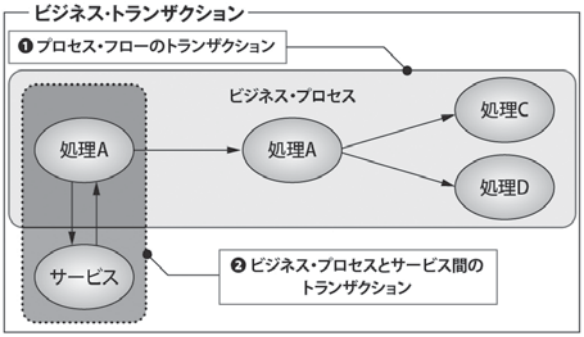

図1. ビジネス・トランザクション

提出日：2008年9月8日　再提出日：2009年3月4日

b) ビジネス・プロセス・エンジンがフロー制御のために内部処理で使用するリソース
c) ビジネス・プロセス・エンジンの製品固有の挙動

そこで，本論文ではこれら 3 つの考慮点を見落とさずにビジネス・トランザクション設計を行うためのアプローチを提示し，実機検証によってその有効性を示す．以下，2 章でビジネス・トランザクションに求められる要件および標準仕様について整理し，3 章でその設計のアプローチを提示する．4 章で WPS を使用した検証結果からアプローチの有効性を示す．

2. ビジネス・トランザクション

2.1 ACIDトランザクションとビジネス・トランザクション

トランザクションの目的はデータの整合性を確保することである．そのために，従来の密結合なシステムでは 1UOW（Unit Of Work）での確実なリソース更新が求められ，ミドルウェア製品による障害時の自動回復処理が一般的であった．このような厳密なトランザクションに求められる要件は，①原子性（Atomicity），②一貫性（Consistency），③隔離性（Isolation），④持続性（Durability）の 4 点であり，本論文ではこうしたトランザクションを「ACID トランザクション」と呼ぶ．

一方，SOA による疎結合なサービス連携が進むと，ビジネス・プロセスは企業間をまたがるなど長時間実行され，複数 UOW から構成されることとなる．UOW が複数に分かれると，ACID トランザクションのような厳密なリソース更新は不可能であり，別 UOW で更新の戻し処理を行う「コンペンセーション」処理により，ビジネス・プロセス全体でデータ整合性を確保する仕組みが必須となる．つまり，SOA システムでのトランザクションには，①長時間にわたる実行，②複数 UOW の制御，③コンペンセーションといった新しい要件が求められる．これが「ビジネス・トランザクション」である．

2.2 WS-BPEL

ビジネス・プロセス・エンジンの多くは標準仕様である WS-BPEL（Web Services Business Process Execution Language：以下，BPEL）［4］をサポートしている．BPEL は，前節のビジネス・トランザクションに求められる 3 要件を満たすプロセス制御のための仕様であり，**図 1 の①**に示すプロセス・フローは，BPEL を用いて定義することとなる．

BPEL では，呼び出すサービスをアクティビティーとして定義し，そのアクティビティーの呼び出し順序をフローとして定義する．アクティビティーは，パートナー・リンクという要素を介して，サービス（WSDL：Web Services Description Language）とひも付けられている（**図 2**）．

これにより，BPEL 自身は実装を意識しない抽象プロセスとして定義でき，プロセス・フローのトランザクション（**図 1 の①**）のみを，サービス実装と切り離して設計できる．これは，BPEL の利点の 1 つではあるが，ビジネス・プロセスとサービス間のトランザクション（**図 1 の②**）を見落としがちな原因の 1 つともなっている．

3. ビジネス・トランザクション設計アプローチ

3.1 ビジネス・トランザクションを構成する2つのトランザクションの設計

前述のように，ビジネス・トランザクションは，ビジネス・プロセスのフローに着目し，障害時のリカバリーを ACID トランザクションで実施するのか，コンペンセーションで実施するのかという観点（**図 1 の①**）と，ビジネス・プロセスとサービス間のトランザクションの UOW を 1 つにするかどうかという観点（**図 1 の②**）で分類することができる．以降では，前者を「BPEL トランザクション」，後者を「サービス呼び出しトランザクション」と呼び，それぞれ考察していく．

ビジネス・トランザクションを検討する際には，まず前提としてリカバリー方針を立てる．データ不整合発生時のリカバリーは，ACID トランザクション

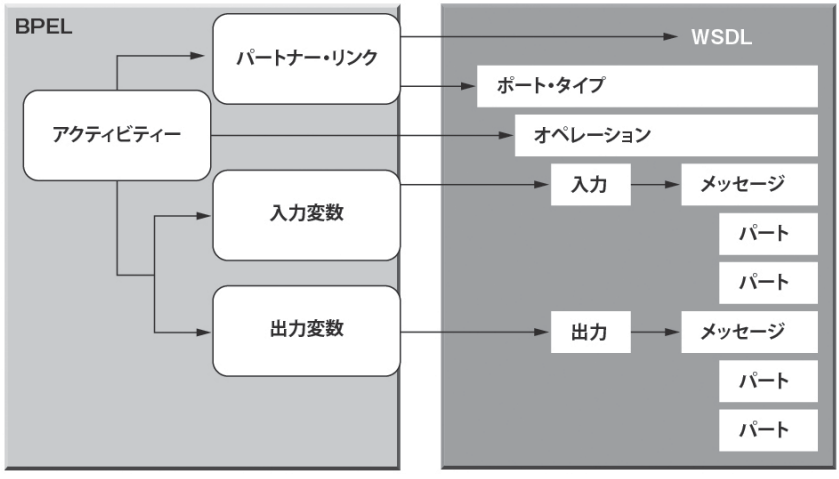

図2. BPELとWSDLの関係

での回復やコンペンセーションだけでなく，手動での対応も組み合わさることが多い．従って，自動化させることを主たる方針とするのか，パフォーマンスなどを考慮して自動化させる範囲を限定するのか，といった基本方針を立てておくことで，ある要件を実現するリカバリー方法が複数ある場合に適切な方法を選択する助けとする．

なお，システム全体のアーキテクチャーを踏まえて使用されるリソースを洗い出し，その CRUD（Create, Read, Update, Delete）分析を行うといった従来のトランザクション設計方法は踏襲し，その上で，特にビジネス・プロセスに注目した場合に実施すべきアプローチを解説する．

(1) BPELトランザクション

最初に BPEL トランザクションを設計する．BPEL は，業務処理の粒度でアクティビティーを抽出し，その呼び出しフローを定義する．従って，BPEL トランザクションを検討する際には，業務の観点で障害発生時にフロー全体としてどのようにデータ整合性を確保したいかを考慮して設計を進めていく．

① 更新系アクティビティーの抽出

障害時にデータ不整合が発生し得るのはアクティビティーの処理内容が更新系の場合である．更新系のアクティビティーを抽出し，検討が必要な範囲を明確化する．

② BPELトランザクションのリカバリー要件の整理

次に，抽出した更新系のアクティビティーについて，そのリカバリー要件を整理する．リカバリーの方法は**表 1** に示すように整理できる．BPEL トランザクションで自動的にリカバリーする場合は，複数アクティビティーの実行を 1UOW にして障害時に初期状態までトランザクションをロールバックさせるのか，コンペンセーションでコミット済みの状態を別 UOW で戻す処理をするのか，の検討が必要となる．

表1. BPELトランザクション・リカバリー方法

リカバリー方法		説明
手動		人の判断によるリカバリー
自動	ACIDトランザクション	複数アクティビティーを1UOWで実行し，ロールバック
	コンペンセーション	別UOWでの戻し処理

③ ACIDトランザクション設計

複数のアクティビティー間で厳密なリソース更新が求められる場合，2 つ以上のアクティビティーを 1UOW で実行する必要が生じる．この場合は ACID トランザクションによるリカバリーが必要となるが，複数アクティビティーを 1UOW で実行する方法は製品実装により異なるため，ビジネス・プロセス・エンジンに応じた設計を行う．

④ コンペンセーション設計

複数の更新系アクティビティーをおのおの別の UOW で実行し，障害時にその更新処理を戻したい場合，BPEL のコンペンセーション機能を使用する．コンペンセーションはアクティビティー単位や,複数アクティビティーをまとめたスコープ単位など，複数の設計パターンが考えられる．例外のハンドリングと併せて,最適な実装パターンを検討する．

(2) サービス呼び出しトランザクション

長時間実行される BPEL プロセスは，プロセス管理用にデータベースを使用することが一般的である．従って，**図 3** に示すように，サービス側のリソース更新とプロセスのステータス更新を 1UOW にしないと，**図 3** ⑤のステータス更新で障害が発生した場合に，フロー制御のトランザクションがロールバックされ，**図 3** ④サービス呼び出しが再実行される可能性がある．

二重送信が許されないなどの厳密なリソース更新が求められる場合には，サービス・リクエスターとしての BPEL と呼び出すサービス間でのトランザクションの原子性を保つために 2 フェーズ・コミット（2PC）を考慮する必要がある．

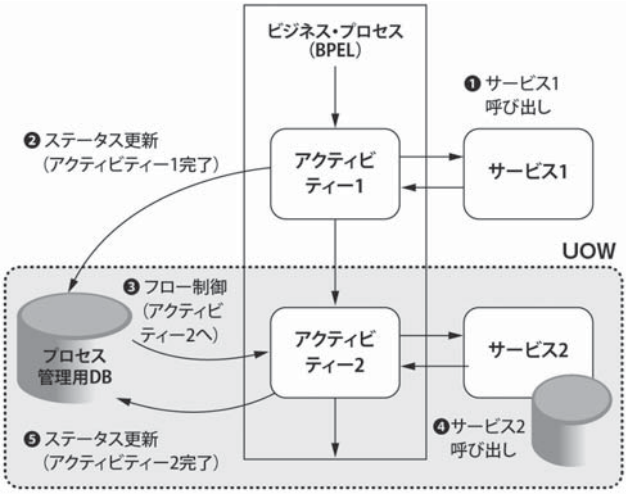

図3. ビジネス・プロセスとサービス間での2PC

以下，サービス呼び出しトランザクション検討のステップについて，Web サービス（SOAP/HTTP）を例に解説する.

① サービス実装の確認

BPELトランザクションで抽出した更新系アクティビティーについて，対応するサービスの実装内容を確認する．具体的には，処理方式（同期／非同期）やバインディング情報（プロトコル），使用するリソースなど確認する.

② サービス呼び出しトランザクションのリカバリー要件の整理

ここでは,BPELトランザクション特有のコンペンセーションは選択肢にないため，手動リカバリーか，ACIDトランザクションによるロールバックのいずれかが選択肢となる.

③ サービスの2PC可否確認

ACID を考慮する場合，まず，サービス側がプロセスとの 2PC 可能な実装となっているかを確認する必要がある．SOAP/HTTP バインディングで呼び出される Web サービスを例にすると，2PC 実現のためには，プロセスおよびサービスは Web Services – Atomic Transaction（WS-AT）[5]に対応している必要がある[6]．ホスト処理をラップしていて内部は非同期処理での実装となっているサービスなどの場合，プロセスとサービス間での 2PC は実現できない．このような場合は，BPELトランザクションの検討に戻り，コンペンセーションでの対応を検討する.

④ プロセス⇔サービス間の2PC設計

サービスが 2PC 可能な場合は，プロセスとサービス間でトランザクションを結合するための設定を行う（具体的な設定方法は使用するビジネス・プロセス・エンジンに依存する）.

BPELトランザクションとサービス呼び出しトランザクションの検討は，数回の反復を繰り返して精査していく.

3.2 ウォーク・スルーおよび実機による検証

前節の 2 つのトランザクションを組み合わせて，ビジネス・トランザクションの最終的な構成が決まる．複雑なトランザクション処理となるため,処理フローを記述したシーケンス図にトランザクション・スコープを記述し，処理内容，特にリソース更新と UOW の関係をウォーク・スルーすることを推奨する．この際，ビジネス・プロセス・エンジンの内部処理によるリソース更新までをシーケンス図に落とし込むことが重要である．プロジェクト上，特に重要なプロセスやトランザクション処理が複雑なプロセスについては,プロジェクトの早い段階でウォーク・スルーを実施する.

また，BPEL はベンダーによる仕様の拡張を許しているため,各社が独自の BPEL 拡張機能を有する．加えて，ビジネス・プロセス・エンジンの実装はベンダー依存となっているため，ウォーク・スルーに加えてプロトタイピングによる実機検証を行うことを強く推奨する．特に，障害テストにおいて，シーケンス図をベースに障害ポイントを網羅的に洗い出し，エンジン自身の障害ケースも含めてテストすることが重要である.

3.3 そのほかの考慮点

そのほかの考慮点として，Enterprise Service Bus を介すなどサービス呼び出しトランザクションが多層にわたる場合の問題判別・パフォーマンスの考慮や，コンペンセーション失敗時のリカバリーの考慮などが挙げられる．また，ベンダー各社の BPEL 拡張機能を標準化する動きがあるため，BPEL のポータビリティーを考慮する場合は，標準化の動向を随時確認しておく必要がある.

4. 設計アプローチの有用性検証

3 章で論じたトランザクション設計アプローチの有用性を実証するため，WPS を使用して図 4 の商品受注プロセスのトランザクションの設計を実施した.

商品受注プロセスは「受注」,「在庫確保」,「クレジット計上」,「配送手配」の 4 つの業務処理で構成されており，それぞれ対応する Web サービスを呼び出す．これらの業務処理をおのおの BPEL のアクティビティーとして定義した．なお，各 Web サービスは，図 4 に示したバインディングを使用することが決定しているものとした.

トランザクションにかかわる要件は以下を前提とした.

(a) 受注後の在庫確保を保証する．在庫が確保できな

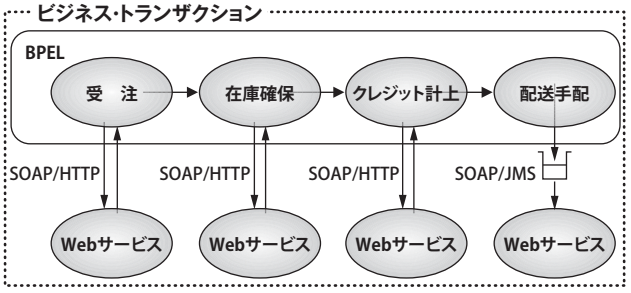

図4. 商品受注プロセス

い場合は，注文は受け付けられない．
(b) 配送手配サービスは完了に時間を要するため，配送結果を待たずに依頼のみを行う．
(c) 配送手配サービスへの依頼が完了するまでは注文の取り消しを可能とする．
(d) 各 Web サービスは実行時にデータを更新する．各 Web サービスの 2 重実行は許されない．
(e) 手動リカバリーによる運用は避ける．

4.1 BPELトランザクションの設計

上記要件を基に，BPELトランザクションの設計を行った．

① 更新系アクティビティーの抽出

要件（d）より，各アクティビティーではデータの更新が行われるため，すべてのアクティビティーを対象としてトランザクションを検討した．

② BPELトランザクションのリカバリー要件の整理

要件（e）より手動リカバリーは避け，ACIDトランザクションまたはコンペンセーションによる自動リカバリーを検討した．以下に自動リカバリーのおのおのの設計について述べる．

③ ACIDトランザクションの設計

要件（a）より在庫確保できなければ受注を受け付けないようにするためには，「受注」・「在庫確保」アクティビティーを 1UOW で実行する必要がある．しかし，BPEL はビジネス・トランザクションのための仕様であり，サービス呼び出しごとに UOW が分かれるのが通常である．そこで，WPS に BPEL 拡張機能として用意されている，「ショート・ランニング・プロセス」を用いることとした．ショート・ランニング・プロセスは，複数アクティビティーを 1UOW で実行するプロセスであるため，「受注」・「在庫確保」アクティビティーをショート・ランニング・プロセスとして定義し，商品受注プロセスから 1 サービスとして呼び出す設計とした．これにより，BPEL トランザクションにおける「受注」・「在庫確認」アクティビティーの ACID トランザクションが保証された．

なお，WPS においてアクティビティー単位に UOW が分かれるプロセスは，「ロング・ランニング・プロセス」と呼ばれる．このロング・ランニング・プロセスには，「トランザクション動作」という属性があり，コミットのタイミングを複数アクティビティーでまとめるといったトランザクションの制御が可能である．ただし，この機能は，あくまでもアクティビティーごとに UOW が分かれるプロセスを前提としているため，正常時のトランザクション最適化のみが可能であり，ACID トランザクションを保証するものではないことに注意する必要がある．

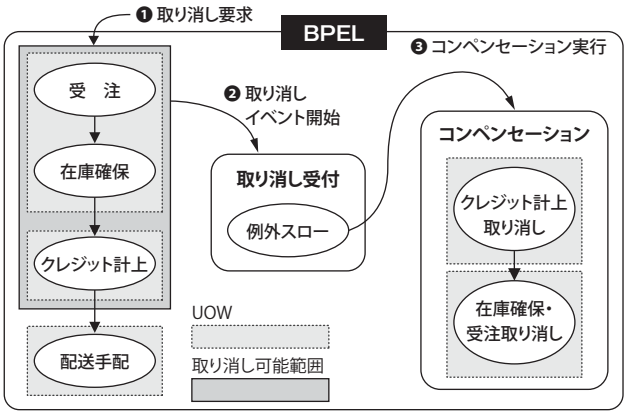

図5. コンペンセーション処理の実装

④ コンペンセーション設計

要件（c）により，配送手配サービスへの依頼が完了するまでは注文の取り消しを可能とする必要があるが，トランザクション処理が終了している更新系のアクティビティーが存在すると，ACID トランザクションによるリカバリーは実現できない．従って，BPEL のコンペンセーション機能を使用し，図 5 のように実装した．

最終的に，BPEL トランザクションは「受注」+「在庫確保」，「クレジット計上」，「配送手配」の 3 つの UOW から構成され，配送手配アクティビティーが完了するまでは，ユーザーからの「取り消し要求」の受信により例外を発生させ，例外処理からコンペンセーションによるリソース更新の戻し処理を呼び出す設計とした．

4.2 サービス呼び出しトランザクションの設計

次に，サービス呼び出しトランザクションの設計を行う．

① サービス実装の確認

各サービスは，図 4 に示したバインディング（SOAP/HTTP または SOAP/JMS）で呼び出され，要件（d）よりすべてのサービスがデータ更新を伴う．

② サービス呼び出しトランザクションのリカバリー要件の整理

要件（e）より，手動リカバリーを避けるためには，ACID トランザクションによる自動リカバリーのみとなるため，プロセス⇔サービス間のトランザクションを 2PC で行う必要がある．

③ サービスの2PC可否確認

「受注」・「在庫確保」・「クレジット計上」の各サービスは，SOAP/HTTP の Web サービスである．SOAP/HTTP は Web Services-Interoperability（WS-I）Basic Profile にも規定されている標準のバインディングであり，プロセス⇔サービス間での 2PC は，同じく標準仕様である WS-AT を使用することで実現できる．SOAP/HTTP の各 Web サービスは，新規開発のサービスを前提とし，WS-AT に対応可能であると仮定した．

一方，「配送手配」サービスは SOAP/JMS バインディングで呼び出す．キューを介して Web サービスを呼び出すため，宛先キューへのメッセージ送信のトランザクションが保証されればよい．その後，メッセージを受信して Web サービスを実行することはサービス側の責任，また，メッセージの信頼性を確保することはメッセージング側の責任となる．SOAP/JMS の実装はベンダー依存となっており，標準化の動きはあるものの，現時点では異なる製品間での相互運用性はない．従って，プロセス制御とメッセージ送信のトランザクションを 2PC により 1UOW とするには，ベンダー固有の機能を使用することとなる．今回は，配送手配サービスが WebSphere Application Server（WAS）上で稼働しているものとし，WAS・WPS で共通して使用可能な IBM 固有の enableTransactionalOneWay［7］というプロパティーを使用した．

表2. 2PCのためのSCA QoS設定

処理方式	プロセス・リファレンス		サービス・インターフェース
	トランザクションの中断	非同期呼び出し	トランザクションの結合
同期	False	コミット	True
非同期	False	コミット	任意※

※ 設定項目はトランザクション動作に影響しない

④ プロセス⇔サービス間の2PC設計

WPS はサービス呼び出しのフレームワークとして Service Component Architecture（SCA）［8］を採用している．SCA では Quality of Service（QoS）設定により，バインディングによらず，統一的にサービス呼び出しトランザクションを設定することができる．プロセス⇔サービス間での 2PC を有効にするためには，サービスの処理方式（同期／非同期）に応じて**表 2** に示す QoS 設定を行う．サービスから応答を受け取って処理を進める必要がある「受注」・「在庫確保」・「クレジット計上」の 3 つのサービスは同期呼び出しとし，結果を受け取らない「配送手配」サービスは非同期呼び出しとし，おのおの対応する QoS 設定を行った．

バインディングに依存しない共通の QoS 設定を行った後，前ステップで確認したバインディングごとの 2PC の設定を行った．

4.3 ウォーク・スルーおよび実機による検証

個別に設計してきた 2 つのトランザクションが組み合

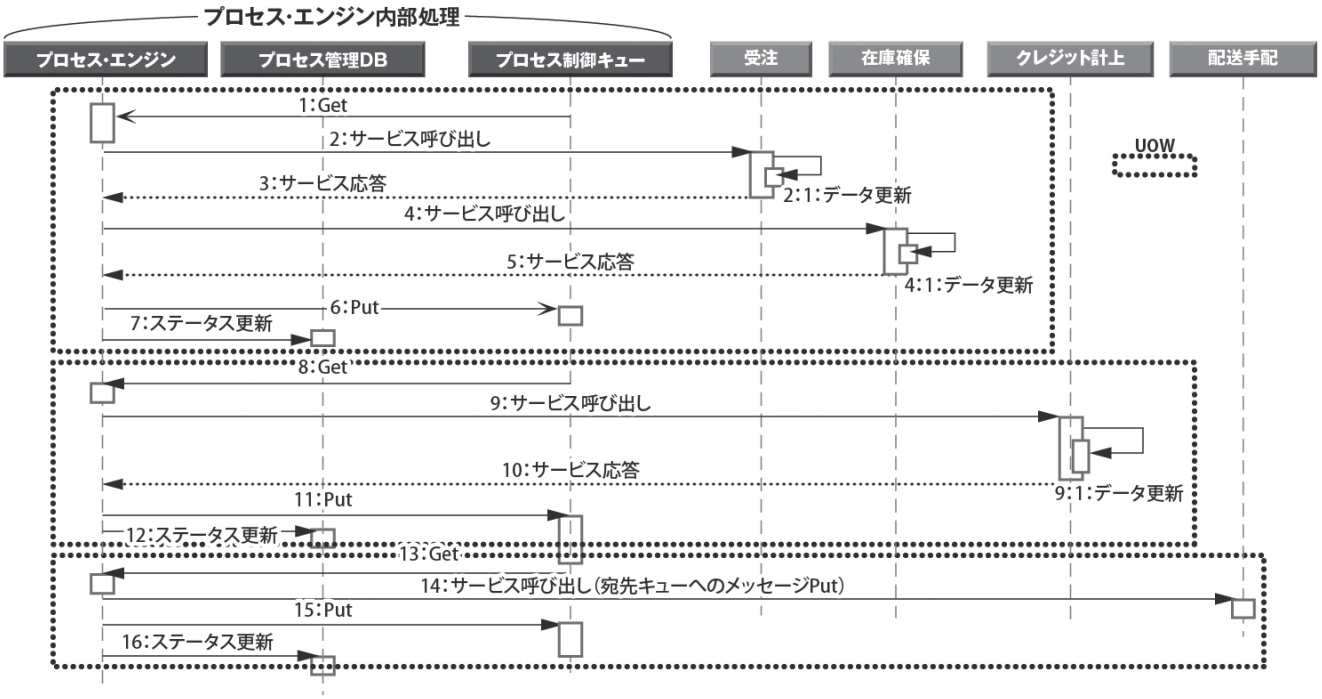

図6. シーケンス図を用いたウォーク・スルー

わされた結果を確認するため，WPS の内部処理を含めたシーケンス図を作成した．まず，そのシーケンス図に BPEL トランザクションの UOW を反映した．次に，各 BPEL トランザクションの UOW と，対応するサービス呼び出しトランザクションの UOW が 1 つになるか，分かれるかを考慮しながら，最終的な設計結果のトランザクション・スコープを図示した（図 6）．

次に，設計したプロセスを WPS にデプロイし，障害テストを実施した．障害テストは，シーケンス図をベースに障害ポイントを網羅的に洗い出し，エンジン自身の障害ケースも含めて実施した．その結果，BPEL トランザクションで設計した 3 つの UOW が，おのおのサービス呼び出しのトランザクションを含んでいることが確認でき，障害時にも要件を満たす設計であることが確認できた．

5. おわりに

本論文では，ビジネス・トランザクション設計において見落とされがちな 3 つの課題を解決するために，①「BPEL トランザクション」と「サービス呼び出しトランザクション」の 2 つの視点で設計を進め，②ビジネス・プロセス・エンジンの内部リソースも含めたシーケンス図を用いてウォーク・スルーを実施し，③シーケンス図を基に障害ポイントを洗い出しエンジン独自実装による挙動を確認するアプローチを提示した．

本アプローチを適用した実機検証の結果，①特に見落とされがちな「サービス呼び出しトランザクション」を意識的に設計項目として取り上げ，漏れのないビジネス・トランザクション設計を実現し，②プロセス制御の内部処理も含めたトランザクション全体の構成を正しく確認し，③ WPS のトランザクション挙動を障害時も含めて網羅的に確認し，ビジネス・トランザクション設計における本アプローチの有用性を示した．

なお，本アプローチを適用した場合でも，一度で意図したトランザクション設計が可能とは限らないため以下の注意が必要である．

SOA サービスの実装パターンは多岐にわたるため，Web サービスの「WS-AT」に相当する各バインディング特有のトランザクション仕様を把握し，必要に応じて検証する必要がある．また，サービス⇔プロセス間での 2PC が不可能なサービスを使用する場合は，データの整合性を保証するためにサービスを二重呼び出し可能にするなど，考慮点が増えることも認識しておく必要がある．上記のようなケースでは，本アプローチを繰り返し，要件を満たすまでトランザクション設計を洗練していく必要がある．

参考文献

［1］ 福場芳正：“B to B システム実現に向けての諸考察：ビジネス ACID ほか，” ProVISION, No.34, pp.42-51（2002）．

［2］ 小宮山光雄：“フレキシブルな企業間アプリケーション統合の実践，” ProVISION, No.50, pp.54-60（2006）．

［3］ 星島洋一，東郷巌：“SOA におけるフロー制御の実装設計手法，” ProVISION, No.51, pp.88-95（2006）．

［4］ Web Services Business Process Execution Language 2.0, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.html（2006.08.23）．

［5］ Web Services Atomic Transaction Specification, http://docs.oasis-open.org/ws-tx/wsat/2006/06（2007.07.12）

［6］ 長妻令子，大崎博靖：“分散サーバー上でのリアルタイム・トランザクション処理，ProVISION, No.55, pp.101-107（2007）．

［7］ Invoking one-way JAX-RPC Web service requests transactionally using the JMS transport, http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/topic/com.ibm.websphere.express.doc/info/exp/ae/twbs_jmsonewayreq.html（2008.06.03）．

［8］ WebSphere Process Server V6 解体新書 Service Component Architecture ガイド，http://www.ibm.com/jp/software/websphere/developer/wbi/diamond/01.html（2006.01.18）．

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
ソフトウェア・テクノロジー・センター
ビジネス・インテグレーション

北尾 寧宏 Yasuhiro Kitao

［プロフィール］
1998 年，日本アイ・ビー・エム　システムズ・エンジニアリング株式会社に入社．IT スペシャリストとして WebSphere Application Server を中心とした多数の Web システムの構築に携わる．その後，SOA 関連の WebSphere 製品の技術サポートを行い，ビジネス・プロセスを制御するアプリケーション (BPEL) の設計・構築に従事する．

日本アイ・ビー・エム
システムズ・エンジニアリング株式会社
ソフトウェア・テクノロジー・センター
ビジネス・インテグレーション

亀田 綾 Aya Kameda

［プロフィール］
WebSphere Business Process Management（WBPM）製品群を担当する IT スペシャリスト．WBPM 製品群の研修開発やプロジェクト参画を通じ，SOA や BPM（Business Process Management）の推進活動を行っている．