ローレベル BPMN パターン

- BPEL 生成可能な BPMN ダイアグラム -

2006 年 6 月

特定非営利活動法人 UML モデリング推進協議会

BPMN 研究会

目次

1　はじめに

UMTP（UML モデリング推進協議会）は、情報技術分野におけるモデリング技術の普及と、策定されたモデルの共有促進を目的として活動する NPO（特定非営利活動法人）である。

その活動は、モデリング技能の認定サービスを、中国・韓国を含む国内外に提供しながら、モデリング技術者の連携を容易とすることと、それを通じて情報技術分野のビジネス連携やソフトウェア連携に貢献することに主眼を置いている。

今日、UML（Unified Modeling Language）が国際規格（ISO/IEC19501）として規格化され、情報交換、あるいはソフトウェア開発などで広く利用されている。UMTP は、UML に力点を置きながらも、モデリング技術全般について、調査・研究を鋭意、展開している。

一方、Web サービスなど新たな Web 技術は、企業内外に存在するソフトウェアを、サービスと見なして連携・統合させることを可能としており、ソフトウェア構築のためのモデリング技術だけでなく、異なるビジネス間の連携を可能とするモデリング技術が求められている。

BPMN（Business Process Modeling Notation）は、そのような背景から、BPMI. Org（OMG）によって標準化が進められているモデル表記法である。特に、Web サービスの統合とその実装を容易とするために、ビジネスプロセスのモデル化と、そのモデルを BPEL（Business Process Execution Language）などによる実装仕様へマッピングさせることを目的としている。

2004 年に発足した UMTP の BPMN 研究会では、BPMN の調査とその適用に関する活動を展開した後、2005 年度から、外部協議会である XML コンソーシアムと連携しながら、BPMN による上位モデルの実装へのシームレスなマッピング容易化をテーマとして活動している

その一環として、BPMN による実装独立の階層と、BPEL などによる実装依存の階層との間を対応付けるモデルパンターンの共通化を試みている。

ビジネス・プロセス・モデルのシームレスな実装ために、大きく、3 つのレベルのモデル階層を定義しながら研究を展開している。2005 年度は、その内のローレベル・プロセス・モデルのパターン化とその利用ガイドを策定した。

本報告書は、2005 年度の成果として、ローレベル・プロセス・パターンについての研究成果を述べるものである。UMTP 会員、XML コンソーシアム会員だけでなく、広く、内外利用者の適用評価を期待すると共に、これらのパターンが、広く共通のものとして受け容れられることを願うものである。

平成 18 年 4 月
UML モデリング推進協議会（UMTP）
副会長　堀内　一

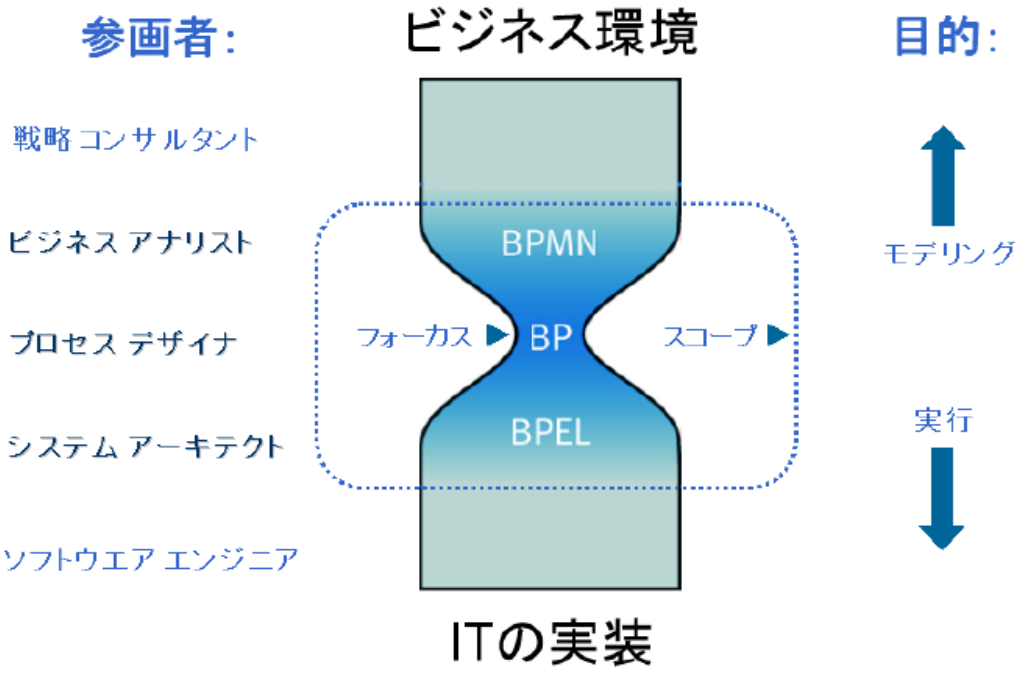
2　ビジネスプロセス設計のプロセス

ビジネスプロセスのモデリングをするに当たり、目的とする成果物の姿や作業手順が決められている方が作業効率の点でも成果物の品質の点でも望ましい。

ここではモデリングの成果物および作業手順に関する事柄を述べる。

2．1　成果物について

BPMI.org では下図のようにビジネスプロセスのモデリングに関する参画者とその目的を定義している。



図：BPMI. orgの砂時計

このうち、ビジネスアナリスト・プロセスデザイナ・システムアーキテクトの三者が BPMN によるモデリングを行うと考えられる。本書では BPMI.org の定義に従い、BPMN によるビジネスプロセスモデルを三層の階層を持つものとして解説する。

モデラー（モデリング実施者）と成果物との関係は、次のとおり。

- ビジネスアナリスト …… ハイレベルビジネスプロセスモデル
- プロセスデザイナ …… ミドルレベルビジネスプロセスモデル
- システムアーキテクト … ローレベルビジネスプロセスモデル

BPMN は、ビジネスプロセスに適用されるモデリング概念だけをサポートしており、いわゆ

る「ハイレベルモデル（ハイレベルビジネスプロセスモデルとは異なる）」はサポート対象外とされる。

BPMN でサポートしていないモデルの例を次に示す。

- 戦略
- ビジネス ルール
- 組織構成およびリソース
- 業務内容のブレークダウン
- データおよび情報モデル

また、BPMN が、データ（メッセージ）の流れや、アクティビティへのデータの成果物の関連付けを示していたとしても、それはデータフロー図ではない。

なお、他の参画者については、戦略コンサルタントはコンセプトレベルモデルと呼ばれるもの（例：バランストスコアカード、バリューチェーン 等）が、ソフトウェアエンジニアはUMLを用いたサービスモジュール単位の各種モデルが、それぞれの成果物として考えられる。

成果物には明確な利用用途があり、これを「モデルの目的」と表現する。また、3種類の成果物は階層的な関係にあり、それぞれのモデルで表現されるビジネスプロセスの粗さ（これを「粒度」と呼ぶ）が決められている必要がある。

成果物とモデルの目的との関係は次のとおり。

- **ハイレベルビジネスプロセスモデル**
  目的：企業レベル・企業間レベルでのビジネスプロセスの可視化、経営陣によるビジネスプロセスの大局把握。

- **ミドルレベルビジネスプロセスモデル**
  目的：ビジネスプロセスの可視化と仮説検証、モニタリング

- **ローレベルビジネスプロセスモデル**
  目的：最も詳細なビジネスプロセスの可視化。BPEL4WS の設計およびレビュー、BPEL4WS の生成

成果物とモデルの粒度との関係は次のとおり。

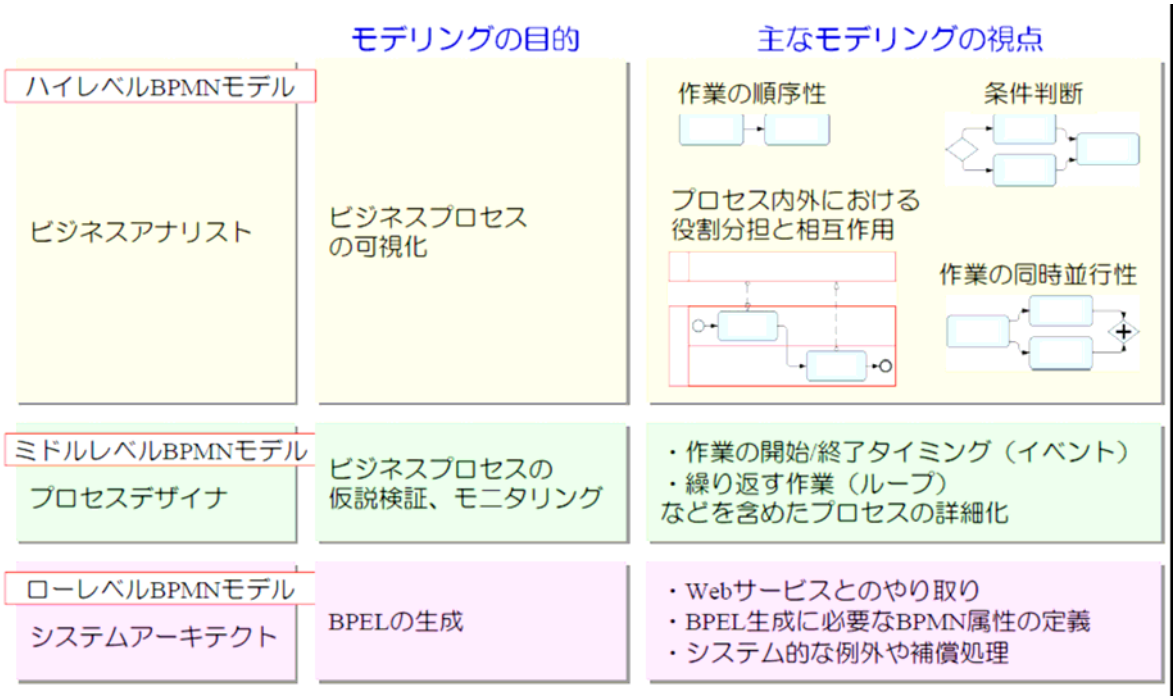
- **ハイレベルビジネスプロセスモデル**
  企業における最も大きな単位のビジネスプロセスや企業間プロセス連携を表現する。

全てのアクティビティは折りたたまれたサブプロセスである。
例：販売管理、顧客管理、在庫管理

- **ミドルレベルビジネスプロセスモデル**
  多くの場合、複数の部署や担当者にまたがるプロセスであり、時系列的に複数のサブプロセスが進行することにより完結する。
  折りたたまれたサブプロセスとして表現されたアクティビティが多く使われ、タスクとして表現され、それらのタスクを用いてビジネスプロセスが表現されているもの。
- 例：受注、出荷、売上、入金、返品

- **ローレベルビジネスプロセスモデル**
  多くの場合、表現されるビジネスプロセスは、一人もしくは単一の部署での一連の作業であり、トランザクション処理の粒度に一致させる場合が多い。
  本書で定義する「ローレベルパターン」が用いられる他にはサブプロセスは存在せず、BPMNで表記されたイベントやタスクが、BPEL4WS の命令にマッピングされる。
  例：売上登録をする、在庫引当入力をする、航空券を予約する

以上の関係をまとめると、以下の表のようになる。



２．２　モデリングの作業手順

ビジネスプロセスのモデリングを行う場合には、段階的詳細化を行う「トップダウンアプローチ」と、段階的概念化を行う「ボトムアップアプローチ」がある。

通常はラフスケッチとしてハイレベルのプロセスモデルを作成し、順を追ってミドルレベル、ローレベルと詳細化していく「トップダウンアプローチ」が用いられる。

詳細なプロセスをモデリングし部品化するといった本書の取り組みは「ボトムアップアプローチ」の一例だが、通常は、ビジネスの仕組みを変更することにより他の業務・部署・外部組織・システム全体への影響がどうであるかを検証する目的で「ボトムアップアプローチ」が用いられる。

■　ローレベル BPMN パターンを用いた作業手順

ローレベルBPMNモデルとして、より詳細な記述が必要な範囲を決定する。ミドルレベルのBPMN モデルが存在する場合は、「タスク」や「折りたたまれたサブプロセス」がその範囲に含まれるはずである。

ローレベルBPMNモデルは設計と実装をシームレスにつなぐ事を目的とするので、BPMNで記述したビジネスプロセス図からプロセス実行言語である BPEL4WS が生成されることを強く意識したモデリングが行われる。

本書で述べられている「ローレベル BPMN パターン」は BPEL4WS の生成が検証されているため、「部品（折りたたまれたサブプロセス）」として利用することにより、モデリング作業の生産性と品質の向上が期待できる。

3　BPMN 概説

3．1　BPMN とは

BPMN はビジネスプロセス図（Business Process Diagram。以下，BP 図）によって業務の手順を分かりやすく記述するための表記ルールである。ビジネスプロセスの上流設計を行うビジネスアナリストなどを対象とし、IT 技術者でなくても容易に理解できるグラフィカルな表記法を採用している。本節では、BPMN が開発された目的と、その特徴について述べる。

3．1．1　開発の目的

近年、M&A やアウトソーシング・ビジネスの拡大によって、企業を取り巻く環境は急速に変化している。そこでは、経営環境の変化に応じてビジネスプロセスを柔軟に変更すると共に、IT システムへ迅速に実装したいというニーズが高まっている。一方で、現状では、ビジネスプロセスと IT システムとのタイムリーな連携を実現することは容易ではない。これは、上流設計を行うビジネスアナリストと、ビジネスアナリストが定義したビジネスプロセスを詳細化・実装する IT 技術者との間で、専門知識のギャップが生じており、円滑なコミュニケーションが困難であるという要因が大きく影響している。さらに、システム導入後においても、システム管理者が運用、モニタリング、メンテナンスを円滑に実施するためには、上流設計および実装に関する知識を共有できる環境にあることが求められる。そこで、このようなビジネスプロセスに関わる人同士のコミュニケーション上のギャップを埋めるために、IT 技術者でなくても理解できるシンプルで分かりやすい表記ルールが求められるようになった。

また、ワークフローシステムなどのビジネスプロセス管理を実装するツールでは、複数のベンダが、プラットフォームに依存した固有の表記法を利用している。SOA（Service Oriented Architecture）などの登場により、自社内外の組織の枠を超えたビジネスプロセス連携に対する必要性が高まりつつあり、ベンダやプラットフォームに依存しない共通的な表記ルールが求められるようになった。

以上のような経緯から、IT 技術者でなくても理解しやすく、また、プラットフォームに依存しない標準的なビジネスプロセス表記ルールの開発を目的として、BPMN が策定された。

3．1．2　BPMN の特徴

BPMN の主な特徴を以下に示す。

1) シンプルな表記法で豊富な表現力を実現

BPMN は、シンプルで分かりやすい表記法を有する一方で、ビジネスプロセスの持つ多様性を十分に表現することができる。これら相反する 2 つの要件を同時に実現するため、グラフィカルな図形要素を体系化すると共に、基本要素とその詳細であるバリエーションによる構成を採用している。基本要素は直感的に意味を判断しやすい図形を採用してシンプルに記述でき、基本要素にバリエーションを追加することで基本的な見た目を変更せずに詳

細化が図れるようにしている（基本要素については、2.2 節参照）。
また、BPMN では、システム間の処理だけでなく人間対人間の業務フローの記述にも対応しているため、業務全体のビジネスプロセスの可視化およびビジネスアナリストと IT 技術者間などのコミュニケーションに利用することができる。

2) 実行言語（BPEL）の生成による迅速な実装が可能
BPMN は、実行言語である Business Process Execution Language For Web Services（以下、BPEL と記述、http://www.oasis-open.org 参照）の生成が可能である。BPEL は、定義されたプロセスに従って制御を行うオーケストレーションを記述する標準実行言語である。BPMN では、BPEL とのマッピング・ルールが規定されており、BPMN から BPEL を自動生成するための変換ツールなどが開発されている。このため、業務側で発生したプロセスの変更を、迅速かつ的確に IT システムへ実装することが可能となる。

３．２　BPMN 表記法

BPMN は、BP 図の表記ルールおよびビジネスの意味的要素であるビジネスセマンティックを定義するものである。BPMN の表記法については、BPMI が発行した Introduction BPMN[1]などで規定されており、本節ではこれら規定に基づいて、BPMN に初めて接する方を対象として基本的な BP 図の図形表記とその意味について述べる。

BPMN では、一般のビジネスモデラーがなじみ易いシンプルな図形表記を採用している。また、それら図形表記を体系立てて整理すると共に、バリエーションを持たせることにより、シンプルでありながら豊富な表現力を実現している。

BPMN における BP 図は、以下の4つの基本要素によって構成される。

- フローオブジェクト（Flow Objects）
  ビジネスプロセスの処理の振る舞いを記述する要素
- 接続オブジェクト（Connecting Objects）
  フローオブジェクト同士を接続するための要素
- スイムレーン（Swimlanes）
  プロセスの関係者を明確化するための要素
- 成果物（Artifacts）
  BP 図の理解を助ける補足情報を記述するための要素

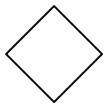
以降、各基本要素について詳述する。

３．２．１　フローオブジェクト（Flow Objects）

フローオブジェクトは、ビジネスプロセスにおける処理の振る舞い記述するための図形要素である。フローオブジェクトは、3つの基本要素で構成されており、種類が少なく、かつ、直感的に

理解しやすい図形を採用している。そのため、ビジネスモデラーは多くの種類の図形を覚え、それらを使い分ける必要がなく、比較的短時間でBPMNをマスターすることが可能である。BPMNの3つのフローオブジェクトを下表に示す。

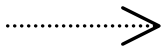
表3-1. フローオブジェクトの基本要素

項番	要素	表記法	説明
1	アクティビティ		ビジネスプロセスで実行される個々の処理を表す。アクティビティは、単一の原子または複合物の非原子を表現することができる。また、タスクだけでなく、サブプロセスを表現することも可能である。ビジネスプロセスは、複数のアクティビティを有し、それらの処理順序や条件などを明確化したものと定義できる。
2	イベント		ビジネスプロセスの中で発生する開始、終了などの事象を表す。ビジネスプロセスの実行を制御するために使用するもので、プロセスの流れに影響を及ぼす。イベントは、開始、中間、終了の3つのタイプがある(左図参照)。
3	ゲートウェイ		シーケンスフローの分岐および収束を表す。ビジネスプロセスのフローを複数に分流、または複数のフローを合流させたりする場合に使用するもので、パスの分岐、併合、経路結合を制御する。

３．２．２　接続オブジェクト（Connecting Objects）

接続オブジェクトは、複数のフローオブジェクト同士を接続してビジネスプロセスを表現する。フローオブジェクトは、ビジネスプロセスの基本構造を表しており、接続オブジェクトによって互いに接続されることで、ひとつのダイアグラムを構成する。BPMN の 3 つの接続オブジェクトを下表に示す。

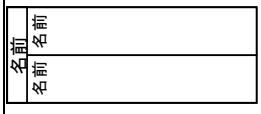
表3-2. 接続オブジェクトの基本要素

項番	要素	表記法	説明
1	シーケンスフロー		アクティビティが実行される順序を表す。シーケンスフローは、あくまで処理の流れを表すものであり、データの流れを表すものではない。
2	メッセージフロー		2つのプール間におけるメッセージの送受信を表す。異なるプール間におけるオブジェクト同士、オブジェクトとプール、または、プール同士を接続する場合に使用する。
3	関連		成果物オブジェクト(データ、グループ、注釈)とフローオブジェクトや接続オブジェクトとの関連付けを表す。各アクティビティに関するインプットとアウトプットを記述ために使用する。

３．２．３　スイムレーン（Swim lanes）

スイムレーンは、プロセスの関係者を明確化するために、フローオブジェクトを四角形の枠で囲んで表現する。スイムレーンによって、アクティビティを視覚的なカテゴリに分離することで、異なる機能または責任を図示することが可能となる。BPMN では下表の 2 つの図形表記を用いる。

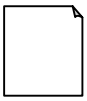
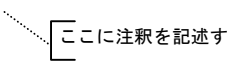
表3-3. スイムレーンオブジェクトの基本要素

項番	要素	表記法	説明
1	プール		プロセスの関係者を表現する。プロセスの関係者には、企業などのビジネスエンティティやプロバイダーなどのビジネスロールが存在する。プール内に具体的なプロセス記述のない抽象プロセス(ブラックボックス)とすることも可能である。
2	レーン		プールをシステムの役割や部署などによって分割する。アクティビティの整理・分類および役割を明示する際に使用する。

３．２．４　成果物（Artifacts）

成果物は、BP 図を理解し易くすることを目的とした、ドキュメント追記のためのオブジェクトである。BPMN では、モデラーとモデリングツールに柔軟性を与えるというコンセプトに則して、基本表記を柔軟に拡張し、業界固有のモデリングニーズに適した要求仕様を追記できるように設計されている。そのため、モデル化する対象業界および対象業務に合わせて、さまざまな成果物を要求仕様としてダイアグラムに追加することが可能である。BPMN 仕様バージョン 1.0 で規定している3種を下表に示す。

表3-4. 成果物オブジェクトの基本要素

項番	要素	表記法	説明
1	データ オブジェクト		ビジネスプロセスに関連するデータを表す。アクティビティの入出力データやアクティビティ間を流れるデータを表現する場合に使用する。ただし、あくまでも補足情報であるため、フローの制御に影響を与えてはならない。
2	グループ		類似目的のアクティビティなどをグルーピングする。グループ分けによって、ドキュメンテーションや分析を行う場合に使用する。また、グループは、プールを超えて表される分散トランザクションのアクティビティを識別するためにも使用される。データオブジェクトと同様に、シーケンスフローに影響を与えてはならない。
3	注釈		BP図の読み手に対して、補足としてテキスト情報を記述する際に使用する。

4　BPEL 概説

4．1　モデル駆動開発と BPMN，BPEL，WSDL

経営におけるITの重要性の認識が高まり、事業戦略とシステム化計画はますます密接に関連するようになってきている。ビジネス環境の変化が激しい近年、変化に応じて柔軟にビジネスプロセスを変更したいと考える企業が増え、柔軟性に対応できる情報システム構造を持つことが課題となっている。このような状況で、ITを含む業務改善/改革のマネジメント手法であるBPMや、ネットワーク上の分散システムを一元的なアーキテクチャ・スタイルにより統合する仮想化概念であるSOAが注目されている。

BPMとSOAの実現化を支えるエンジニアリング・アプローチとして、モデル駆動開発がある。現実の事業や業務において着目する論点に絞り、単純化したビジネスプロセスの分析モデルを作成し、業務の改善策を決定する。それらのモデルを詳細化して機能やワークフローを厳密に定義した設計モデルに変換し、さらに実装言語へとつなげる。近年、複数のWebサービスを手順に従い呼び出すワークフローを形成するための記述言語であるBPELが事実上の標準となり、ワークフローモデルの標準的な表記法となるBPMNが誕生したことで、ビジネスプロセス・モデルを軸にしたモデル駆動開発が実現可能になってきている。

BPMNの仕様においてBPMNとBPELの要素の対応が関係付けられており、BPELは呼び出すWebサービスのインタフェース記述のWSDL（Web Services Description Language, http://www.w3.org/tr/wsdlを参照）を仕様に包含している。これにより、BPMNによるビジネスプロセス・モデルの作成からWebサービスとワークフローの定義まで、一貫性のある開発支援環境を構築することができる。ビジネスプロセスを軸にしたモデル駆動開発のプロセスは図4.1のようになる。

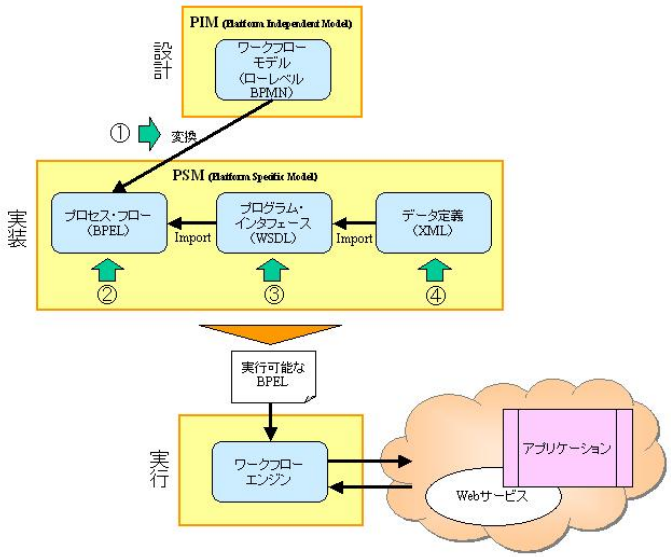


図 4.1　BPMN を起点とするモデル駆動開発イメージ

モデル駆動開発では、CIM（Computation Independent Model）、PIM（Platform Independent Model）、PSM(Platform Specific Model)の順に三つの異なる観点から分析・設計を行う。

ビジネスプロセスを軸にしたモデル駆動開発のCIM分析では、システムに対する要求を獲得するためにシステム化対象のドメインのビジネスプロセスを分析する。この段階のビジネスプロセス分析は業務課題の解決を意図したシステムを意識しないモデルを作成する。このモデルにより業務の流れを大枠で捉えた上で、システム境界を決定し他システムとの関連を考察する。

PIM設計では、CIMのビジネスプロセス・モデルからシステム化対象範囲のプロセスを抽出し、実装に適した大きさのワークフローに分解し処理内容を詳細化する。詳細化の作業ではシステムの構造を意識して、メッセージ交換規約、例外処理を含むワークフローを厳密に定義する。

PSM作成では、PIMで設計したワークフローのモデルをBPELなどワークフロー・エンジンが処理可能な記述言語に変換する。変換したワークフローを雛形として、変数や内部処理の実装記述を行い、また、呼び出すWebサービスのインタフェース宣言の修正や追加を行うことで完成する。

本書が対象とするローレベルBPMNモデルは実行可能なBPELへ変換可能なワークフローモデルであり、モデル駆動開発におけるPIMに相当する。そして、これをベースに生成されるBPELとWSDL、XMLはPSMに相当する。現在、ローレベルBPMNモデルからBPELへの自動変換が可能な各種のツールがソフトウェア・ベンダーから提供されている（①）。変換されたBPELを実行可能なものにするために、変換後のBPELに機能拡充のための要素の追加（②）や、関連する既存Webサービスのインタラクションのために WSDLを参照（Import）することが必要となる。Webサービスを新規に開発するのであれば、Webサービスのインタフェースを設計しWSDLを定義する（③）。Webサービスの設計において交換するメッセージに既存のデータ構造を包含する場合、そのデータ構造を定義するXML SchemaをWSDLで参照（Import）する（④）。本書では、これらの一連の動きによりBPMNを起点としたモデル駆動開発を行うことを想定している。

４．２　実行可能な BPEL の利用モデル例

BPMNで記述したローレベルBPMNモデルを実行可能なBPELに変換するアプローチは万能ではなく、いくつかの有効な利用モデルが考えられる。以下に有効な利用モデルとして2つの例を紹介する。

４．２．１　B2B コラボレーションプロセス

2つ以上のプール間の相互作用（メッセージ交換）をローレベルBPMNで表現しBPELに変換するモデルである。

このモデルの場合、各プールにおけるメッセージ送受信処理が実行可能なBPELの対象領域となる（図4.2）。BPEL変換のためには、プール内のプロセスを分解し実装を意識したローレベルBPMNモデルとして詳細化する（図4.3）。

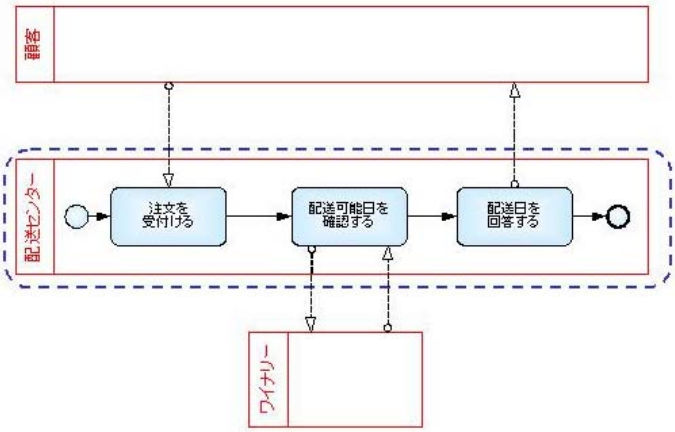


図4.2　B2B コラボレーションプロセス(ミドルレベル BPMN モデル:CIM レベル)

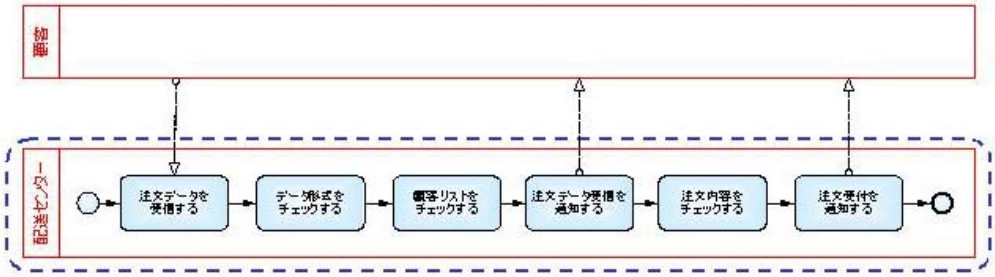


図 4.3　B2B コラボレーションプロセス(ローレベル BPMN モデル:PIM レベル)

４．２．２　単一事業組織内のビジネスプロセス連携の自動化

単一事業組織内の部署間のビジネスプロセス連携をBPELにより自動化するモデルである(図4.4)。

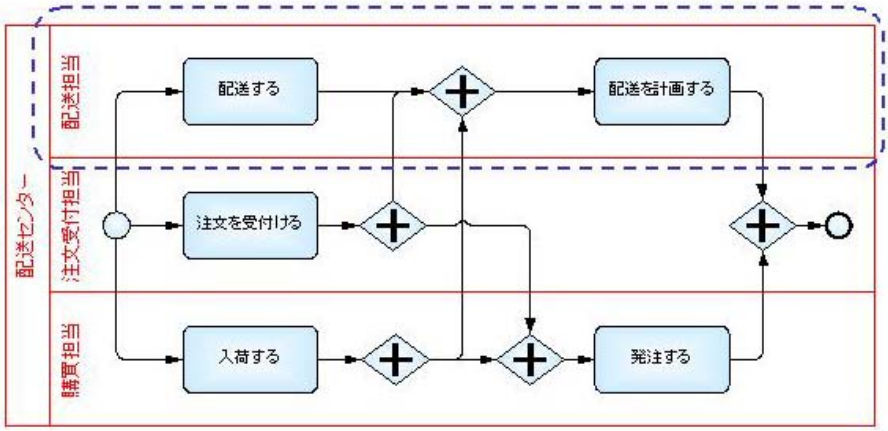


図 4.4　内部ビジネスプロセス(CIM)

このモデルの場合、BPEL によるプロセスの自動化は 2 つのの実装方式がある。一つは図 4.4 のモデル全体を単一の処理系（ワークフロー・エンジン）で実現する集中方式と、もう一つはレーン毎に処理系を分けて協調する分散方式である。集中方式では、アプリケーションのファンクションレベルにまで詳細化したローレベル BPMN モデルと BPEL を一元管理する。業務統制を行うことが容易でシステムの性能や品質を高めることが比較的容易である。その半面、プールやレーンをまたぐフローを管理・制御することが複雑になり、また、要求の変更を局所化することが難しい。分散方式では、レーン毎に業務要求に即したきめ細かなフローの実現が容易で、処理フローもレーン横断的なものに比べ単純である。その反面、事業内を横断する統合的な業務の監視は別途仕組みが必要になり、また、分散するワークフロー間の相互運用性の維持が負荷になる。図 4.5 は、レーン毎にワークフローを分離した分散方式を示している。

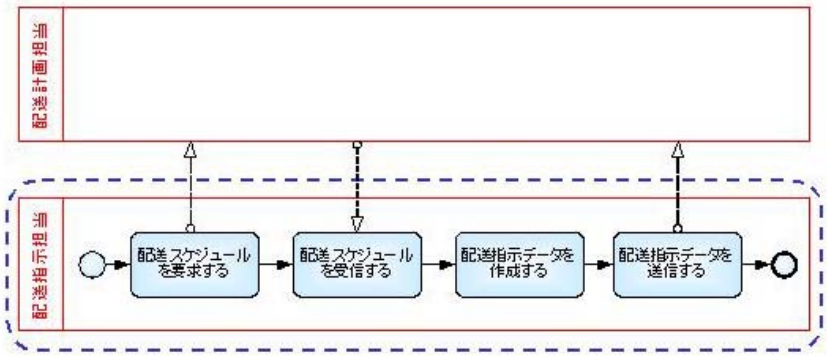


図 4.5 内部ビジネスプロセス（PIM）

４．３ 言語構造

４．３．１ プログラム・インタフェース -WSDL の言語構造-

BPELの言語構造において、Webサービスのインタフェースを記述するWSDL（Web Services Description Language, http://www.w3.org/tr/wsdlを参照）を用いてサービス間のインタラクションを定義する。そのため、WSDLの言語構造を把握しておくことはBPELを理解するうえで欠かせない。

WSDLの言語構造は、そのWebサービスが提供するサービスがやりとりするメッセージの形式や構造を定義する部分（インタフェース）と、サービスへのアクセス方法とアドレスを定義する部分（実装）とに大きく二分される。このようにインタフェースと実装が分離されていることにより、サービスの実装部分のみが異なる場合でも、サービスのインタフェース定義は再利用が可能となっている。

インタフェースを定義する部分は*types*要素、*messages*要素、*portType*要素により定義される。type要素は、データ項目やデータ項目を組み合わせたデータ構造の定義を行う。*messages*要素は、type要素で定義したデータ項目とデータ構造を用い順序を考慮した組み合わせにより入出力メッセージを定義する。*portType*要素は、message要素で定義したメッセージを組み合わせ論理的なイ

ンタフェースを定義する。サービスへのアクセス方法とアドレスを定義する部分では、サービスが「どのように」、「どこで」提供されるかを*binding*要素（サービスとのアクセス方法）、*service*要素（サービスのアドレス）で定義する。binding要素は、portTypeで定義した論理的なインタフェースにトランスポート・プロトコルなど考慮した実装のためのインタラクション方法を指定する。service要素は、アクセスポイントのURI指定など実際に提供するWebサービスを表明する。

WSDLは*definitions*要素で表現される定義部に、データ構造（*types*要素）、メッセージ（*message*要素）、インタフェース（*portType*要素）、アクセス方法（*binding*要素）、アドレス（*service*要素）の定義をそれぞれ記述する。図4.6にWSDLの言語構造のイメージを示す。

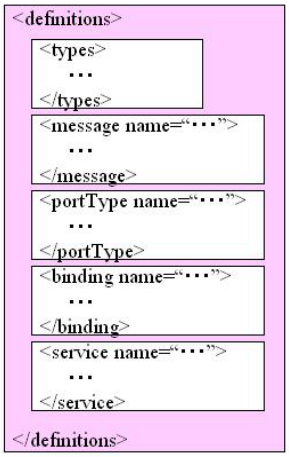


図 4.6 WSDL の言語構造

（出典:『Web サービスプラットフォームアーキテクチャ』エスアイビーアクセス）

４．３．２　プロセス・フロー　-BPEL の言語構造-

BPELはWSDLで定義されたWebサービスのインタフェースを連携させることで、複雑なプロセスを実現することができる。BPELで定義されたプロセスは、それ自体が複数のWebサービスを連携させた複合的なWebサービスとなる。

BPELプロセスは、外部パートナーとの関係、プロセスデータの宣言、様々な目的のためのハンドラ、実行対象となるアクティビティ（ここでいうアクティビティはBPMNにおけるアクティビティとは異なる）という要素から構成される。BPELの言語構造を図4.7に示す。

図 4.7 BPEL の言語構造

プロセスの実行に必要な外部パートナーとの関係は、*partnerLinks*で記述する。アクティビティ間のメッセージ受け渡しやパートナーとビジネスプロセスの間で送受信されるに使用する変数やなどのプロセスで使用されるデータは、*variables*で定義する。予期せぬ問題を取り扱うために、様々な目的のためのハンドラを定義することが出来る(*faultHandlers*、*compensateionHandlers*、*eventHandlers*)。

実行対象となるアクティビティには、構造を持つものと持たない(基本的な)ものがある。基本的なアクティビティはWebサービスとのインタラクション(*invoke*、*receive*、*reply*)やデータ処理のための特別なアクティビティ(*assign*)、その他の様々なタイプの動作を記述するアクティビティ(*pick*、*empty*、*wait*、*terminate*、*throw*、*compensation*)である。BPELではこれらの基本的なアクティビティを構造化されたアクティビティ(*sequence*、*flow*、*switch*、*while*)を使用して結合することにより、より複雑なビジネス・ロジックを作ることができる。

5　ローレベル BPMN パターン

5．1　パターンの利用目的

ローレベル BPMN パターンは、以下の 3 つの目的での利用を前提としている。

（目的１）パターンの組み合わせにより BPEL 生成可能な BPMN モデルを設計する

後述の「6．1．BPEL にマッピングされない BPMN 要素」で示すように、全ての BPMN ダイアグラムから BPEL を生成できるとは限らない。また、BPEL にマッピングされる BPMN 要素であっても、使い方によっては、マッピングされるケース、マッピングされないケースがある。

したがって、ローレベルプロセスモデルを設計するためには、BPMN による表記ルールを理解しただけでは不充分であり、BPEL へのマッピングも把握しなければならない。BPMN 仕様書には BPEL へのマッピング仕様が全て記述されているものの、表形式での文章記述が主体であり、全てを理解するためにはそれなりの労力がかかってしまう。

マッピング仕様の全てを把握しなくても、パターンの組み合わせにより BPEL 生成可能な BPMN モデルが設計できる。それがローレベル BPMN パターンの最も重要な利用目的である。

（目的２）BPMN ダイアグラムの表記における統一化を図る

下図の例のように同じ BPEL を生成する等価な表記が存在している。

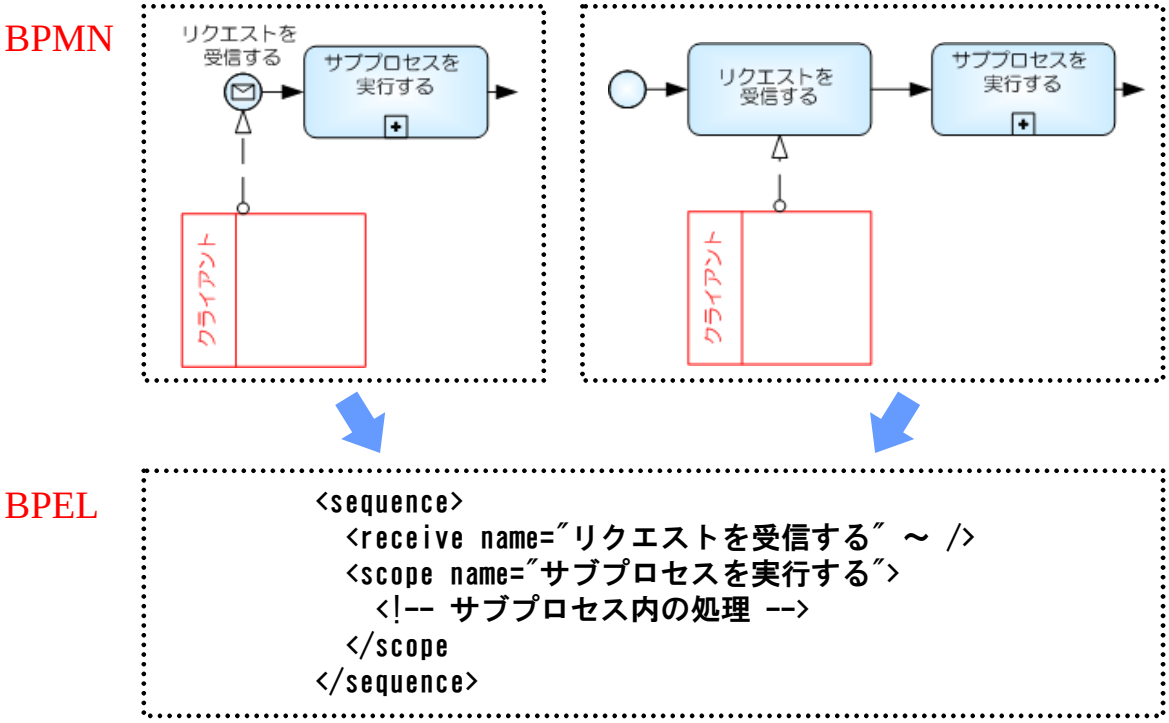


図 5.1　同じ BPEL にマッピングされる BPMN ダイアグラム

これら等価な表記が存在する理由としては、ハイレベルプロセスモデルからローレベルプロセスモデルまでといった広範囲での利用ができるように「シンプルな表記法」と「複雑な振る舞いで

も表現できる表記法」の両方を用意していること、業務イベントに着目したモデリング手法など様々なプロセスモデリング手法にも対応できるようにしていることなどが挙げられる。

表記の統一化を促進するものとして、モデリング方法論やモデリング基準（ガイドライン）の整備、サンプルモデルの充実などが挙げられるが、パターンの組み合わせによるモデリングも大きな効果が期待できる。そこで、パターン集は、できる限り表記の統一化を図り、パターンと等価な表記がある場合には特記事項欄にその旨を記載するようにしている。

（目的３）BPMN と BPEL を習得する補助テキストとして利用する

BPMN を習得しようとしている人にとっては、パターンを表すダイアグラムとシナリオ（時系列に箇条書きで記述したもの）、プロセス実行言語である BPEL へのマッピング結果、パターン適用例として記述したサンプルモデルなどを通じ、BPMN のダイアグラムが表す振る舞いを正確に理解することができる。

また、BPEL ではどのような機能が実現できるのかなど、BPEL の概要を知りたいと考えている人にとっては、ビジュアルでわかりやすい BPMN のダイアグラムを通じて BPEL 構文の骨格や機能の実現範囲を容易に理解することができる。

５．２　パターンを分類するカテゴリ

ローレベル BPMN パターンは、実行順序、経路選択、ループなど業務フローの制御パターンを表す「フロー制御」、クライアントや Web サービスとのメッセージ交換パターンを表す「サービス連携」といった 2 つのカテゴリに分類する。さらに、フロー制御は 4 つのサブカテゴリ、サービス連携は2 つのサブカテゴリに分類する。の参照性と理解の容易性の観点から、その使用目的、振る舞いの特徴などを考慮して、以下の 8 つのカテゴリに分類する。

以下、カテゴリによる分類視点について、サブカテゴリ毎に記述する。

【フロー制御に関するサブカテゴリ】

（１）基本制御

あるアクティビティが完了したら、次にどのアクティビティを開始するといった実行順序を表す基本的な制御パターン。あらかじめ定められた静的な実行順序を表すパターンであり、条件判定によって次に実行するアクティビティやアクティビティの繰り返し回数が決まるなど、動的にアクティビティの実行が決定するパターンを除く。

（２）経路選択

複数に分流するフローのうち、条件判定によってどのフローに流れるのかを決定するといった経路選択を表す制御パターン。

（３）ループ制御

タスク、サブプロセス、またはそれらを含むあらかじめ決められた振る舞いを繰り返し実行するループ制御を表すパターン。

（４）例外処理

「例外的なメッセージを受け取った」、「タイムアウトが発生した」など、例外の発生により正常時のフローとは異なる例外フローに流れを切り換えるといった例外処理を表すパターン。

【サービス連携に関するサブカテゴリ】

（５）サービス要求受信

パターンを利用して設計する対象プロセス自体を Web サービスとして公開し、その Web サービスを利用するクライアントからのリクエストの受信し、そのリクエストに対するレスポンスをクライアントに送信するといったメッセージ交換を表すパターン。

（６）サービス呼び出し

パターンを利用して設計する対象プロセスが Web サービスを実行制御するといったオーケストレーションに関するメッセージ交換パターン。

５．３　パターン一覧

表 5.1　カテゴリ一覧

カテゴリ名		No	パターン名	頁
フロー制御	基本制御	1	逐次実行	22
		2	並列実行（AND）	25
		3	タイマーイベント待ち	28
		4	プロセス即時終了	30
	経路選択	5	排他選択（XOR）	33
		6	包含選択（OR）	37
	ループ制御	7	標準ループ	42
		8	マルチインスタンスループ	46
	例外処理	9	メッセージ例外	50
		10	タイマー例外	54
		11	エラー処理	58
		12	補償処理	62
サービス連携	サービス要求受信	13	同期型受信応答	66
		14	非同期型受信応答	69
		15	非同期型受信	72
		16	タイマー通知送信	75
	サービス呼出	17	同期型要求	78
		18	非同期型要求	80
		19	複数応答受信（順次）	82
		20	複数応答受信（順不同）	84
		21	複数応答受信（選択）	87
		22	複数応答受信（任意応答）	91

５．４　パターン集

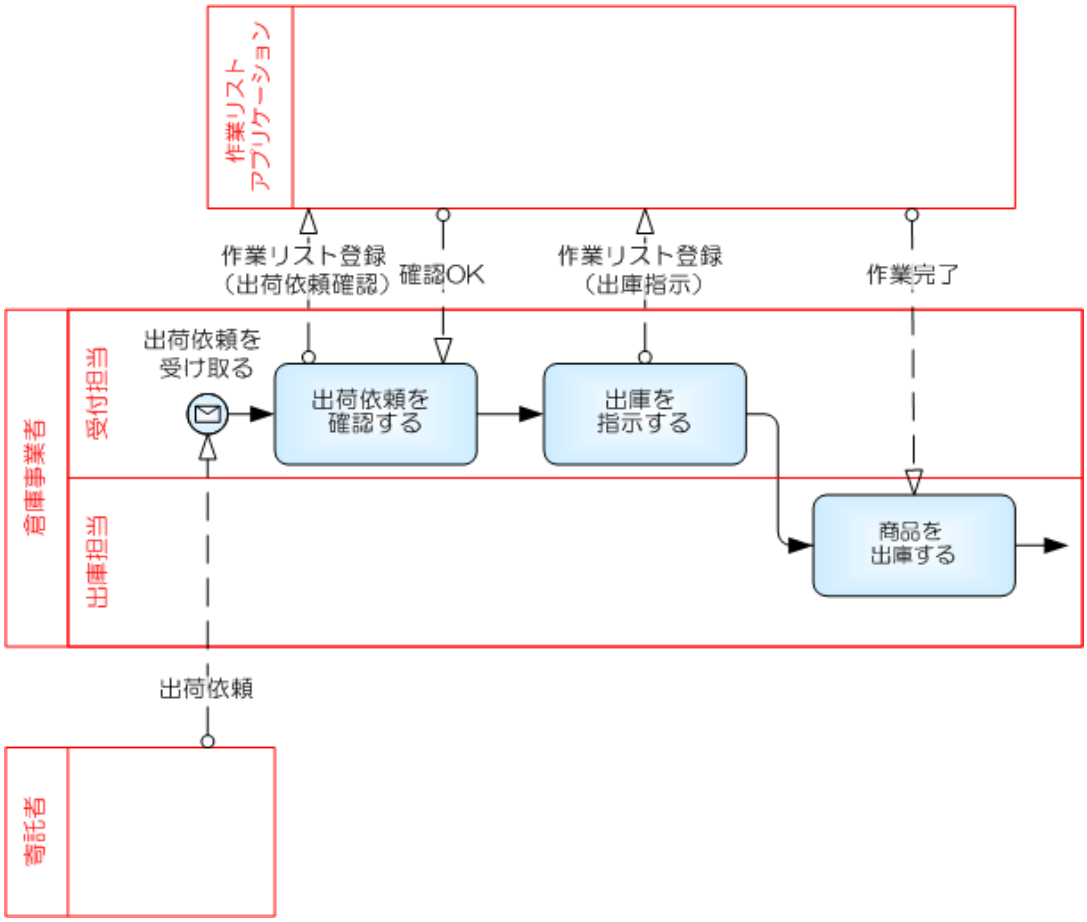
表 5.2　パターン集における記述項目

項目		記述内容
カテゴリ		パターンを分類するカテゴリを記述する。
パターン名称		パターンの役割を端的に表すパターン名称を記述する。
目的		パターンを使用する目的を記述する。 目的以外にも、振る舞い概要、特徴など、他のパターンとの違いを明確にするために必要な情報がある場合には、この欄に記述する。
ビジネスプロセス図	ダイアグラム	パターンを表す代表的な BPMN モデル。 なお、同じ振る舞いを表す BPMN モデルがある場合には、「特記事項」欄に記述する。
	シナリオ	上記のダイアグラムの振る舞いを時系列に箇条書きで記述する。
BPEL		上記のダイアグラムに対応した BPEL を記述する。
BPEL 説明		パターンを表す重要な BPEL 要素の説明や注意事項などを記述する。
特記事項		「BPMN 仕様」「BPEL マッピング」といった 2 つの視点で特記事項を記述する。 BPMN 仕様では、上記のダイアグラムと同じ振る舞いを表す異なる表現、BPMN 属性に関する補足説明、BPMN 仕様で誤解されやすい事項などを記述する。 BPEL マッピングでは、上記 BPEL コードの補足説明を記述する。
パターン適用例		パターンを仮想業務に当てはめた具体的な BPMN モデルのサンプルを記述する。

カテゴリ		フロー制御　基本制御
パターン名称		逐次実行
目的		前のアクティビティが終了すると、次のアクティビティが開始するといった順序付けされた一連のアクティビティを表すために使用する。
ビジネスプロセス図	**ダイアグラム**	
	シナリオ	① 「タスク A」を実行する ② 「タスク A」が完了したら、「タスク B」を実行する ③ 「タスク B」が完了したら、「タスク C」を実行する

BPEL	<sequence> <empty name="タスク A" /> <empty name="タスク B" /> <empty name="タスク C" /> </sequence>
BPEL 説明	逐次実行の振る舞いを<sequence>によって表す。 <sequence>と</sequence>で囲まれた各要素が記述された順序に従い逐次実行される。
特記事項	【BPMN 仕様】 ◇ アクティビティの実行順序を表すシーケンスフローは、1 つのプロセス内（＝1 つのプール内）でのみ使用することができる。異なるプロセス（プール）間ではシーケンスフローは使用できず、その間の相互作用はメッセージフローを使用し、メッセージのやり取りとして表す。

【パターン適用例】



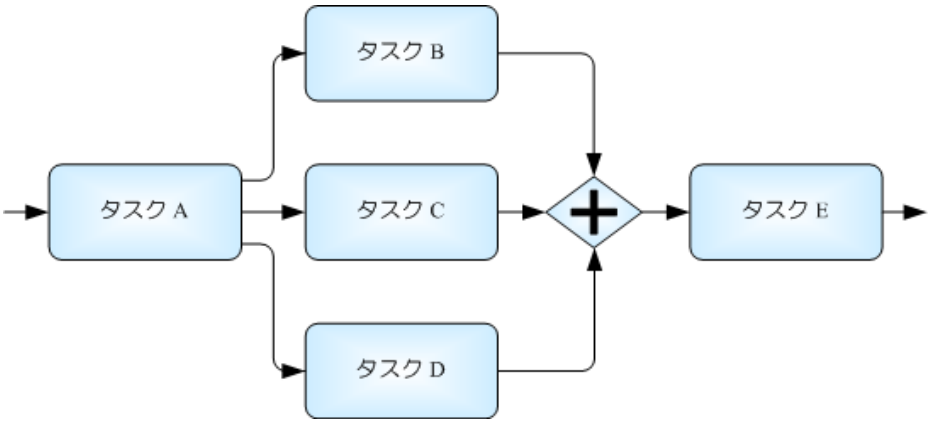
① 寄託者からの出荷依頼を受け取る

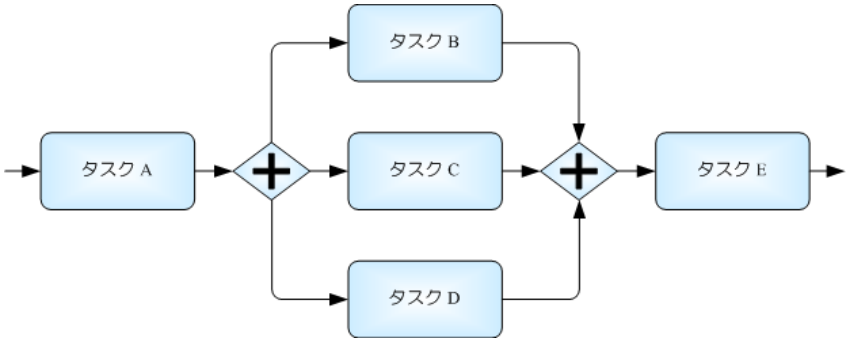
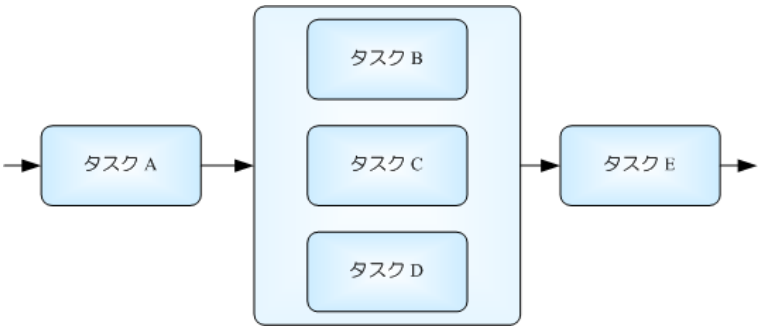
② 受付担当者が出荷依頼の内容を確認する

> **【注記】**
> 上記の例では、作業リストアプリケーションによって、受付担当者と出庫担当者が確認、作業指示、作業完了報告をするものとしている。作業リストアプリケーションはシステムと人とのインターフェースの役割を持つものであり、受付担当者または出庫担当者がログインすると、担当作業の一覧が表示され、参照や登録などを実施する。

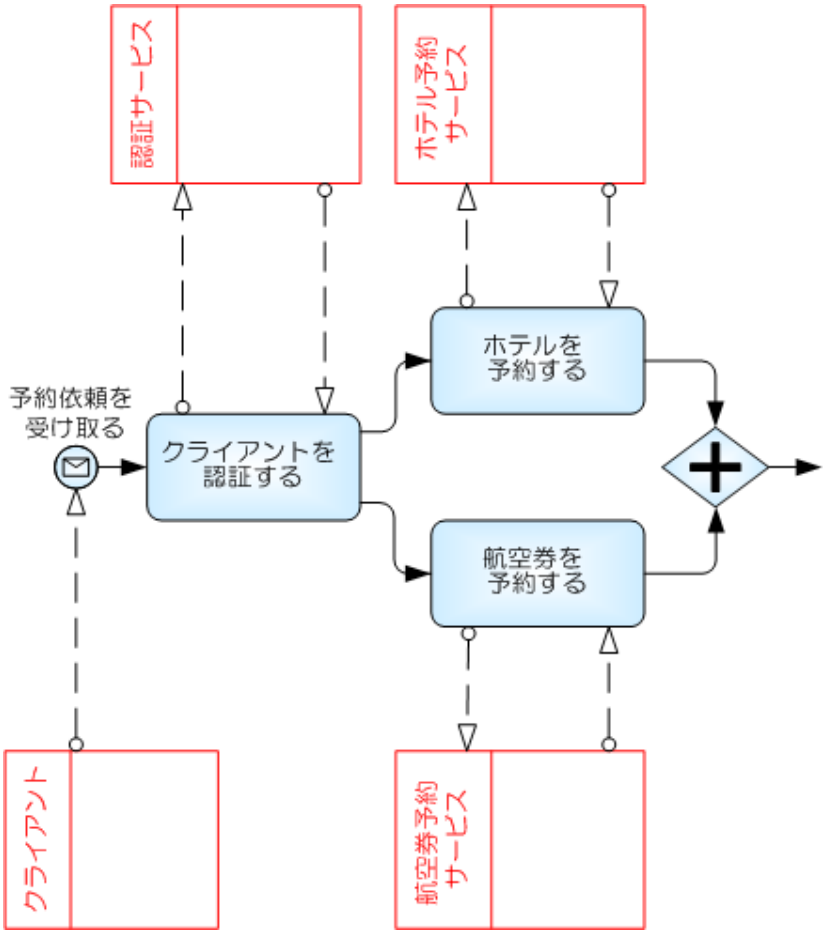
③ 受付担当者が出庫担当者に対して出庫を指示する

④ 出荷担当者が商品を出庫する

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　基本制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>並列実行</td></tr>
<tr><td colspan="2">目的</td><td>2 つ以上のアクティビティを同時に開始するといった並列実行の振る舞いを表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① 「タスク A」を実行する
② 「タスク A」が完了したら、3 つのタスク「タスク B」、「タスク C」、「タスク D」を同時並列で実行する
③ 3 つのタスク「タスク B」、「タスク C」、「タスク D」のすべてが完了したら、「タスク E」を実行する</td></tr>
</table>

BPEL	<sequence> 　<empty name="タスク A" /> 　<flow> 　　<empty name="タスク B" /> 　　<empty name="タスク C" /> 　　<empty name="タスク D" /> 　</flow> 　<empty name="タスク E" /> </sequence>
BPEL 説明	並列実行の振る舞いを<flow>によって表す。 <flow>と</flow>で囲まれた各要素が全て同時に開始される。
特記事項	【BPMN 仕様】 ◇ 並列的な分流を表す AND ゲートウェイを使用しても同じ振る舞いを表現することができる。  ◇ 下図のようにサブプロセスを「並列ボックス」として使用することで並列実行の振る舞いを表現することができる。 

【パターン適用例】



① クライアントからの予約依頼を受け取る

② 認証サービスを呼び出し、クライアントを認証する

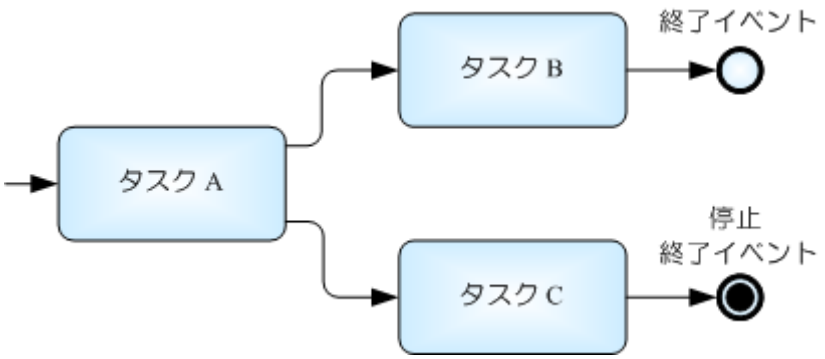
③ ホテル予約サービスと航空券予約サービスを同時に呼び出し、ホテルと航空券を予約する

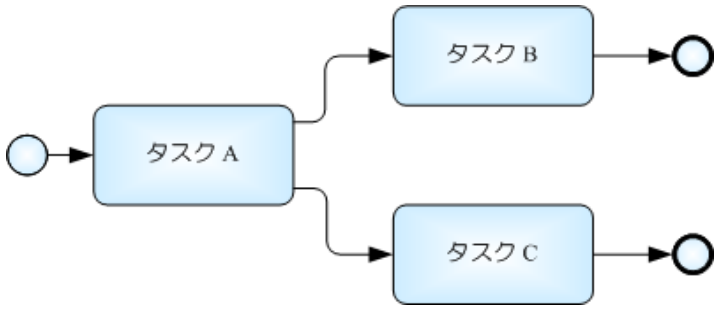
<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　基本制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>タイマーイベント待ち</td></tr>
<tr><td colspan="2">目的</td><td>プロセスの途中で、あらかじめ決められた時間が経過することを待つといった時間による待機状態を表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク A」が完了したら、あらかじめ決められた時間が経過するまで待機する
※あらかじめ決められた時間として、「17 時になるまで待つ」などの定時刻指定、「タスク A 完了後、1 時間経過するのを待つ」などの経過時間指定の 2 通りがある
③ 時間が経過したら「タスク B」を実行する</td></tr>
</table>

BPEL	<sequence> <empty name="タスク A" /> <wait until="*****" /> <empty name="タスク B" /> </sequence>
BPEL 説明	タイマーイベント待ちの振る舞いを<wait>によって表す。 上記の BPEL は定時刻指定の例であり<wait until="*****" />となっているが、経過時間指定の場合には<wait until="*****" />となる。
特記事項	【BPMN 仕様】 ◇ あらかじめ決められた時間の設定は、タイマー中間イベントの BPMN 属性に設定する。定時刻指定の場合には TimeDate 属性に日付または時刻を設定する。経過時間指定の場合には TimeCycle 属性に「タスク A」が完了してからの経過時間を設定する。

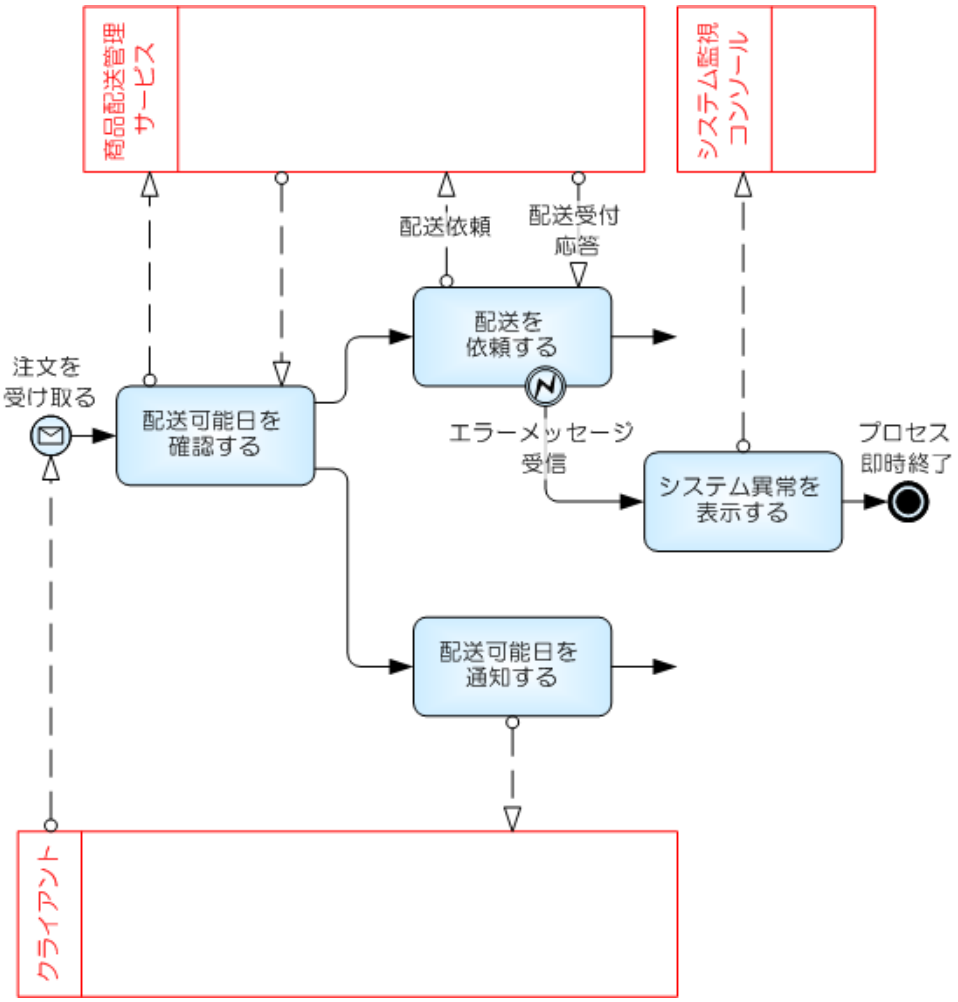
【パターン適用例】

①サブプロセス「注文を受け付ける」を実行する
②サブプロセス「注文を受け付ける」の完了が 17 時前ならば 17 時になるまで待ち、17 時を過ぎていれば待ずに次の作業を実行する
③サブプロセス「配送計画を立てる」を実行する

カテゴリ		フロー制御　基本制御
パターン名称		プロセス即時終了
目的		分流により他に並列実行中のアクティビティがある場合でも、プロセスを即時終了させる振る舞いを表すために使用する。
ビジネスプロセス図	**ダイアグラム**	
	シナリオ	①「タスク A」を実行する ②「タスク A」が完了したら、「タスク B」、「タスク C」を同時並列で実行する ③「タスク C」が完了したら、プロセスを終了する ※「タスク C」が完了した時、「タスク B」が実行中であってもプロセスを終了する ※「タスク B」が完了した時、「タスク C」も完了していればプロセスを終了するが、「タスク C」が実行中であればプロセスを終了しない。

BPEL	<sequence> <empty name="タスク A" /> <flow> <empty name="タスク B" /> <sequence> <empty name="タスク C" /> <terminate /> </sequence> </flow> </sequence>
BPEL 説明	プロセスを即時終了させる箇所に<terminate />を記述する。
特記事項	【BPMN 仕様】 ◇ 停止終了イベント以外の各種終了イベントは、分流により同時並列のフローがある場合には全てのフローが終了イベントに到達した時にプロセスを終了させる。例えば下図において、プロセスが終了するのは「タスク B」、「タスク C」の両方が完了するまでプロセスを終了しない。そのため、エラー発生時のフローなど、他のアクティビティが実行中であっても即時にプロセスを終了させたい場合には、本パターンを使用する。 

【パターン適用例】



① クライアントからの注文を受け取る

② 商品配送管理サービスを呼び出し、配送可能日を確認する

③ 商品配送管理サービスを呼び出し、配送を依頼すると同時に、クライアントに配送可能日を通知する

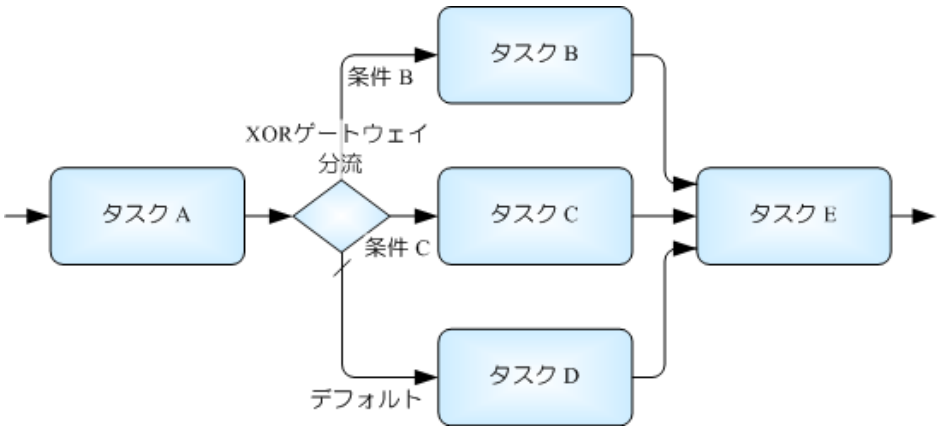
以下、商品配送管理サービスからエラーメッセージ(※注)を受け取った場合のフロー

※注:Web サービスの WSDL において<operation>タグ内の<fault>タグで規定されたメッセージ

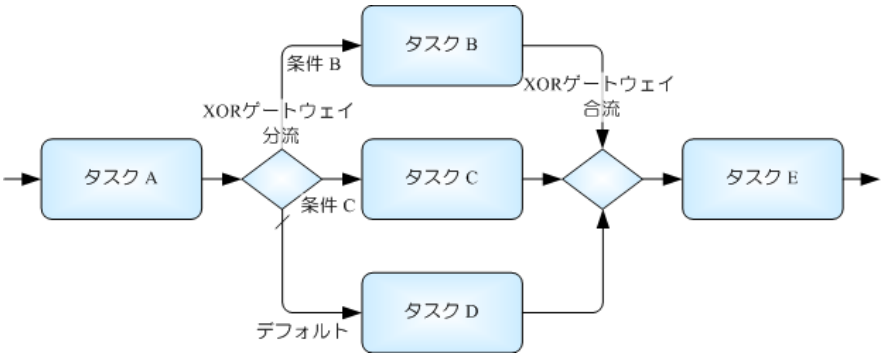
④ エラーメッセージを受け取る

⑤ システム監視コンソール(サービス)を呼び出し、システム異常を表示する

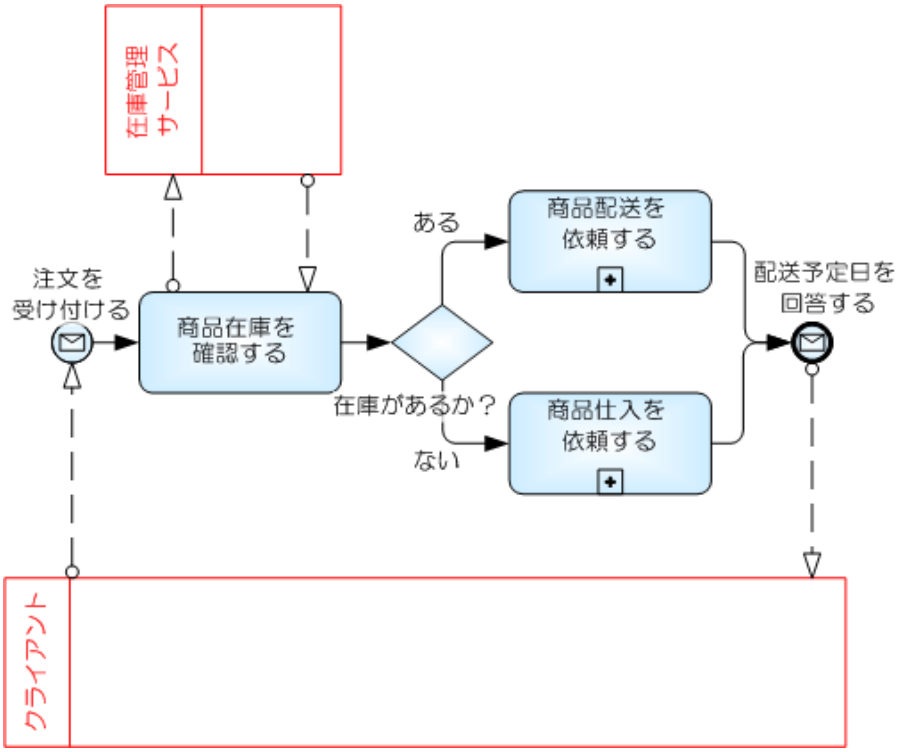
⑥「配送可能日を通知する」以降のフローが実行中であっても、プロセスを即時終了させる

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　経路選択</td></tr>
<tr><td colspan="2">パターン名称</td><td>排他的選択（XOR）</td></tr>
<tr><td colspan="2">目的</td><td>2 つ以上のフローから、条件判定によって 1 つのフローだけが選択されるといった排他的なフロー選択の振る舞いを表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
② 条件判定により後続フローの何れに進むかを決定する
②-1「条件 B」が「真」の場合には「タスク B」を実行する
②-2「条件 B」が「偽」であり、「条件 C」が「真」の場合には「タスク C」を実行する
②-3「条件 B」および「条件 C」が共に「偽」の場合には「タスク D」を実行する（デフォルトフローに進む）
③「タスク B」、「タスク C」または「タスク D」が完了したら、「タスク E」を実行する</td></tr>
</table>

BPEL	<sequence> <empty name="タスク A" /> <switch name="XOR ゲートウェイ分流"> <case condition="条件 B"> <empty name="タスク B" /> </case> <case condition="条件 C"> <empty name="タスク C" /> </case> <otherwise> <empty name="タスク D" /> </otherwise> </switch> <empty name="タスク E" /> </sequence>
BPEL 説明	排他的選択の振る舞いを<switch>、<case>、<otherwise>によって表す。 XOR ゲートウェイ(分流)から出力される制御フローの後続を<case>と</case>で囲み、デフォルトフローの後続を<otherwise>と</otherwise>で囲む。

特記事項	**【BPMN 仕様】** ◇ 下図のように合流を表す XOR ゲートウェイを使用しても同じ振る舞いを表現することができる。  ◇ XOR ゲートウェイは下図のように、ゲートウェイ内に英字「X」を記述し、XOR の振る舞いである旨を明示することもできる。  ◇ 「条件 B」および「条件 C」の条件判定式は、XOR ゲートウェイ（分流）に後続するシーケンスフローの ConditionExpression 属性に Xpath 式で設定する。 ◇ デフォルトフローは省略することができるが、その場合には何れかのフローが必ず選択されるように条件を設定する必要がある。（何れのフローも選択されない場合にはフローが途中で途絶えていまうため、無効なモデルになる。）

【パターン適用例】



① クライアントからの注文を受け取る

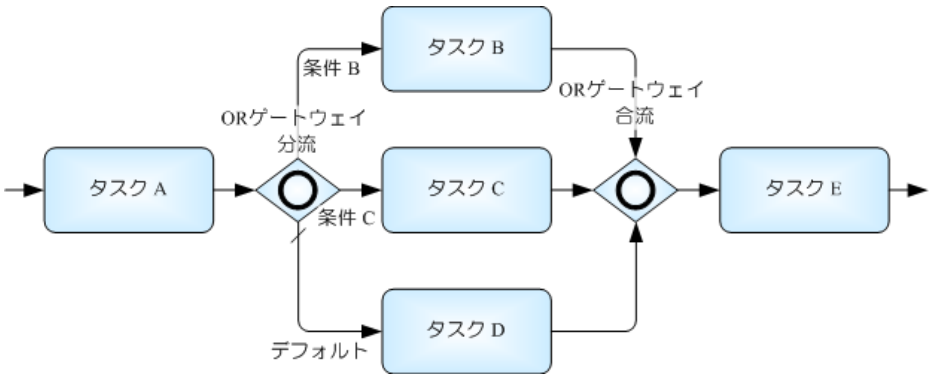
② 在庫管理サービスを呼びだし、商品在庫を確認する

③ 在庫の有無により、後続の何れのサブプロセスを実行するかを決定する

③-1 在庫がある場合には、「商品配送を依頼する」サブプロセスを実行する

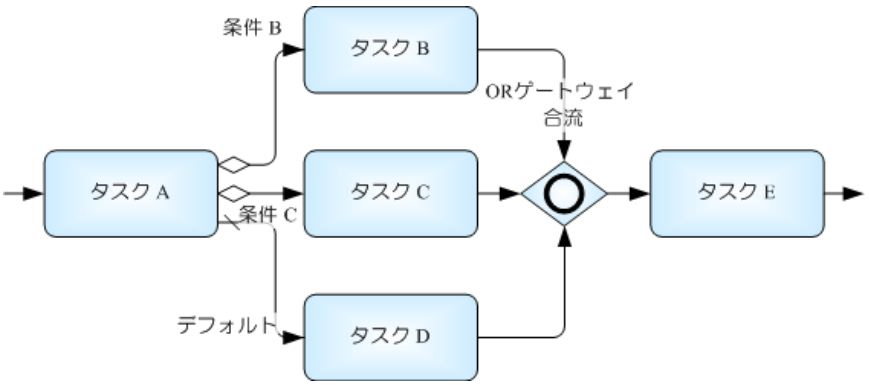
③-2 在庫がない場合には、「商品仕入を依頼する」サブプロセスを実行する

④ 上記のサブプロセスが完了したら、クライアントに配送予定日を回答する

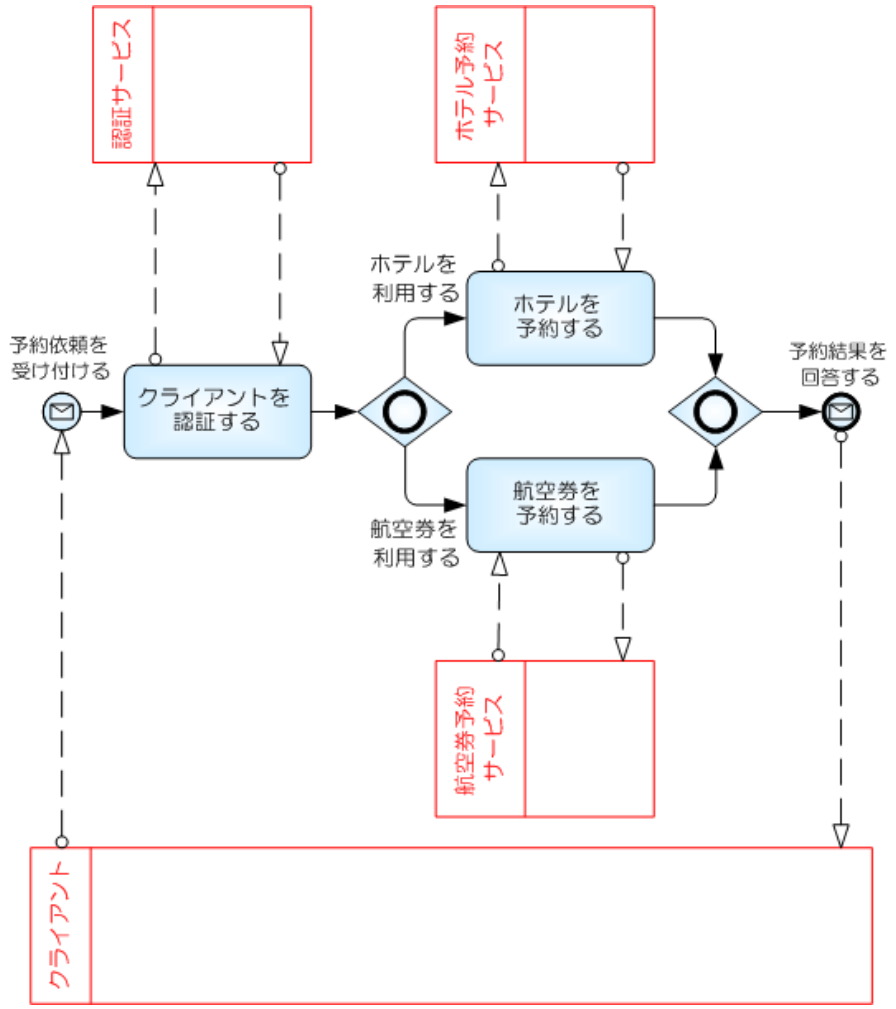
<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　経路選択</td></tr>
<tr><td colspan="2">パターン名称</td><td>包含的選択（OR）</td></tr>
<tr><td colspan="2">目的</td><td>2 つ以上のフローから、条件判定によって1つまたは複数のフローが選択されるといった包含的なフロー選択の振る舞いを表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① 「タスク A」を実行する
② 条件判定により後続フローに進むかを決定する
②-1 「条件 B」が「真」の場合には「タスク B」を実行する
②-2 「条件 C」が「真」の場合には「タスク C」を実行する
※「条件 B」と「条件 C」が共に「真」の場合には「タスク B」と「タスク C」を実行する
②-3 「条件 B」および「条件 C」が共に「偽」の場合には「タスク D」を実行する（デフォルトフローに進む）
③ ②で実行された全てのタスクが終了したら、「タスク E」を実行する</td></tr>
</table>

BPEL コード	<variables>
	  <variable name="OR ゲートウェイ分流_DefaultRequired"
	  type="xsd:boolean" />
	</variables>
	<sequence>
	  <empty name="タスク E" />
	  <sequence>
	  <assign name="OR ゲートウェイ分流_DefaultRequired_initialize">
	  <copy>
	  <from expression="true()" />
	  <to variable="OR ゲートウェイ分流_DefaultRequired" />
	  </copy>
	  </assign>
	  <flow name="OR ゲートウェイ分流">
	  <switch>
	  <case condition="条件 B">
	  <sequence>
	  <empty name="タスク B" />
	  <assign name=
	  "OR ゲートウェイ分流_DefaultRequired_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable=
	  "OR ゲートウェイ分流_DefaultRequired" />
	  </copy>
	  </assign>
	  </sequence>
	  </case>
	  </switch>
	  <switch>
	  <case condition="条件 C">
	  <sequence>
	  <empty name="タスク C" />
	  <assign name=
	  "OR ゲートウェイ分流_DefaultRequired_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable=
	  "OR ゲートウェイ分流_DefaultRequired" />
	  </copy>
	  </assign>
	  </sequence>
	  </case>
	  </switch>
	  </flow>
	（次頁に続く）

BPEL（続き）	<switch> <case condition= "bpws:getVariableData("OR ゲートウェイ分流_DefaultRequired")"> <empty name="タスク D" /> </case> </switch> </sequence> <empty name="タスク E" /> </sequence>
BPEL 説明	包含的選択の振る舞いを<flow>、<switch>、<case>によって表す。 OR ゲートウェイ（分流）から出力されるフローに対応した<switch>と</switch>の対を出力本数分だけ記述し、<flow>と</flow>で囲むことによって各フローに進むかの条件判定を同時並列で実行する。 制御フローの場合は、<case>内に記述された条件判定によって、そのフローに進むかを決定する。 デフォルトフローの場合は、他のフロー（制御フロー）が選択されていないことを確認するためのトラッキング変数（OR ゲートウェイ分流_DefaultRequired）によって判定し、他のフローが実行されてなければデフォルトフローの後続に進む。

特記事項	**【BPMN 仕様】** ◇ 下図のように OR ゲートウェイを使用せず、アクティビティから直接出力される制御シーケンスフローとデフォルトシーケンスフローを使用しても同じ振る舞いを表現することができる。  ◇ 「条件 B」および「条件 C」の条件判定式は、XOR ゲートウェイ分流に後続するシーケンスフローの ConditionExpression 属性に Xpath 式で設定する。 ◇ デフォルトフローは省略することができるが、その場合には何れかのフローが必ず選択されるように条件を設定する必要がある。（何れのフローも選択されない場合ではプロセスフローが途中で途絶えるため、無効なモデルとなってしまう。） **【BPEL マッピング】** ◇ デフォルトフローを省略した場合には、前記の BPEL において **OR ゲートウェイ分流_DefaultRequired** に関連するコードが全て不要になる。

【パターン適用例】



① クライアントからの予約依頼を受け付ける

② 認証サービスを呼び出し、クライアントを認証する

③ 予約依頼のメッセージを確認し、ホテル利用と航空券利用の必要性を判定し、後続のタスクを実行するかを決定する

③-1 ホテルのみを利用する場合には、ホテル予約サービスだけを呼び出し、ホテルを予約する

③-2 航空券のみを利用する場合には、航空券予約サービスだけを呼び出し、航空券を予約する

③-3 ホテルと航空券を利用する場合には、ホテル予約サービスと航空券予約サービスを同時並列で呼び出し、ホテルと航空券を予約する

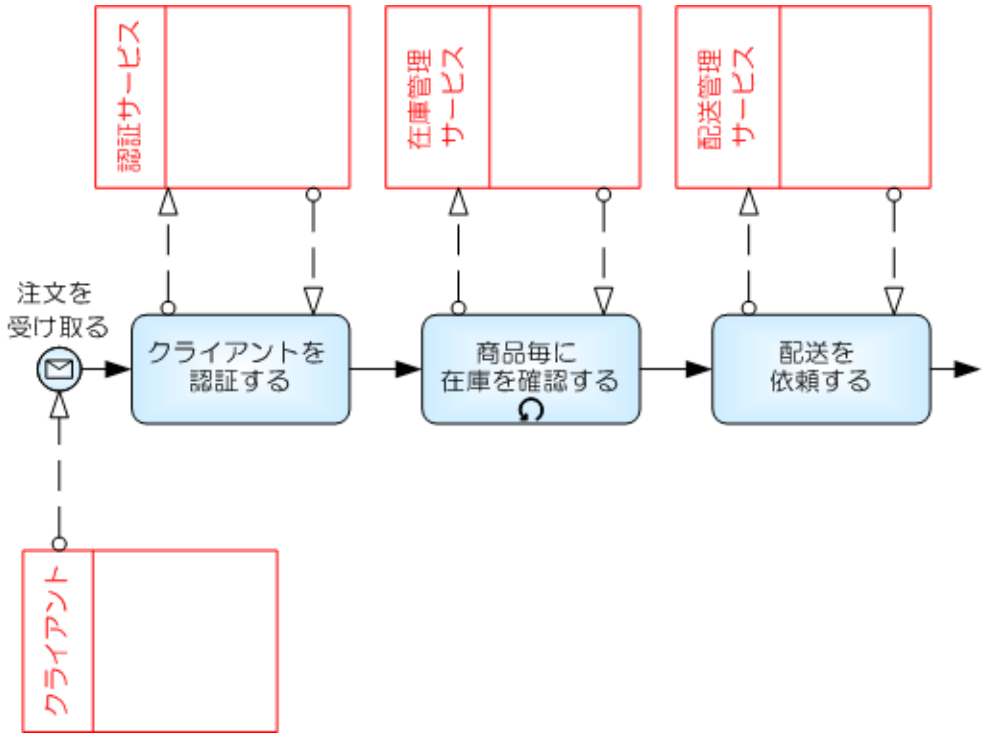
④ ③で必要と判定された全ての予約が完了したら、クライアントに予約結果を回答する

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　ループ制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>標準ループ</td></tr>
<tr><td colspan="2">目的</td><td>アクティビティ(タスクまたはサブプロセス)を繰り返し実行するといった振る舞いを表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク　A」が完了したら、あらかじめ決められたループ条件を満たすまで、「タスク B」を繰り返し実行する
※前の「タスク B」の完了を待ってから、次の「タスク B」を開始する
③ ループ条件が満たされたら、「タスク C」を実行する</td></tr>
</table>

BPEL	<variables> <variable name="タスク B_loopCounter" type="xsd:integer" /> </variables> <sequence> <empty name="タスク A" /> <sequence> <assign name="タスク B_loopCounter_initialize"> <copy> <from expression="0" /> <to variable="タスク B_loopCounter" /> </copy> </assign> <while condition="ループ条件"> <sequence> <empty name="タスク B" /> <assign name="タスク B_loopCounter_increment"> <copy> <from expression= "bpws:getVariableData("タスク B_loopCounter")+1" /> <to variable="タスク B_loopCounter" /> </copy> </assign> </sequence> </while> </sequence> <empty name="タスク C" /> </sequence>
BPEL 説明	標準ループの振る舞いを<while>によって表す。 カウンタ変数(タスク B_loopCounter)によって繰り返し回数をカウントする(必要に応じて、繰り返し条件の判定やレポート目的で使用する)。

特記事項	【BPMN 仕様】
	◇ 繰り返し条件の判定式は、LoopCondition 属性内の Expression 属性に Xpath 式で設定する
	◇ 繰り返し条件を該当アクティビティの実行前、実行後の何れで判定するかを TestTime 属性に設定（実行前=Before、実行後=After）する
	
	【BPEL マッピング】
	◇ 前記の BPEL は TestTime 属性を Before に設定した例である。After に設定した場合には、<while>の前に繰り返し実行されるアクティビティが 1 回実行されるようにする（下記の BPEL を参照）
	
	`</sequence>`
	`<empty name="タスク B" />`
	`<while condition="ループ条件">`
	`<sequence>`
	`<empty name="タスク B" />`
	`<assign name="タスク B_loopCounter_increment">`
	`<copy>`
	`<from expression="bpws:getVariableData("タスク B_loopCounter")+1" />`
	`<to variable="タスク B_loopCounter" />`
	`</copy>`
	`</assign>`
	`</sequence>`
	`</while>`
	`</sequence>`

【パターン適用例】



① クライアントからの注文を受け取る

② 認証サービスを呼び出し、クライアントを認証する

③ 注文のメッセージを確認し、商品の種類の数だけ在庫管理サービスを呼び出し、在庫を確認する(1 つの商品の在庫確認が完了したら、次の商品の在庫を確認する)

④ 配送管理サービスを呼び出し、配送を依頼する

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　ループ制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>マルチインスタンスループ</td></tr>
<tr><td colspan="2">目的</td><td>同じアクティビティ(タスクまたはサブプロセス)を同時並列で実行する振る舞いを表すために使用する。
※あらかじめ決められたループ条件に従い、同じアクティビティの開始を繰り返し指示し、開始を指示されたアクティビティは同時並列で実行されることから、この振る舞いを BPMN 仕様では「マルチインスタンスループ」と呼ぶ。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① 「タスク A」を実行する
② 「タスク A」が完了したら、あらかじめ決められたループ条件を満たすまで、「タスク B」を繰り返し開始する
※前の「タスク B」の完了を待たずに、次の「タスク B」を開始する
③ ②で開始された全ての「タスク B」が完了したら、「タスク C」を実行する</td></tr>
</table>

BPEL

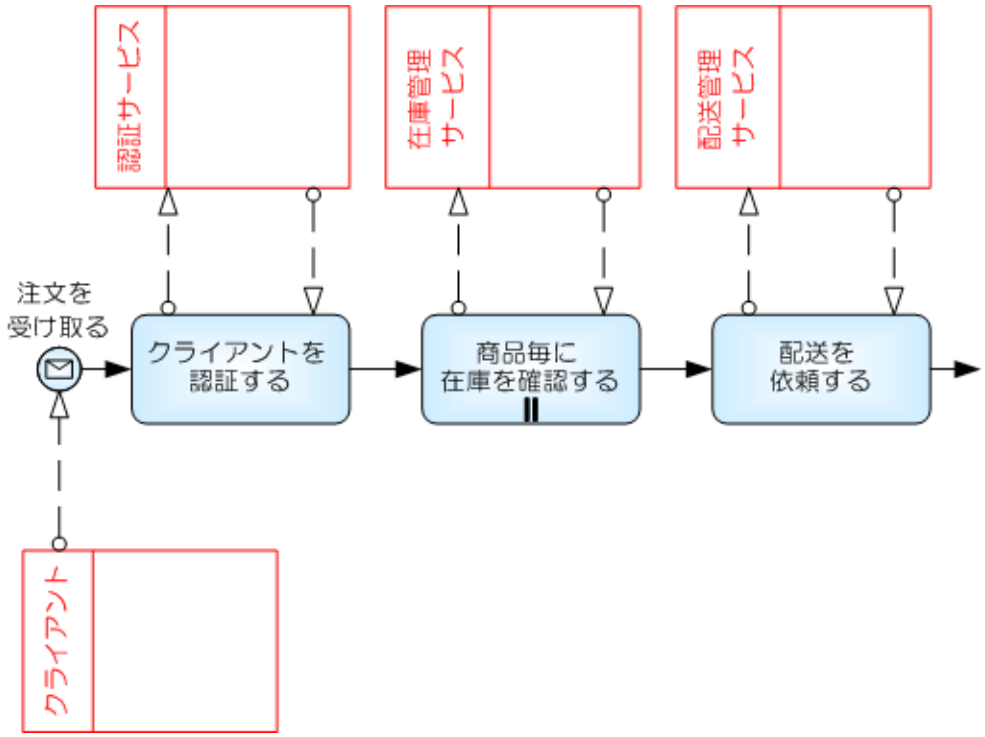
```
<variables>
  <variable name="タスク B_loopCounter" type="xsd:integer" />
  <variable name="タスク B_forEachCount" type="xsd:boolean" />
</variables>
<sequence>
  <empty name="タスク A" />
  <sequence>
    <assign name="タスク B_loopCounter_initialize">
      <copy>
        <from expression="0" />
        <to variable="タスク B_loopCounter" />
      </copy>
    </assign>
    <assign name="タスク B_forEachCount_initialize">
      <copy>
        <from expression="ループ条件" />
        <to variable="タスク B_forEachCount" />
      </copy>
    </assign>
    <while condition=
           "bpws:getVariableData("タスク B_loopCounter")
           >= bpws:getVariableData("タスク B_forEachCount")">
      <sequence>
        <invoke name="Spawn_Process_For_タスク B"
                partnerLink="*****" portType="*****"
                operation="*****" variable="*****" />
        <assign name="タスク B_loopCounter_increment">
          <copy>
            <from expression=
             "bpws:getVariableData("タスク B_loopCounter")+1" />
            <to variable="タスク B_loopCounter" />
          </copy>
        </assign>
      </sequence>
    </while>
    <assign name="タスク B_reset_loopCounter">
      <copy>
        <from expression="0" />
        <to variable="タスク B_loopCounter" />
      </copy>
    </assign>
    <while condition=
           "bpws:getVariableData("タスク B_loopCounter")
           >= bpws:getVariableData("タスク B_forEachCount")">
```

（次頁に続く）

<table>
<tr><td>BPEL（続き）</td><td>
<pre>
      <sequence>
        <receive name="タスク B_Completed"
                partnerLink="*****" portType="*****"
                operation="*****" variable="*****" />
        <assign name="タスク B_loopCounter_increment">
          <copy>
            <from expression=
             "bpws:getVariableData("タスク B_loopCounter")+1" />
            <to variable="タスク B_loopCounter" />
          </copy>
        </assign>
      </sequence>
    </while>
  </sequence>
  <empty name="タスク C" />
</sequence>

（以下、「Spawn_Process_For_タスク B」プロセス）
<sequence>
  <receive name="Spawn_Process_For_タスク B"
           createInstance="yes"
           partnerLink="*****" portType="*****"
           operation="*****" variable="*****" />
  <empty name="タスク B" />
  <invoke name="タスク B_Completed"
          partnerLink="*****" portType="*****"
          operation="*****" variable="*****" />
</sequence>
</pre>
</td></tr>
<tr><td>BPEL 説明</td><td>マルチインスタンスの振る舞いを繰り返しを表す<while>、並列実行の対象となるアクティビティを実行する Spawn_Process、Spawn_Process を呼び出す<invoke>、Spawn_Process が完了したことを受け取る<receive>によって表す。
カウンタ変数（タスク B_loopCounter）によって繰り返し回数をカウントしながら、ループ条件を表す変数（タスク B_ForEachCount）になるまで Spawn_Process を開始する。
カウンタ変数とループ条件を表す変数を使用して繰り返し<receive>を実行しながら、開始した全ての Spawn_Process の完了を待つ。</td></tr>
<tr><td>特記事項</td><td>【BPMN 仕様】
◇ 繰り返し条件の判定式は、MI_Condition 属性内の Expression 属性に Xpath 式で設定する</td></tr>
</table>

【パターン適用例】

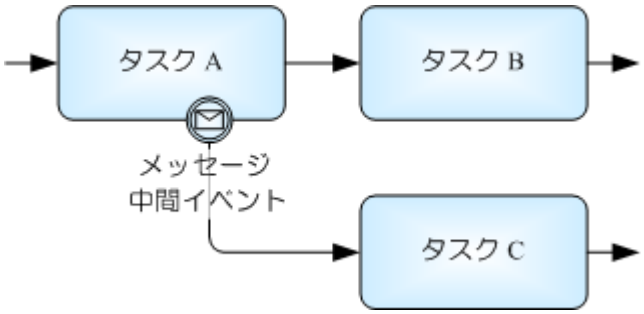


① クライアントからの注文を受け取る

② 認証サービスを呼び出し、クライアントを認証する

③ 注文のメッセージを確認し、商品の種類の数だけ同時並列で在庫管理サービスを呼び出し、在庫を確認する(全ての商品の在庫確認を同時並列で実行する)

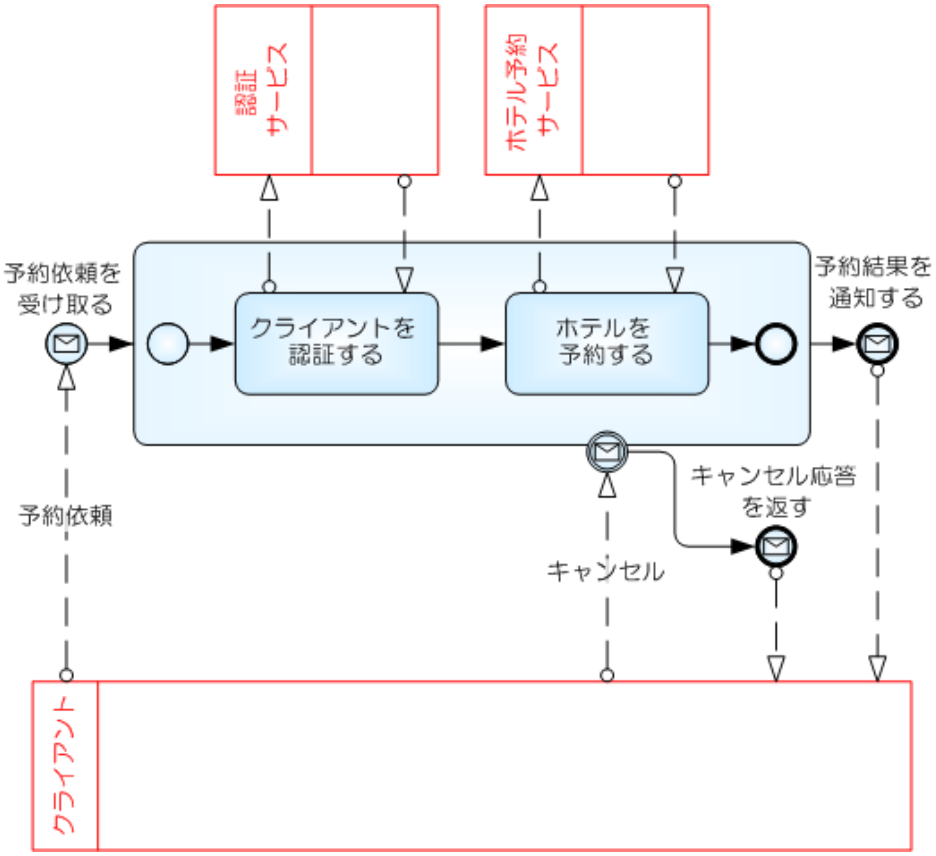
④ 配送管理サービスを呼び出し、配送を依頼する

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　例外処理</td></tr>
<tr><td colspan="2">パターン名称</td><td>メッセージ例外</td></tr>
<tr><td colspan="2">目的</td><td>あるアクティビティを実行中に例外的なメッセージを受け取った場合の例外処理を表すために使用する。メッセージを受け取ると、そのアクティビティの実行を中断し、アクティビティが正常に完了した場合の通常フローとは異なる例外フローに流れを切り換える。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク A」が完了する前に例外メッセージを受け取るか、受け取らないかによって後続フローの何れに進むかを決定する
②-1 例外メッセージを受け取らず「タスク A」が完了した場合には、「タスク B」を実行する
②-2「タスク A」が完了する前に例外メッセージを受け取った場合には「タスク C」を実行する (メッセージ中間イベントから出力する例外フローに進む)</td></tr>
</table>

BPEL	<variables>
	  <variable name="タスク A_normalCompletion"
	  type="xsd:boolean" />
	</variables>
	<sequence>
	  <scope name="タスク A">
	  <faultHandlers>
	  <catch faultName="メッセージ中間イベント_Exit ">
	  <sequence>
	  <empty name="タスク C" />
	  <assign name="タスク A_normalCompletion_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable="タスク A_normalCompletion" />
	  </copy>
	  </assign>
	  </sequence>
	  </catch>
	  </faultHandlers>
	  <eventHandlers>
	  <onMessage partnerLink="*****" portType="*****"
	  operation="*****" variable="*****">
	  <throw faultName="メッセージ中間イベント_Exit" />
	  </onMessage>
	  </eventHandlers>
	  <sequence>
	  <assign name="タスク A_normalCompletion_initialize">
	  <copy>
	  <from expression="true()" />
	  <to variable="タスク A_normalCompletion" />
	  </copy>
	  </assign>
	  <empty name="タスク A" />
	  </sequence>
	  </scope>
	  <switch>
	  <case condition=
	  "bpws:getVariableData("タスク A_normalCompletion")">
	  <empty name="タスク B" />
	  </case>
	  </switch>
	</sequence>

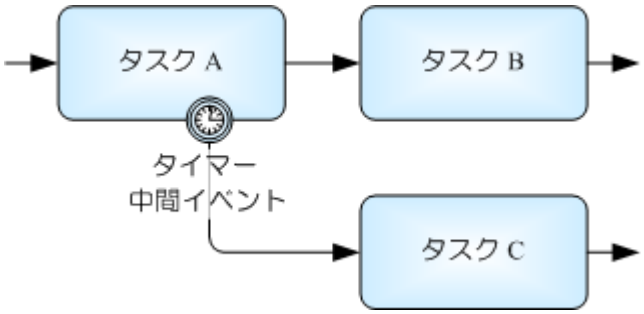
BPEL 説明	メッセージ例外の振る舞いを例外メッセージを受け取る**<eventHandlers>**と例外時の振る舞いを記述する**<faultHandlers>**によって表す。 **<eventHandlers>**では、**<onMessage>**によって例外メッセージを受けて、メッセージを受けたら**<throw>**によって**<faultHandlers>**にエラーを渡す。 **<eventHandlers>**では、エラーを受ける**<catch>**内に例外フローの振る舞いを記述する。 メッセージ中間イベントを境界に配置したアクティビティが正常に完了したことを表す変数(**タスク A_normalCompletion**)を使用し、例外が発生していないことを**<switch>**内の**<case>**で判定し、発生していない場合のみ該当アクティビティに後続する通常フローを実行する。
特記事項	

【パターン適用例】



① クライアントからの予約依頼を受け取る
② 認証サービスを呼び出し、クライアントを認証する
③ ホテル予約サービスを呼び出す

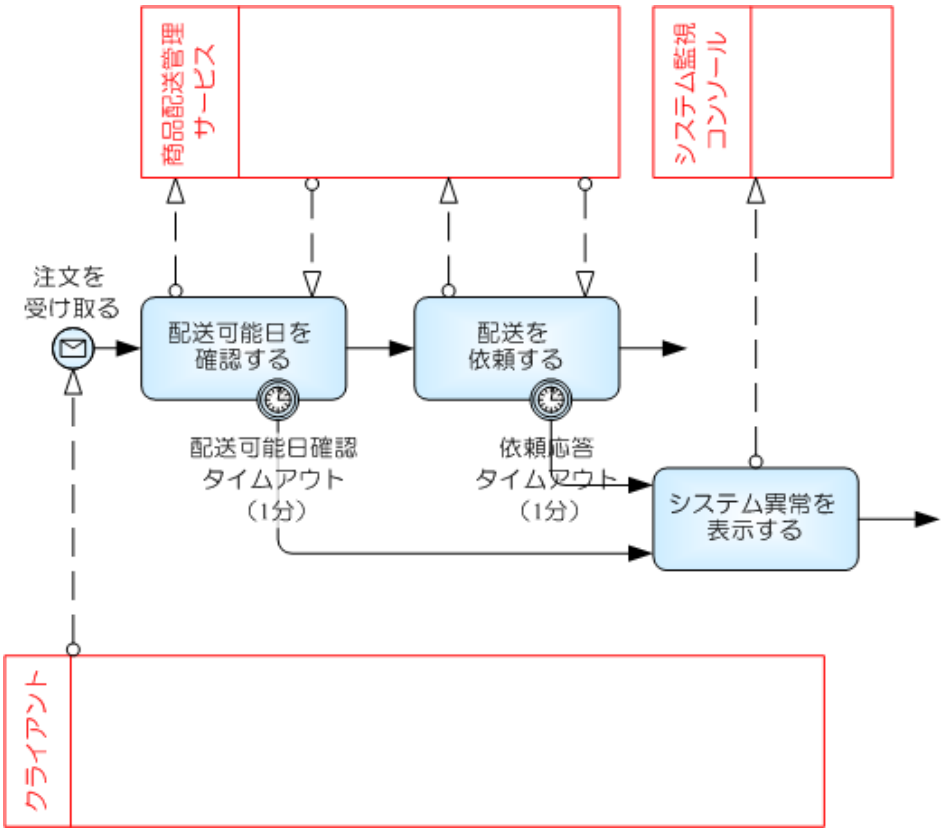
以下、ホテル予約サービスからの応答を受け取る前に、クライアントからのキャンセルを受け取った場合のフロー
④ クライアントからキャンセルを受け取る
⑤ ホテル予約を中断し、クライアントにキャンセル応答を返す

<table>
<tr><td colspan="2">カテゴリ</td><td>例外制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>タイマー例外</td></tr>
<tr><td colspan="2">目的</td><td>あるアクティビティを実行中にタイムアウトが発生した場合の例外処理を表現するために使用する。タイムアウトが発生すると、そのアクティビティの実行を中断し、アクティビティが正常に完了した場合の通常フローとは異なる例外フローに流れを切り換える。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク A」が完了する前にタイムアウトが発生するかによって後続フローの何れに進むかを決定する
②-1 タイムアウトが発生せずに「タスク A」が完了した場合には「タスク B」を実行する
②-2「タスクA」が完了する前にタイムアウトが発生した場合には「タスクC」を実行する (タイマー中間イベントから出力する例外フローに進む)</td></tr>
</table>

BPEL	<variables>
	  <variable name="タスク A_normalCompletion"
	  type="xsd:boolean" />
	</variables>
	<sequence>
	  <scope name="タスク A">
	  <faultHandlers>
	  <catch faultName="タイマー中間イベント_Exit">
	  <sequence>
	  <empty name="タスク C" />
	  <assign name="タスク A_normalCompletion_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable="タスク A_normalCompletion" />
	  </copy>
	  </assign>
	  </sequence>
	  </catch>
	  </faultHandlers>
	  <eventHandlers>
	  <onAlarm until="*****">
	  <throw faultName="タイマー中間イベント_Exit" />
	  </onAlarm>
	  </eventHandlers>
	  <sequence>
	  <assign name="タスク A_normalCompletion_initialize">
	  <copy>
	  <from expression="true()" />
	  <to variable="タスク A_normalCompletion" />
	  </copy>
	  </assign>
	  <empty name="タスク A" />
	  </sequence>
	  </scope>
	  <switch>
	  <case condition=
	  "bpws:getVariableData("タスク A_normalCompletion")">
	  <sequence>
	  <empty name="タスク B" />
	  </sequence>
	  </case>
	  </switch>
	</sequence>

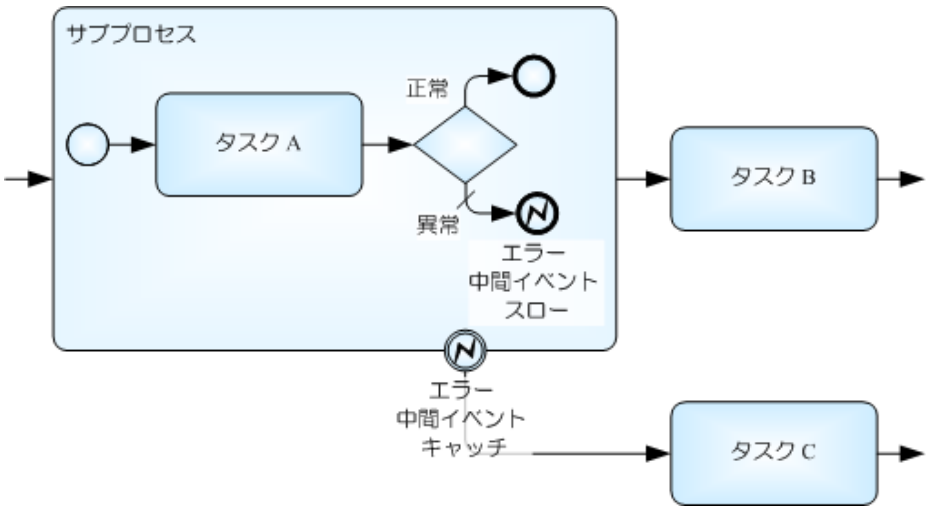
BPEL 説明	タイムアウトを捕捉する**<eventHandlers>**と例外時の振る舞いを記述する**<faultHandlers>**によって表す。 **<eventHandlers>**では、**<onAlarm>**によってタイムアウトを捕捉し、タイムアウトが発生したら**<throw>**によって**<faultHandlers>**にエラーを渡す。 **<eventHandlers>**では、エラーを受ける**<catch>**内に例外フローの振る舞いを記述する。 メッセージ中間イベントを境界に配置したアクティビティが正常に完了したことを表す変数(**タスク A_normalCompletion**)を使用し、例外が発生していないことを**<switch>**内の**<case>**で判定し、発生していない場合のみ該当アクティビティに後続する通常フローを実行する。
特記事項	**【BPMN 仕様】** ◇ タイムアウトの発生タイミングは、タイマー中間イベントの BPMN 属性に設定する。あらかじめ定められた日付または時刻を設定する場合には TimeDate 属性に設定し、タイマー中間イベントを境界に配置したアクティビティの開始からの経過時間を設定する場合には TimeCycle 属性に設定する。 **【BPEL マッピング】** ◇前記の BPEL コードは、タイムアウトの発生タイミングをタイマー中間イベントの TimeDate 属性にした例である。TimeCycle 属性に設定した場合には、前記の**<onAlarm until="*****">**要素の代わりに**<onAlarm for="*****">**要素にマッピングされる。

【パターン適用例】



① クライアントからの注文を受け取る
② 配送可能日を確認するために、商品配送管理サービスを呼び出す

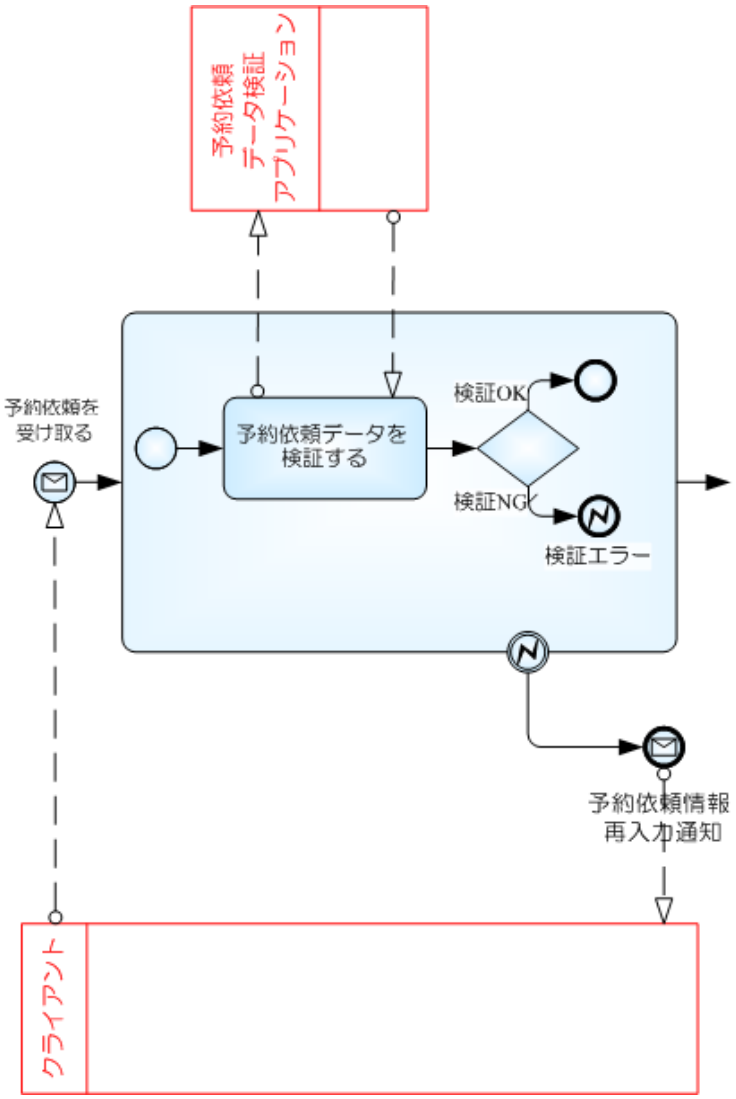
以下、商品配送管理サービスからの応答を受け取る前に、配送可能日確認タイムアウトが発生した場合のフロー
③ 配送可能日確認タイムアウトが発生する
④ システム監視コンソール(サービス)を呼び出し、システム異常を表示する
※「配送を依頼する」以降の通常フローは実行されない

<table>
<tr><td colspan="2">カテゴリ</td><td>フロー制御　例外処理</td></tr>
<tr><td colspan="2">パターン名称</td><td>エラー処理</td></tr>
<tr><td colspan="2">目的</td><td>あるアクティビティを実行中にエラーが発生した場合の例外処理を表現するために使用する。エラーが発生すると、そのアクティビティの実行を中断し、アクティビティが正常に完了した場合の通常フローとは異なる例外フローに流れを切り換える。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク A」が完了したら、実行結果が正常か、異常かによって後続フローの何れに進むかを決定する
②-1 正常な場合には「タスク B」を実行する
②-2 異常な場合には「タスク C」を実行する（エラー中間イベントから出力する例外フローに進む）</td></tr>
</table>

BPEL	<variables>
	  <variable name=
	  "サブプロセス_normalCompletion" type="xsd:boolean" />
	</variables>
	<sequence>
	  <scope name="サブプロセス">
	  <faultHandlers>
	  <catch faultName="エラーコード">
	  <sequence>
	  <empty name="タスク B" />
	  <assign name="サブプロセス_normalCompletion_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable="サブプロセス_normalCompletion" />
	  </copy>
	  </assign>
	  </sequence>
	  </catch>
	  </faultHandlers>
	  <sequence>
	  <assign name="サブプロセス_normalCompletion_initialize">
	  <copy>
	  <from expression="true()" />
	  <to variable="サブプロセス_normalCompletion" />
	  </copy>
	  </assign>
	  <sequence>
	  <empty name="検証タスク" />
	  <switch name="XOR ゲートウェイ">
	  <case condition="条件式">
	  <empty />
	  </case>
	  <otherwise>
	  <throw faultName="エラーコード" />
	  </otherwise>
	  </switch>
	  </sequence>
	  </sequence>
	  </scope>
	  <switch>
	  <case condition=
	  "bpws:getVariableData("サブプロセス_normalCompletion")">
	  <empty name="タスク A" />
	  </case>
	  </switch>
	</sequence>

BPEL 説明	異常を判定する**<switch>**と例外時の振る舞いを記述する**<faultHandlers>**によって表す。 **<swith>**では、**<case>**によって正常か、異常かを判定し、異常の場合には**<throw>**によって**<faultHandlers>**にエラーを渡す。 **<eventHandlers>**では、エラーを受ける**<catch>**内に例外フローの振る舞いを記述する。 メッセージ中間イベントを境界に配置したアクティビティが正常に完了したことを表す変数(**タスク A_normalCompletion**)を使用し、例外が発生していないことを**<switch>**内の**<case>**で判定し、発生していない場合のみ該当アクティビティに後続する通常フローを実行する。
特記事項	**【BPMN 仕様】** ◇ エラー中間イベント(キャッチ)では、エラーを特定するコードを ErrorCode 属性に設定する。ErrorCode 属性を設定しない場合には、全てのエラーを捕捉する ◇ 正常か、異常かを判定する XOR ゲートウェイ、エラー中間イベント(スロー)を省略しても良い ◇ 省略した場合には、明示されない任意のエラーを表し、エラー中間イベント(キャッチ)はBPEL 実行環境において発生する事前定義の任意エラーを捕捉する **【BPEL マッピング】** ◇ エラー中間イベント(キャッチ)においてErrorCode 属性を設定しない場合には、前記の BPEL において、**<catch faultName="エラーコード">**が**<catchAll>**になる

【パターン適用例】



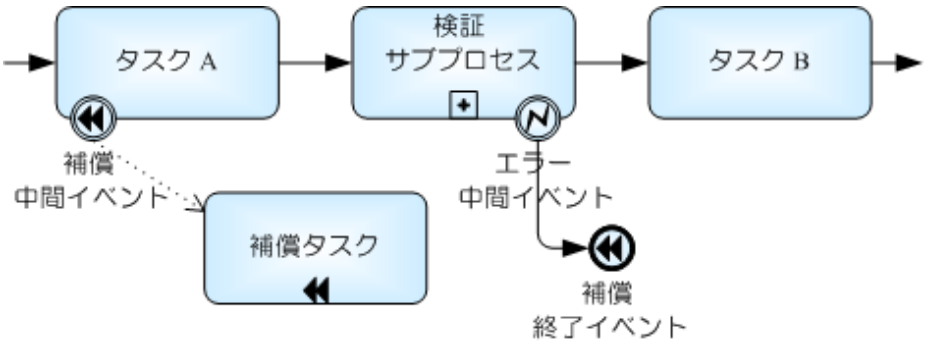
① クライアントからの予約依頼を受け取る

② 予約依頼データ検証アプリケーションを呼び出し、予約依頼データを検証する

以下、検証結果が NG の場合のフロー

③ 予約依頼データ検証アプリケーションから検証 NG を受け取ると、検証エラーを発生させる

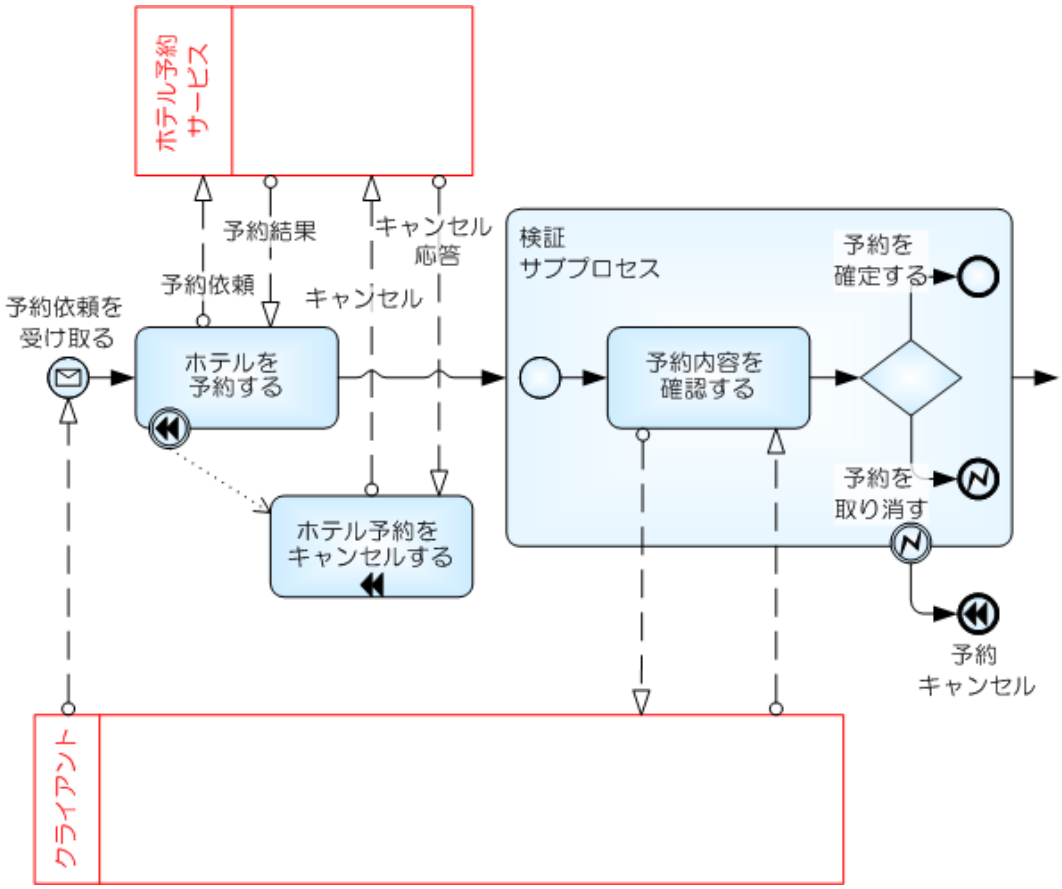
④ クライアントに予約依頼情報再入力通知を送る

<table>
<tr><td colspan="2">カテゴリ</td><td>例外制御</td></tr>
<tr><td colspan="2">パターン名称</td><td>補償処理</td></tr>
<tr><td colspan="2">目的</td><td>アクティビティの実行結果が望ましくないと判断されるときに、そのアクティビティを実行する前の状態に戻すといった補償処理を表現するために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>①「タスク A」を実行する
②「タスク A」が完了したら、「検証サブプロセス」を実行する
※「タスク A」の実行結果が望ましいものかを判定し、望ましくないと判定される場合にはエラーを発生させる
③「検証サブプロセス」においてエラーが発生するかによって後続フローの何れに進むかを決定する
③-1 エラーが発生せずに「検証サブプロセス」が完了した場合には「タスク B」を実行する
③-2 エラーが発生が発生した場合には「補償タスク」を実行する (「タスク B」は実行せず、「補償タスク」が完了したらプロセスを終了する)</td></tr>
</table>

BPEL	<variables>
	  <variable name="検証サブプロセス_normalCompletion"
	  type="xsd:boolean" />
	</variables>
	<sequence>
	  <scope name="タスク A">
	  <compensationHandler>
	  <empty name="補償タスク" />
	  </compensationHandler>
	  <empty name="タスク A" />
	  </scope>
	  <scope name="検証サブプロセス">
	  <faultHandlers>
	  <catchAll>
	  <sequence>
	  <compensate scope="タスク A" />
	  <assign name="検証サブプロセス
	  _normalCompletion_reset">
	  <copy>
	  <from expression="false()" />
	  <to variable="検証サブプロセス
	  _normalCompletion" />
	  </copy>
	  </assign>
	  </sequence>
	  </catchAll>
	  </faultHandlers>
	  <sequence>
	  <assign name="検証サブプロセス
	  _normalCompletion_initialize">
	  <copy>
	  <from expression="true()" />
	  <to variable="検証サブプロセス_normalCompletion" />
	  </copy>
	  </assign>
	  <empty />
	  </sequence>
	  </scope>
	  <switch>
	  <case condition=
	  "bpws:getVariableData("検証サブプロセス
	  _normalCompletion")">
	  <empty name="タスク B" />
	  </case>
	  </switch>
	</sequence>

BPEL 説明	補償が必要な場合(「タスク A」の実行結果が望ましくない場合)に発生するエラーを捕捉する**<faultHandlers>**と補償時に必要な処理を記述する**<compensationHandler>**によって表す。 **<faultHandlers>**では、**<compensate>**によって**<compensationHandler >**に補償が必要であることを示す通知を渡す。 補償が必要なエラーが発生しなかったことを表すエラー変数(**検証サブプロセス_normalCompletion**)を使用して判定し、エラーが発生していない場合のみ該当アクティビティに後続する通常フローを実行する。
特記事項	**【BPEL マッピング】** ◇ BPEL では補償処理(**<compensationHandler>**)の実行を指示する**<compensate>**要素を**<faultHandlers>**要素内に配置する必要がある。そのため、BPMN の記述においても、**<compensate>**要素にマッピングされる「補償終了イベント」を**<faultHandlers>**要素にマッピングされる「エラー中間イベントから出力される例外フロー」に配置する必要がある。

【パターン適用例】



① クライアントからの予約依頼を受け取る

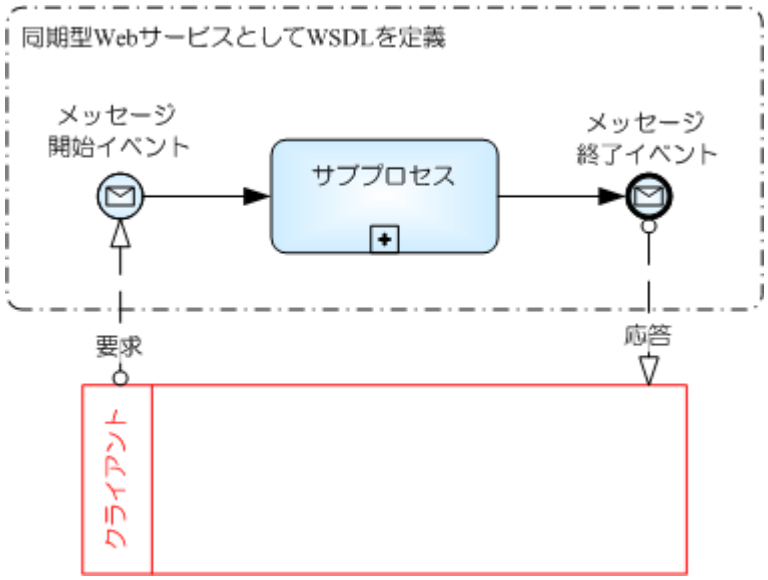
② ホテル予約サービスを呼び出し、ホテルを予約する

③ クライアントに予約内容を渡し、クライアントは予約を確定するか、予約を取り消すかを判断する

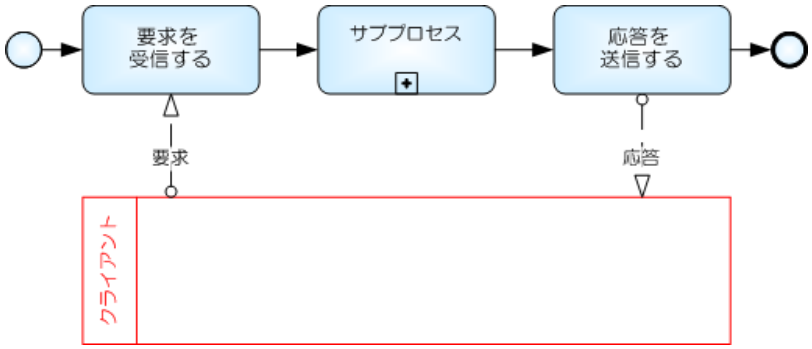
以下、予約を取り消す場合のフロー

④ クライアントから予約を取り消すことを表すメッセージを受け取る

⑤ ホテル予約サービスを呼び出し、ホテル予約をキャンセルする

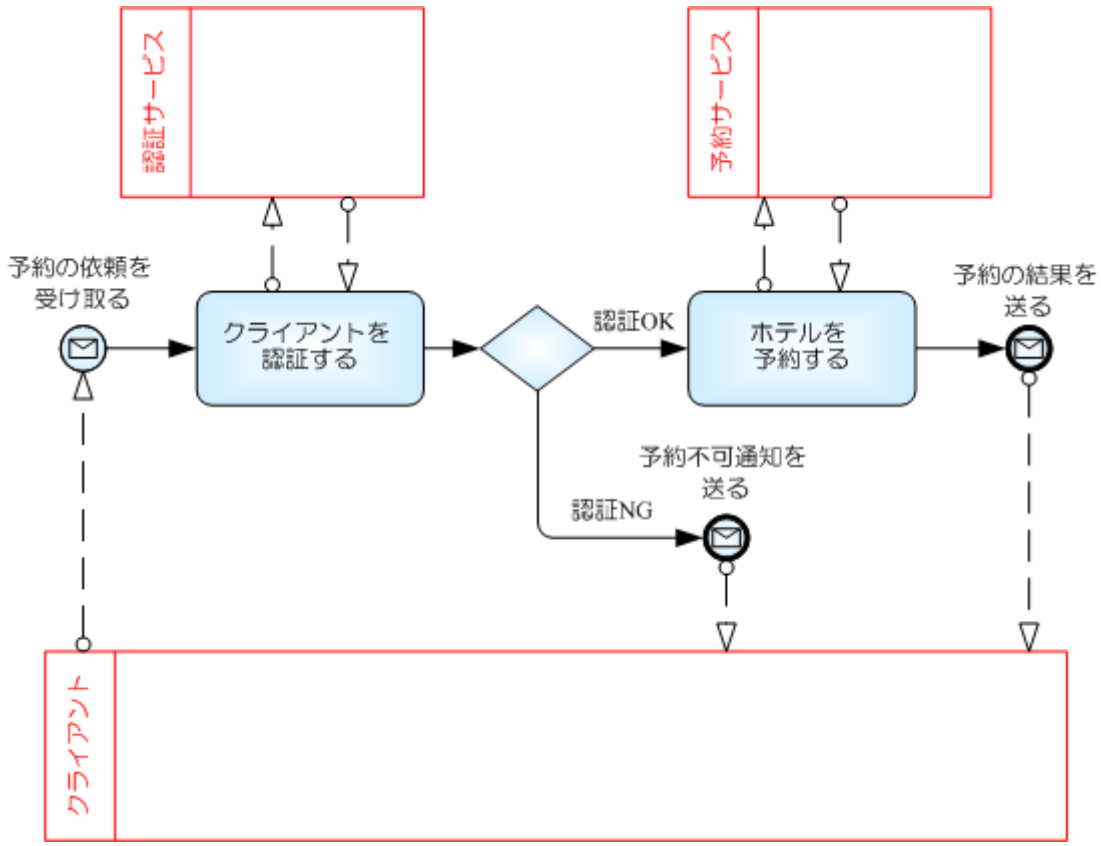
<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス要求受信</td></tr>
<tr><td colspan="2">パターン名称</td><td>同期型受信応答</td></tr>
<tr><td colspan="2">目的</td><td>自プロセスを同期型 Web サービスとして公開し、クライアントからのサービス要求を受信後、クライアントに応答を送信するといったメッセージの送受信を表すために使用する。クライアント側では、要求送信後には応答待ちの状態になり、待っている間は他の処理を実行することができない。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① クライアントからの要求を待ち、要求を受け取ったらプロセスを開始する
② サブプロセスを実行する
③ サブプロセスの実行が完了したら、クライアントに応答を送信する

※ クライアントは、要求を送信してから応答を受信するまでの間、他の処理を実行できない</td></tr>
</table>

<table>
<tr><td>BPEL</td><td><pre>
<sequence>
  <receive name="メッセージ開始イベント"
           createInstance="yes"
           partnerLink="*****" portType="*****"
           operation="*****" variable="*****" />
  <scope name="サブプロセス">
    <empty />
  </scope>
  <reply name="メッセージ終了イベント"
         partnerLink="*****" portType="*****"
         operation="*****" variable="*****" />
</sequence>
</pre></td></tr>
<tr><td>BPEL 説明</td><td>クライアントからの要求を受け取る<receive>とクライアントに応答を送信する<reply>によって表す。
<receive>では、createInstance="yes"として要求受信時にプロセスをインスタンス化する。</td></tr>
<tr><td>特記事項</td><td>【BPMN 仕様】
◇ 下図のように、タスクを使用してクライアントからの要求受信と応答送信を表すこともできる
 

◇ タスクを使用してクライアントからの要求受信を表す場合には、そのタスクの TaskType 属性を Receive に設定し、Instantiate 属性を True に設定する
◇ タスクを使用してクライアントからの応答受信を表す場合には、そのタスクの TaskType 属性を Send に設定する</td></tr>
</table>

【パターン適用例】

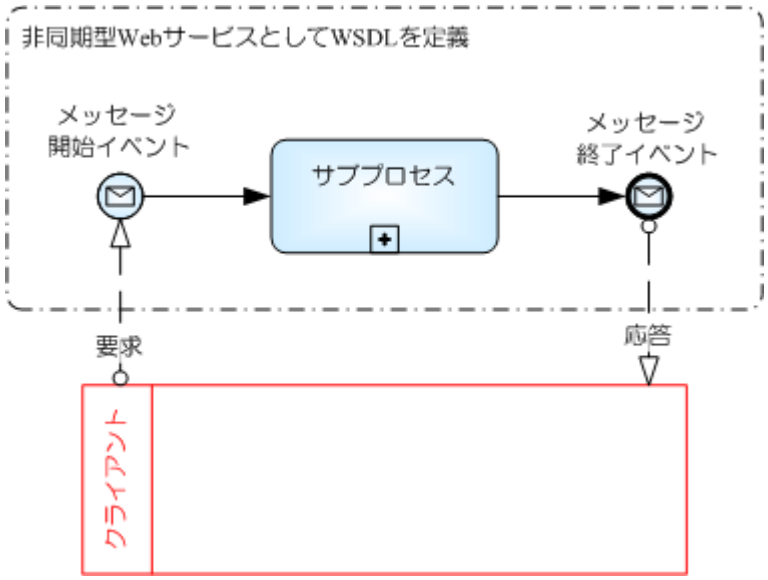


① クライアントからの予約依頼を受け取る
（予約依頼を受け取るとプロセスをインスタンス化し、プロセスを開始する）
② 認証サービスを呼び出し、クライアントを認証する
③ 認証サービスからの応答を確認し、後続の何れのフローに進むかを決定する

認証 OK の場合
④-1 ホテル予約サービスを呼び出し、ホテルを予約する
⑤ クライアントに予約結果を送る

認証 NG の場合
④-2 クライアントに予約不可通知を送る

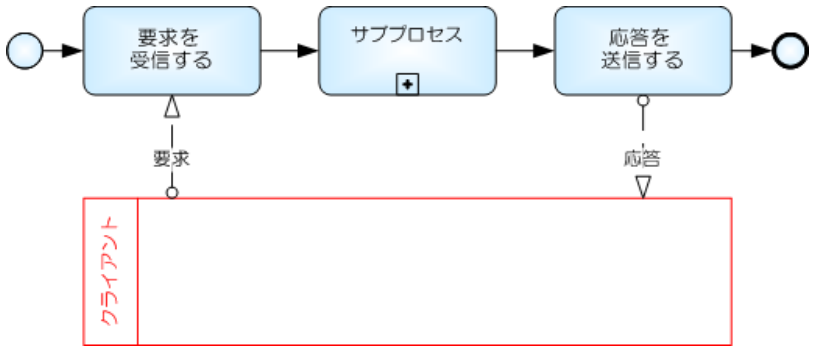
※ クライアントは予約依頼を送信してから、予約結果または予約不可通知を受信するまでの間、他の処理を実行できない

<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス要求受信</td></tr>
<tr><td colspan="2">パターン名称</td><td>非同期型受信応答</td></tr>
<tr><td colspan="2">目的</td><td>自プロセスを非同期型 Web サービスとして公開し、クライアントからのサービス要求を受信後、クライアントに応答を送信するといったメッセージの送受信を表すために使用する。クライアント側では、要求送信後には応答待ちの状態にならず、待っている間に他の処理を実行することができる。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① クライアントからの要求を待ち、要求を受け取ったらプロセスを開始する
② サブプロセスを実行する
③ サブプロセスの実行が完了したら、クライアントに応答を送信する

※ クライアントは、要求を送信してから応答を受信するまでの間、他の処理を実行できる</td></tr>
</table>

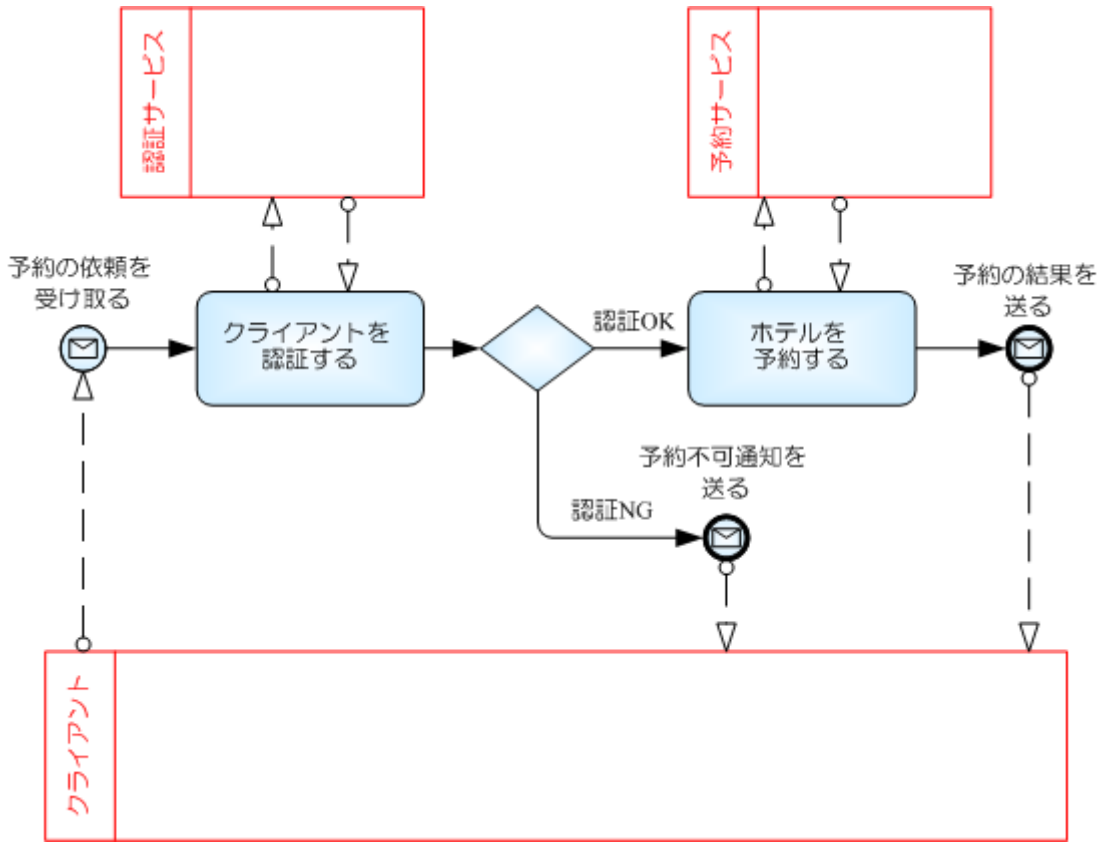
<table>
<tr><td>BPEL</td><td>

```
<sequence>
  <receive name="メッセージ開始イベント"
           createInstance="yes"
           partnerLink="*****" portType="*****"
           operation="*****" variable="*****" />
  <scope name="サブプロセス">
    <empty />
  </scope>
  <invoke name="メッセージ終了イベント"
          partnerLink="*****" portType="*****"
          operation="*****" variable="*****" />
</sequence>
```

</td></tr>
<tr><td>BPEL 説明</td><td>クライアントからの要求を受け取る<receive>とクライアントに応答を送信する<invoke>によって表す。
<receive>では、createInstance="yes"として要求受信時にプロセスをインスタンス化する。</td></tr>
<tr><td>特記事項</td><td>【BPMN 仕様】
◇ 下図のように、タスクを使用してクライアントからの要求受信と応答送信を表すこともできる
 

◇ タスクを使用してクライアントからの要求受信を表す場合には、そのタスクの TaskType 属性を Receive に設定し、Instantiate 属性を True に設定する
◇ タスクを使用してクライアントからの応答受信を表す場合には、そのタスクの TaskType 属性を Send に設定する</td></tr>
</table>

【パターン適用例】

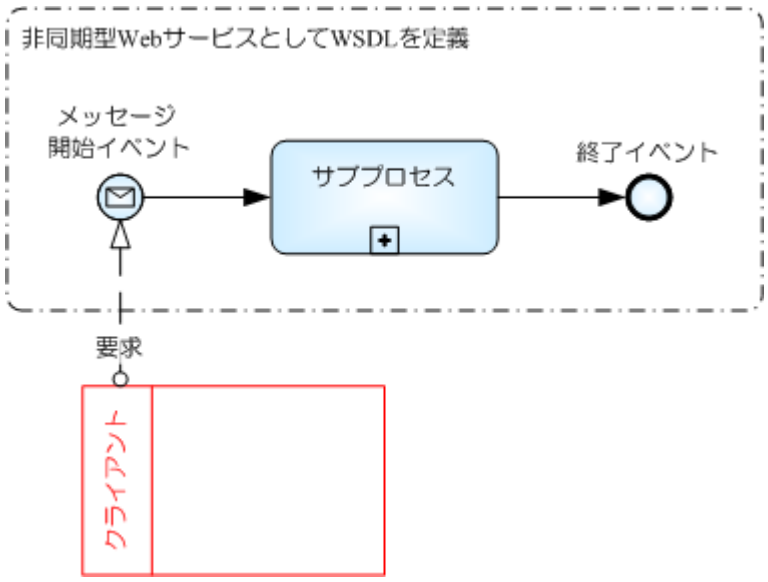


① クライアントからの予約依頼を受け取る
（予約依頼を受け取るとプロセスをインスタンス化し、プロセスを開始する）
② 認証サービスを呼び出し、クライアントを認証する
③ 認証サービスからの応答を確認し、後続の何れのフローに進むかを決定する

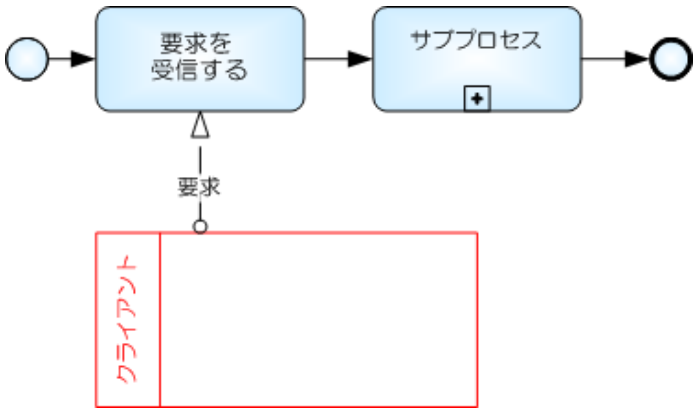
認証 OK の場合
④-1 ホテル予約サービスを呼び出し、ホテルを予約する
⑤ クライアントに予約結果を送る

認証 NG の場合
④-2 クライアントに予約不可通知を送る

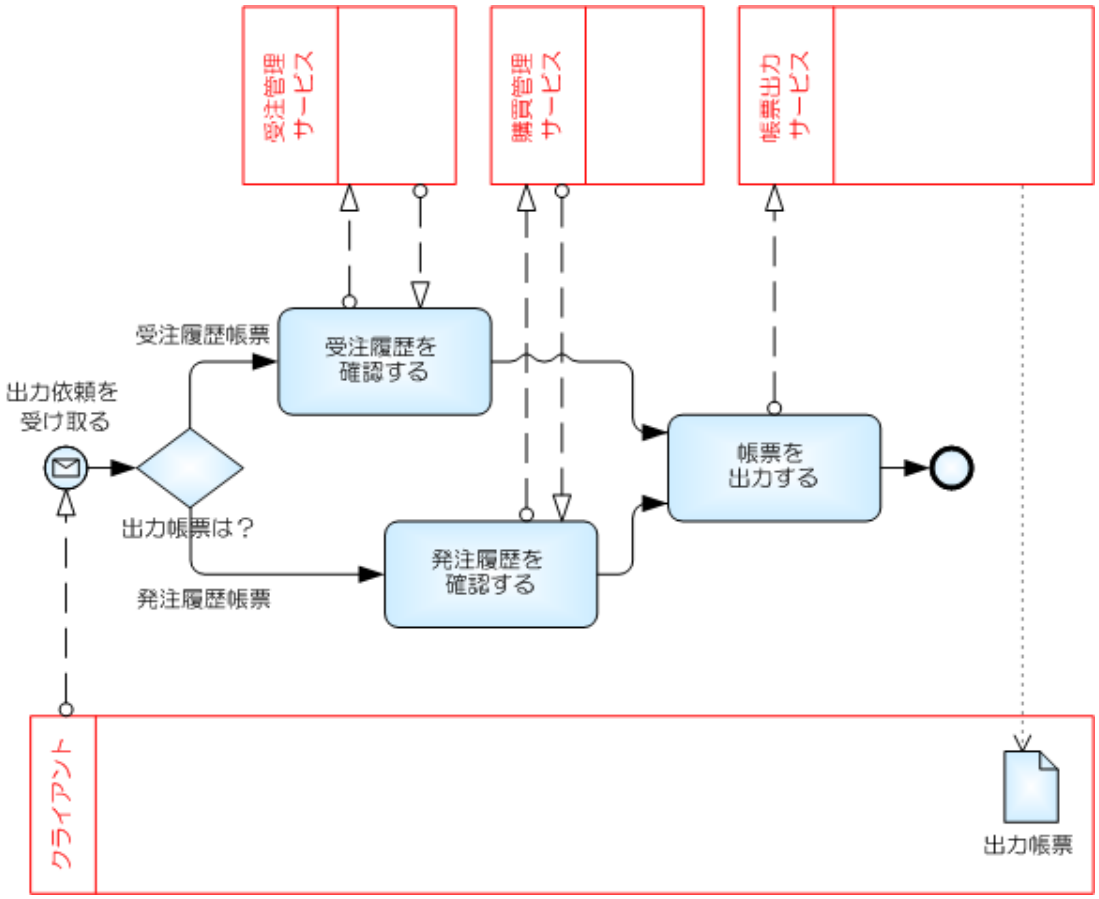
※ クライアントは予約依頼を送信してから、予約結果または予約不可通知を受信するまでの間、他の処理を実行できる

<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス要求受信</td></tr>
<tr><td colspan="2">パターン名称</td><td>非同期型受信</td></tr>
<tr><td colspan="2">目的</td><td>自プロセスを非同期型 Web サービスとして公開し、クライアントからのサービス要求を受信後、ある決められた処理を実行するがクライアントに応答を送信しない片方向メッセージの受信を表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① クライアントからの要求を待ち、要求を受け取ったらプロセスを開始する
② サブプロセスを実行する
③ サブプロセスが完了したら、プロセスを終了する</td></tr>
</table>

BPEL	<sequence>
	  <receive name="メッセージ開始イベント"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	  <scope name="サブプロセス">
	  <empty />
	  </scope>
	</sequence>

BPEL 説明	クライアントからの要求を受け取る<receive>によって表す。 <receive>では、createInstance="yes"として要求受信時にプロセスをインスタンス化する。
特記事項	【BPMN 仕様】 ◇ 下図のように、タスクを使用してクライアントからの要求受信を表すこともできる  ◇ タスクを使用してクライアントからの要求受信を表す場合には、そのタスクの TaskType 属性を Receive に設定し、Instantiate 属性を True に設定する

【パターン適用例】



① クライアントからの出力依頼を受け取る

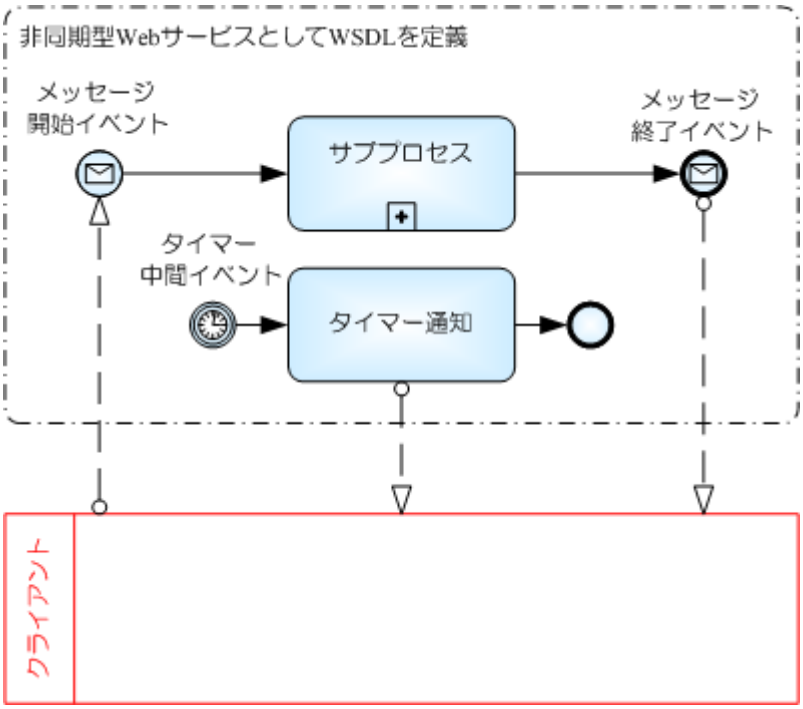
（出力依頼を受け取るとプロセスをインスタンス化し、プロセスを開始する）

② クライアントからの出力依頼で指定された帳票の種類によって、後続の何れのフローに進むかを決定する

②-1 受注履歴帳票の場合は、受注管理サービスを呼び出し、受注履歴を確認する

②-2 発注履歴帳票の場合は、発注管理サービスを呼び出し、受注履歴を確認する

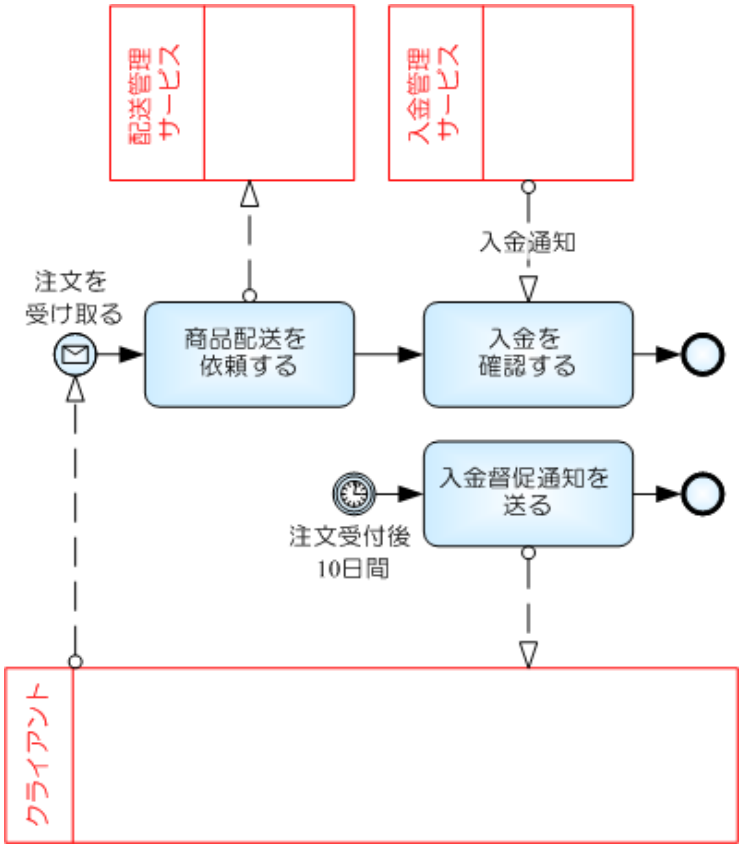
③ 帳票出力サービスを呼び出し、帳票を出力する

カテゴリ		サービス連携　サービス要求受信
パターン名称		タイマー通知送信
目的		タイマーによる監視を行い、タイムアウトが発生した場合のみ、クライアントにタイマー通知（タイムアウトが発生した旨を伝えるメッセージ）を送るといった振る舞いを表すために使用する。タイムアウトが発生しても、通常フローは中断せず、クライアント側も応答を待ち続ける点が、前述のタイマー例外パターンとは異なる。
ビジネスプロセス図	**ダイアグラム**	
	シナリオ	① クライアントからの要求を待ち、要求を受け取ったらプロセスを開始する ② サブプロセスを実行する ③ サブプロセスが完了する前に、タイマー中間イベントであらかじめ決められた時間が経過した場合のみ、タイマー通知を送信する ④ サブプロセスが完了したら、クライアントに応答を送信する ※ タイマー通知を送信した場合でも、クライアントに応答を送信する

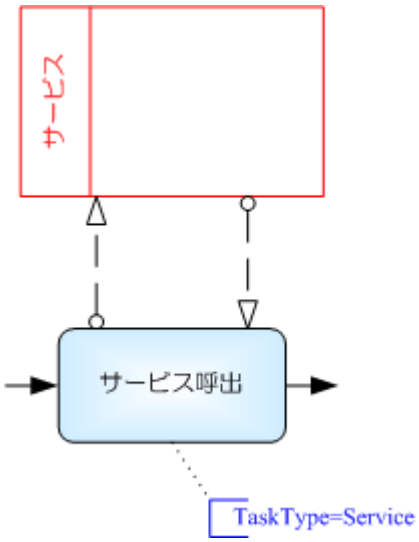
BPEL	<eventHandlers>
	  <onAlarm until="*****">
	  <invoke name="タイマー通知"
	  partnerLink="*****" portType="*****"
	  operation="*****" inputVariable="*****" />
	  </onAlarm>
	</eventHandlers>
	<sequence>
	  <receive name="メッセージ開始イベント"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	  <scope name="サブプロセス">
	  <empty />
	  </scope>
	  <invoke name="メッセージ終了イベント"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	</sequence>

BPEL 説明	タイマー監視を行う<eventHandlers>によって表す。 <eventHandlers>では、<onAlarm>でタイマー監視を行い、タイムアウトが発生した場合には<invoke>でタイマー通知をクライアントに送る。
特記事項	【BPMN 仕様】 ◇ タイムアウトの発生タイミングは、タイマー中間イベントの BPMN 属性に設定する。あらかじめ定められた日付または時刻を設定する場合には TimeDate 属性に設定し、タイマー中間イベントを境界に配置したアクティビティの開始からの経過時間を設定する場合には TimeCycle 属性に設定する。 【BPEL マッピング】 ◇前記の BPEL コードは、タイムアウトの発生タイミングをタイマー中間イベントの TimeDate 属性にした例である。TimeCycle 属性に設定した場合には、前記の<onAlarm until="*****">要素の代わりに<onAlarm for="*****">要素にマッピングされる。

【パターン適用例】

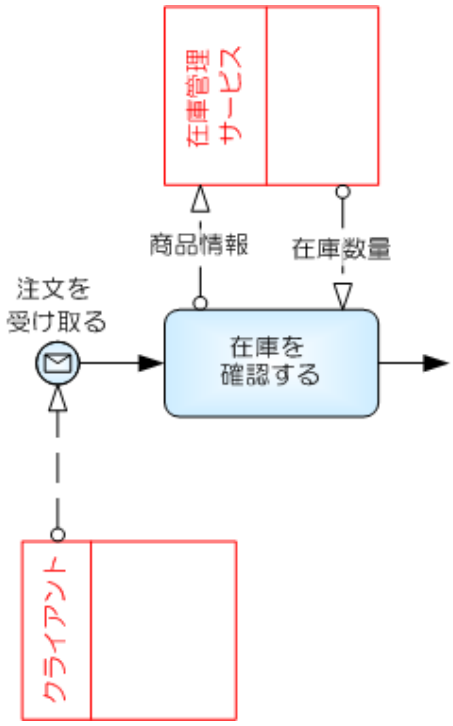


① クライアントからの注文を受け取る
（注文を受け付けるとプロセスをインスタンス化し、プロセスを開始する）
② 配送管理サービスを呼び出し、商品配送を依頼する
③ 入金管理サービスからの入金通知の受信を待つ
④ 入金通知の受信前に、注文受付後 10 日間の期限がきた場合には、クライアントに入金督促通知を送る
⑤ 入金管理サービスから入金通知を受信したら、プロセスを終了する

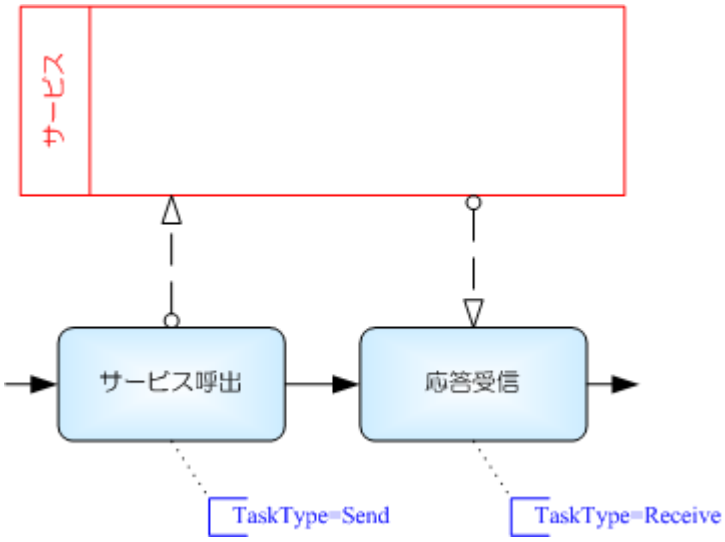
<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス呼出</td></tr>
<tr><td colspan="2">パターン名称</td><td>同期型要求</td></tr>
<tr><td colspan="2">目的</td><td>同期型 Web サービスを呼び出して要求を送信し、その応答を受信するといった同期型メッセージの送受信を表すために使用する。要求を送信したら、応答を受信するまで待機し、その間は他の処理を実行することができない。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① サービスを呼び出し、メッセージを送信する
② メッセージを送信したら、その応答メッセージの受信を待機する
③ 応答メッセージを送信したら、後続のフローに進む</td></tr>
</table>

BPEL	<invoke name="サービス呼出" partnerLink="*****" portType="*****" operation="*****" variable="*****" />
BPEL 説明	同期型 Web サービスの呼び出しは、<invoke>でメッセージの送信、その応答メッセージの受信を表す。
特記事項	

【パターン適用例】



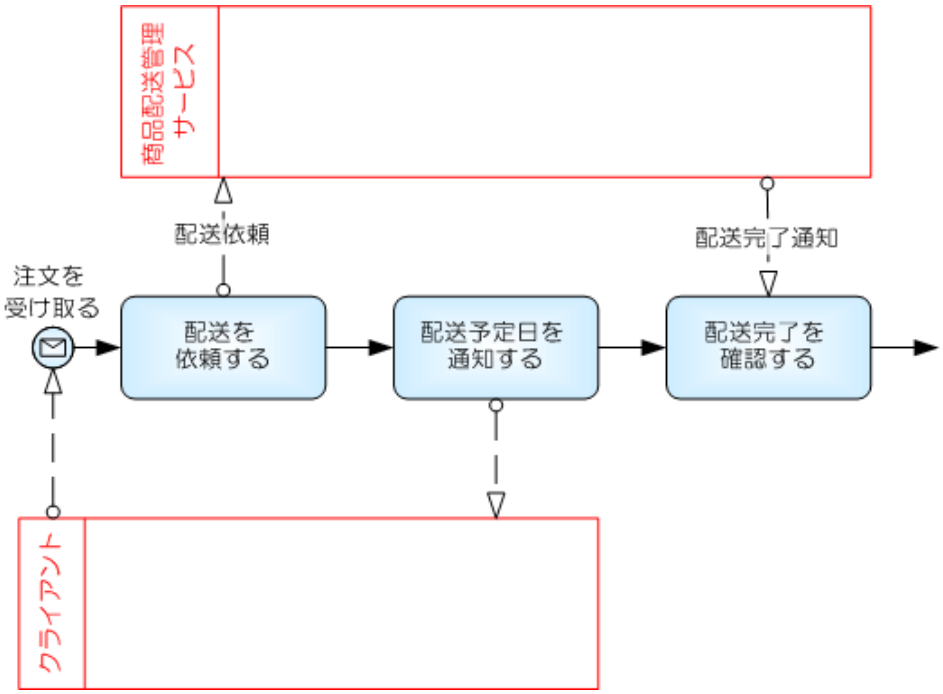
① クライアントからの注文を受け取る
② 在庫管理サービスを呼び出し、商品情報メッセージを送信する
③ 在庫管理サービスから応答メッセージ(在庫数量)の受信を待機する
④ 応答メッセージを受信したら、後続のフローに進む

<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス呼出</td></tr>
<tr><td colspan="2">パターン名称</td><td>非同期型要求</td></tr>
<tr><td colspan="2">目的</td><td>非同期型 Web サービスを呼び出して要求を送信し、その応答を受信するといった非同期型のメッセージ送受信を表すために使用する。要求を送信したら後続のフローに進み、応答を受信するタスクまで進むと応答を待機するため、送受信の間に他の処理を実行することもできる。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① サービスを呼び出し、メッセージを送信する
② メッセージを送信したら、その後続のフローに進み、「応答受信タスク」まで進むと応答メッセージを待つ
③ 応答メッセージを受信したら、その後続のフローに進む</td></tr>
</table>

BPEL	<sequence>
	  <invoke name="サービス要求"
	  partnerLink="*****" portType="*****"
	  operation="*****" inputVariable="*****" />
	  <receive name="応答受信"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	</sequence>

BPEL 説明	非同期型 Web サービスの呼び出しとメッセージの送信は<invoke>を使用し、その応答メッセージの受信は<receive>を使用して表す。
特記事項	

【パターン適用例】

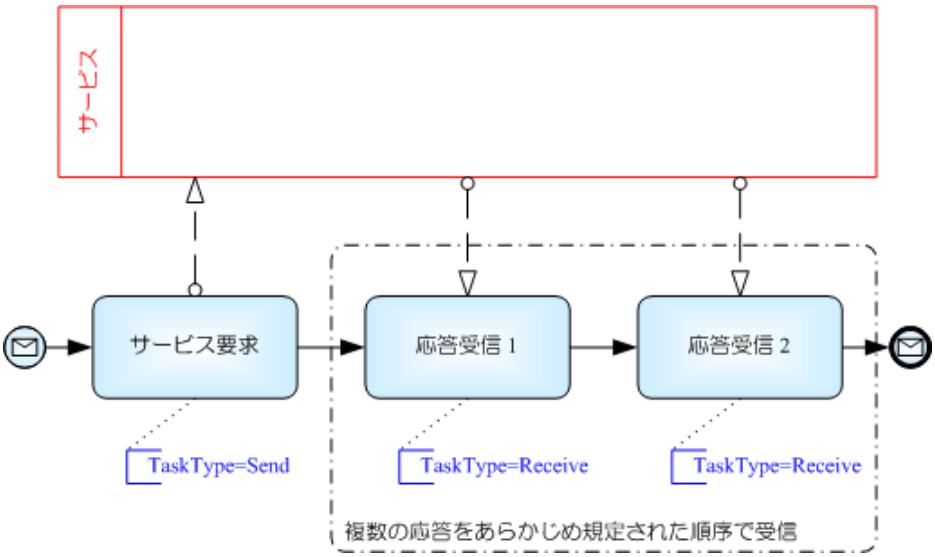


① クライアントからの注文を受け取る

② 商品配送管理サービスを呼び出し、配送依頼メッセージを送信する

③ クライアントに配送予定日を通知する

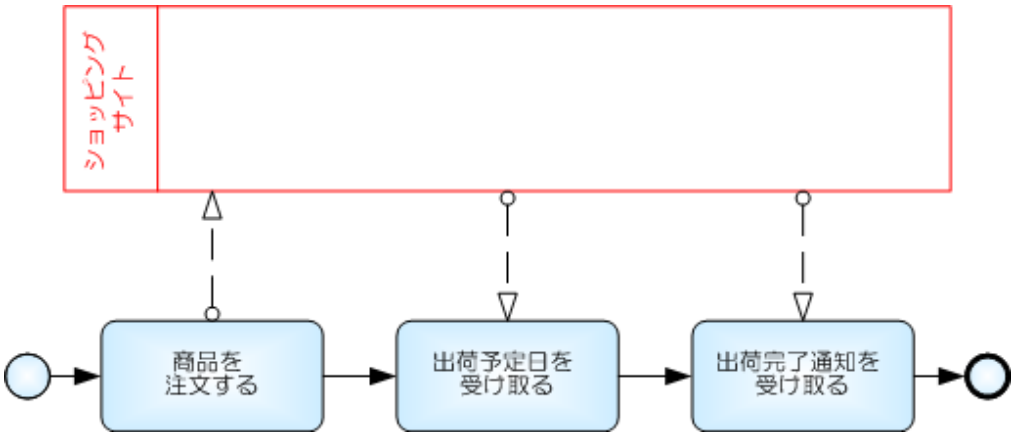
④ ②の配送依頼メッセージに対する応答を待ち、応答メッセージを受信したら、後続のフローに進む

<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス呼出</td></tr>
<tr><td colspan="2">パターン名称</td><td>複数応答受信(順次)</td></tr>
<tr><td colspan="2">目的</td><td>非同期型 Web サービスを呼び出して要求を送信し、複数の応答をあらかじめ規定された順序で受信するといったメッセージの送受信を表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① サービスを呼び出し、メッセージを送信する
② メッセージを送信したら、「応答受信1タスク」で応答メッセージを受信する
③ 「応答受信1タスク」で応答メッセージを受信したら、「応答受信 2 タスク」で応答メッセージを受信する</td></tr>
</table>

BPEL	<sequence>
	  <invoke name="サービス要求"
	  partnerLink="*****" portType="*****"
	  operation="*****" inputVariable="*****" />
	  <receive name="応答受信 1"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	  <receive name="応答受信 2"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	</sequence>

BPEL 説明	非同期型 Web サービスの呼び出しとメッセージの送信は<invoke>を使用し、その応答メッセージの受信は<receive>を使用して表す。 複数の応答を受信するために複数記述される<receive>は、あらかじめ規定された順序に従い<sequence>と</sequence>で囲み、応答を 1 メッセージずつ順番に受信する。
特記事項	

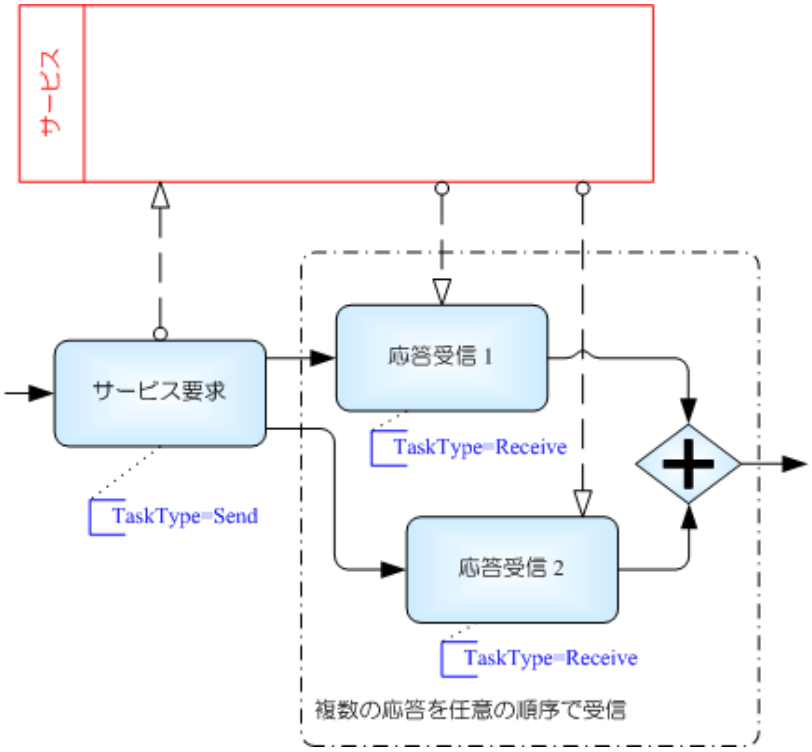
【パターン適用例】



① ショッピングサイトにメッセージを送信し、商品を注文する

② ショッピングサイトから応答メッセージ(出荷予定日)を受信する

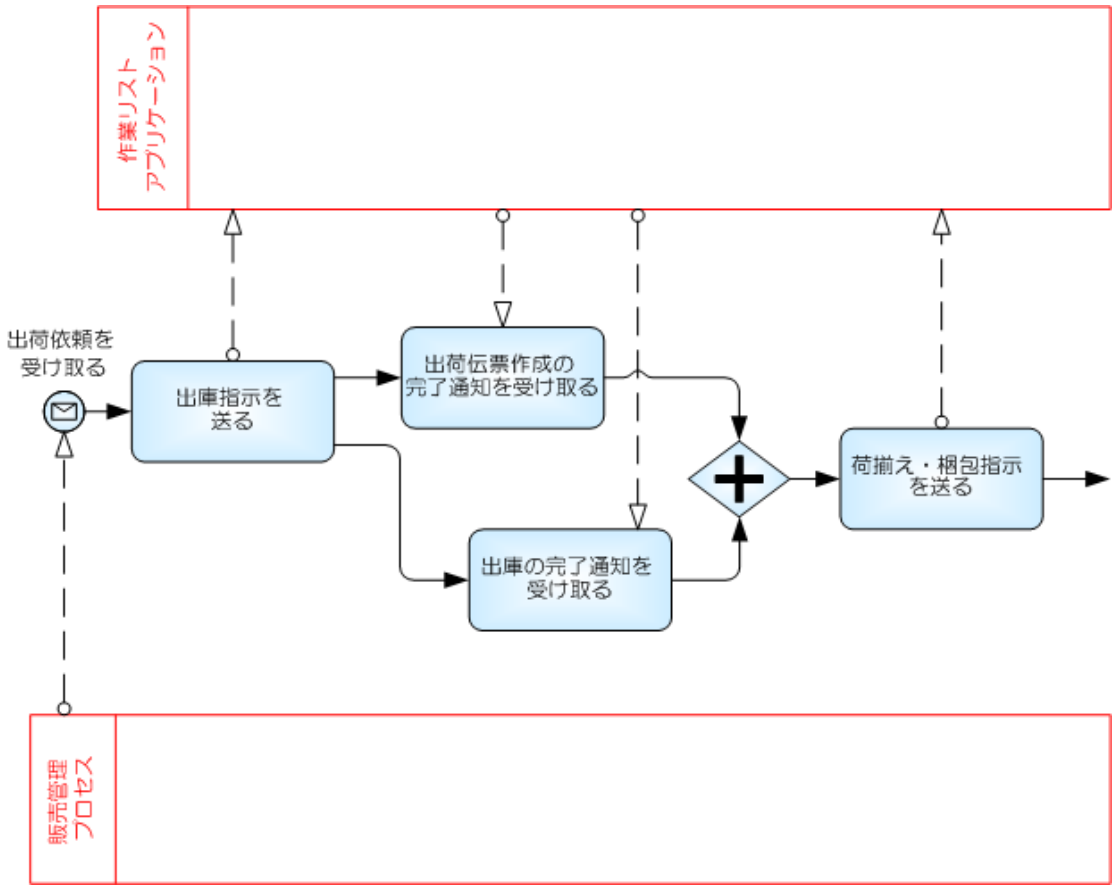
③ ショッピングサイトから応答メッセージ(出荷完了通知)を受信する

<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス呼出</td></tr>
<tr><td colspan="2">パターン名称</td><td>複数応答受信（順不同）</td></tr>
<tr><td colspan="2">目的</td><td>非同期型 Web サービスを呼び出して要求を送信し、複数の応答を任意の順序で受信するといったメッセージの送受信を表すために使用する。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① サービスを呼び出し、メッセージを送信する
② メッセージを送信したら、「応答受信 1」タスクと「応答受信 2」タスクを同時並列で開始し、①で送信したメッセージに対する応答を待つ
③「応答受信 1」タスクと「応答受信 2」タスクが受信する応答メッセージのうち、先に送信されてきたメッセージを受信する
④ ③で受信しなかった応答メッセージを待ち続け、その応答メッセージを受信したら後続のフローに進む</td></tr>
</table>

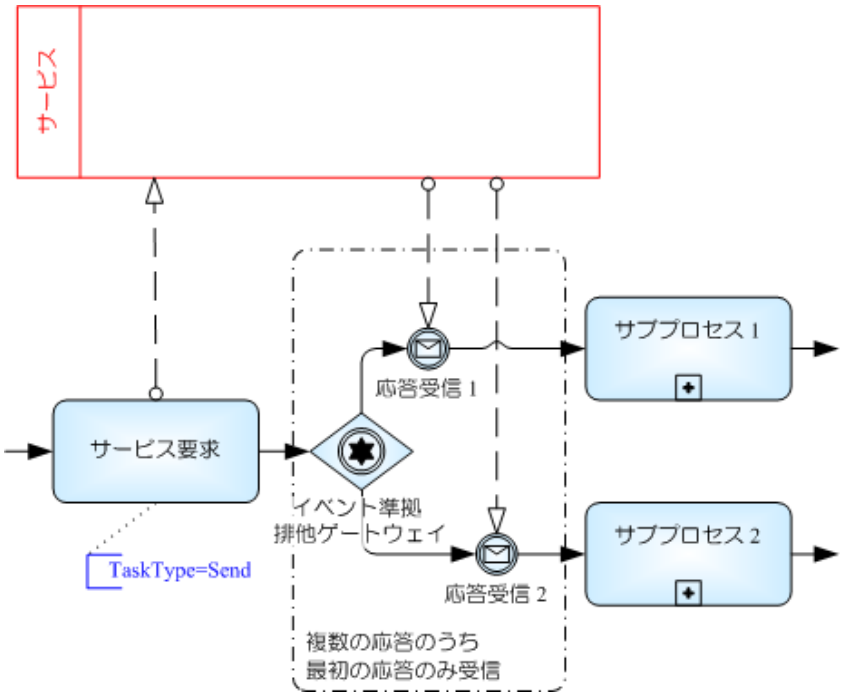
BPEL	<sequence>
	  <invoke name="サービス要求"
	  partnerLink="*****" portType="*****"
	  operation="*****" inputVariable="*****" />
	  <flow>
	  <receive name="応答受信 1"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	  <receive name="応答受信 2"
	  createInstance="yes"
	  partnerLink="*****" portType="*****"
	  operation="*****" variable="*****" />
	  </flow>
	</sequence>

BPEL 説明	非同期型 Web サービスの呼び出しとメッセージの送信は<invoke>を使用し、その応答メッセージの受信は<receive>を使用して表す。 複数の応答を受信するために複数記述される<receive>は、<flow>と</flow>で囲み、同時に応答を待機する。
特記事項	

【パターン適用例】

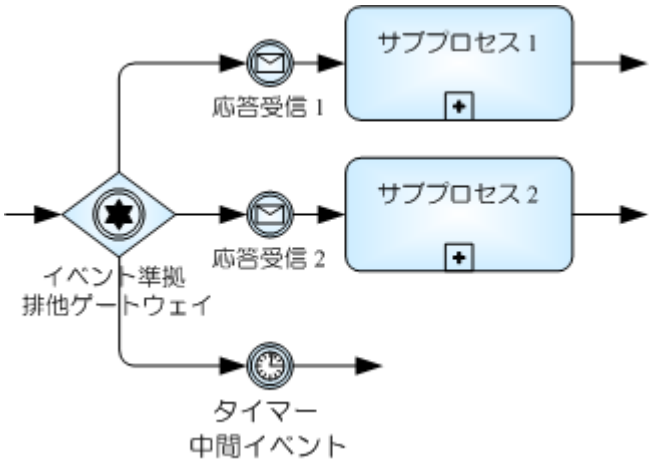


① 販売管理プロセスから出荷依頼を受け取る
② 作業リストアプリケーションに作業指示を送信する
③ 作業リストアプリケーションから「出荷伝票作成の完了通知」または「出庫の完了通知」が送信されてくるのを待機する
④ 「出荷伝票作成の完了通知」または「出庫の完了通知」の何れかが送信されてきたら、その通知を受信し、もう一方の通知が送信されてくるのを待機する
⑤ もう一方の通知も送信されてきたら、その通知を受信し、作業リストアプリケーションに荷揃え・梱包指示を送信する

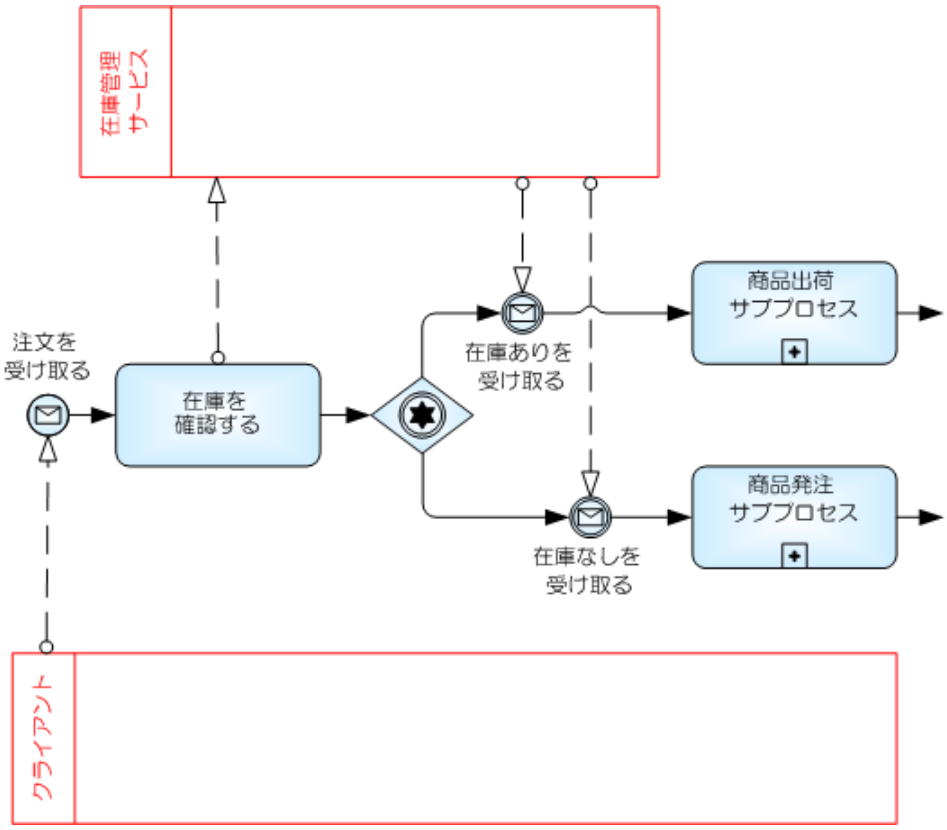
<table>
<tr><td colspan="2">カテゴリ</td><td>サービス連携　サービス呼出</td></tr>
<tr><td colspan="2">パターン名称</td><td>複数応答受信（選択）</td></tr>
<tr><td colspan="2">目的</td><td>非同期型 Web サービスを呼び出して要求を送信し、複数の応答のうち、最初の応答のみ受信するといったメッセージの送受信を表すために使用する。複数のメッセージのうち、何れのメッセージを受信するかによって後続フローを決定といった排他的選択の役割も持つ。</td></tr>
<tr><td rowspan="2">ビジネスプロセス図</td><td>ダイアグラム</td><td> 
</td></tr>
<tr><td>シナリオ</td><td>① サービスを呼び出し、メッセージを送信する
② メッセージを送信したら、「応答受信 1」イベントと「応答受信 2」イベントで、①で送信したメッセージに対する応答を待つ
③ 「応答受信 1」イベントと「応答受信 2」イベントが受信する応答メッセージのうち、先に送信されてきたメッセージを受信する
④ ③で応答を受信したイベントの後続サブプロセスを実行する
④-1 「応答受信 1」イベントで受信した場合には、「サブプロセス 1」を実行する
④-2 「応答受信 2」イベントで受信した場合には、「サブプロセス 2」を実行する
※ その後、③で応答を受信しなかった応答が送られてきても受信せず、そのイベントの後続サブプロセスも実行されない</td></tr>
</table>

BPEL	<sequence>
	  <invoke name="サービス要求"
	  partnerLink="*****" portType="*****"
	  operation="*****" inputVariable="*****" />
	  <pick name="イベント準拠排他ゲートウェイ" createInstance="no">
	  <onMessage partnerLink="*****" portType="*****"
	  operation="*****" variable="*****">
	  <scope name="サブプロセス 1">
	  <empty />
	  </scope>
	  </onMessage>
	  <onMessage partnerLink="*****" portType="*****"
	  operation="*****" variable="*****">
	  <scope name="サブプロセス 2">
	  <empty />
	  </scope>
	  </onMessage>
	  </pick>
	</sequence>

BPEL 説明	非同期型Webサービスの呼び出しとメッセージの送信は<invoke>を使用し、複数の応答メッセージの排他的な選択受信は<pick>を使用して表す。 <pick>内に配置した<onMessage>で各応答メッセージの受信を表し、<onMessage>内には、該当する応答メッセージの受信に後続するフローを記述する。

特記事項	**【BPMN 仕様】** ◇ 複数応答受信（選択）では、応答受信タイムアウトが発生した場合の例外的な振る舞いを表すために前述のタイマー例外パターンを使用せず、下図のようにイベント準拠排他ゲートウェイとタイマー中間イベントをシーケンスフローで接続する。  ◇ イベント準拠排他ゲートウェイに後続する中間イベントは、メッセージ、タイマーの他、ルール、リンク、エラーを使用することができる。 ◇ メッセージ中間イベントを TaskType 属性が Receive であるタスクで代替することができる。 **【BPEL マッピング】** ◇ イベント準拠排他ゲートウェイとタイマー中間イベントをシーケンスフローで接続して応答受信タイムアウトを表す場合には、**<pick>**内に下記の**<onAlerm>**が追加される。 **<onAlarm for="*****">または<onAlarm until="*****"> <!--タイムアウト時の処理--> </onAlarm>** ◇ イベント準拠 XOR ゲートウェイに後続するイベントとして、ルール中間イベント、リンク中間イベントを使用した場合には、ルールが満たされたこと、またはリンクにトークンが発生したことをメッセージとして受け取ることを前提としている。BPEL マッピングはメッセージ中間イベントと同様、**<onMessage>**にマッピングされる。 ◇ BPMN 仕様ではイベント準拠 XOR ゲートウェイに後続するイベントとしてエラー中間イベントを使用できるが、BPEL マッピングの対象外となる。

【パターン適用例】



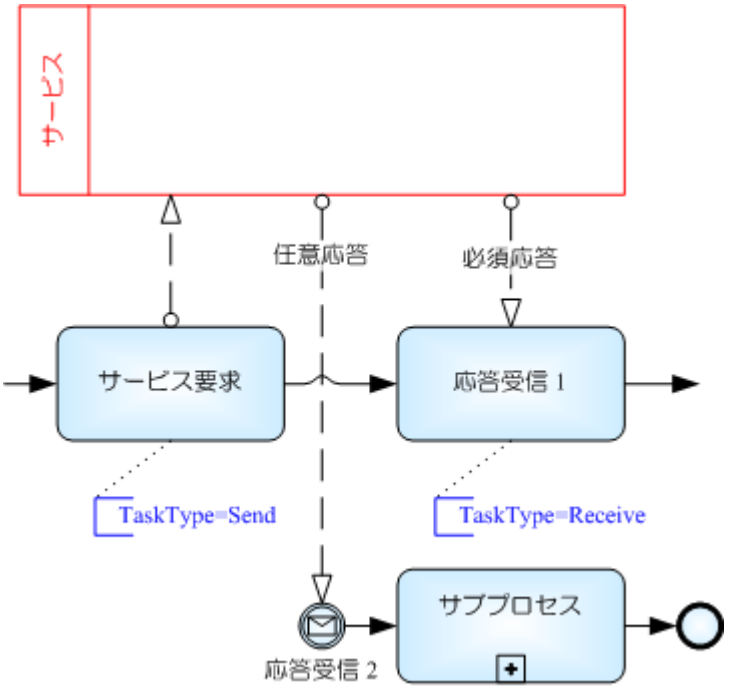
① クライアントからの注文を受け取る

② 在庫管理サービスを呼び出し、メッセージを送信する

③ 在庫管理サービスからの応答メッセージを待ち、その応答によって後続の作業を決定する

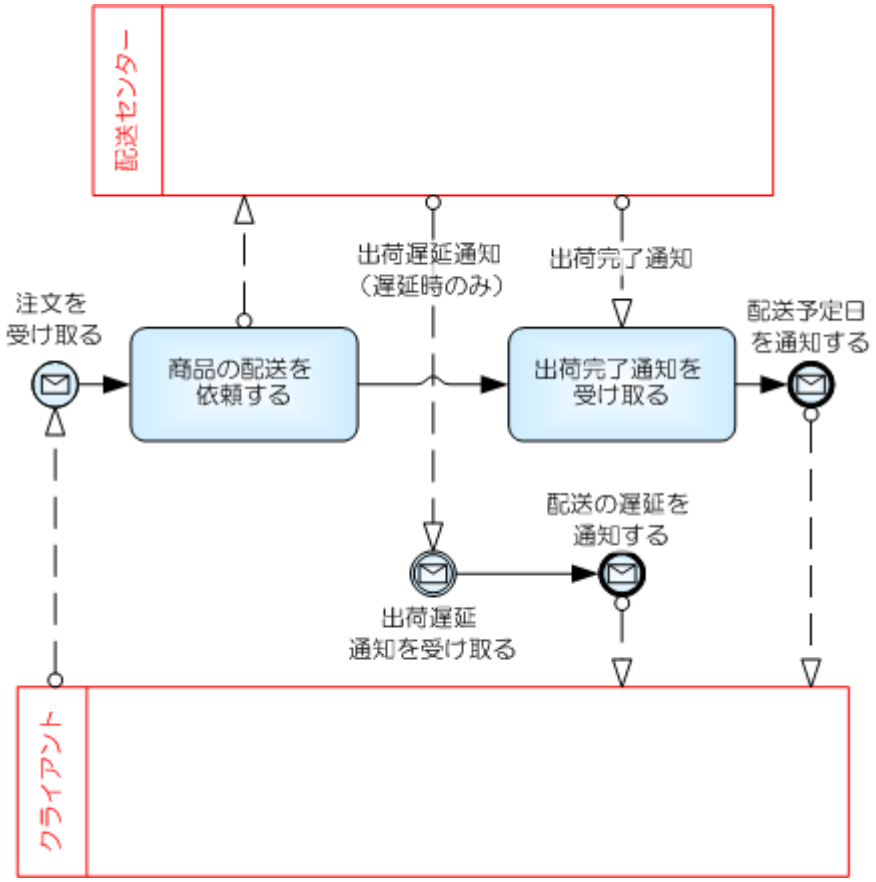
③-1 在庫ありを受け取った場合には、商品出荷サブプロセスを実行する

③-2 在庫ありを受け取った場合には、商品発注サブプロセスを実行する

カテゴリ		サービス連携　サービス呼出
パターン名称		複数応答受信（任意応答）
目的		非同期型 Web サービスを呼び出して要求を送信し、必ず送られてくる必須応答の他、必要に応じて送られてくる任意応答も受信するといったメッセージの送受信を表すために使用する。
ビジネスプロセス図	**ダイアグラム**	
	シナリオ	① サービスを呼び出し、メッセージを送信する ② メッセージを送信したら、「応答受信 1」タスクを開始し、必須応答の受信を待つ ③「応答受信 1」タスクと「応答受信 2」イベントが受信する応答メッセージのうち、先におくられてきたメッセージを受信する ③-1「応答受信 1」タスクが受信した場合には、その後続フローに進む ③-2「応答受信 2」イベントが受信した場合には、サブプロセスを実行する ④ ③で受信しなかった応答メッセージを待ち続ける ④-1「応答受信 1」タスクが受信しなかった場合には必ず待ち続け、受信したら、その後続フローに進む ④-2「応答受信 2」タスクが受信しなかった場合には、プロセスの終了まで待ち続け、終了前に受信した場合のみサブプロセスを実行する

BPEL	<eventHandlers> <onMessage partnerLink="*****" portType="*****" operation="*****" variable="*****"> <scope name="サブプロセス"> <empty /> </scope> </onMessage> </eventHandlers> <sequence> <invoke name="サービス要求" partnerLink="*****" portType="*****" operation="*****" inputVariable="*****" /> <receive name="応答受信 1" partnerLink="*****" portType="*****" operation="*****" variable="*****" /> </sequence>
BPEL 説明	非同期型 Web サービスの呼び出しとメッセージの送信は<invoke>、必須応答の受信は<receive>、任意応答の受信は<eventHandler>内の<onMessage>を使用して表す。
特記事項	

【パターン適用例】



① クライアントからの注文を受け取る

② 「配送センター」プロセスにメッセージを送信し、商品の配送を依頼する

③ 「出荷完了通知」と「出荷遅延通知」のうち、先におくられてきたメッセージを受信する

③-1 出荷完了通知を受信した場合には、クライアントに配送予定日を送信し、プロセスを終了する

③-2 出荷遅延通知を受信した場合には、クライアントに配送の遅延を通知し、「出荷完了通知」を待ち続ける

６　BPMN と BPEL のマッピングに関する補足

６．１　BPEL にマッピングされない BPMN 要素

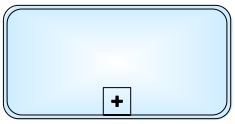
ローレベル BPMN パターンは、BPMN 仕様において BPEL とのマッピングが定義されているものであり、その定義の全てに対応したツールであれば BPEL を生成することができる。したがって、本書で記述されたパターンを組み合わせた BPMN ダイアグラムも、基本的には BPEL を生成することができる。ただし、ローレベル BPMN パターンの列挙にあたって可能な限りパターン網羅性を意識してきたが、まだ初版であり完全な網羅性を確保できていない可能性が大きい。今後、列挙したパターンをミドルレベル BPMN モデルに適用しながら検証し、網羅性を高めるように本書を改訂していく予定である。

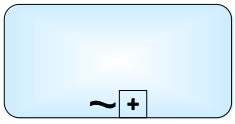
そこで、現段階では、BPEL 生成可能な BPMN ダイアグラムを設計するために、どのような BPMN モデルが BPEL を生成できないかも把握しておくことが重要と考える。以下に BPEL にマッピングされない BPMN 要素を列挙する。

（１）イベント

キャンセル中間イベント: 

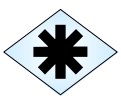
（２）アクティビティ

トランザクション サブプロセス: 

アドホック サブプロセス: 

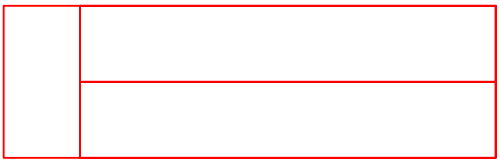
参照タスク:  （TaskType 属性＝Reference）

（３）ゲートウェイ

複合ゲートウェイ: 

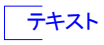
（４）スイムレーン

プール： 

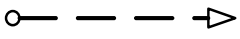
レーン： 

（５）成果物

データオブジェクト： 

注釈： 

（６）接続オブジェクト

メッセージフロー： 

【注記】

Web サービスとやり取りするメッセージは、タスクの BPMN 属性（Message 属性、WebService 属性など）に Web サービスの WSDL で規定された情報を設定する。

メッセージフローが BPEL にマッピングされるのではなく、メッセージフローは Web サービスとのやり取りをモデルとして可視化するためにのみ使用される。

関連：

7　参考文献

[1] Introduction to BPMN、Stephen A. White, IBM Corporation、2004.5、BPMN.org

[2] Business Process Modeling Notation（BPMN） Version1.0、Stephen A. White ,et al、2004.3、BPMN.org

[3] BPMN の概要（Introduction to BPMN 和訳版）、日揮情報ソフトウェア、2004.5

[4] BPMN を活用したビジネスプロセス・モデリング、日揮情報ソフトウェア、2005.3、@IT

【BPMN 研究会】

目的

ビジネスプロセスの分析/設計から実装に至る一連の設計仕様を表記できる新しいダイアグラム技法『BPMN（Business Process Modeling Notation）』に関して、研究会メンバーによる実践を通じて活用するための課題と解決策を研究する

設立：2005 年 1 月

主査：日揮情報ソフトウエア株式会社 岩田アキラ

副主査：NECネクサソリューションズ株式会社 コンサルティング部 冨澤 雅彦

【執筆者】

『２ ビジネスプロセス設計のプロセス』

NECネクサソリューションズ株式会社 コンサルティング部 冨澤 雅彦

『３ BPMN 概説』

株式会社日立製作所 システム開発研究所 uVALUE イノベーションセンター 熊谷 貴禎

『４ BPEL 概説』

日本ユニシス株式会社 ビジネス・イノベーション・オフィス 原園 耕路

『５ ローレベル BPMN パターン』

『６ BPMN と BPEL のマッピングに関する捕捉』

日揮情報ソフトウエア株式会社 技術本部 明庭 聡