プロセス モデリング表記法とワークフロー パターン

Stephen A. White, IBM Corp., United States

オリジナル英文: http://tmitwww.tm.tue.nl/research/patterns/patterns.htm
日本語訳: 2004 年 9 月 日揮情報ソフトウェア株式会社

要約

Wil van der Aalst、Arthur ter Hofstede、Bartek Kiepuszewski、および Alistair Barros による研究により、ビジネス プロセスの振る舞いを記した 21 のパターンが定義されました。ここでは、2 つのプロセス モデリング表記法(Business Process Management Initiative (BPMI)の BPMN ビジネス プロセス図と Object Management Group (OMG)の UML 2.0 アクティビティ図)を用い、両者のワークフロー パターンの描き方についてレビューします。読みやすさに加え、パターンを表すための技術的な能力についても比較します。

序論

Wil van der Aalst、Arthur ter Hofstede、Bartek Kiepuszewski、および Alistair Barros による研究により、ワークフロー サーバーがビジネス プロセスの遂行中に起こし得る振る舞いを記述する目的で、ビジネス プロセスの振る舞いを記した 21 のパターンが定義されました。パターンは、シンプルなものから非常に複雑なものまで多岐にわたり、ビジネス プロセス モデルで見られる振る舞いをほとんど網羅しています。研究者たちが構築したウェブサイト[1]には、これらのパターンに関する説明、例、論文、およびパターンをサポートするワークフロー製品についての評価が掲載されています。

この論文の目的は、2 つのモデリング記法(BPMI の BPMN ビジネス プロセス図と OMG の UML2.0 アクティビティ図)がどれだけワークフロー パターンをグラフィカルに描画できるかを評価することです。それぞれのパターンで、2 つの表記法がどのように扱われているのか比較しています。パターンが技術的および直感的に表されているかに焦点を当てて、比較しています。

BASIC CONTROL PATTERNS

プロセスの振る舞いの簡単な例を、最初の 5 つのパターンで示します。これらは、ビジネス プロセスの

[1] http://tmitwww.tm.tue.nl/research/patterns/patterns.htm

基本的なモデリング パターンを表しています。

SEQUENCE

Sequence パターンは、順序付けされた一連のアクティビティを表すパターンです。前のアクティビティが完了すると、次のアクティビティがスタートします。WfMC2[2]は、この振る舞いを「Sequential Routing」と定義しています。

ビジネス プロセス図

ビジネス プロセス図では、このパターンをシーケンス フローによって接続された一連のアクティビティを使って定義します(図 1 を参照)。シーケンス フローの矢印の方向は、シーケンスの順序を表します。このパターンの振る舞いは、「トークン」という概念を用いて表すことができます。トークンは、ソース オブジェクトからターゲット オブジェクトに向かって、シーケンスの矢印が示す向きに沿って流れていきます。このパターンでは、アクティビティが完了すると、トークンがシーケンス フローを通り、次のアクティビティへと流れていきます。分岐やトークンの制御はありません。BPMN では、トークンの流れ上にある制御(ゲートウェイや条件付きシーケンス フロー)は菱形で表されます。フロー制御の例は、他のワークフロー パターンで紹介します。

この論文で網羅したパターンの振る舞いは、トークンを用いて記述します。

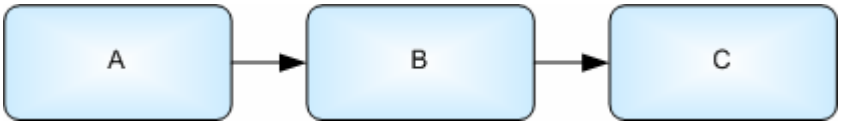

図 1: Sequence——ビジネス プロセス図

アクティビティ図

アクティビティ図では、このパターンをコントロール フローによって接続された一連のアクティビティを使って定義します(図 2 を参照)。コントロール フローの矢印の方向は、シーケンスの順序を表します。ビジネス プロセス図のように、アクティビティが完了すると、トークンがコントロール フローを通り、次のアクティビティへと流れていきます。

図 2: Sequence——アクティビティ図

比較

シーケンス パターンの表記では、ビジネス プロセス図とアクティビティ図の間に主な違いはありませ

[2] The Workflow Management Coalition Terminology & Glossary (1999)

ん。両方とも、アクティビティを角の丸い長方形で表しています。長方形の角の丸みは、モデラーやツール ベンダーによって異なります。両方とも、フローの方向を示すために矢印のついたラインを使用します。ただし、矢印の形は異なります。BPMN ではシーケンス フローを黒塗りの矢印で表しますが、UML ではコントロール フローや標準オブジェクト フローを通常の矢印で表します。UML も黒塗りの矢印をストリーミング オブジェクト フロー エッジに使用します。これは、複数のオブジェクト トークンがアクティビティのインスタンスを新たに生成することなく、アクティビティに入っていけることを指します。このような振る舞いは、BPMN では、関連を使って行ないます(このような振る舞いについての詳細は、UML および BPMN の仕様書を参照してください)。矢印の違いは、以下で記述されたすべてのパターンに当てはめることができます。

PARALLEL SPLIT

Parallel Split パターンは、アクティビティが連続的にではなく、同時に実行されることを可能にするメカニズムです。2 つ以上のアクティビティが同時にスタートするように、プロセスが通るパスは 2 つ以上のパスへと分割されます。WfMC はこの振る舞いを、「AND-Split」と定義しています。

ビジネス プロセス図

ビジネス プロセス図では、Parallel Split パターンを作成するために 3 つのメカニズムを提供します。第 1 のメカニズムでは、フロー オブジェクトは 2 つ以上の出力シーケンス フローを持つことができます(図 3 を参照)。この特別なフロー コントロール オブジェクトは制御されていないため、分岐を必要としません。フローは依存性や制約を受けずにそれぞれのパスを進みます。つまり、フローを制御するパスはないのです。トークンは出力シーケンス フローごとに生成されます。シーケンス フローの分岐は、タスク、サブプロセス、あるいはスタート イベントによって生成されます。

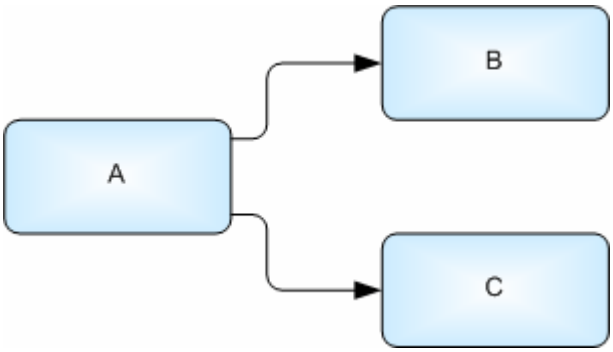

図 3: Parallel Split——ビジネス プロセス図 バージョン 1

第 2 のメカニズムでは、並列ゲートウェイ(内部にプラス記号のある菱形)を使用します(図 4 を参照)。図 3 の中で示したように、複数の出力シーケンス フローを使って同様の振る舞いが表せるため、図 4 で示されるような状況でゲートウェイを使う必要はありません。しかし、モデラーやモデリング ツールによっては、「ベスト プラクティス」として、分岐のためにゲートウェイを使用することがあります。トークンがゲートウェイに到着すると、各出力シーケンス フローからトークンが送り出されます。Exclusive Choice パターンなどでも、この並列ゲートウェイが必要となります(Exclusive Choice パターンの例は図 10 を参照)。

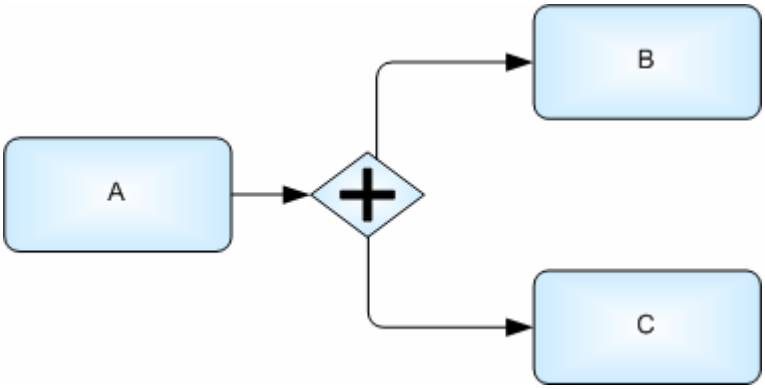

図 4: Parallel Split――ビジネス プロセス図 バージョン 2

第 3 のメカニズムでは、プロセスあるいはサブプロセスがスタート イベントを伴わない場合(これは任意です)、入力シーケンス フローを持っていないアクティビティは、サブプロセスがスタートする時点でスタートします。トークンは、スタートする各アクティビティに供給されます。通常、モデラーは図 5 のように「並列ボックス」を作成してサブプロセスを表します。コンパクトかつ視覚的に区別できる手法で、並列を表しています。

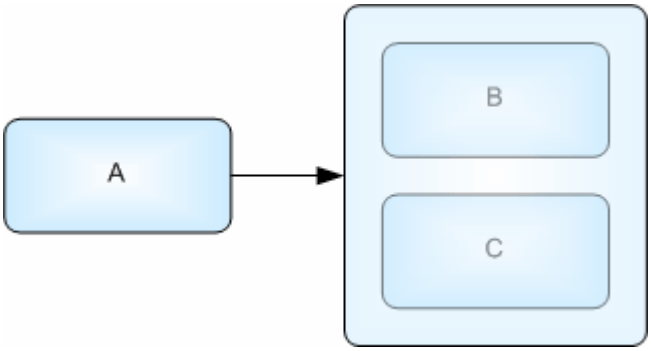

図 5: Parallel Split――ビジネス プロセス図 バージョン 3

アクティビティ図

アクティビティ図では、並列パスを表すためにフォーク ノード(垂直または水平のバー)を使用します(図 6 を参照)。バーから出力されるコントロール フローは、並列フローを示しています。トークンは、出力コントロール フローごとに生成されます。各フローのターゲット アクティビティは、同時にスタートします。

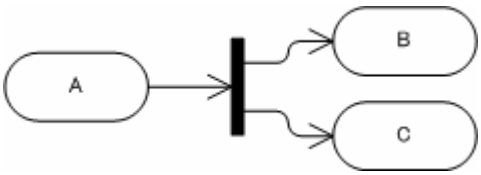

図 6: Parallel Split――アクティビティ図

比較

Parallel Split パターンを提供するためのメカニズムは、2 つの記法間で異なります。ビジネス プロセ

ス図では、複数の出力シーケンス フローおよび「並列ボックス」によって、より単純なメカニズムを提供します。コントロール オブジェクトが必要な場合は、並列ゲートウェイが使用できます。一方、アクティビティ図では、フォーク ノードが必要になります。一般的に、フローを制御するためのアプローチは、2 つの記法間で異なります。ビジネス プロセス図では、アクティビティ間を流れるトークンのフローへの制御または制約を表すために、菱形のみを使用します。内部にある記号は、制御のタイプ(選択や並列など)を示します。菱形が選択されたのは、フローチャートで分岐を表すものとして広く知られていたためです。ビジネス プロセス図では、コアとなるモデリング オブジェクトの数は少なくなっています。アクティビティ図では、トークンのフロー制御を表すために 2 つの異なるタイプのオブジェクト(菱形およびバー)を使用します。これにより、2 つの基本的な制御(選択と並行)を簡単に識別することができます。しかし、より複雑な制御(N out of M join や Synchronization Merge などの並行振る舞い)が必要な場合、両者は明確に区別できません。

SYNCHRONIZATION

Synchronization パターンは、Parallel Split パターンによって分割されたパスを結合します。プロセスを継続するには、事前のアクティビティが完了している必要があります。これは並列パスの「同期」です。WfMC はこの振る舞いを「AND-Join」と定義しています。

ビジネス プロセス図

ビジネス プロセス図には、Synchronization パターン用のメカニズムが 2 つあります。第 1 のメカニズムは、並列ゲートウェイを使います(図 7 を参照)。このゲートウェイは Parallel Split パターンにも用いられ(図 4 を参照)、2 つの機能を同時に実行することができます。ゲートウェイは複数の入力シーケンス フローを受け取ります。各シーケンス フローからすべてのトークンが到着するのを待ってから、ゲートウェイはトークンを通過させます。このメカニズムは、Parallel Split パターンの最初の 2 つのメカニズムに使用されています(図 3 および図 4 を参照)。プロセスのフローは同期化されているため、ここではゲートウェイを使用する必要があります。

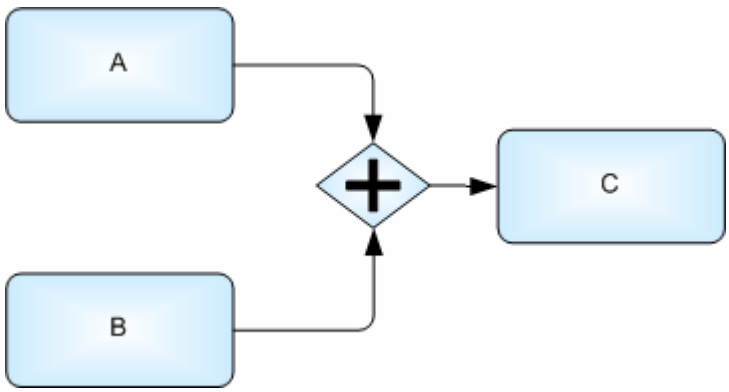

図 7: Synchronization——ビジネス プロセス図 バージョン 1

第 2 のメカニズムは、図 5 のように「並列ボックス」を使う方法です。プロセスやサブプロセスが終了イベントを持っていない場合、出力シーケンス フローを持っていないサブプロセスはすべてのアクティビティが終了した時点で終了します。その後、トークンは包含するプロセス内のサブプロセスを通過します。「並

列ボックス」のメカニズムは、サブプロセスを終了させるメカニズムのサブセットです。サブプロセス内に複数の並列パスがある場合は、終了イベントがなくても、それらのパスは個別に終了することができます。すべての並列パスが終了イベントに達するまで、サブプロセスは完了しません。これはフローの同期です。その後、より高いレベルのプロセスに移動します。

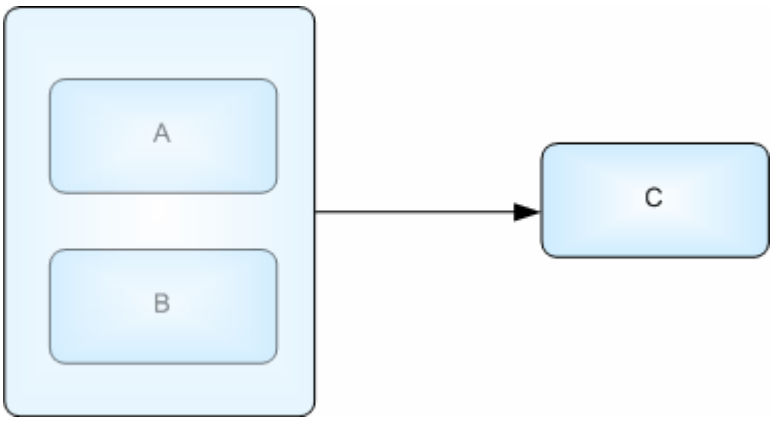

図 8: Synchronization——ビジネス・プロセス図 バージョン 2

アクティビティ図

アクティビティ図は、複数の並列パスを組み合わせるジョイン ノードを使用します(図 9 を参照)。フォーク ノード(図 6 を参照)と同じもののように見えますが、このバーはどちらの機能も持っています。ジョイン ノードは、入力コントロール フローからすべてのトークンが到着することを表しています。その後、1 つのトークンがバーを通過します。

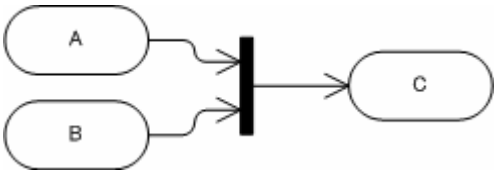

図 9: Synchronization——アクティビティ図

アクティビティ図では、サブ アクティビティを用いて表すこともできます。サブ アクティビティ内に複数の並列パスがある場合は、フロー最終ノードがなくても、それらのパスは個別に終了することができます。すべての並列パスがフロー最終ノードに到着するまで、サブプロセスは完了しません。これはフローの同期です。その後、より高いレベルのプロセスに移動します。

比較

Synchronization パターンを提供するためのメカニズムは、基本的に同じです。しかし、異なる部分もあります。ビジネス プロセス図は、並列ゲートウェイを使用します。一方、アクティビティ図はジョイン ノードを使用します。どちらにも、サブプロセスを使った同期化メカニズムがあります。フロー制御オブジェクトの各利点については、Parallel Split パターンの比較で議論しています。

EXCLUSIVE CHOICE

Exclusive Choice パターンは、フローが複数の排他的な代替パスへと「分割している」プロセスがある状況を指します。このパターンは、プロセスを継続させるために複数の代替パスから 1 つだけ選ぶという点で排他的と言えます。WfMC はこの振る舞いを「OR-Split」と定義しています。

ビジネス プロセス図

Exclusive Choice パターンを作成するには、ゲートウェイを追加する必要があります。具体的には、排他的なデータに準拠したゲートウェイ(Exclusive Data-Based Gateway)が、このパターンに使用されます(図 10 を参照)。ゲートウェイから出力されるシーケンス フローは論理式を持っており、プロセスがどのシーケンス フローで継続されるかが評価されます。トークンがゲートウェイに到着した時点で、論理式は(規定の手順で)評価されます。評価が真であれば、それに対応するシーケンス フローが選ばれ、トークンは選ばれたパスを流れていきます。ゲートウェイに到着したトークンの中から、1 つのトークンだけがゲートウェイを出て行きます。

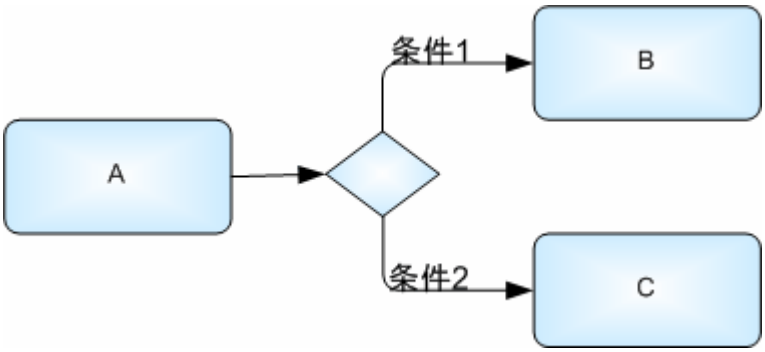

図 10: Exclusive Choice——ビジネス プロセス図

アクティビティ図

アクティビティ図は、分岐ノードを使用して、複数の排他的な代替パスを作成します(図 11 を参照)。分岐ノードから出力されるコントロール フローは論理式を持っており、プロセスがどのコントロール フローで継続されるかが評価されます。トークンがノードに到着した時点で、論理式は(規定の手順で)評価されます。評価が真であれば、それに対応するコントロール フローが選ばれ、トークンは選ばれたパスを流れていきます。ノードに到着したトークンの中から、1 つのトークンだけが分岐ノードを出て行きます。

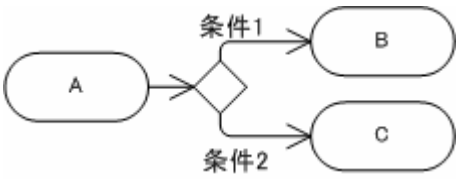

図 11: Exclusive Choice——アクティビティ図

比較

このパターンでは、2 つの記法はほぼ同じ振る舞いをしています。どちらも、菱形を使用して分岐を表

現します。

SIMPLE MERGE

Simple Merge パターンは、複数の代替パスが単一のパスへと連結されるプロセスがある状況を定義します。WfMC はこの振る舞いを「OR-JOIN」と定義しています。

ビジネス プロセス図

Simple Merge パターンは、制御を必要としません。トークンは、2 つのアクティビティ間を制御機構(ゲートウェイ)を介さずに進むことができます。入力シーケンス フローのうちの 1 つだけが実行時にトークンを持つため、複数の入力シーケンス フローが許されています(図 12 を参照)。トークンが到着すると、アクティビティはすぐにスタートします。

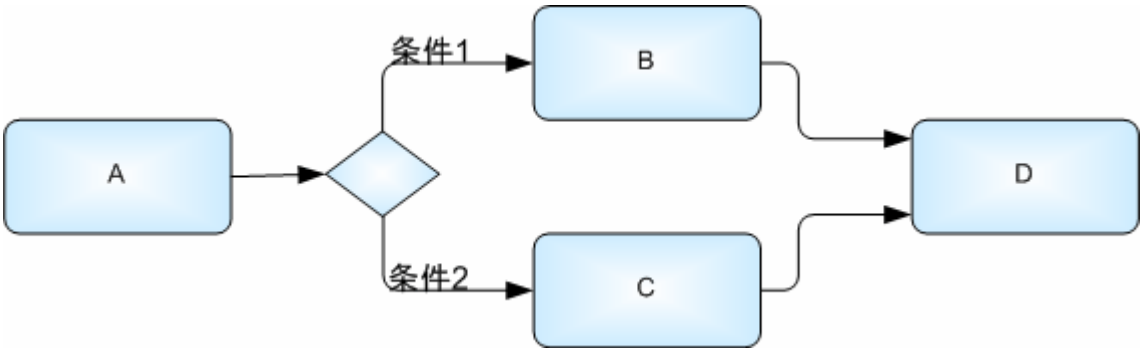

図 12: Simple Merge——ビジネス プロセス図 バージョン 1

排他的ゲートウェイも代替パスを結合するために使用できます(図 13 を参照)。モデラーは「ベスト プラクティス」として、こちらの方が使用しやすいと感じるかもしれません。ゲートウェイに到着した複数のトークンの中から 1 つだけが選ばれる場合、トークンはすぐに出力シーケンス フローを流れていきます。複数のトークンがゲートウェイに到着するモデルを作成する場合は、ゲートウェイが識別者(Discriminator)の役割を果たすことに注意してください(以下を参照)。

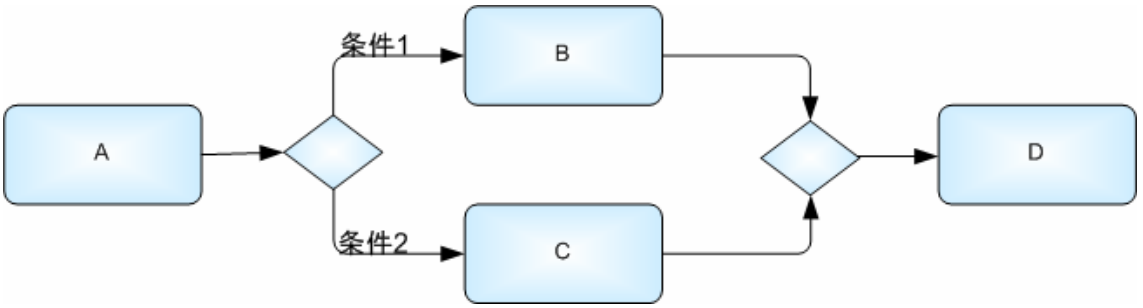

図 13: Simple Merge——ビジネス プロセス図 バージョン 2

アクティビティ図

アクティビティ図は、合流ノードを使用して、複数の代替パスを連結します(図 14 を参照)。図の中で示

される菱形は、分岐ノードと同じもののように見えます。実際、この菱形はどちらの働きもすることができます。トークンがノードに到着すると、トークンは出力コントロール フローを流れていきます。合流ノードは必ずしも必要ではありません。複数の入力コントロール フローを直接アクティビティに接続することもできますが、これはアクティビティ図の振る舞いのモデルとして、適切な方法ではありません。

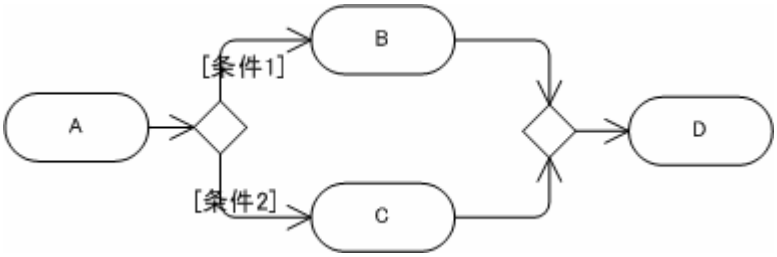

図 14: Simple Merge——アクティビティ図

比較

標準的な方法として、ビジネス プロセス図は複数の入力シーケンス フローといった単純なメカニズムで Simple Merge パターンを表します。排他的ゲートウェイを使用することもできます。これはアクティビティ図にも同様に当てはまります。

ADVANCED BRANCHING AND SYNCHRONIZATION PATTERNS

次の 5 つのパターンでは、ビジネス プロセスのフローの分割や連結などの、より複雑な方法を表しています。

MULTIPLE CHOICE

Multiple Choice パターンは、実行時に複数の代替パスが選ばれるという点で、Exclusive Choice パターンとは異なります。パスが 1 つも選ばれないことも技術的には可能ですが、この場合プロセス フローが予期せず止まることを意味しますので、無効となります。

ビジネス プロセス図

ビジネス プロセス図は、Multiple Choice パターンを扱うために 2 つのメカニズムを提供しています。第 1 のメカニズムでは、アクティビティから出力されるシーケンス フローに条件分岐を置きます。このとき、ゲートウェイ オブジェクトは使用されません。ただし、条件分岐による制御があるため、ミニ ゲートウェイ(小さな菱形)がシーケンス フローの開始点に付けられます。ビジネス プロセス図では、モデルのフローの完全性を保証するため、シーケンス フローに 1 つでも条件分岐がある場合は、複数のシーケンス フローを出力する、または、実行時には必ず 1 つ以上の妥当なシーケンス フローを出力することを条件としています。トークンは、条件が真となった出力シーケンス フローごとに生成されます。したがって、以下の図のように、「タスク A」から出力されたシーケンス フローのどちらか一方、あるいは両方が選択されます。

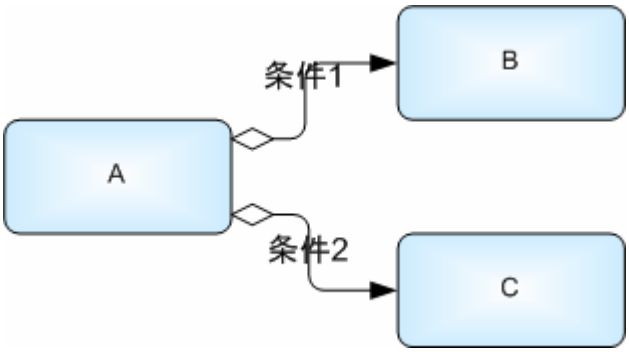

図 15: Multiple Choice——ビジネス プロセス図 バージョン 1

第 2 のメカニズムは、包含ゲートウェイ(内部に円のある菱形)を使用します(図 16 を参照)。ゲートウェイから出力されるシーケンス フローは論理式を持っており、プロセスを継続するために使用されるシーケンス フローが評価されます。トークンがゲートウェイに到着すると、論理式が評価され、真ならば対応するシーケンス フローが選択され、トークンはそのパスを流れていきます。トークンが 1 つ入力されると、出力シーケンス フローごとにトークンが生成されます。

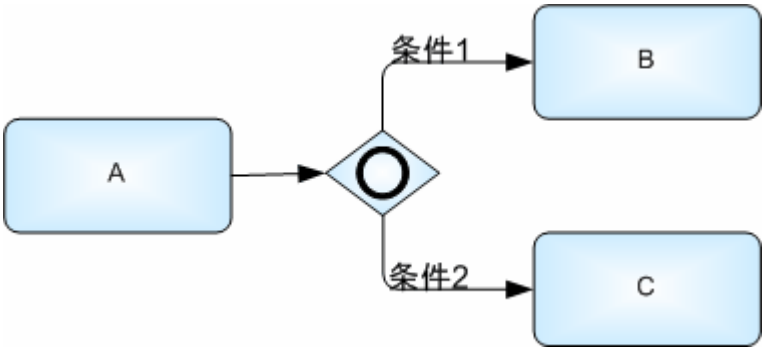

図 16: Multiple Choice——ビジネス プロセス図 バージョン 2

アクティビティ図

アクティビティ図は、フォーク ノードを使用して Multiple Choice パターンを作成し、条件分岐のある出力コントロール フローを表します(図 17 を参照)。フォーク ノードは複数の並列パスを作成します。トークンがノードに到着すると、トークンは条件が真となった出力コントロール フローの数に分割されます。フォーク ノードは必ずしも必要ではありません。アクティビティから出力されるコントロール フローに分岐条件を直接置くこともできます。しかし、この方法では、Multiple Choice パターンは適切にモデル化されているとは言えません。

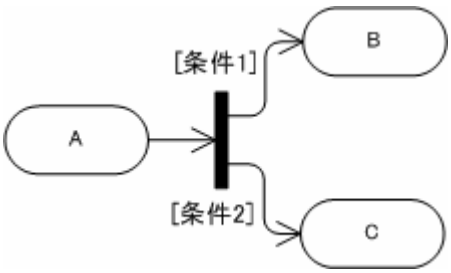

図 17: Multiple Choice——アクティビティ図

比較

2 つの記法の基本的な違いは、ビジネス プロセス図がゲートウェイ(菱形)を使用し、アクティビティ図がフォーク ノードを使用するという点です。アクティビティ図では、分岐ノードが使用されていないのに並列と条件分岐が一緒に表されているため、少し混乱するかもしれません。ビジネス プロセス図では、分岐ポイントを表すために菱形の分岐が使用されます。また、分岐が排他的ではないことを示すために、内部に記号が付けられます。

MULTIPLE MERGE

Multiple Merge パターンは、多数のパスが合流するプロセスのある状況を定義します。ただし、トークンへのフロー制御はありません。トークンがアクティビティに到着すると、そのアクティビティがインスタンス化されます。したがって、多数のパスがトークンを制御しないアクティビティに集まっている場合、マージされるパスごとにアクティビティがインスタンス化されることもあります。さらに、すべてのトークンはそれぞれ独立しており、(もしあれば)残ったプロセスを通り越していきます。Synchronization パターンや Simple Merge パターンと違うのは、Synchronization パターンや Simple Merge パターンのターゲットアクティビティは一度しかインスタンス化されないという点です。ビジネス パーソンは、Multiple Merge パターンのプロセスが暗に重複されていることをすぐに理解できないかもしれません。しかし、これは、ある特定の状況をモデリングするのには大変有益な技術です。

ビジネス プロセス図

Multiple Merge パターンはフロー制御を含んでいないため、ゲートウェイを使用しません。ビジネスプロセス図では、複数のシーケンス フローがアクティビティに入力されることを許可しています。これが、Multiple Merge となります(図 18 を参照)。入力シーケンス フローを流れてくるトークンごとに、アクティビティのインスタンスが生成されます。

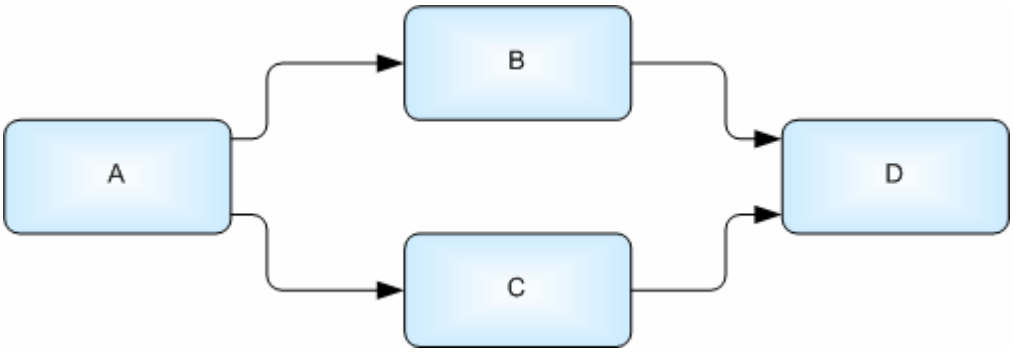

図 18: Multiple Merge——ビジネス プロセス図

アクティビティ図

アクティビティ図は、合流ノードを使用して、複数のパスを合流します(図 19 を参照)。図の中の菱形は分岐ノードと同じもののように見えます。実際、この菱形はどちらの働きもすることができます。トークンがノードに到着すると、出力コントロール フローを流れていきます。ビジネス プロセス図の排他的ゲートウェイとは異なり、合流ノードは図のとおりにマージします。偶然ノードに到着したトークンを止めたりはしません。したがって、もし合流ノードに入るパスが(下記の図のように)分岐していれば、合流ノードの後のアクティビティは複数回にわたってインスタンス化されます。合流ノードは必ずしも必要ではありません。複数のコントロール フローをアクティビティに直接入力することは許可されていますが、これはアクティビティ図の振る舞いをモデル化する適切な方法ではありません。

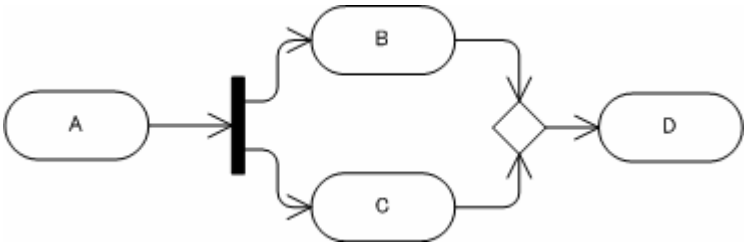

図 19: Multiple Merge——アクティビティ図

比較

ビジネス プロセス図では、このパターンを単純なメカニズムで表すことができます。しかし、このパターンは、どちらの記法を使った場合でも、少し混乱するかもしれません。モデラーには、この振る舞いのモデルに注釈を加えることをお勧めします。

DISCRIMINATOR

Discriminator パターンは、Parallel Split パターンで生成されたパスを組み合わせるもう 1 つの方法です。ここは、複数の並列フローが元に戻るポイントです。Synchronization パターンと異なるのは、最初のトークンがここに到着すると、Discriminator のあるポイントを通過してしまう点です。Parallel Split パターンで生成された残りのトークンは、後から Discriminator に到着しますが、それらは Discriminator によってブロックされます。

ビジネス プロセス図

ビジネス プロセス図では、排他的ゲートウェイを使用して Discriminator パターンの振る舞いを作成します(図 20 を参照)。排他的ゲートウェイに到着するトークンは、1 つ以外すべて排除されます。ゲートウェイは最初のトークンの通過を許可し、残りのトークンをブロックします。

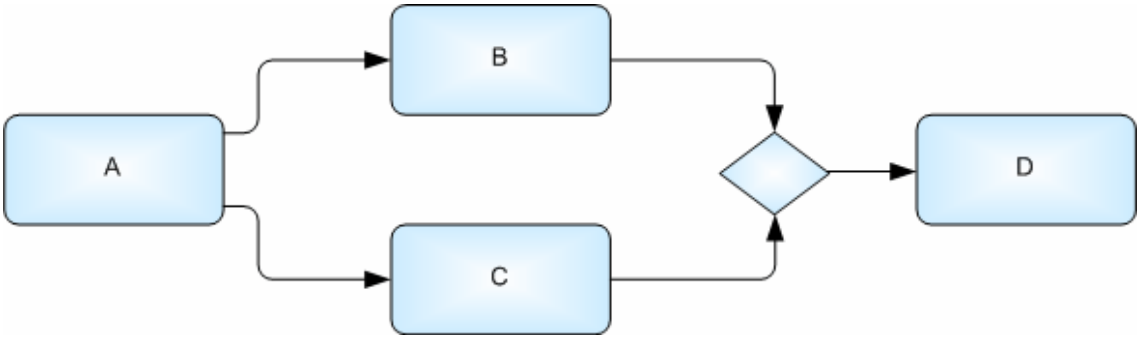

図 20: Discriminator——ビジネス プロセス図

アクティビティ図

アクティビティ図では、複数の機能を使用して Discriminator パターンの振る舞いを作成します(図 21 を参照)。ジョイン ノードはパスを統合するために使用されます。ジョイン ノードには条件式が用意されており、ソース アクティビティが完了したかどうか判断します。最初のトークンがジョイン ノードに到着すると、トークンは出力コントロール フローを流れていきます。ジョイン ノードに到着した他のトークンはブロックされます。

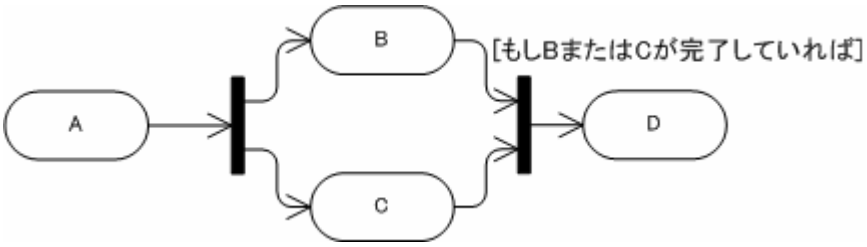

図 21: Discriminator——アクティビティ図

比較

ビジネス プロセス図では、排他的ゲートウェイのマージ機能を使用してこのパターンの振る舞いを表すことができます。アクティビティ図では、このパターンを表すためのメカニズムを持っていません。しかしモデラーは、合流ノードの代わりにジョイン ノードを使って入力コントロール フローを表すことができます。このパターンは、ビジネス プロセス図の方がシンプルかつ自然に表すことができます。

N OUT OF M JOIN

N Out of M Join パターンの振る舞いは、Synchronization パターンや Discriminator パターンにも登場します。合流ポイントを超える必要のある複数の並行した入力トークンの代わりに、モデラーは N Out of M Join パターンを使って、継続して流れるトークンの数を指定できます。残りのトークンは(Discriminator パターンのように)すべてブロックされます。

ビジネス プロセス図

ビジネス プロセス図は、複合ゲートウェイ(内部に「星印」のある菱形)を使用して、N Out of M Join パターンを表します(図 22 を参照)。複合ゲートウェイは、このような状況のために用意されています。モデラーは、ゲートウェイに式を追加できます。ゲートウェイは出力シーケンス フローから出て行くトークン数を決定します。その後到着するトークンは、ゲートウェイによってブロックされます。

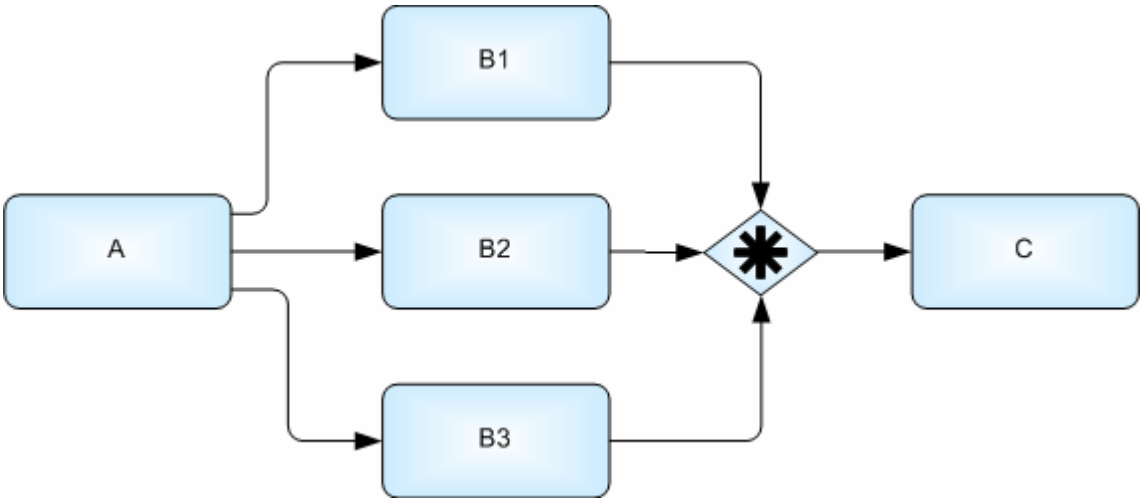

図 22: N Out of M Join——ビジネス プロセス図

アクティビティ図

アクティビティ図は、ジョイン ノードを使用して複数の並列パスを組み合わせます(図 23 を参照)。Synchronization パターンのように、ジョイン ノードは、すべての入力コントロール フローからトークンが到着することを示しています。しかし、ジョイン ノードに追加される条件は、1 つのトークンがバーを通り過ぎる前に、いくつのトークンが必要なのかを決めています。その後到着するトークンは、ジョイン ノードによってブロックされます。

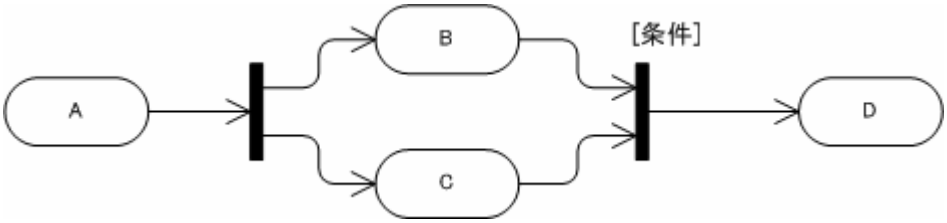

図 23: N Out of M Join——アクティビティ図

比較

基本的に他のワークフロー パターンで使用されるフロー制御メカニズムの比較と同じです。ビジネス プロセス図は、菱形を使用して適切な振る舞いを示します。アクティビティ図は、フロー制御に菱形またはバーを使用します。これらのメカニズムの長所は上記で議論したとおりです。

SYNCHRONIZING MERGE

このパターンは、Synchronization パターンの変形です。目的は、すべての並列パスからアクティビティにやってくるトークンを同期させることです。トークンがいくつ到着するかを事前に知ることが出来ない点が課題となります。この状況は、Multiple Choice パターンによって生成されます。したがって、Synchronization Merge パターンでは、上流でトークンがいくつ生成されたかを判断する必要があります。その後、それらのトークンを同期します。その他のトークンを待つことはしません。

ビジネス プロセス図

ビジネス プロセス図は、上流で包含的ゲートウェイによってパスを作成し(図 10 を参照)、再度、包含的ゲートウェイによってそれらを結合します。

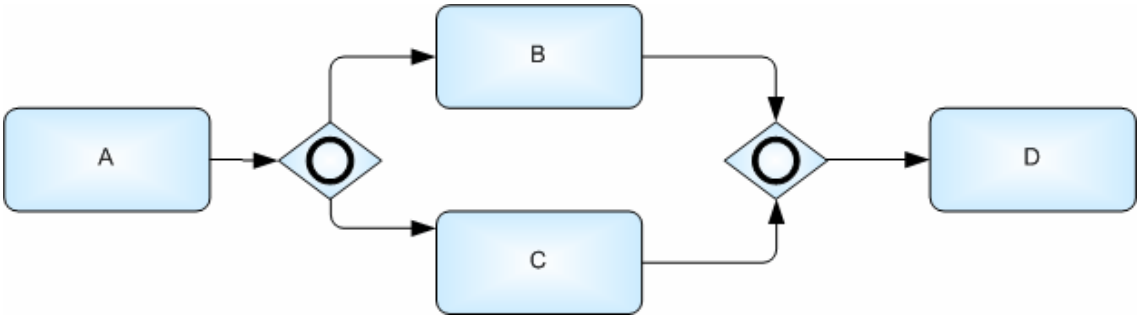

図 24: Synchronization Merge——ビジネス プロセス図

アクティビティ図

アクティビティ図は、ジョイン ノードを使用して Synchronization Merge パターンを表します。(図 25 を参照)。ジョイン ノードには条件式があり、トークンが出力される前に、いくつのトークンがコントロールフローから入力されるかを制御します。

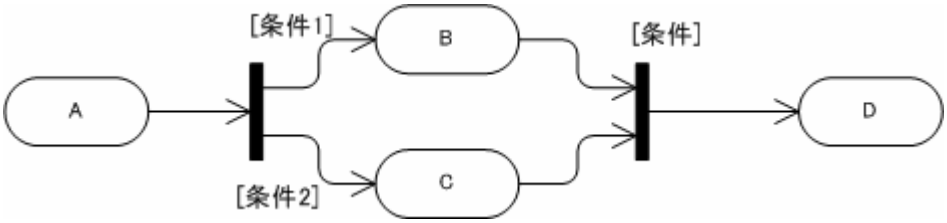

図 25: Synchronization Merge——アクティビティ図

比較

これまでのパターンのように、2 つの記法の違いは基本的には他のワークフロー パターンで使用された他のフロー制御メカニズムと同じです。ビジネス プロセス図は、適切な振る舞いを示すために菱形の変形を使用します。アクティビティ図は、フロー制御のために菱形あるいはバーのいずれかを使用します。これらのメカニズムの長所は上記で議論したとおりです。

STRUCTUAL PATTERNS

Structural パターンに属する 2 つのパターンは、ループや別々のプロセス パスの独立した振る舞いをカバーします。

ARBITRARY CYCLES

Arbitrary Cycles パターンは、プロセス中のあるセクションが繰り返される(ループされる)ことを可能にするためのメカニズムです。このパターンでは、不規則なループやブロック構造ではないループが認められています。すなわち、ループするセグメントには、1 つ以上の入口または出口が存在することができます。このパターンは、valid だが複雑なループを 1 つの画で表します。ブロック構造のループのみを表現できる記法では、1 つの画の中にすべてのプロセスやプロセスレベルを表示することはできません。また、直感的ではなくなります。

ビジネス プロセス図

ビジネス プロセス図では、上流のアクティビティにシーケンス フローを接続することで Arbitrary Cycles パターンを作成できます(図 26 を参照)。

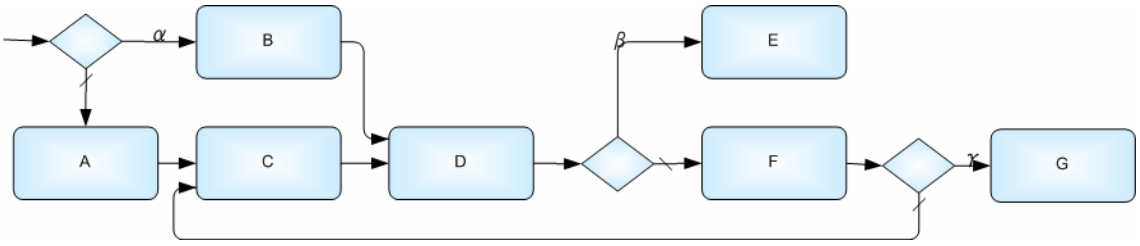

図 26: Arbitrary Cycles——ビジネス プロセス図

アクティビティ図

アクティビティ図では、上流のアクティビティにコントロール フローを接続することで Arbitrary Cycles パターンを作成できます(図 27 を参照)。

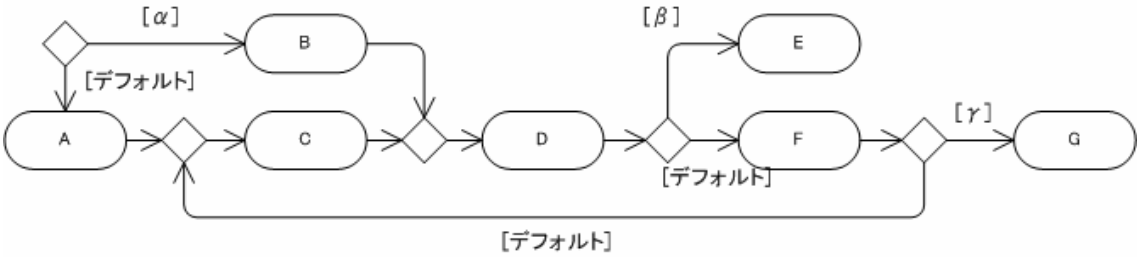

図 27: Arbitrary Cycles——アクティビティ図

比較

ビジネス プロセス図とアクティビティ図の間で、このパターンの表記に大きな差はありません。どちらの記法も、ブロック構造フォーマットを認めています。上流のアクティビティを下流のアクティビティに対し非

ブロック構造の形で接続することで、ループを可能にします。違いは、ビジネス プロセス図では、シーケンス フローが複数入力されるところでマージ ゲートウェイを使用していない点です。アクティビティ図では、ここに合流ノードを使用しています。つまり、同じ振る舞いを表すために、アクティビティ図はビジネス プロセス図よりも若干多くのオブジェクトとラインを使用します。

IMPLICIT TERMINATION

このパターンは、他の並行パスの終了なしに、プロセスのパスを終了できます。これは、すべてが最終ノードに到達した時点ですべてのプロセスが終了するという記法とは反対のものです。ビジネス プロセス図やアクティビティ図を含む多くの記法では、どちらの機能も提供されています。

ビジネス プロセス図

ビジネス プロセス図の終了イベント(太い線の円)は、パスが完成したことを表します(図 28 を参照)。ビジネス プロセス図の終了イベントは 7 種類あります。それらは 1 つを除いてすべて、Implicit Termination パターンに使用できます。終了イベントはすべて、特別な終了結果を示すために内部に記号を持っています。ターミネート終了イベントは、スタートしていなかったり、まだアクティブなアクティビティがあったりしても、その名前が示す通り、全プロセスを終了します。

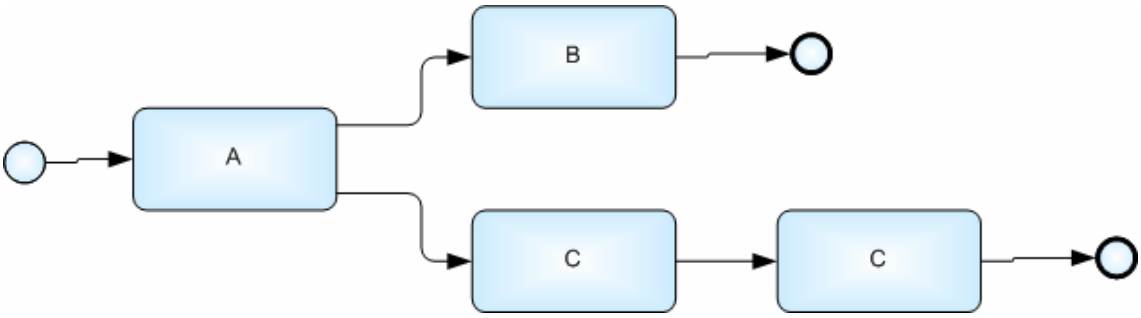

図 28: Implicit Termination——ビジネス プロセス図

アクティビティ図

アクティビティ図のフロー最終ノード(「X」が内部にある円)は、あるパスが完了したことを示します。これは、Implicit Termination パターンに使用されます(図 29 を参照)。アクティビティ最終ノード(小さな円が内部にある円)は、スタートしていなかったり、まだアクティブなアクティビティがあったりしても、全プロセスを終了します。

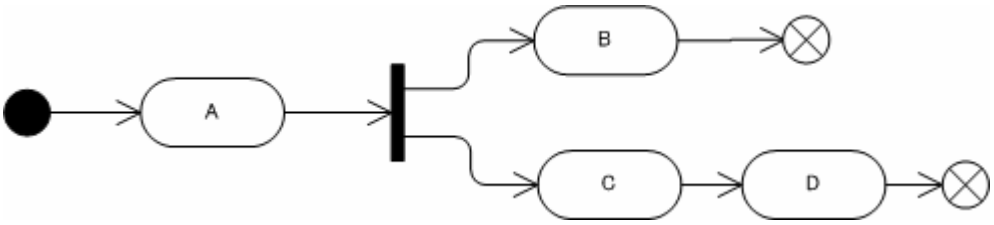

図 29: Implicit Termination——アクティビティ図

比較

2 つの記法はこのパターンを表すために同様のメカニズムを提供します。どちらの記法も、素直な方法で Implicit Termination パターンを表します。違いは、ビジネス プロセス図では、モデラーがプロセスの終了結果にビジネス情報を追加できる点にあります。終了イベントはそれぞれ、マージ、例外、キャンセル、補償、他のプロセスへのリンク、結果の組み合わせなどを表します。

PATTERNS INVOLVING MULTIPLE INSTANCE

次の 4 つのパターンは、複数のインスタンスやアクティビティのコピーがどのように作成されるかについて表したものです。標準ループ パターンを使用して扱うこともできますが、記述された状況の多くは標準ループよりも複雑です。

MI WITH A PRIORI DESIGN TIME KNOWLEDGE

このパターンは、アクティビティがどのように既知数回、並行にインスタンス化されるかを表します。つまり、モデラーは、プロセス モデル中にアクティビティがいくつ活動しているか、いくつ定義する必要があるかを知っていることになります。このパターンでは、アクティビティのコピーがスタートした後、何が起こるかを正確には定義していません。もしプロセスが継続前に、アクティビティ (およびトークン)のコピーをすべて同期させる必要がある場合は、MI requiring Synchronization パターンを使う必要があります。その後、アクティビティのコピーごとにプロセスが継続する場合(つまり、トークンがすべて独立して継続することを許す場合)は、MI with no A Priori Knowledge パターンを使う必要があります。

ビジネス プロセス図

ビジネス プロセス図のアクティビティは、LoopType 属性を「Multi-Instance」にし、MI_InstanceGeneration 属性を「Parallel」に設定することで、複数のコピーを表すことができます(2 本の平行線をアクティビティの中央下部に描きます。図 30 を参照)。アクティビティの MI_Condition 属性には、静的な数をセットして、このパターンの振る舞いを作成できます。

図 30MI WITH A RIORI DESIGN TIME KNOWLEDGE——ビジネス・プロセス図

アクティビティ図

アクティビティの周りに拡張区画を使って、アクティビティのコピーを複数作成できます(図 31 を参照)。これは、アクティビティの境界線にある入出力拡張ノード(小さな長方形)によって表されます。これがアクティビティの内部を拡張区画で示すコンパクトな方法であることに注意してください。並行に実行されるアクティビティのコピーを作成するために、concurrency 属性に「Parallel」がセットされます。入力区画にあるコレクション内の要素の数は、区画の中身のコピーがいくつ振る舞うかを決定します。要素の数は静的に定義され、このパターンの振る舞いを生成します。

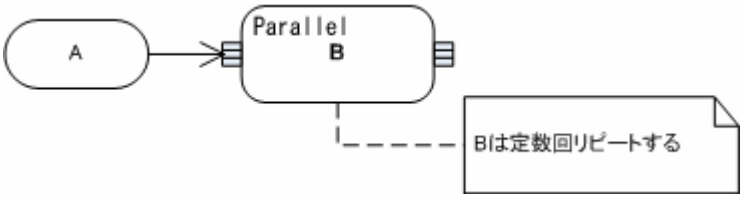

図 31: MI WITH A PRIORI DESIGN TIME KNOWLEDGE——アクティビティ図

比較

このパターンにおいて、ビジネス プロセス図とアクティビティ図の間に主な機能的な差はありません。どちらの記法も、アクティビティの複数インスタンス化を許可しています。記法は異なりますが、振る舞いは同じです。しかし、アクティビティ図の拡張区画は、アクティビティのピンと混同するかもしれません。

MI WITH A PRIORI RUNTIME KNOWLEDGE

このパターンは、プロセスが実行されるまでコピーの数が分からないため、事前に数をセットできない点を除けば、MI with A Priori Design Time Knowledgeパターンに似ています。さらに、このパターンでは、コピーは並行だけでなく連続的にも実行できます。以下の例は、コピーが連続して実行されたものです。

ビジネス プロセス図

ビジネス プロセス図のアクティビティは、LoopType 属性を「Multi-Instance」にし、MI_InstanceGeneration 属性を「Serial」に設定することで、複数のコピーを表すことができます(ループシンボルは、アクティビティの中央下部に描きます。図 32 を参照)。LoopCondition 属性は、実行されるコピーの数を動的に決定するためにセットされます。この振る舞いは、アクティビティの後の排他的ゲートウェイと、同一のアクティビティ(例えば、図の中のタスク「B」)をマージするためのルーピング バックを使って作成することもできます。

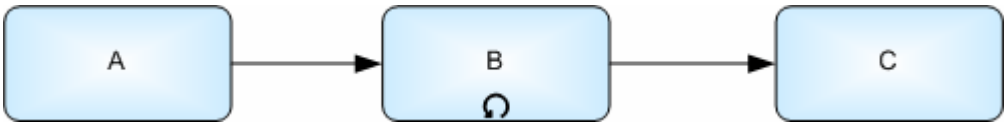

図 32: MI WITH A PRIORI RUNTIME KNOWLEDGE——ビジネス プロセス図

アクティビティ図

アクティビティ図では、アクティビティの周りに拡張区画を使って、アクティビティのコピーを複数作成します(図 33 を参照)。これは、アクティビティの境界線の入出力拡張ノード(小さな長方形)によって表されます。アクティビティのコピーの連続した振る舞いを作成するために、concurrency 属性に「iterative」セットされます。入力区画にあるコレクション内の要素の数は、プロセスの継続前に区画の中身のコピーがいくつ振る舞うかを動的に決定することができます。この振る舞いは、アクティビティの後の分岐ノードと、同じアクティビティ(例えば、図の中のアクティビティ「B」)をマージするためのルーピング バックを使って作成することもできます。

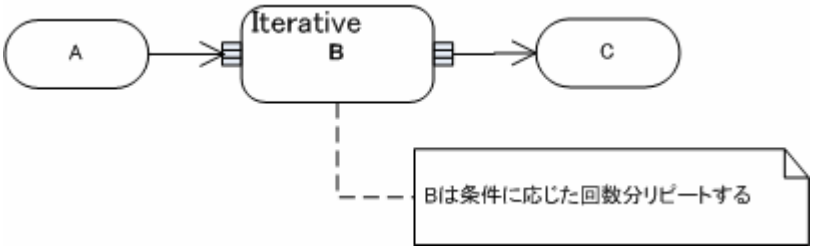

図 33: MI WITH A PRIORI RUNTIME KNOWLEDGE――アクティビティ図

比較

MI with A Priori Design Time Knowledge パターンと同様、このパターンにおいても、ビジネス プロセス図とアクティビティ図の間に主な差はありません。どちらの記法も、このパターンを作成するためのメカニズムを提供しています。

MI WITH NO A PRIORI KNOWLEDGE

このパターンは、コピーが作成される前にアクティビティのコピーの数を決定できない点で、前の 2 つのパターンと異なります。正確な数は、それらのコピーの実行中に決定されます。これは、標準のループや multi-instance パターンが、この振る舞いを作成するのに十分ではないことを意味します。ループより複雑な形は、以下の 2 つの例で示します。アクティビティの特定のパターンおよびフロー制御メカニズムは、このパターンを作成するために必要です。ターゲット アクティビティ(アクティビティ「B」)は、別のコピーが必要かどうか決める他のアクティビティ(アクティビティ「C」)とペアになる必要があります。これら 2 つのアクティビティは、ループ内に置かれます。アクティビティは、ループが繰り返される前に完了する必要はありません。しかし、ターゲット アクティビティ(アクティビティ「B」)のコピーはすべて、最終アクティビティ(アクティビティ「E」)をスタートする前に完了する必要があります。

ビジネス プロセス図

ビジネス プロセス図では、このパターンを作成するために、ゲートウェイとアクティビティのコンビネーションが使用されます(図 34 を参照)。タスク「B」および「C」は、並列のゲートウェイによって並行してスタートします。しかし、排他的ゲートウェイがタスク「C」に続きます。タスク「B」の別のコピーが必要な場合は、上流の排他的マージ ゲートウェイにシーケンス フローを接続することになります。マージ ゲートウェイは、並列ゲートウェイの同期振る舞いを回避するために必要となります。

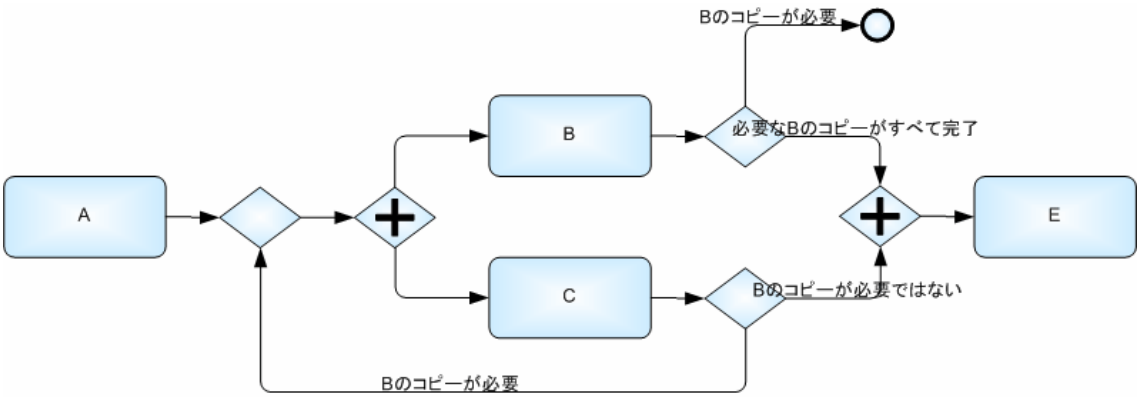

図 34: MI WITH NO A PRIORI KNOWLEDGE——ビジネス プロセス図

アクティビティ図

アクティビティ図では、分岐ノード、ノード マージ、フォーク ノード、およびアクティビティのコンビネーションを使用してこのパターンを表します(図 35 を参照)。アクティビティ「B」および「C」は、フォーク ノードによって並行でスタートします。しかし、分岐ノードがアクティビティ「C」に続きます。アクティビティ「B」の別のコピーが必要な場合は、上流の合流ノードにコントロール フローが接続されることになります。合流ノードは、フォーク ノード(ジョイン ノード)の同期振る舞いを回避するために必要となります。

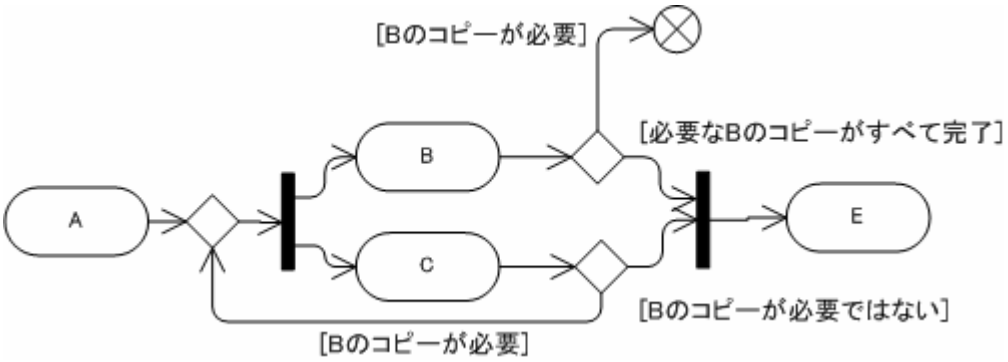

図 35: MI WITH NO A PRIORI KNOWLEDGE——アクティビティ図

比較

このパターンは、2 つの記法で同様に扱われます。基本的に、オブジェクトのパターンは、見た目は違いますが、分岐メカニズムは同じです。

MI REQUIRING SYNCHRONIZATION

このパターンは、プロセスが継続する前に繰り返されたアクティビティ(並行で実行)のコピーがすべて完了する必要があるという点を除いて、MI with A Priori Runtime Knowledge パターンと同じです。

ビジネス プロセス図

ビジネス プロセス図のアクティビティは、LoopType 属性を「Multi-Instance」にし、MI_InstanceGeneration 属性を「Parallel」に設定することで、複数のコピーを表すことができます (2

本の平行線をアクティビティの中央下部に描きます。図 36 を参照)。LoopCondition 属性は、実行されるコピーの数を動的に決定するためにセットされます。MI_FlowCondition 属性は、プロセスが継続する前にアクティビティのコピーがすべて終わることを保証するために、「All」にセットする必要があります(これは、1 つのトークンだけがプロセスを通じて継続することを意味します)。

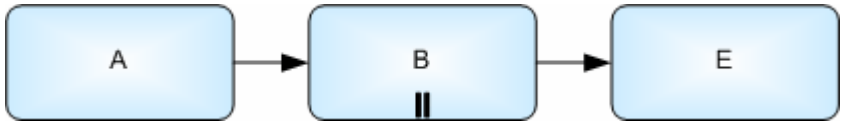

図 36: MI REQUIRING SYNCHRONIZATION——ビジネス プロセス図

アクティビティ図

アクティビティ図では、アクティビティの周りに拡張区画を使って、アクティビティのコピーを複数作成します(図 37 を参照)。これは、アクティビティの境界線にある入出力拡張ノード(小さな長方形)によって表されます。アクティビティのコピーの並行した振る舞いを作成するために、concurrency 属性に「Parallel」にセットされます。入力区画にあるコレクション内の要素の数は、プロセスの継続前に区画の中身にあるコピーがいくつ振る舞うかを動的に決定することができます。拡張区画の定義された振る舞いは、プロセスが区画を通過する前に、区画内ですべてのアクティビティを同期化します。

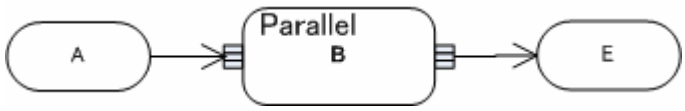

図 37: MI REQUIRING SYNCHRONIZATION——アクティビティ図

比較

このパターンは基本的に、他の 3 つの multiple instance パターンと同じです。

STATE-BASED PATTERNS

このグループの 3 つのパターンは、プロセス エンジンの直接制御外の要因によって、ビジネス プロセスの振る舞いがどのように影響されるか定義しています。

DEFERRED CHOICE

このパターンは、一種の排他的な分岐を表しており、Exclusive Choice パターンに似ています。しかし、パスを分岐する根拠は異なります。Exclusive Choice パターンは、プロセス データの評価に基づきます。Differed Choice パターンは、プロセス間に生じるイベントに基づきます。イベントが生じると(つまり選択が行われると)、他の代替パスは無効になります。

ビジネス プロセス図

ビジネス プロセス図は、排他的データ準拠ゲートウェイを使用します(図 38 を参照)。この分岐は、代

替パスがプロセス中のポイントで生じるイベントに準拠した分岐ポイントを表しています。このソリューションは、ゲートウェイがシーケンス フローを制御するために追加されたオブジェクトにサポートされるという点で、ビジネス プロセス図特有のものです。中間イベントがゲートウェイに続きます。トークンがゲートウェイに到着すると、トークンはそこで待機します。あるイベント(通常はメッセージの受取)が、どのパスが選択されるかを決めます。タイマーのような他のタイプのイベントを使用することもできます。代替パスのうち 1 つだけが選ばれます。最初に生じるイベントは、ゲートウェイからの他のパスを排除し、得られるパスを引き起こします(メッセージの中間イベントの代わりに、受理タスクを使用できることに注目)。このイベントは、シーケンス フローを下っていく場所で、ゲートウェイからトークンを「引っぱります」。

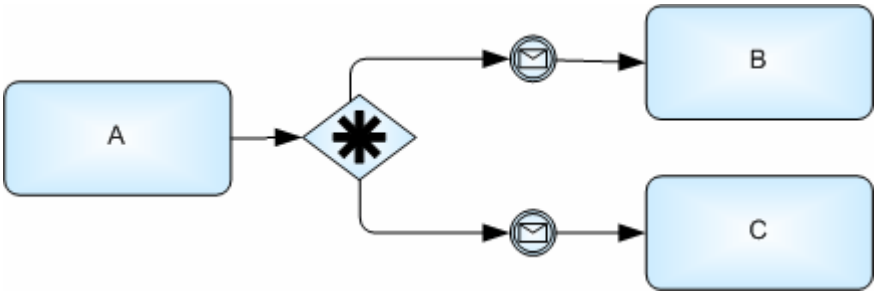

図 38: Differed Choice——ビジネス プロセス図

アクティビティ図

アクティビティ図は、このパターンの振る舞いを表すために、フォーク ノード、割り込み可能区画、および受信シグナル アクションを使用します(図 39 を参照)。選択が行なわれるべきポイントでは、フォークノードが選択オプションの数と等しい並列パスの数を作成します。したがって、複数のトークンが作成されます。これらの並列パスは、割り込み可能区画を横断します。並列パスはそれぞれ、区画内の受信シグナル アクションに結びつきます。第 1 のシグナルが到着し、区画を出るために割り込みエッジが使用されると、他のシグナルの受信は不可になります。他のトークンが割り込み可能区画によって消費されている一方で、トークンのうちの 1 つは、その目的地への割り込み可能なエッジを横断します。

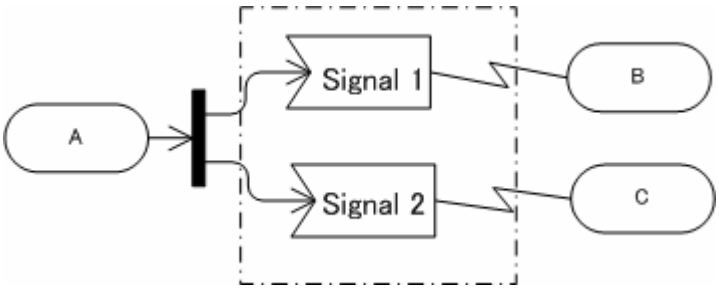

図 39: Differed Choice——アクティビティ図

比較

どちらの記法も、このパターンの振る舞いを十分に表すことができます。しかし、使用されるメカニズムは異なります。アクティビティ図では、フォーク ノードが、シグナルのトリガを 1 つだけ選択する複数のパスを生成して、振る舞いを扱っている一方、ビジネス プロセス図では、分岐とイベントで振る舞いを扱っています。フォークおよび割り込み可能区画では、プロセスで分岐が行われていることが直感的にはわ

かりません。

INTERLEAVED PARALLEL ROUTING

このパターンにある「parallel」という言葉には、少し語弊があるかもしれません。このパターン中のアクティビティは、連続して実行する必要があるからです。しかし、順序は定まっていません。連続したアクティビティの実行は、アクティビティが同じ資源を共有するのか、更新するのかという要求によって異なります。アクティビティの実行者は、アクティビティの順序を決定します。

ビジネス プロセス図

この振る舞いを扱うために、ビジネス プロセス図は、Ad-Hoc プロセス(オブジェクトの下部中央にある「~(チルダ)」が付いたサブ プロセス オブジェクト)の概念を持っています。このプロセスは、任意の順に実行できるアクティビティのコレクションです(図 40 を参照)。タスク「B」および「D」は、インターリーブド パラレル ルーティング パターンを描く図形の部品です。Ad-Hoc サブプロセスは、モデラーが定義した条件を持っており、完了する前に満たされます。さらに、Ad-Hoc プロセスの ordering 属性には、「Serial」がセットされます。その結果、アクティビティは 1 つずつ実行されます。

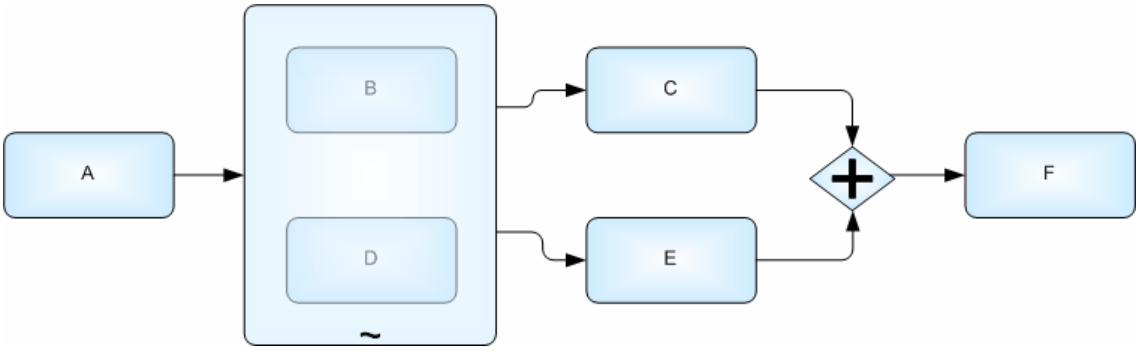

図 40: Interleaved Parallel Routing——ビジネス プロセス図

アクティビティ図

アクティビティ図は、Ad-Hocプロセスの概念を持っておらず、このパターンのための適切な図を持ちません。この振る舞いを作成する最も単純な方法として、フォーク ノードとジョイン ノードの間にアクティビティを置く方法があります(図 41 を参照)。アクティビティは、並行で実行されているように見えます。しかし、同じ資源を使用するなど、各アクティビティに制約が追加されているはずです。このメカニズムは、2 つのアクティビティを同時に実行できないことを表します。しかし、注釈以外に制約を表す図式はありません。

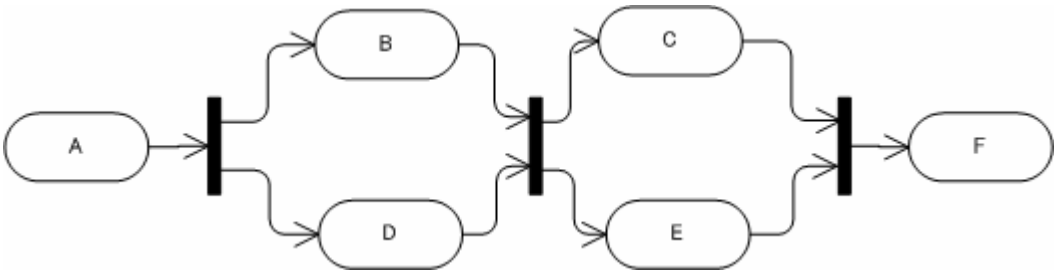

図 41: Interleaved Parallel Routing——アクティビティ図 バージョン

モデラーは、代替パスを作成して、振る舞いをより明示的にできます。代替パスは、Deferred Choice パターンを利用し、アクティビティの重複も使用します(図 42 を参照)。アクティビティ「B」および「D」は、Interleaved Parallel Routing パターンを描くための図形の一部です。これで、希望の振る舞いを描くことができます。しかし、パターン中のアクティビティの数が 2 つ以上の場合、図の理解は難しくなります。

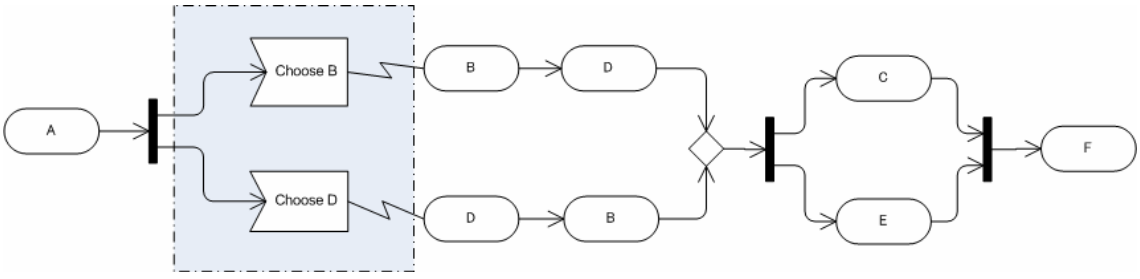

図 42: Interleaved Parallel Routing——アクティビティ図 バージョン 2

もう 1 つ、UML(状態図)で Interleaved Parallel Routing パターンを作るためのより複雑な方法があります。しかし、平均的なビジネス モデラーが、これらのソリューションを理解するのは非常に難しことです。

比較

このパターンは、他のパターンと比較して、記法に大きな違いがあります。ビジネス プロセス図は、Ad-Hoc プロセス用の機能を内蔵していますが、アクティビティ図では、それに似た機能をすべて使い、アクティビティの重複などを使用しながらレイアウトする必要があります。結果として、ビジネス プロセス図の方が、コンパクトで理解しやすい図を提供できます。また、理解しやすいフォーマットではありませんが、UML にも振る舞いを扱う図があります(状態図)。図 41 からわかるように、Interleaved Parallel Routing パターンと Parallel Split パターンの違いがわかりにくくなっています。

MILESTONE

プロセス内にポイントがあり、そこで特定のイベントが起きるか、条件があるかを知ることは重要です。これらのイベントあるいは条件をマイルストーンと呼ぶことができます。プロセス モデルは、こういったマイルストーンを識別し、反応する必要があります。このパターンは、複数の方法によって実現できます。マイ

ルストーンがプロセスによってどのように使用されるかという正確な性質に依存しています。私たちは、ワークフロー研究者[3]によって提供される例のうちの★2 つ目を★レビューします。

ビジネス プロセス図

下記の例は、正常なシーケンス フローを使用できない場合に、情報がビジネス プロセス図のある部品から別の部分までどのように渡されるかを示します(図 43 を参照)。この情報は、マイルストーンの発生を反映します。例では、ある別のサブプロセス内のアクティビティ(タスク「B」)がスタートできる前に、あるサブプロセス内のアクティビティ(タスク「B」)の完了が要求されます。タスク「B」に続く Link End イベントは、タスク「D」に先行する Link Start イベントを引き起こすために使用されます。Association Link は関係を強化します。

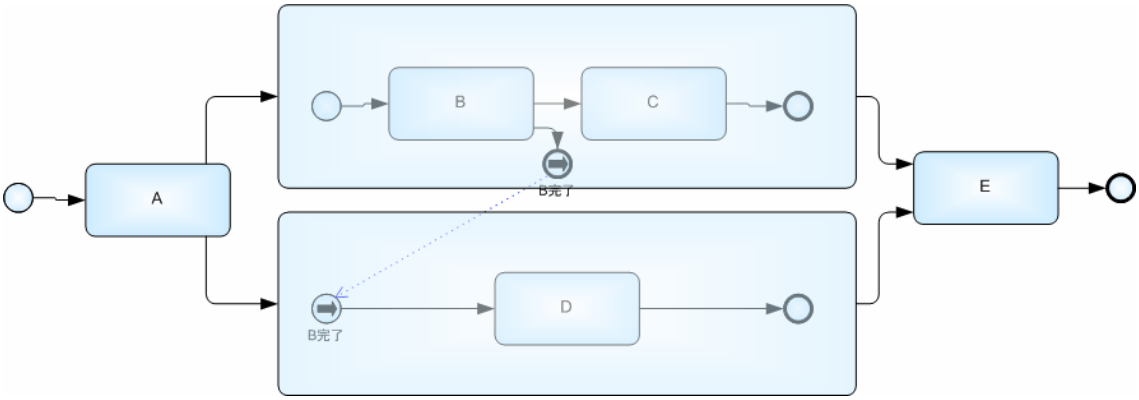

図 43: Milestone——ビジネス プロセス図 (例 2)★

アクティビティ図

★第 2 の例★は、正常なコントロール フローを使用できない場合に、情報がプロセスのある部分から別の部分までどのように渡されるかを示します(図 44 を参照)。例では、ある別のサブ アクティビティ中のアクティビティ(アクティビティ「D」)がスタートできる前に、あるサブ アクティビティ中のアクティビティ(アクティビティ「B」)の完了が要求されます。アクティビティ「B」に続くシグナル(右側を指している長方形)のブロード キャストは、アクティビティ「D」に先行するシグナル(左側に刻み目のある長方形)の受信を引き起こすために使用されます。

[3] http://tmitwww.tm.tue.nl/research/patterns/milestone.htm

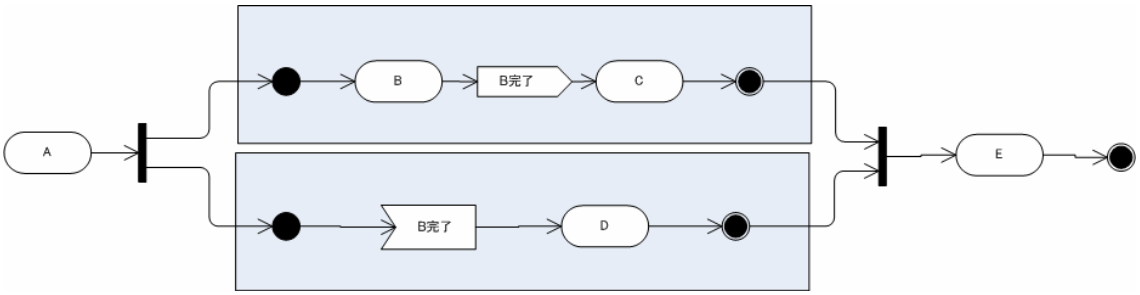

図 44: Milestone——アクティビティ図(例 2)★

比較

どちらの記法も、このパターンの振る舞いを同様のメカニズムで十分に扱うことができます。しかし、第1の例は、例外処理用のメカニズムを導入します。両方のメカニズムは同じ振る舞いを提供します。しかし、アクティビティの境界に中間イベントを直接付けるビジネス プロセス図のメカニズムの方が、シグナル アクションの受信を備えた割り込み可能区画よりも、より視覚的に表すことができます。

CANCEL PATTERNS

次の 2 つのパターンは、あるアクティビティの完了が、どのようにアクティビティやアクティビティ グループの取り消しを引き起こすかを示します。

CANCEL ACTIVITY

このパターンは、競合する 2 つのアクティビティが存在できることを表します。アクティビティのうちの 1 つが完了すると、それは別のアクティビティがプロセスを止めるべきであるという合図になります。したがって、取り消しを示すメカニズム、および、このシグナルに基づいたアクティビティの中断メカニズムが必要となります。

ビジネス プロセス図

ビジネス プロセス図は、アクティビティの境界に付けられる、中間イベントによって例外処理を実行します(図 45 を参照)。アクティビティが実行されている間に中間イベントのトリガが生じると、アクティビティは中断されます。また、トークンは中間イベントによってアクティビティを抜け、次のオブジェクトに向かってシーケンス フローを進んでいきます。以下の図の中で、例外中間イベントのトリガは、タスク「B」に続く中間イベントであり、一般的に、プロセスのフローです(アクティビティの境界へ付けられていません)。この中間イベントは、取り消し信号を「throw」します。また、タスク「C」の境界へ付けられた中間イベントは、信号を「catch」します。

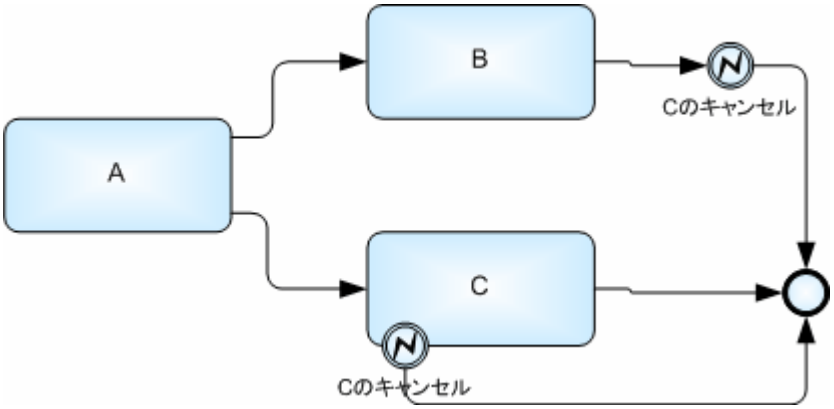

図 45: Cancel Activity——ビジネス プロセス図

アクティビティ図

アクティビティ図は、複数のアクティビティを囲む割り込み可能区画を通って、例外処理を実行します(図 46 を参照)。たとえば、何かが、アクティビティが完了したりシグナルを受け取ったりするなどして、トークンに区画の端(図中の破線)を横断させた場合、区画内のアクティビティはストップします。そして、フローは中断する端を継続します。下記の図では、例外のトリガはシグナルの受信です。シグナルはアクティビティ「B」の完了直後にブロード キャストされます。送信シグナル アクションは、取り消しシグナルを「throw」します。また、割り込み可能区画の受信シグナル アクションはシグナルを「catch」します。

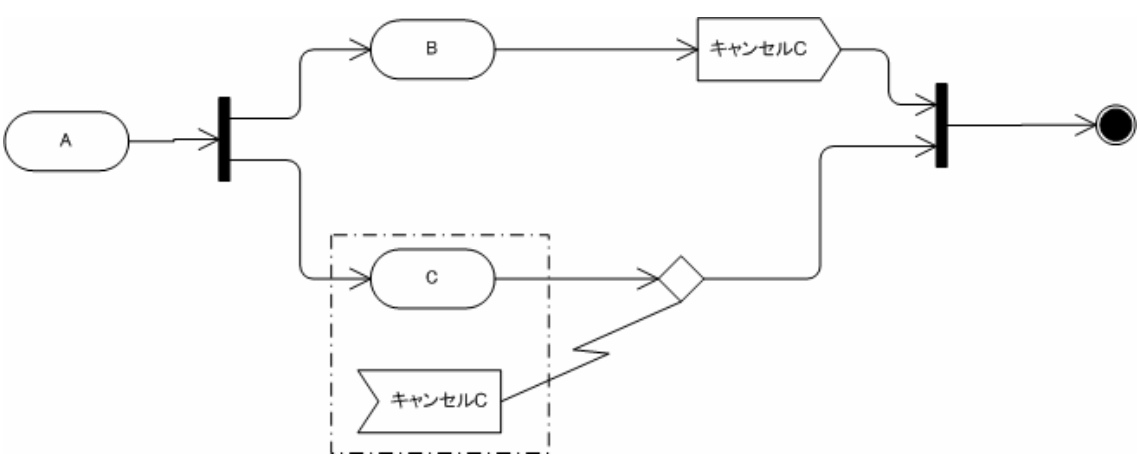

図 46: Cancel Activity——アクティビティ図

比較

2 つの記法は同様の方法で、例外の throw および catch を扱います。アクティビティの境界に中間イベントを付けるというビジネス プロセス図のメカニズムは、例外がどこで適用されるかをより直感的に表しています。さらに、ビジネス プロセス図のメカニズムはアクティビティ、タスク、あるいはサブプロセスを完了させるためだけに適用されます。アクティビティ図の割り込み可能区画は、リスクの高い手法です。フォーク パターンやジョイン パターン内にあるアクティビティなどの、グループのサブ セットであるアクティビティを中断してしまう場合があります。

CANCEL CASE

このパターンは Cancel Activity パターンの拡張です。このパターンでは、すべてのプロセスが取り消されます。

ビジネス プロセス図

基本的に、Cancel Activity パターンと同じです。しかし、中間イベントはタスクではなく他のアクティビティを含んだサブプロセスの境界に付けられます(図 47 を参照)。図では、サブ プロセスを取り消します。

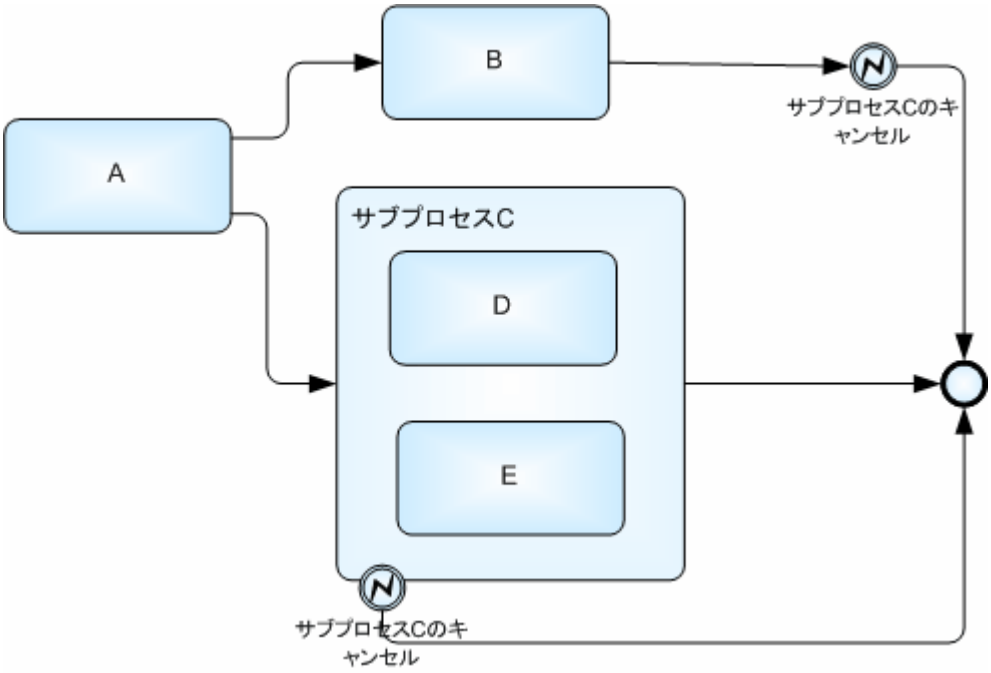

図 47: Cancel Case——ビジネス プロセス図(例 1)

モデラーが全工程を取り消す場合は、もっと単純です。必要なのは、取り消しが示されるべき位置での最終イベントの使用です(図 48 を参照)。トークンが最終イベントに到着すると、全プロセスを止めることができます。

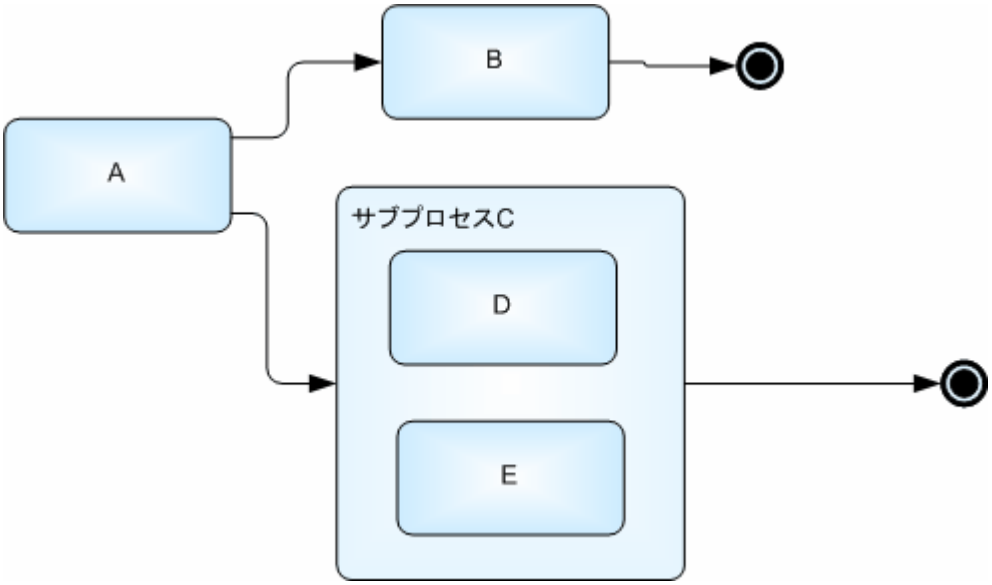

図 48: Cancel Case——ビジネス プロセス図(例 2)

アクティビティ図

基本的に Cancel Active パターンと同じです。しかし、割り込み可能区画は、単純なアクティビティではなく、他のアクティビティを含んでいる合成アクティビティを囲んでいます(図 50 を参照)。図では、サブプロセスを取り消します。

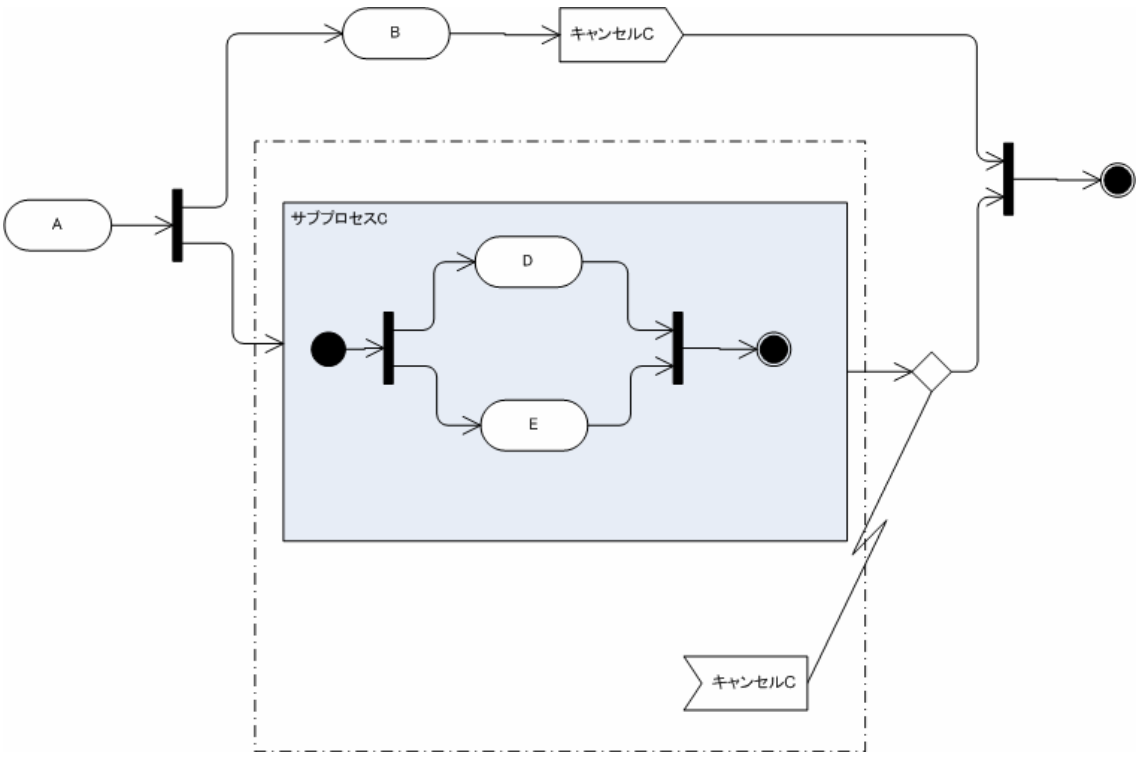

図 49: Cancel Case——アクティビティ図(例 1)

モデラーが全工程を取り消す場合は、もっと単純です。モデラーは、取り消しが示されるべき位置でアクティビティ最終ノードを使用する必要があります(図 50 を参照)。トークンがアクティビティ最終ノードに到

着すると、全プロセスが停止します。

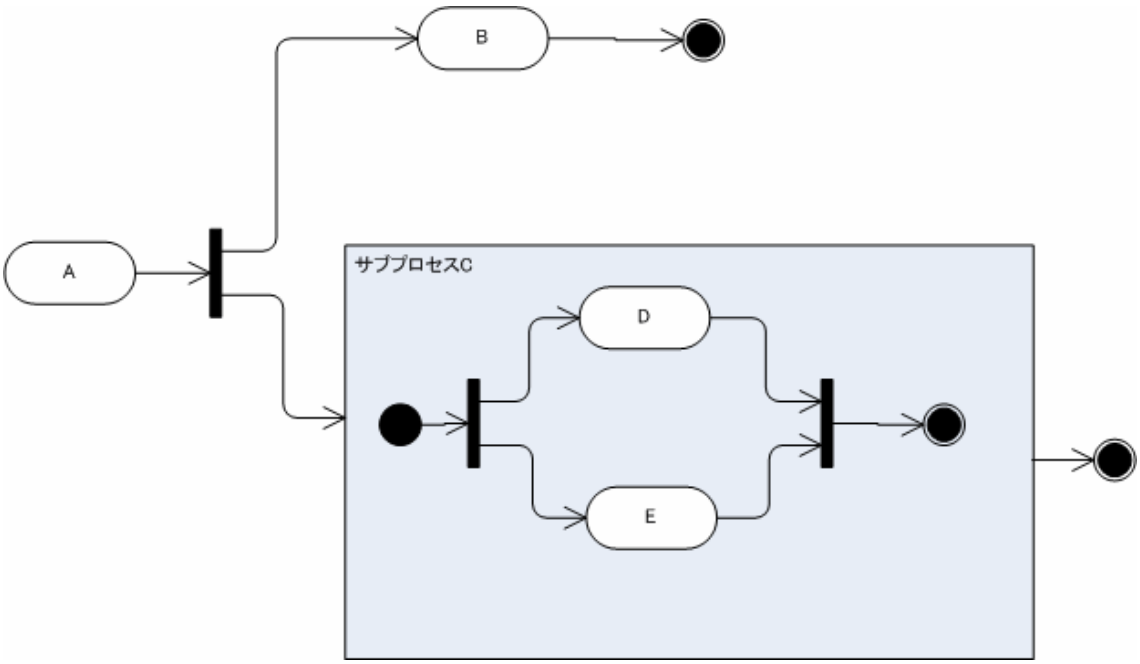

図 50: Cancel Case——アクティビティ図

比較

Cancel Active パターンの比較と同じです。どちらの記法も、全工程の取り消しモデルとほぼ同じ図式のオブジェクトを備えています(ビジネス プロセス図の最終イベントとアクティビティ図のアクティビティ最終ノード)。

結論

ビジネス プロセス図とアクティビティ図が、21 のワークフロー パターンをどのようにモデル化するかという調査から、どちらの記法も十分にパターンをモデル化できることが実証されました。ただ、1 つだけ例外があり、メタモデルには適切な構造がありますが、アクティビティ図は Interleaved Parallel Routing パターンを適切に表すための図式を持っていませんでした。ほとんどのパターンで両者とも同様のソリューションを提供していますが、これは、両者の見た目がいかに近いかを表しています。どちらも、同じ目的を表す同じ形のものを多く共有しています(たとえば、アクティビティには角の丸い長方形、分岐には菱形など)。モデリング オブジェクトの形にはいくつか違いがあります。上記のワークフロー パターンで示したように、BPMN のコア オブジェクトの方が数が少なく、モデリング プロセス中に起こる複雑な状況を扱うために、オブジェクトのバリエーションが用意されています。もう 1 つの違いは用語です。たとえば、アクティビティ図は開始ノードを持っていますが、ビジネス プロセス図は開始イベントを持っています。ワークフロー パターンは、プロセス アクティビティ間の制御フローに焦点を当てているため、その他の違いについては、この論文では省いています。しかし、アクティビティ間のデータのフローを考慮した場合、他にも違いはあります。

2 つの表記法は、同じような基礎的な問題を解決する(ビジネス プロセスの手続きを図解する)ことを目的に設計されているため、類似しています。しかし、異なるターゲット ユーザーに合わせて設計されていることから、2 つの記法には違いがあります。BPMN は、ビジネスマンが使うための記法を作るという目的で、2 年前から作成が始まりました。UML は、ソフトウェア開発のモデリングを標準化するという目的で、数年前から作成が始まりました。アクティビティ図はその試みの一環です。UML 2.0 では、アクティビティ図がビジネス ピープルにとって使いやすいものとなるようにアップ グレードされましたが、まだ技術色が残っています――同じく OMG で制定されている EDOC という非常に技術色の強いモデリング技術の影響を受けています。

ビジネス プロセス定義メタモデルが OMG の RFP プロセスを通じて開発されたため、2 つの図はビュー(図形)の特徴を共有しています。ビジネス プロセス定義メタモデルは、UML 2.0 のメタモデルを拡張したものであるため、アクティビティは元々ビューです。BPMN については、ビジネス プロセス定義メタモデルにマッピングする形で示されました。

アクティビティ図とビジネス プロセス図は非常に似ており、同じメタモデルのビューとなります。将来、相互変換される可能性もあります。OMG は、ビジネス プロセスやビジネス ルールを含む、ソフトウェア開発レベルより高いレベルでのビジネス モデリングに関わっていくことを約束しています。よって、元は BPMI で開発されたビジネス プロセス図は、将来的には OMG で開発されている高レベルのビジネス モデリング インフラストラクチャの一部として開発されている可能性があります。ビジネス プロセス図のビジネス プロセス定義メタモデルへのマッピング、およびこの論文でのワークフロー パターンでのビジネス プロセス図とアクティビティ図との比較は、そのようなプロセスの初期の活動であるといえます。