BEA WebLogic Integration™

BPM ワークフローの設計
ベスト プラクティス ガイド

リリース 7.0
マニュアルの日付 : 2002 年 6 月 28 日

著作権

限定的権利条項

商標または登録商標

WebLogic Integration BPM ワークフローの設計 ベスト プラクティス ガイド

パート番号	日付	ソフトウェアのバージョン
なし	2002 年 6 月	7.0

このマニュアルの内容

バージョン	1.0
日付	１　２００４
ソース	d:\build\antwork\bestprgd\preface.fm
マニュアル名	WebLogic Integration BPM ワークフローの設計 ベスト プラクティス ガイド
最終更新 :	Cheryl Solis、Doc Mgr、San Jose Frameworks

このマニュアルでは、ワークフロー設計で推奨される方法を説明し、よく使用されるプログラミング パターンに従ってワークフローを構築するためのガイドラインを示しています。このマニュアルの目的は、将来バージョンの WebLogic Integration に移行するときに作業がスムーズに進むようなワークフローを作成するガイドを提供することです。

このマニュアルの内容は以下のとおりです。

- 「BEA WebLogic Integration Studio では、ビジネス プロセス ワークフローを設計し、モニタする設計環境を提供しています。Studio ワークフローを作成している、またはワークフロー管理システムとして Integration Studio を使用する予定の場合、ワークフローの設計にベスト プラクティスを念頭に置くようにすることをお勧めします。以下の節では、将来の製品リリースに移行するときに、スムーズに移行作業が行なえるためのワークフロー構築のガイドラインを示します。」では、そのようなベスト プラクティスのガイドラインの目的および使用について説明します。
- 「よく使用されるワークフロー パターンに対するワークフロー設計の推奨例」では、よく使用されるワークフロー パターンの推奨される設計方法を紹介します。
- 「タスク ノードの使用法」では、提示したワークフロー パターンに関連するユーザ タスク ノードおよび自動タスク ノードの実装に関するガイダンスを示します。

- 「例外処理」では、例外処理ブロックをワークフロー内に構成する場合のガイドラインを提供します。
- 「Studio プラグインを使用する場合のガイドライン」では、移植性の高いワークフローを設計する場合の WebLogic Integration Studio のプラグイン機構の使用についての疑問に答えます。

対象読者

このマニュアルは、主としてビジネス アナリスト、アプリケーション開発者、および BEA WebLogic Integration Studio を使用して既存のワークフローを作成したことがある、または Weblogic Integration Studio によってワークフローを設計する予定の開発者一般を対象にしています。Weblogic Integration Studio プラットフォームおよび Java プログラミングに読者が精通していることを前提として書かれています。

e-docs Web サイト

BEA 製品のドキュメントは、BEA Systems, Inc. の Web サイトで入手できます。BEA のホーム ページで [製品のドキュメント] をクリックするか、または「e-docs」という製品ドキュメント ページ（http://edocs.beasys.co.jp/e-docs/index.html）を直接表示してください。

このマニュアルの印刷方法

Web ブラウザの [ファイル | 印刷] オプションを使用すると、Web ブラウザからこのマニュアルを一度に 1 ファイルずつ印刷できます。

このマニュアルの PDF 版は、Web サイトで入手できます。BEA WebLogic Integration PDF を Adobe Acrobat Reader で開くと、マニュアルの全体（または一部分）を書籍の形式で印刷できます。PDF を表示するには、BEA WebLogic Integration ドキュメントのホーム ページを開き、[PDF 版] ボタンをクリックして、印刷するマニュアルを選択します。

Adobe Acrobat Reader がない場合は、Adobe の Web サイト（http://www.adobe.co.jp/）で無料で入手できます。

関連情報

次の BEA WebLogic Integration マニュアルには、ここで挙げるベスト プラクティス ガイドラインの使用、および BEA WebLogic Integration Studio によるワークフローの構築方法に関する情報が掲載されています。

BEA WebLogic Integration および特に BEA WebLogic Integration Studio に関する一般情報の詳細については、以下の資料を参照してください。

- Business Process Management の概要
- WebLogic Integration BPM ユーザーズ ガイド
- WebLogic Integration Studio ユーザーズ ガイド
- WebLogic Integration Worklist ユーザーズ ガイド
- BPM クライアント アプリケーション プログラミング ガイド

サポート情報

BEA WebLogic Integration のドキュメントに関するユーザからのフィードバックは弊社にとって非常に重要です。質問や意見などがあれば、電子メールで **docsupport-jp@bea.com** までお送りください。寄せられた意見については、BEA WebLogic Integration のドキュメントを作成および改訂する BEA の専門の担当者が直に目を通します。

電子メールのメッセージには、BEA WebLogic Integration 7.0 リリースのドキュメントをご使用の旨をお書き添えください。

本バージョンの BEA WebLogic Integration について不明な点がある場合、または BEA WebLogic Integration のインストールおよび動作に問題がある場合は、BEA WebSUPPORT (**websupport.bea.com/custsupp**) を通じて BEA カスタマ サポートまでお問い合わせください。カスタマ サポートへの連絡方法については、製品パッケージに同梱されているカスタマ サポート カードにも記載されています。

カスタマ サポートでは以下の情報をお尋ねしますので、お問い合わせの際はあらかじめご用意ください。

- お名前、電子メール アドレス、電話番号、ファクス番号
- 会社の名前と住所
- お使いの機種とコード番号
- 製品の名前とバージョン
- 問題の状況と表示されるエラー メッセージの内容

表記規則

このマニュアルでは、全体を通して以下の表記規則が使用されています。

表記法	適用
太字	用語集で定義されている用語を示す。
〔Ctrl〕+ 〔Tab〕	複数のキーを同時に押すことを示す。
斜体	強調または書籍のタイトルを示す。

等幅テキスト	コード サンプル、コマンドとそのオプション、データ構造体とそのメンバー、データ型、ディレクトリ、およびファイル名とその拡張子を示す。等幅テキストはキーボードから入力するテキストも示す。
	例
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

太字の等幅テキスト	コード内の重要な箇所を示す。 *例* `void **commit** ( )`
斜体の等幅テキスト	コード内の変数を示す。 *例* `String *expr*`

表記法	適用
すべて大文字のテキスト	デバイス名、環境変数、および論理演算子を示す。 *例* LPT1 SIGNON OR
{ }	構文の中で複数の選択肢を示す。実際には、この括弧は入力しない。
[ ]	構文の中で任意指定の項目を示す。実際には、この括弧は入力しない。 *例* `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
|	構文の中で相互に排他的な選択肢を区切る。実際には、この記号は入力しない。
...	コマンドラインで以下のいずれかを示す。 ■ 引数を複数回繰り返すことができる ■ 任意指定の引数が省略されている ■ パラメータや値などの情報を追加入力できる 実際には、この省略記号は入力しない。 *例* `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	コード サンプルまたは構文で項目が省略されていることを示す。 実際には、この省略記号は入力しない。

目次

BPM ワークフロー設計上のベスト プラクティス

BEA WebLogic Integration Studio では、ビジネス プロセス ワークフローを設計し、モニタする設計環境を提供しています。Studio ワークフローを作成している、またはワークフロー管理システムとして Integration Studio を使用する予定の場合、ワークフローの設計にベスト プラクティスを念頭に置くようにすることをお勧めします。以下の節では、将来の製品リリースに移行するときに、スムーズに移行作業が行なえるためのワークフロー構築のガイドラインを示します。

- 「ベスト プラクティスをお勧めする理由」
- 「よく使用されるワークフロー パターンに対するワークフロー設計の推奨例」
- 「並行実行」
- 「イベント選択」
- 「タイムアウト付きイベント」
- 「イベントによる取消し」
- 「実行タイムアウト」
- 「ワークフロー パターンにおけるアクションの使用法」
- 「タスク ノードの使用法」
- 「Studio プラグインを使用する場合のガイドライン」

ベスト プラクティスをお勧めする理由

現行の WebLogic Integration Studio 実装では、構造化していないワークフローでも構築することができます。しかし、ワークフロー管理の関係業界、および近年意識されつつあるワークフロー規格においても、構造化ワークフローの設計を推奨しています。将来バージョンの WebLogic Integration Studio では構造化ワークフローの使用が必須となる予定で、ワークフロー規格が成熟し、その使用が妥当と思われた段階でこれに準拠する予定です。このマニュアルでは、現在の WebLogic Integration Studio で使用するための構造化ワークフローのパターンを説明します。ここで挙げるガイドラインに従うことにより、よりよい設計のワークフローが構築でき、構造化ワークフロー パラダイムへの移行がスムーズに行なえるようになります。

注意： 以下のガイドラインは、従うべきビジネス ロジックの実装方法を提起するものではありません。

ビジネス プロセスの設計

ワークフロー定義は、複雑なビジネス プロセス、方針、および手順をカプセル化するものです。その複雑さを考慮して、ベスト プラクティス ガイドラインでは、共通する一般的なワークフロー パターンに関連した事項を提示します。コンセプトによっては、そのコンセプトの望ましい実装方法、および避けるべき事項を具体的に示します。

提示されたワークフロー パターンやプログラミングの推奨方法は、それが対応するコンセプトの唯一の実装方法というわけではありません。提示内容は、一般的な設計原理がわかりやすいように、また自動移行するときに最適とは言えないモデルないしワークフロー パターンを明確に指摘するために意図的に単純化してあります。当然のことながら、実際のビジネス事例では、それらのコンセプトについて異なる実装または設計が必要な場合もあるでしょうし、実際のワークフロー設計は、このマニュアルで提示される設計パターンよりもおそらくより複雑なものになると思われます。

このガイドで提示される実装例は、ワークフロー設計を最適化する必要度の評価手段、および目安として使用してください。ここで提示されるガイドラインに盲従することは、実際のビジネス プロセスを反映していない、不適切なワークフローになる場合もありますので、決してお勧めできません。

WebLogic Integration Studio を使用したワークフローの設計

ここで規定する望ましいベスト プラクティスは、WebLogic Platform リリース 7.0 の WebLogic Integration Design Studio コンポーネントを使用してワークフローを設計することを想定しています。WebLogic Integration Studio を使用したワークフローの設計に関する詳細は、以下の各トピックのマニュアルを参照してください。

- Business Process Management の概要
- WebLogic Integration BPM ユーザーズ ガイド
- WebLogic Integration Studio ユーザーズ ガイド
- WebLogic Integration Worklist ユーザーズ ガイド
- BPM クライアント アプリケーション プログラミング ガイド

よく使用されるワークフロー パターンに対するワークフロー設計の推奨例

以下の節では、ワークフロー構築におけるさまざまな推奨パターンを提示します。これらのガイドラインは、よくある共通のワークフロー問題に対する適切な対処法を提示することを目的としています。ビジネス プロセスに関して多くのワークフローで見られる一般的なパターンには、以下のようなものがあります。

- 「並行実行」
- 「イベント選択」
- 「タイムアウト付きイベント」
- 「イベントによる取消し」
- 「実行タイムアウト」

ここからは、それらのパターンに関する説明、およびそれらのワークフローの推奨される実装について具体的に記載します。

並行実行

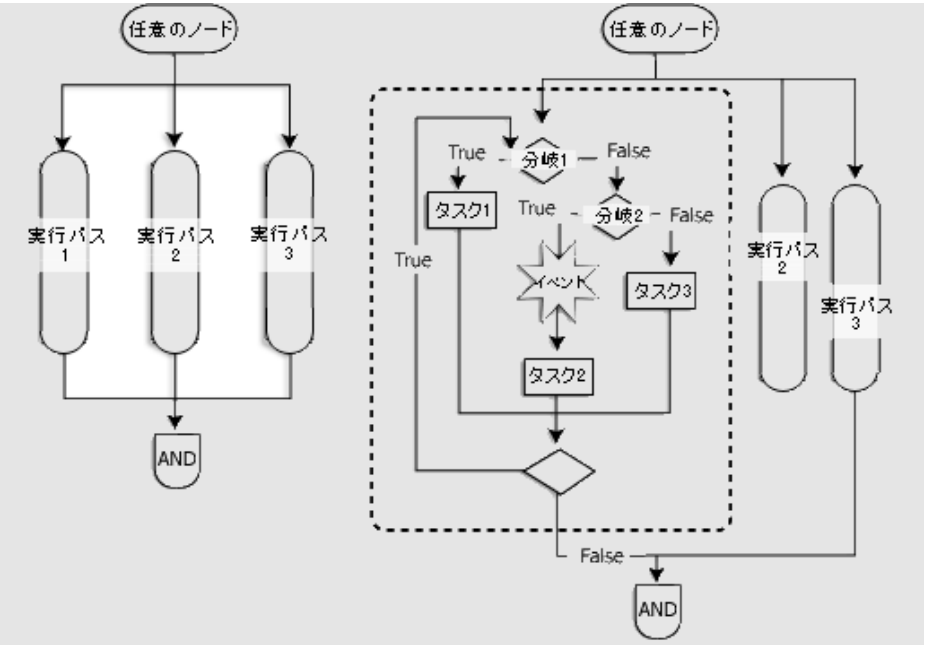
「並行実行」パターンは、1 つのワークフロー ノードから出るすべての並行パスが、同時に実行され、その並行処理の終了地点の AND ノードで再び 1 つになる場合を表します。このワークフロー パターンを適切に実装するためには、次のガイドラインに従ってください。

1. **1 つのワークフロー ノードから出るすべての並行パスが同じ AND ノードに合流するようにします。**起点が異なる複数のノードになっている並行パスは、同じ AND ノードに合流することはできません。言い換えれば、並行実行域では、必ず単独の入口ポイントと出口ポイントを持たなければならない、ということです。これらのパスは、パスの構成要素も同様に単独の入口ポイントと出口ポイントを持つ、という条件が満たされていれば、複合化は自由にできます。図 1 は、並行実行のワークフロー パターンを示しています。

   注意： BPM では、並行実行は、同時には実行されません。同一のスレッドでそれぞれのパスが実行され、まず最初のパスが終了するまで進行

し、次に新しいパスに移り、すべてのパスが進行できなくなるか、またはすべてのパスが **AND** ノードに進む状態になるまで、そのように作業が進行します。開始パスがどれになるかは一定ではありません。

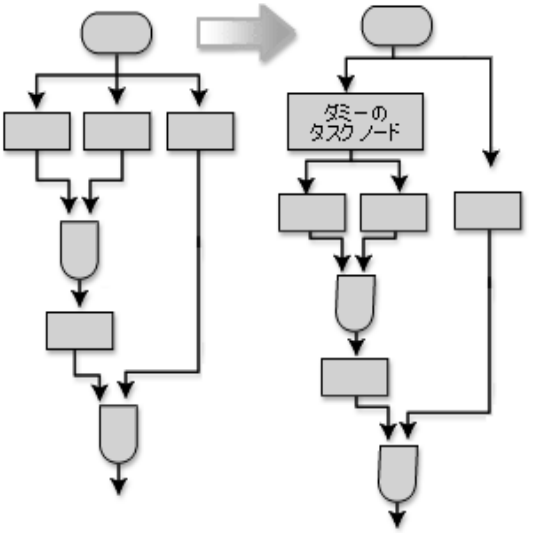
図 1　並行実行パターンの実装



2. 並行パスのサブセットが別のノードで合流する必要がある場合、そのサブセットの起点としてダミーのタスク ノードを設定し、そのダミー タスク ノードは無条件で終了しているものとマークします。これによって、ネストされた並行処理が機能的に作成され、並行実行パスが単独の入口ポイントと出口ポイントを持つ原則が確実に守られます。

   図 2 は、複数のタスク ノード、複数の入口ポイントを持つワークフローが、ダミー タスク ノードを使用することによって、並行実行パスの入口ポイントが 1 つになり、1 つの AND ノードに合流するように変わるしくみを示しています。

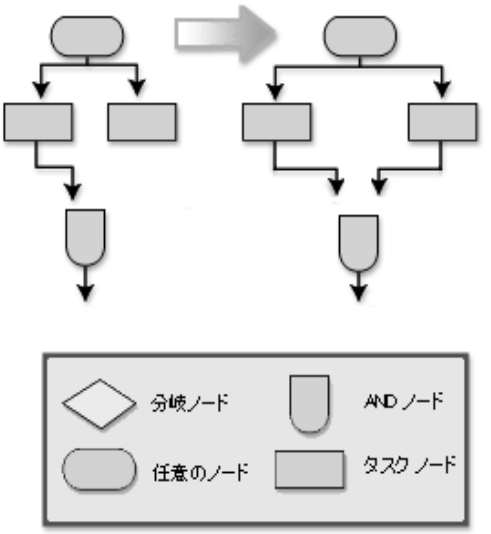
図 2　単独の入口ポイントと出口ポイント方式を使って、ネストされた並行処理を作成する



3. パスが途中で消失するようにすることはできません。後続するパスがないノードにパスがつながっていると、そのパスは消失してしまいます。消失しないように修正するには、そのパスが直接 AND ノードにつながるように線を追加します。図 3 の左側の部分は、消失するパスがあるワークフローを示しています。図 3 の右の部分は、このワークフローをどのように実装したらよいかが示されています。

   注意： 消失するパスは、「タイムアウト付きイベント」に示されているワークフロー以外、設定できません。

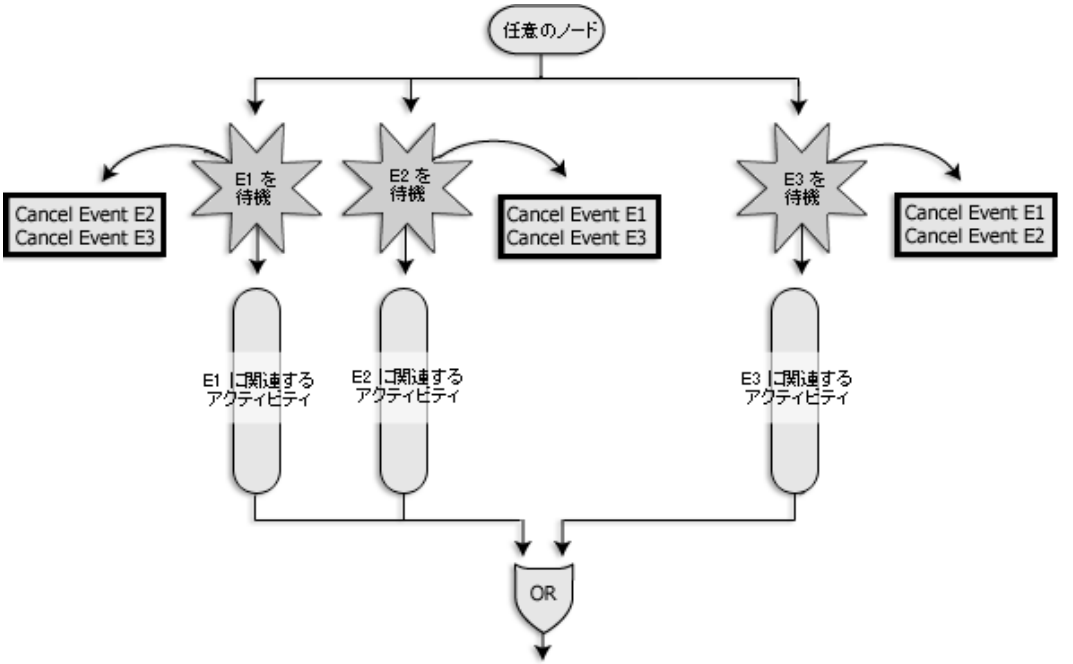
図 3　必ず 1 対の入口ポイントと出口ポイントを持つようにする



イベント選択

「イベント選択」パターンは、1 つのノードから分岐するすべての並行パスが一定のイベントが発生するのを待ち、最初に起きたイベントによって、どのパスが実行されるか決まるものです（図 4 を参照）。複数の並行パスのうち、実行されるパスは、どのイベントが最初に発生するかによってただ 1 つだけです（その他のパスはすべて抑止され、発動しません）。実行パスが互いに排他的になることを保証するため、次のガイドラインに示されているように イベントの取消しアクションを使用します。

図 4 「イベント選択」パターンの実装



「イベント選択」ワークフロー パターンを適切に実装するには、以下の手順に従ってください。

1. **1** つのノードから出るすべての並行パスがイベント ノードで開始されるようにします。図 4 では、3 つの並行パスが [任意のノード] を出て、各パスが 3 つのイベント ノードで開始するようになっています。ノードの数は、すべてのノードがイベント ノードで開始される限り、必要に応じていくつでも設定できます。
2. イベントの取消し アクションを使用して、相互排他性が確立するようにします。各イベント ノードに定義されたアクション リストには、兄弟イベント ノードごとに イベントの取消し アクションを 1 つずつ入れます。

   注意： 2 つのノードが同じ親を持つ場合、その 2 つのノードは兄弟となります。

a. イベントの取消し アクションでは、当該ワークフロー パターンのイベント兄弟以外のイベント ノードは一切取り消さないようにします。この イベントの取消し アクションによって、並行パスの 1 つだけが実行できるようになることが保証されます。この イベントの取消し アクションは、[イベント ノード] の [アクション] タブで定義できます。

b. イベントが互いに排他的であることが保証されている場合でも、イベントの取消し アクションを組み込むようにします。そのようにすることにより、移行ツールによるこのワークフロー パターンの移行がよりスムーズにできるようになります。

3. **イベントの取消し アクションは、そのイベント ノードの [アクション] タブで宣言されている他のどのアクションよりも前に配置します。**できれば、イベントの取消し アクション以外のアクションは、タスク ノードに配置し、そのノードをイベント ノードの後に置くようにしてください。

4. **すべての並行パスが同一の OR ノードに合流するようにします。**図 4 は、すべての並行パスが同一の OR ノードに合流するパターンを示しています。

5. **実行パスには、複合構成要素（サブ並行処理、ネストされた「イベント選択」パターン、およびループなど）を自由に入れることができます。**

タイムアウト付きイベント

「タイムアウト付きイベント」パターンは、互いに排他的なイベントのセットが、一定の時間枠内または期限前に発生する必要がある場合を表しています。タイムアウト前に発生した最初のイベントの実行パスは完了するまで実行され、他のパスはタイムアウト パスも含めて抑止されます。タイムアウト前にどのイベントも発生しない場合は、タイムアウト パスが実行されます。タイムアウト後にイベントが発生しても、対応する実行パスが開始されることはありません。タイムアウト ロジックは 時限イベント アクションか タスク期日を設定 アクションのどちらかを使用すれば実装できますが、タスク期日を設定 アクションのほうがより明確でわかりやすい構成になるので、タスク期日を設定 アクションを使用することをお勧めします。図 5 は、このワークフロー パターンの適切な実装形態を示しています。時限イベント アクションを使ったベスト プラクティスの説明については、「タイムアウト付きイベント」を参照してください。

「タイムアウト付きイベント」ワークフロー パターンを適切に実装するには、以下のガイドラインに従います。

1. パスは、**1** つを除いて他はすべてイベント ノードで開始するようにします。それぞれのイベント ノードには、一定のイベントが発生したことを示す変数を設定します。図 5 では、2 つのイベント ノード、E1 と E2 が示されています。

2. その唯一の例外パスを開始するノードとして、タスク ノードを **1** つ設定します（ここでは「タイムアウト タスク」ノードと呼びます）。タイムアウト タスク ノードの後には分岐ノードを設定し、イベント ノードで設定された変数値を検査します。図 5 では、この分岐ノードは「イベント発生?」という名前になっています。「タイムアウト タスク」ノードを使用するにあたってのガイドラインは、次のとおりです。

   a. タイムアウト タスク ノードのアクティブ化リストにあるアクションは、タスク期日を設定 だけです。このアクションにより、タイムアウト タスク ノードの期日が設定されます。タイムアウト タスク ノードの 実行時リストまたは 完了マーク時 リストには、どのようなアクションも含まれません。

   b. 最後のサブアクション（タイムアウト アクション）では、タイムアウト タスク ノードを完了とマークする必要があります。図 5 の右側分岐にその使用例が示されています。タスクに完了マークを付ける アクションを使用して、タイムアウト起動後にタイムアウト タスク ノードから実行が継続されるようにします。他のサブアクションはあってはなりません。

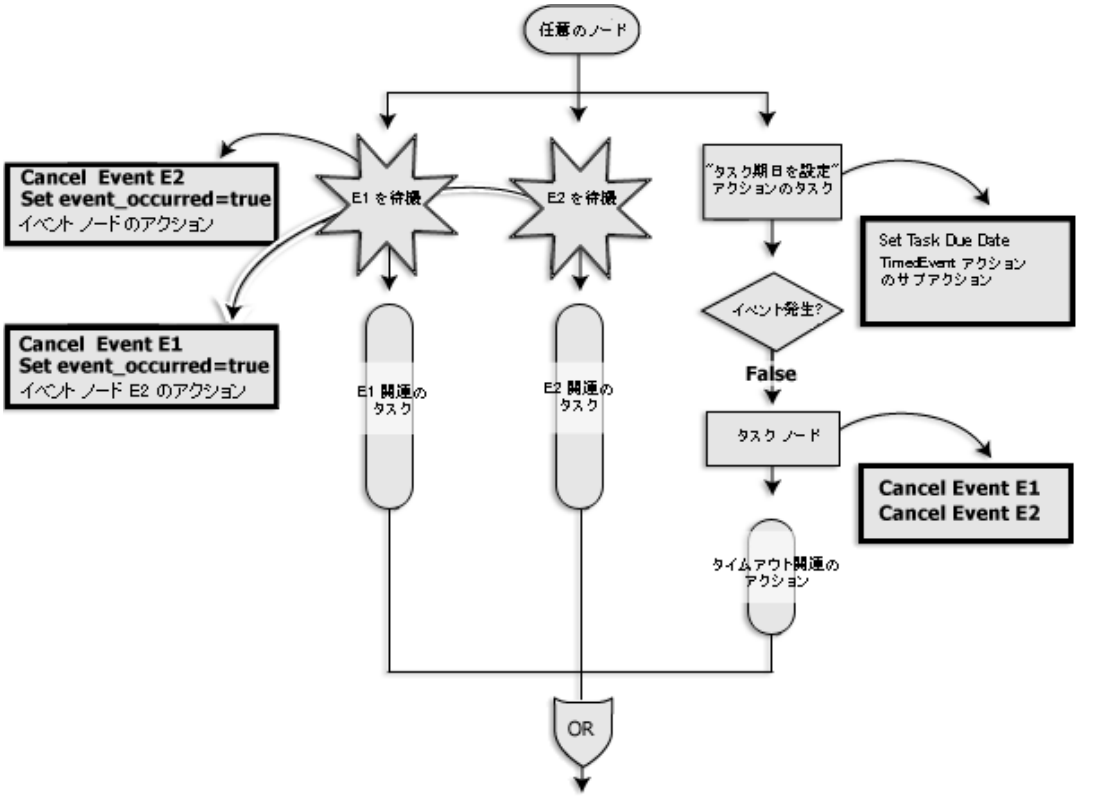
3. すべての実行パスが同一の **OR** ノードに合流するようにします。

4. タイムアウトが実際に発生したかどうかをチェックします。タイムアウト タスク ノードから出るパスは、イベントが発生したかどうかとは無関係に、期日到来によって実行が開始されます。イベントが実際に発生した場合は、タイムアウト パスの実行は停止される必要があります。イベント発生によりそのように停止させるためには、タイムアウト タスク ノードの後に分岐ノードを挿入します。この分岐ノードは、イベントによって設定される変数の値を検査します。変数が設定されていれば、イベントが発生済みであることを示しています。その場合、対応する分岐ノードの分岐が消失するようにします。これは、その分岐の次にノードを指定しなければ自然にそのようになります。*false* の場合、分岐ノードの分岐は次のノードに移るようにします（図 5 で「イベント発生?」の**分岐**ノードの後の分岐を参照）。

5. タイムアウトが発生したら、他のイベントはすべて取り消します。タイムアウト パスには イベントの取消し アクションを設定し、イベントが起動することのないようにすべてのイベントを取り消します。できれば、これらの イ

ベントの取消し アクションは分岐ノードの直後のタスク ノードに配置してください。図 5 は、分岐ノード後に 2 つの イベントの取消し、E1 と E2 がある実装例を示しています。

図 5 「タイムアウト付きイベント」パターンの実装



イベントによる取消し

「イベントによる取消し」パターンは、実行パスが特定のイベントによって取り消される（メッセージ到着により、以前に発注されていた注文を取り消す場合など）場合を表しています。実行パスが実行途中で取り消すことができるかどうかによって、2 つのタイプが考えられます。

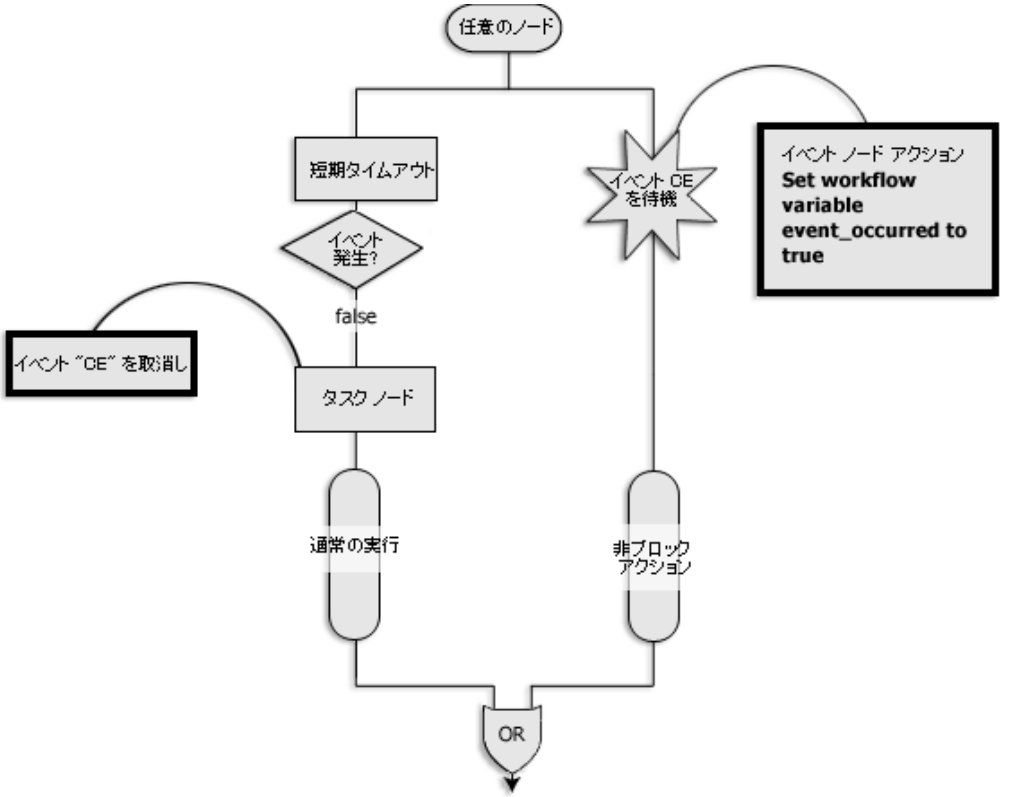
1. 中途取消しなし：実行パスがまだ開始されていない場合、キャンセル イベントにより実行パスの実行が中止されます。実行パスの実行が開始されている場合、キャンセル イベントは無効となり、実行パスは完了まで実行が継続されます。
2. 中途取消しあり：キャンセル イベントは、取消し対象となるパスの実行期間全体にわたって有効となります。言い換えれば、実行パスは、キャンセル イベントを受信したら、実行の途中でも中止できることになります。ビジネス ロジックによっては、キャンセル イベント受信前に実行された作業部分を元に戻す（取り消す）ため、一定のクリーンアップ処理が必要な場合があります。

どちらのタイプのほうが実際のビジネス ロジックに近いかを判断した上で、選択したタイプを次のガイドラインの提示に従って実装してください。最初の方法、すなわち「中途取消しなし」のタイプのほうが、部分的に完了した作業を取り消すクリーンアップ処理の必要がないため、実装が簡単です。 2 番目の方法、「中途取消しあり」はより強力で、取消し対象の実行パスの実行期間が長い場合はこちらの方法が唯一の選択肢ということもあります。

「中途取消しなし」タイプ

図 6 は、取消し対象のパスが実行前にキャンセル イベントを受信しない限りパスが実行される場合の、「イベントによる取消し」パターンの推奨される実装を示しています。このタイプを適切に実装するには、次の手順に従ってください。

図 6　「イベントによる取消し」パターンの「中途取消しなし」タイプの実装



1. キャンセル パスの最初は、キャンセル イベント（**CE** と表記）を待機するイベント ノードにする必要があります。
2. イベント ノードが適切に登録され、キャンセル メッセージの受信によってただちに開始されるようにするには、通常実行パスの最初に短期間実行をブロックする（静止状態を入れる）タスク ノードを置きます。静止状態を入れることにより、イベント ノードが確実に登録されます。キャンセル イベントが受信されると、それによって取消しを待機していたイベント ノードが開始されます。
3. イベント ノードに変数を設定するアクションを追加します。この変数の値は、通常実行パスを継続して実行するかどうかを判断するため、通常実行パスによって静止状態終了後ただちに検査されます。

4. 分岐ノードを、手順 3.「イベントによる取消し」に示された変数値を検査するタイムアウト ノードの後に追加します。変数が設定された場合、この分岐ノードの *true* 分岐は、その分岐の次のノードを定義しないことにより、消失するようにします。*false* 分岐については、通常実行を開始するノードに移るようにします。図 6 はこのガイドラインを示しており、「イベント発生 ?」分岐ノードの後に *false* 分岐が配置されています。

5. 通常実行パスが実行開始されたら、イベント ノードがその後に開始されることはあってはなりません。キャンセル イベントを取り消す イベントの取消し アクションを追加することにより、イベント ノードを無効化します。このアクションは、分岐ノードの直後に配置する必要があります。図 6 は、分岐ノードの直後のタスク ノードとそれに関連付けられた イベントの取消し (CE) アクションを示しています。

6. どちらのパスも同じ **OR** ノードに合流させます。

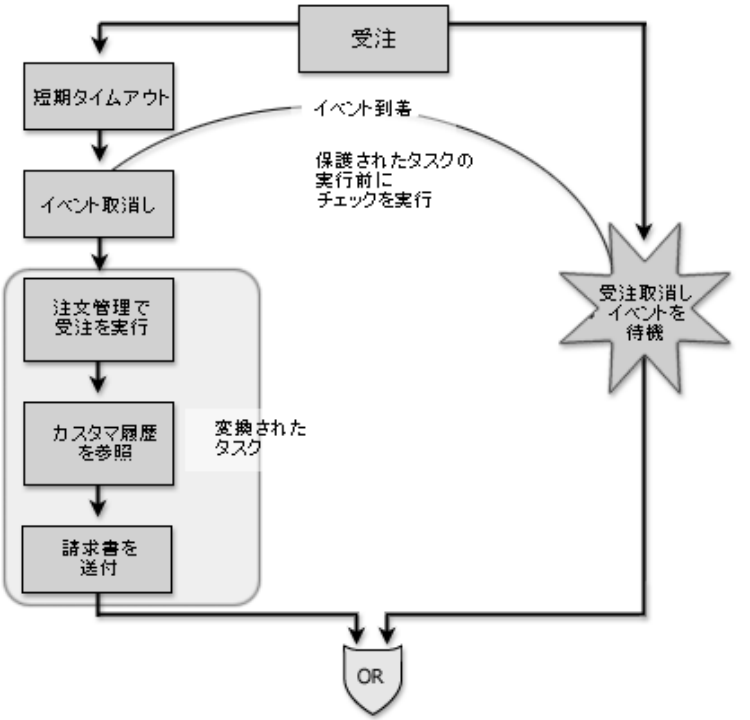
「中途取消しなし」の「イベントによる取消し」は、重要タスクを実行する前に特定のイベントが発生したかどうかをチェックするように設計されたパターンです。このパターンの簡単な例は、受注処理ワークフロー内に見られます（図 7 を参照してください）。受注処理ワークフロー内のタスクとしては、次のようなものが考えられます。

1. 顧客ステータスのチェック
2. 受注ステータスの実行
3. 注文の有効性チェック

   注意： 次の手順に進む前であれば、任意の時点で注文を取り消すことができます。図 7 に示されているように、依頼済みの注文の実行を処理する保護タスクのセットより前であれば、キャンセル イベント ノードを挿入できます。

4. 注文管理システムに対する注文の実行指示

図 7　受注処理における「中途取消しなし」タイプの例



「中途取消しあり」タイプ

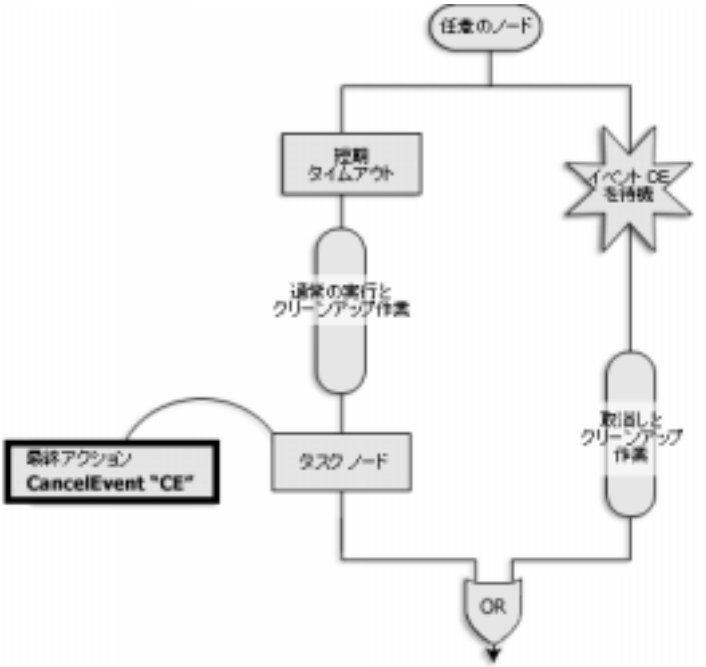
図 8 は、「中途取消しあり」タイプの推奨実装を示しています。ただし、このタイプの実装では、クリーンアップ ロジックが正しく実装されるために、取消し対象の通常実行パスとキャンセル パスとの間により高度な調整が必要です。

一般に、キャンセル パスでは、必要なクリーンアップ ロジックを実行する必要があります。このクリーンアップ ロジックは、通常実行パスがどこまで先に進んだかによって異なる可能性があります。通常実行パスの進行度の検出は、通常実行パスに一定の変数を設定し、キャンセル イベントの発生時にキャンセル パスでそれらの変数を検査することにより行なうことができます。同様に、通常実行パスでは、キャンセル パスがアクティブ化したことを検出し、キャンセル パスでは実行されない一定のクリーンアップ アクションを実行する必要がある場合があります。作成したワークフローが実際のビジネス プロセスを正確に反映

したものになるための適切な調整を実装する作業は、ワークフロー作成者自身が適宜行なってください。クリーンアップ処理は 1 つのパス上で行なうことを推奨します。

図 8 は、「中途取消しあり」タイプの実装の骨格だけを示したものです。取消しおよびクリーンアップ ロジックの詳細は、記載されていません。

図 8 「イベントによる取消し」パターンの「中途取消しあり」タイプの実装



以下の一般的な手順は、このワークフロー パターンを適切に実装する方法を示しています。

1. キャンセル パスの最初は、キャンセル イベント（**CE** と表記）を待機するイベント ノードにする必要があります。
2. 通常実行パスの最初は、12 ページの「「中途取消しなし」タイプ」に示したように、短期間実行をブロックするタスク ノードにします。これによって、イベント ノードが適切に登録されます。

3. **通常実行パスの最後のアクションでは、イベントの取消し アクションを使用して イベントの取消し （CE）を取り消すことにより、イベント ノードを無効化する必要があります。**これによって、通常実行パスが実行完了し、OR ノードに到達した以降にイベント ノードが起動することがなくなります。図 8 では、左側分岐の最後にイベント ノードを完了するためのタスク ノードを設定してこの処理を行なっています。
4. **どちらのパスも同じ OR ノードに合流させます。**キャンセル イベント / メッセージでイベント ノードがアクティブ化するのは、任意の時点ではなく、通常実行パスが静止状態に入ったときに限られますので注意してください。そうなっていることにより、通常実行パスが静止状態に入るポイントは普通は少数ですから、クリーンアップ ロジックが単純化されます。したがって、取り消し / クリーンアップの組み合わせ作業がごく少ない箇所で済みます。

実行タイムアウト

「実行タイムアウト」パターンは、「イベントによる取消し」パターンと似ていますが、パスの実行を取り消す原因はイベントではなくタイムアウトです。図 9 は、このワークフロー パターンの推奨される実装形態を示しています。

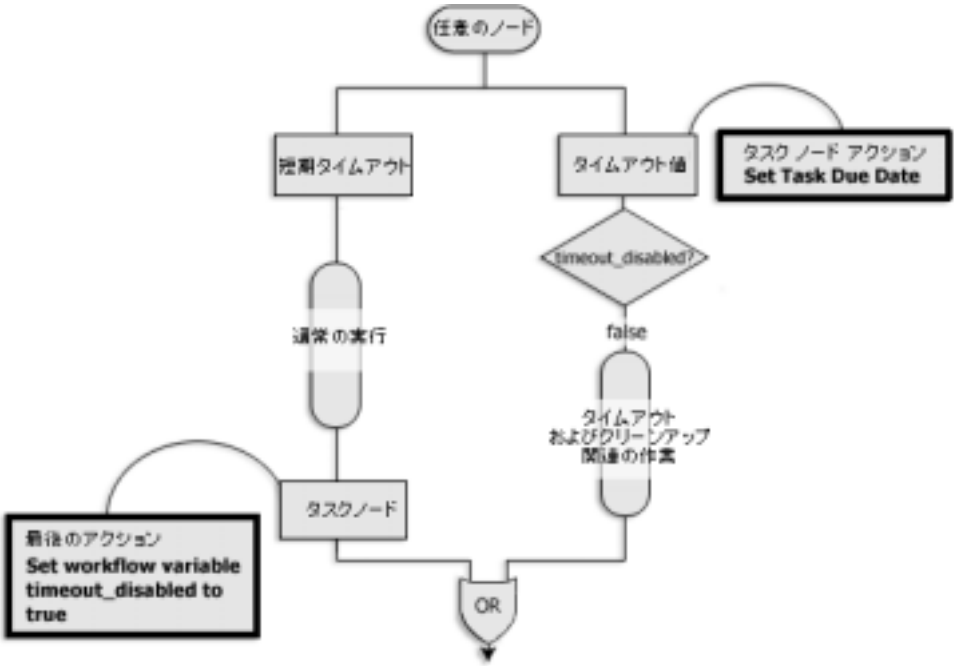
実行タイムアウト パターンでは、タイムアウトが起動した場合の、通常実行パスとタイムアウト パスとの間に一定レベルの調整が必要な場合があります。さらに、複雑なクリーンアップ ロジックが必要なこともあります（これに関しては「イベントによる取消し」の節を参照してください）。クリーンアップに必要なワークフロー アクションの実装は、ワークフロー作成者自身が行なってください。この節では、このワークフロー パターンの実装の骨格のみを示します。

「実行タイムアウト」パターンを実装する場合、以下の手順を実行してください。

1. **タイムアウト パスの最初は、タイムアウト アクションを含むタスク ノードにする必要があります。**タイムアウトは、時限イベント アクションまたは タスク期日を設定 アクションのいずれを使用しても実装できますが、タスク期日を設定 のほうがよりわかりやすいコードになるので、タスク期日を設定 アクションを使用することをお勧めします。通常実行パスとタイムアウト パスは、これらのパスに含まれる構成要素が単独の入口ポイントと出口ポイントをそれぞれ持つようになっている限り、どちらも自由に複合化することができます。
2. **どちらのパスも同じ OR ノードに合流させます。**

3. タイムアウト パスは、通常実行パスが実行終了したら（すなわち **OR** ノードに到達したら）、無効化するようにしなければなりません。そのためには、通常実行パスの最後で変数を設定し、分岐ノードをタイムアウト ノードの後に置いて、そこでこの変数の値をチェックするようにします。図 9 では、「timeout_disabled」という名前のブール変数が実行パスの最後で *true* に設定されており、この値が分岐ノードでチェックされます。タイムアウト パスが無効化されない場合、タイムアウト パスが進行します。

図 9 「実行タイムアウト」パターンの実装



ワークフロー パターンにおけるアクションの使用法

表 1 は、このマニュアルで前述したワークフロー パターンに関連するアクションをリストにしたものです。自動移行できるようにワークフローを最適化するには、ワークフローでこれら特定のアクションを使用するときに 表 1 に示すガイドラインに従ってください。

注意： 非同期アクションのコールバック アクション リストには、Set Task as Done 以外のアクション（ワークフローを開始 など）は定義しないでください。

表 1 : ワークフローアクションの使用に関するガイドライン

アクション	使い方
イベントの取消し	このアクションは、「イベント選択」、「タイムアウト付きイベント」、または「イベントによる取消し」の各パターンの実装時に相互排他性を確立するためにのみ使用する。
タスクに完了マークを付ける	このアクションは、自己に完了マークを付ける場合、または「タイムアウト付きイベント」パターンを実装する場合にのみ使用する。
タスクの完了マークを外す	このアクションは使用しないこと。
タスク期日を設定	このアクションは、「タイムアウト付きイベント」パターンまたは「実行タイムアウト」パターンのタイムアウト ロジックを実装する場合にのみ使用する。
タスクを実行	このアクションは、自己を自動実行する場合にのみ使用する。
OR 結合	OR 結合 は、前述のワークフロー パターンの実装、または複数の開始ノードの結合にのみ使用する。

表 1 : ワークフローアクションの使用に関するガイドライン

アクション	使い方
AND 結合	AND 結合 は、「並行実行」パターンを実装する場合にのみ使用する。

タスク ノードの使用法

WebLogic Integration Studio のタスク ノードには、任意の数のアクションを 作成時、アクティブ時、実行時、および 完了マーク時 リストに入れることができます。これらのリストは、WebLogic Integration Studio の [タスクのプロパティ] ダイアログで定義します。[タスクのプロパティ] ダイアログの詳細については、『WebLogic Integration Studio ユーザーズ ガイド』を参照してください。

一般に、タスクには次の 2 種類があります。

表 2 : タスクの種類

タスク	使い方
ユーザ タスク	ユーザに割り当てられているタスク。ユーザ タスクは、ユーザまたはロールによって明示的に実行されなければ、実行されることはない。
自動タスク	ユーザの介在が不要なタスク。自動タスクは、通常ユーザの介在なく一連のアクションを実行し、自己に完了マークを付けて次のノードへと実行を進める。

ユーザ タスクのガイドライン

ユーザ タスクを実装する場合、以下のガイドラインに従ってください。

1. アクティブ時 リスト には、事前割り当てタスクを配置します。アクティブ時 リストの最後のアクションは、3 つのタスク割り当てアクション のいずれかでなければなりません。それらのアクションは次のとおりです。
   a. ユーザにタスクを割り当て
   b. ロールにタスクを割り当て
   c. ルーティング テーブルを使用してタスクを割り当て

2. Executed リストには、事後割り当てタスクを配置します。これらのアクションは、ユーザが明示的にタスクを実行する場合に実行されます。Executed リストの最後のアクションは、タスクに完了マークを付けるため、タスクに完了マークを付ける アクションにする必要があります。
3. タスクに関連するアクション（タスク コメントを設定 および タスク期日を設定 など）はすべて「タスクの割り当て」アクションの直前に置くようにします。

自動タスクのガイドライン

自動タスクを実装する場合、以下のガイドラインに従ってください。

1. アクションはすべて アクティブ時 リストに配置します。
2. Executed リストにはアクションを配置しないでください。
3. アクティブ時 リストの最後のアクションは、タスクに完了マークを付けるアクションにして、タスクに完了マークを付ける必要があります。

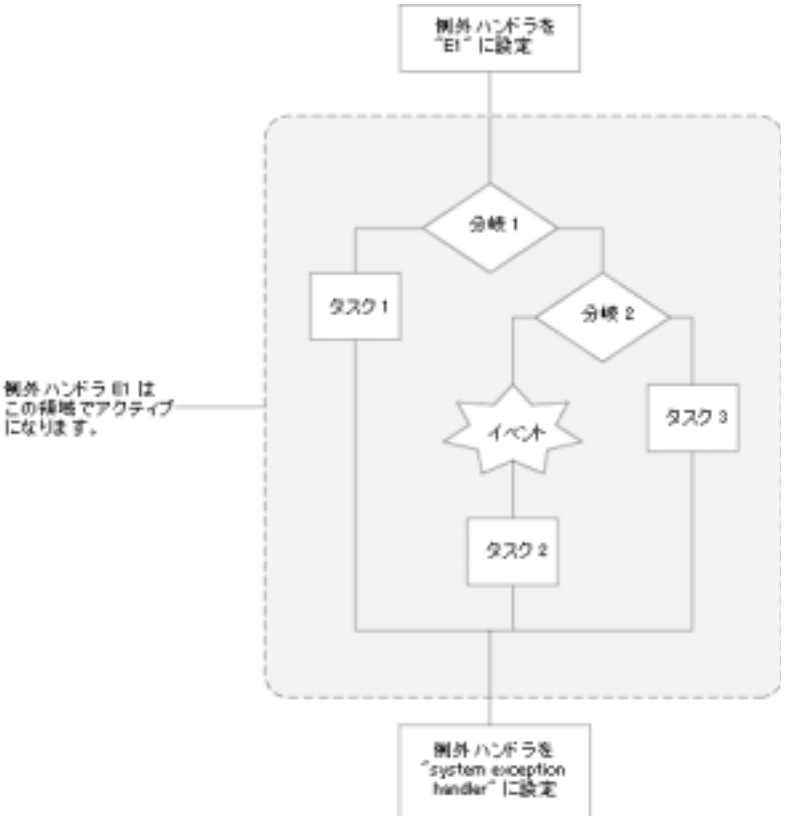
例外処理

将来バージョンの WebLogic Integration では、構造化された例外処理を使用する予定です。将来の WebLogic Integration 製品リリースにスムーズに移行できるようにするため、例外ハンドラを設定する時に、以下の事項を考慮するようにしてください。

1. 例外ハンドラをワークフローの中ほどに設定する場合、順番的に最も前にあるノードのデフォルトの例外ハンドラに対して後ろになるようにそのハンドラを設定し、設定ノードおよび解除ノードによって、出入りするパスがない適切なブロックが形成されるようにします。次の図、図 10 では、この種の例外ハンドラ ブロックを示しています。一般的に、例外ハンドラがアクティブになる範囲はできるだけ小さくするようにしてください。
2. 「タイムアウト付きイベント」パターンまたは「実行タイムアウト」パターンを使用する場合、タイムアウト パス内の例外ハンドラは、変更および追加しないようにします。

3. 「イベントによる取消し」パターンを使用する場合、例外ハンドラを変更しないようにします。

図 10　例外ハンドラ ブロックの例



Studio プラグインを使用する場合のガイドライン

WebLogic Integration Studio では、一部のワークフロー コンポーネントの機能を拡張するプラグインをサポートしています。プラグインの機能はそのままでも継続しますが、できればカスタマイズしたプラグインに対して EJB を使用して、自動移行ができるようにワークフローを最適化してください。『WebLogic Integration BPM プラグイン プログラミング ガイド』を参照してください。

あ

い

か

さ

し

た

は

ふ

へ

ま

ゆ

れ

わ