Novell exteNd Composer™ Process Manager

4.2

www.novell.com

ユーザガイド

Novell®

保証と著作権

米国 Novell, Inc.
1800 South Novell Place
Provo, UT 85606

www.novell.com

Novell exteNd Composer Process Manager ユーザガイド
2003 年 1 月
000-000000-000

オンラインマニュアル： この製品およびその他の Novell 製品のオンラインマニュアルや更新情報については、www.novell.com/documentation を参照してください。

目次

このガイドについて

目的

このガイドでは、eXtend Composer の Process Manager を使用して、Web サービスに完全または部分的に依存する、長時間実行され、多くの場合に大規模な自動化プロセスを構築する方法について説明します。このガイドは、『eXtend Composer ユーザガイド』の代わりではなく、補助マニュアルとなることを目的としています。

対象読者

このガイドは、自動化された動作 (つまり、ビジネスプロセスモデル) の連携システムの設計と配備に携わるユーザーを対象にしています。このようなシステムの開発に参加するすべてのユーザーは、このガイドをお読みになることをお勧めします。

前提条件

XML 関連の標準 (スキーマ、XSL、および XPath を含む)、ドキュメントオブジェクトモデル、および WSDL の例と動機に加え、ファイルのパッケージ (JAR/EAR/WAR ファイル) に関する基本的な J2EE の概念について精通している必要があります。

追加のマニュアル

Novell exteNd のユーザガイドおよびその他のマニュアルの完全なセットは、Novell マニュアルの Web サイト (**http://www.novell.com/documentation-index/index.jsp**) を参照してください。

構成

このマニュアルは、次のように編成されています。

章	説明
第 1 章、「Composer とプロセス管理へようこそ」	Process Manager およびプロセスモデルの重要な概念の定義と概要を説明します。

章	説明
第 2 章、「プロセスのモデル化の準備」	Process Manager の設計時の概念とユーザインタフェースの機能を簡単に説明します。
第 3 章、「Process Designer のユーザインタフェース」	プロセスの設計時に配慮を要する重要な要因の概要を説明し、さまざまな方法を示します。簡単な例をウォークスルー形式で説明しています。
第 4 章、「プロセスの作成とテスト」	プロセス配備オプション、および Process Administrator コンソールを使用してプロセスインスタンスを管理する方法を説明します。
第 5 章、「高度なヒント」	Web Service Receive アクティビティおよび Synchronize Subprocesses アクティビティを含む方法とともに、キューに登録されているワークアイテムを含むユーザ中心のワークフローを実装する方法についても説明します。
第 6 章、「配備の準備」	プロセスのパッケージ化と配備に関する概念について説明します。
第 7 章、「プロセスのランタイム管理」	実行中のプロセスの監視および制御に使用できる管理コンソールの概要を説明します。
付録 A、「テスト」	設計時のテストとサーバサイドテストの重要な違いを説明します。
付録 B、「パフォーマンスの調整」	パフォーマンスが非常に重要な環境において、プロセスサーバのパフォーマンスを向上させるために調整できるパラメータについて説明します。
付録 C、「用語集」	プロセス管理のさまざまな用語を定義します。

PDF マニュアルについて

PDF マニュアルは、Acrobat Reader 3.0 以降を使用して表示できます。Reader の最新バージョン (本マニュアル執筆時点では 5.1) は、次の URL で無料でダウンロードできます。

http://www.adobe.com/products/acrobat/readstep.html

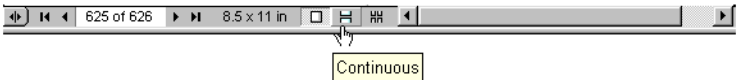
ナビゲーション

このガイドを Acrobat Reader で表示する際は、さまざまなナビゲーション機能を使用できます。

- (ウィンドウの左側にある) ［ブックマーク］フレームには、ガイドのコンテンツが章名、見出し、および小見出し別にリストされます。コンテンツツリーにリストされている各トピックは、クリックできるリンクです。ツリーノードの下層にあるサブツリー全体 (すべてのチャイルド) を開くには、<Ctrl> キーを押しながらペアレントノードをクリックします。［ブックマーク］フレーム自体の表示 / 非表示を切り替えるには、<*F5*> キーを押します。
- ガイドの目次 (iii ページ) の各項目は、クリックできるリンクです。クリックすると、説明テキストに直接移動します。これは索引でも同様です。
- Web アドレス (URL) が表示されている場合は、アドレスをクリックすると、通常は、そのサイトがブラウザに表示されます。URI は、青色でない場合や下線が引かれていない場合でも、一般的にホットリンクです。ホットリンクであるかどうかについては、マウスを URI の上に移動することによって確認できます。ホットリンクである場合は、カーソルが矢印から指の形に変わります。
- 章内および章間の相互参照もクリックできます。相互参照の場合も、マウスを相互参照に合わせたときのカーソルの形から、クリックできることがわかります。
- ガイド内の特定のページに直接移動するには、<*Ctrl*>+<*N*> キーを使用します。ページ番号の入力を求めるダイアログボックスが表示されます。
- テキスト検索を実行するには、<*Ctrl*>+<*F*> キーを使用します。

テキストのコピーと貼り付け

- PDF のテキストをクリップボードにコピーするには、最初にテキスト選択ツールを有効にして (ショートカット : キーボードの <V>) から、ドラッグしてテキストを選択します。
- 複数の PDF ページにまたがっている大きなテキスト部分を選択するには、まず、Acrobat ウィンドウの最下部にあるボタンバーで、「［連続］ページモード」アイコンをクリックします (次の図を参照してください)。次に、<Shift> キーを押しながらドラッグしてテキストを選択します (または、<Ctrl>+<A> キーを使用してすべてを選択します)。［コピー］を選択して、選択したテキストをクリップボードにコピーできます。



- クリップボード内の選択範囲を Word などのアプリケーションに貼り付ける場合にテキストのスタイルを保持するには、貼り付け先のアプリケーションの［編集］メニューから、［形式を選択して貼り付け］(利用可能な場合) を選択します。たとえば、Microsoft Word では、これにより、テキストのスタイルを保持したまま、クリップボードのテキストを RTF (Rich Text Format) に貼り付けることができます。

印刷マニュアル

このマニュアルの PDF バージョンは、SilverStream から印刷および製本済みの形式で最小限の費用にて入手することもできます。詳細については、**ebizintegration@silverstream.com** にお問い合わせください。

サーバ側のインストールとセットアップ

Process Manager をサーバ側にインストールした後に、Composer プロセスサーバのデータベースオプションを手動で設定する必要があります。インストールを完了するには、次の手順に従います。

前提条件

データベースセットアップ手順を実行する前に、次の条件を満たしていることを確認する必要があります。

- SilverStream、WebSphere、または WebLogic アプリケーションサーバがインストールされている必要があります。
- Sybase、Oracle、または IBM DB2 データベースシステムがインストールされている必要があります (データベースの互換性に関する最新のリストについては、Composer Process Manager のリリースノートを参照してください)。
- 次の手順で、アプリケーションサーバが既存のデータベースまたは新しいデータベースにアクセスするようセットアップされている必要があります。
  - データベースシステムの管理機能を使用して、新しいデータベースを作成します。
  - 必要に応じて、データベースの ODBC データソースを作成します。
  - アプリケーションサーバの管理機能を使用して、新しく作成したデータベースをアプリケーションサーバのデータソースとして追加します (接続プールを含む)。

Process Manager データベースのセットアップ

プロセスサーバは、重要なプロセスデータをランタイムで保存するためにデータベースを使用します。この目的に使用するデータベースを指定してから、次の手順に従って、プロセスサーバをこのデータベースに「バインド」する必要があります。

注記 : 次の手順を実行する前に、前の節 (前述の「前提条件」) で説明されているように、Process Manager 専用のデータベースを作成してください。

➢ **Process Manager データベースをセットアップする**

1 アプリケーションサーバを起動します。

2 Composer Process Manager をアプリケーションサーバにインストールします (まだインストールされていない場合)。

3 Web ブラウザを使用して、アプリケーションサーバ上の Process Manager コンソール (**http://< ホスト名 >:< ポート >/eXtendComposerProcess/**) にアクセスします。コンソールのスクリーンショットおよび追加の情報については、189 ページ「[Process Database Info]」を参照してください。

4 次のように、Process Engine が未設定の状態であることを確認します。

- Process Engine Status に「Shut Down」と表示されている
- Process Database Info に「Invalid Configuration」と表示されている

5 [**Configure**] ボタンをクリックします。[Process Database Configuration] の画面が表示されます (189 ページ以降の「[Process Database Info]」を参照してください)。

6 Sybase、Oracle、または IBM DB2 の中から、Process Manager が使用するデータベースのタイプを選択します。

7 Process Manager が使用するデータベースに対して、アプリケーションサーバに固有の [**Pool Name**] を入力します。例 :

- SilverStream: `Databases/< データベース名 >/DataSource`
- WebSphere: `jdbc/<DBPoolName>`
- WebLogic: 任意の JNDI データソース名

WebSphere と WebLogic の両方では、接続プールを作成するときにユーザが指定した JNDI 名が使用されます。したがって、*ProcessPool* という名前の接続プールを *ProcessJNDI* という JNDI 名で作成する場合は、[Process Database Configuration] 画面 (190 ページの図を参照) で、Composer のプロセスコンソールの [Pool Name] フィールドに、「*ProcessJNDI*」と入力する必要があります。

8 データベースに対して、[**User Name**] (「dba」など) および [**Password**] (<sql> など) を入力します。

注記 : [Process Database Configuration] 画面を使用して Process エンジンのデータベースを設定する場合 (190 ページを参照)、SilverStream または WebSphere サーバの [**Username**] および [**Password**] は、データベースのユーザ名 / パスワードになります (たとえば、Sybase データベースでは「dba/sql」)。ただし、WebLogic では、Process データベースの設定に必要なユーザ名 / パスワードは、WebLogic サーバのユーザ名 / パスワードです (たとえば、「system/weblogic」)。

9 [**Save**] ボタンを押します。正常に保存された場合は、[**Initialize**] ボタンが表示されます。

10 Process Manager データベーステーブルをセットアップするために、[**Initialize**] ボタンを押します。

11 正常に初期化された場合は、[**Status**] に「Connected - Ready」と表示されます。

12 ［**Return to Main**］ボタンを押して、Process Manager コンソールにアクセスします。

13 Process Manager エンジンを起動するために、［**Start**］ボタンを押します。正常に起動した場合は、［Process Engine Info］の［Status］に「Running」と表示されます。

14 Silverstream アプリケーションサーバの場合のみ、Process Manager エンジンが実行されたら、サーバ管理コンソールにアクセスして、Process Manager データベースを同期します。

注記： データベースを初期化したり、別のデータベースに変更したりするには、最初にエンジンを停止してから、前述の手順を繰り返す必要があります。

Process Manager をお使いになる前に：5 分間のチュートリアル

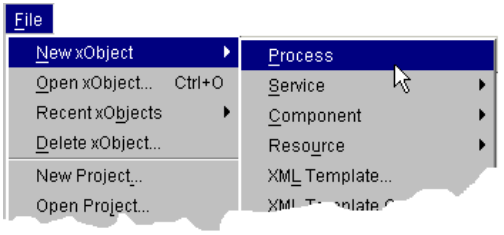
Process Manager でのプロセスの作成は、簡単で直接的です。次の手順では、基本的な手順を説明しています。関連する概念のより詳しい説明については、「Process Designer のユーザインタフェース」と「プロセスの作成とテスト」の章 (およびこのガイドの他の関連部分) を参照することをお勧めします。

次に示す基本的な手順は、常に同じです。

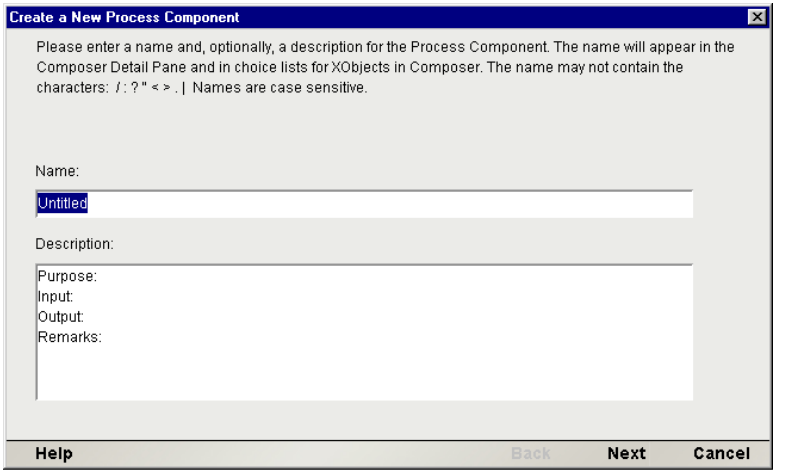
- アクティビティをプロセスグラフに配置する
- アクティビティを相互にリンクする
- 関連するデータマッピングを指定する

➢ プロセスを作成する

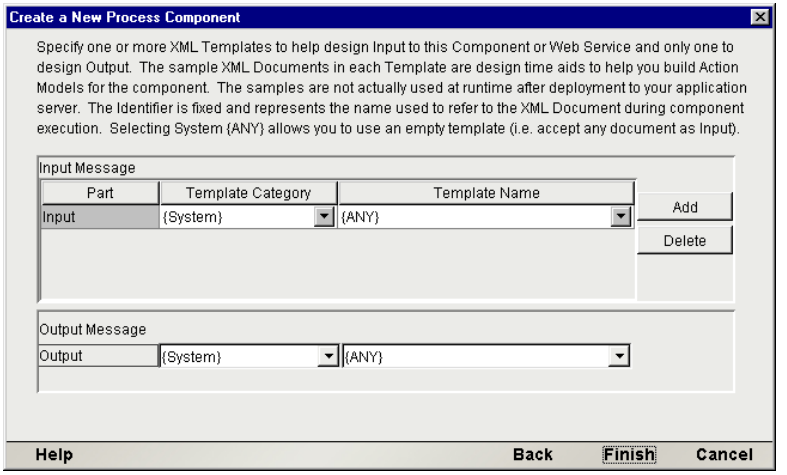
1 Composer を起動します。次に示すように、［**File**］メニューから、［**New xObject**］>［**Process**］の順に選択します。



2 ［Create a New Process Component］ダイアログボックスが表示されます。プロセスの名前を［**Name**］に入力します。

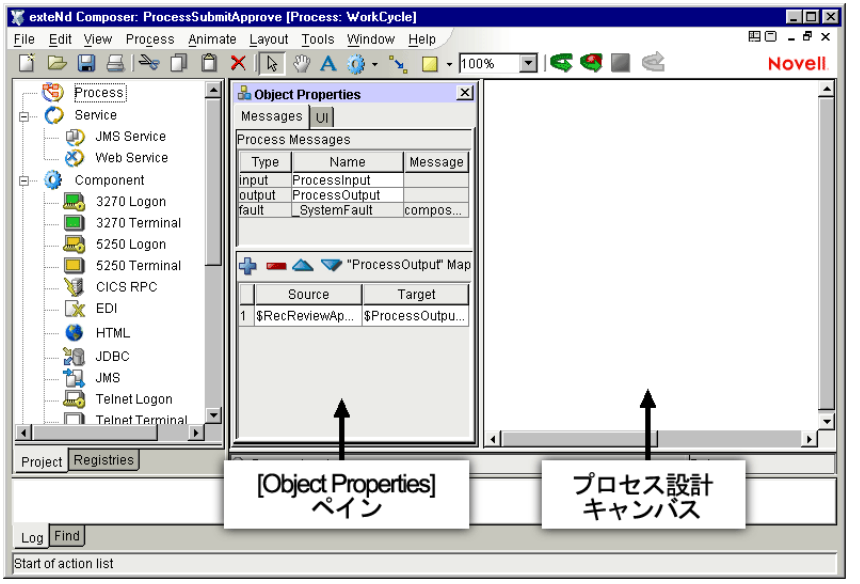


3 [**Next**]をクリックして、ウィザードの 2 番目の (最後の) ダイアログボックスを表示します。



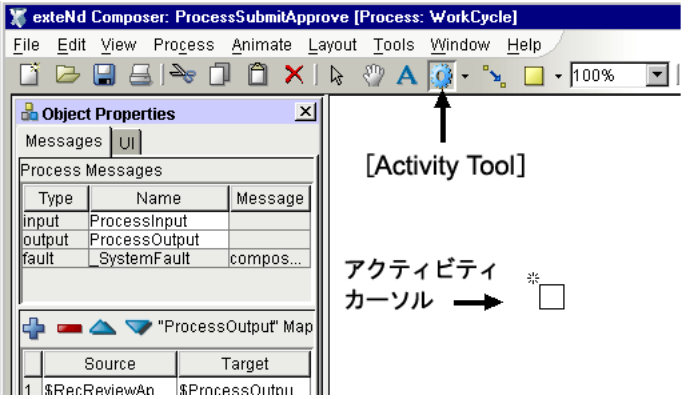
4 2 番目のダイアログボックスでは、プロセスの入力および出力メッセージの **XML テンプレート**を選択できます。Composer のその他のコンポーネントを通常の方法でセットアップするときと同様に、XML テンプレートを選択します (『*Composer ユーザガイド*』を参照してください)。

5 ダイアログボックスを閉じます。通常はネイティブ環境ペインである場所に、**空白のキャンバス** (プロセスを作成する領域を表します) が表示されます。

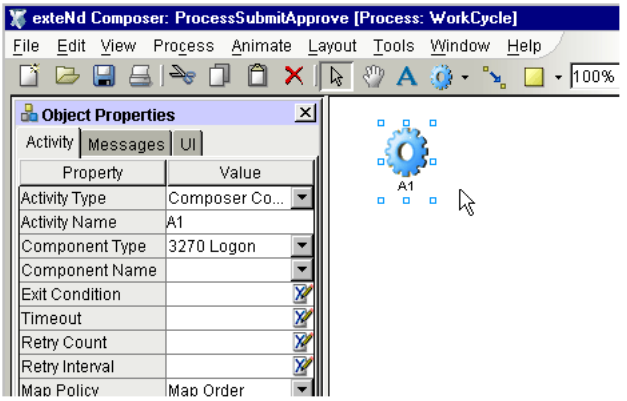


6 上の図に示すように［Object Properties］ペインが表示されている必要があります。表示されていない場合は、［**View**］メニューの［**Object Properties**］コマンドを使用して、表示 / 非表示を切り替えます。このペインを操作しやすい場所にドラッグしたい場合は、切り離す (移動する) ことができます。

7 **アクティビティツール**をクリックして選択します (次の図を参照してください)。カーソルの形が変わります。

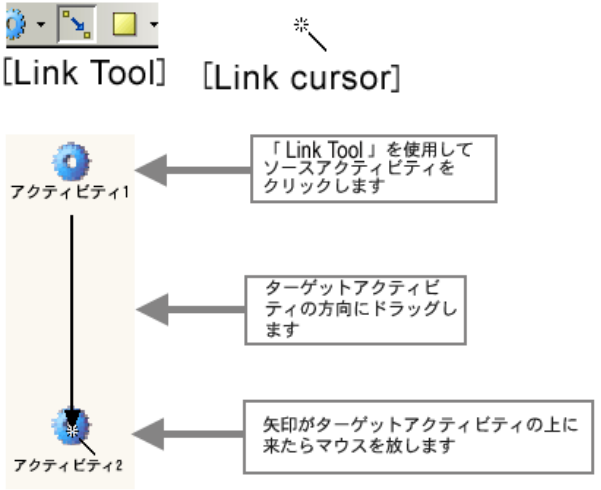


8 空白のキャンバスの任意の場所をクリックします。新しいアクティビティが作成され、境界の周りに青い拡大ハンドルが表示されます。



9 新しいアクティビティを選択した状態で (新しいアクティビティにフォーカスがある状態で)、［**Copy**］および［**Paste**］を選択して、コピーを 1 つ作成します (または、もう一度アクティビティツールを使用して、キャンバス上に別のアクティビティを作成します)。これで、**A1** と **A2** の 2 つのアクティビティが作成されました。

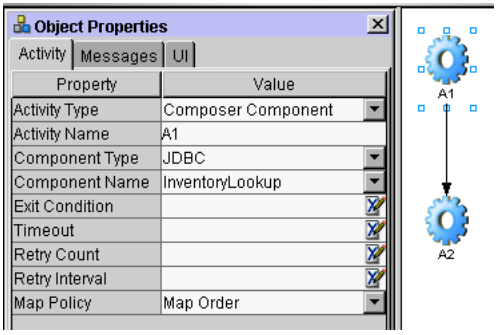
10 ツールバーの**リンクツール**を選択します。次に示す方法で、2 つのアクティビティをリンクで接続します。



11 続いて、アクティビティをコンポーネントに関連付けます (具体的な実装)。［Object Properties］ペインがまだ表示されていない場合は、［**View**］>［**Object Properties**］の順に選択して表示します。最初のアクティビティをクリックします。［Object Properties］ペインが更新され、最初のアクティビティの現在のプロパティが表示されます。

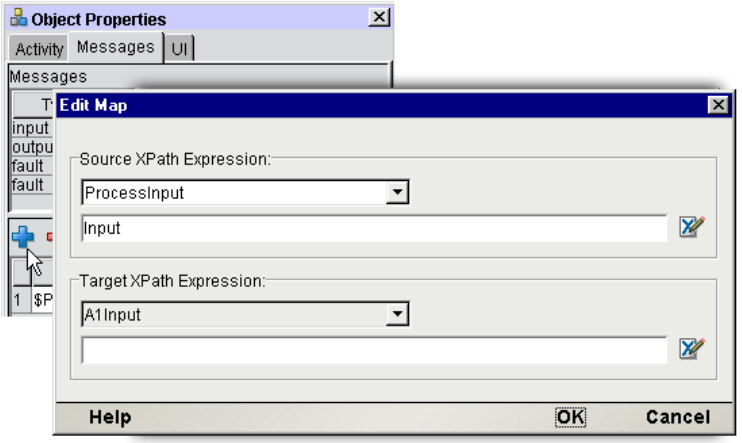
12 ［**Component Type**］の横のドロップダウンメニューから、最初のアクティビティの実装として使用するコンポーネントのタイプ (XML Map、JDBC など) を選択します。

13 ［**Component Name**］の横のドロップダウンメニューから、実際のコンポーネントを選択します (このリストには、現在のプロジェクトにすでに存在するコンポーネントの名前が表示されます)。次の図に、現時点でのオブジェクトプロパティの状態を示します。ここでは、*InventoryLookup* という名前の JDBC コンポーネントを選択したことを想定しています。



14 ［Object Properties］ペインの［**Messages**］タブをクリックします。

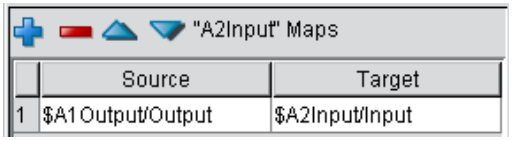
15 このアクティビティに入力を関連付けるので、［Messages］タブの青い**プラス記号**をクリックします。ダイアログボックスが表示されます。



16 これはプロセスの最初のアクティビティなので、メッセージの「**ソース**」として、［*ProcessInput*］を指定できます (これがデフォルトです)。［*ProcessInput*］は、前の手順 3 で入力に指定した XML テンプレートに対応するデータ構造を持ちます。

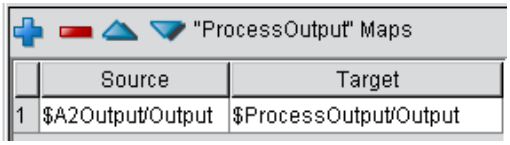
17 マップの**ターゲット**は **A1** (または現在選択されているアクティビティの名前) なので、ターゲットメッセージとして［*A1Input*］を、ターゲットメッセージの一部として［*Input*］をそれぞれ指定します (次の図を参照してください)。「入力部分」は、コンポーネント内の入力 DOM に対応していると考えることができます。

18 **アクティビティA2** (作成した2番目のアクティビティ)に対して、手順10～16を繰り返します。ここでも、選択されているアクティビティから外へではなく、選択されているアクティビティ (A2) の「中に」データをマッピングすることに注意してください。ここでは、現在のアクティビティの入力の［**Source**］は、前のアクティビティの出力 (*A1Output*) になります。したがって、データマッピングは次のようになります。



19 この簡単なプロセスの出力は、単に 2 番目のアクティビティの出力になります。このためには、A2 から *ProcessOutput* へのマッピングを明示的に指定するために、もう 1 つデータマッピングが必要です。このデータマッピングをセットアップするには、キャンバスの空白の部分をクリックして、すべてのアクティビティを選択解除します。続いて、［**Object Properties**］ペインに戻ります。ペインが変更されて、プロセス全体のプロパティが反映されていることがわかります。

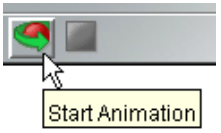
20 青い**プラス記号**をクリックして、次に示すデータマッピングを作成します。



21 作業を「**保存**」します。

以上で作業は完了です。最初のプロセスが作成されました。

ここで、プロセスをアニメーション表示して、期待どおりに動作することを確認することをお勧めします。ツールバーの**アニメーションツール** (次の図を参照) を使用して、最初から最後までプロセスを順番に実行する作業を開始できます。



追加のツールバーツールを使用して、アクティビティの実装を構成するコンポーネントをステップインまたはステップオーバできます。これらのコンポーネントの 1 つをステップインした場合、アニメーションは、アクションモデルレベルでリアルタイムに続行されます。つまり、コンポーネント内の任意のアクションをステップインまたはステップオーバして、ステップの順番にコンポーネントを実行できます。最後のアクションが実行されると、コンポーネントは戻されて、ユーザはプロセスグラフレベルに戻り、次のアクティビティへアニメーションを続行できます。

配備方法

Composer で作成したコンポーネントまたはサービスと同様に、プロセスは単に Composer xObject であるため、他のプロジェクトと同じ手順 (同じ Deployment Wizard を使用) に従って、プロジェクトの一部として配備されます。明確な理由から、配備したプロセスを実行するには、ターゲットアプリケーションサーバに Composer サーバとプロセスサーバの両方がインストールされている必要があります。配備の前に、すべての Composer 製品のサーバ側のインストールが完了していることを確認してください。

コンポーネントがサービスから呼び出されなければならないのと同様に、プロセスも Composer サービスから呼び出されなければなりません。このためには、単に任意のサービスのアクションモデルに Process Execute アクションを配置してから、サービスを配備します (サービスの入力メッセージをプロセスに直接渡すことができます。後述する「Process Execute アクション」の説明を参照してください)。サービスがパブリック URL 上に配備されている場合は、受信リクエストによって、関連付けられているプロセスの新しいインスタンスがトリガされます。それらのインスタンス、および関連付けられているすべてのアクティビティのステータスは、プロセスサーバコンソールを使用して監視できます (このガイドの「プロセスのランタイム管理」を参照してください)。

Composer サービスの配備の詳細については、必ず、ご使用のアプリケーションサーバ環境に合った『*eXtend Composer Server ユーザガイド*』を参照してください。

1 Composer とプロセス管理へようこそ

SilverStream eXtend Composer Process Manager へようこそ。このガイドは、Composer のコア機能が説明されている『eXtend Composer ユーザガイド』に付属しています。このガイドでは、Composer のコア機能に精通していることを想定しているので、『Composer ユーザガイド』をご覧になっていない場合は、このガイドを使用する前に読んで内容を確認してください。

作業を始める前に、まず Process Manager を既存の Composer 環境にインストールしておく必要があります。同様に、サーバベースのプロセスを実行する前に、使用しているアプリケーションサーバに Composer プロセスサーバソフトウェアをインストールしておく必要があります。

Process Manager を正しく使用するには、次の内容に精通している必要があります。

- ビジネスプロセス管理 (BPM) の概念
- 配備する特定のアプリケーションサーバ環境 (SilverStream、WebSphere、WebLogic など)
- XML、XSD (スキーマ)、および XPath
- WSDL (Web Services Description Language)
- Java WAR (Web Archive) ファイル
- サービスを作成および配備する eXtend Composer (eXtend Developer Workbench) の使用方法
- 基本構造化プログラミング概念およびオブジェクト指向設計パターン

また、Web Services Flow Language についての知識があるとさらに便利です。完全な仕様については、次を参照してください。

http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

この章では、主要な BPM (または「ワークフロー」) の概念について説明します。これにより、Web サービス、J2EE アプリケーション、および Composer アプリケーションと自動化されたワークフローとの関係についてさらに理解を深めることができます。

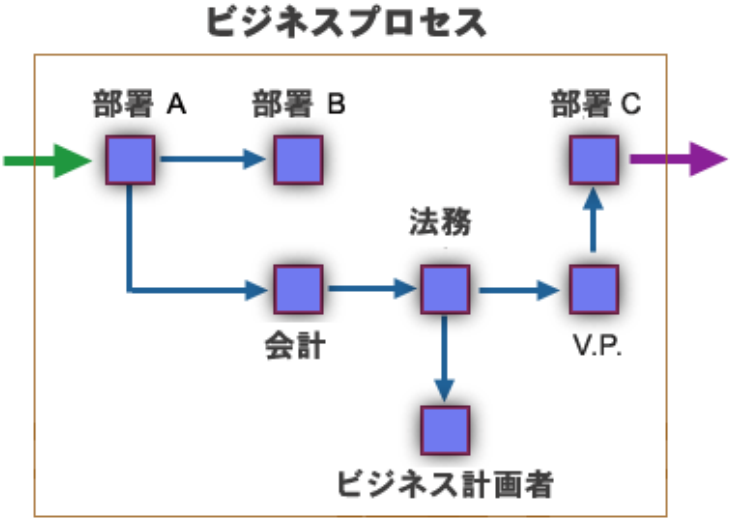
プロセス管理とは

ビジネスプロセス管理の目的は、業務処理を明確に定義された「タスクのシステム」としてモデル化することであり、これにより、参加者は、定められたコレオグラフィに従って目標を達成することができます。

このようなモデルでの高レベルな作業単位は、通常「プロセス」と呼ばれ、基盤となる操作の動的な性質を強調しています。このようなシステムを通した作業の流れのため、多くの場合、モデルは「ワークフロー」をカプセル化すると言われます。

自動化されていないワークフローは次のようになります。

図 1-1



この仮のフローモデルでは、(会社の内外の) さまざまな関係者によってさまざまなタスクが明確に定義されたシーケンスで達成されます。プロセス全体をトリガする入力は電話をかけることで、出力は契約書に署名することです。プロセスには、定義された役割と責任を持つ特定可能なプレーヤがいます。各参加者がその職務を行うと、目標が達成されます。

「自動化された」ビジネスプロセスは、(独自のルールおよび責任を持ったそれぞれの) 企業のアプリケーションの観点から同じ操作をモデル化しようとします。高い柔軟性を実現するため、アプリケーションは Web サービスとして実装される場合があります (必須ではありません)。ユーザ入力に対応するため、一部のアプリケーションにはユーザ用のプレゼンテーション層があります。一部の状況では、モデル全体が人為的な処理を必要としないで、ソフトウェア内で実現される場合があります。

自動化プロセス管理の利点

BPM の最終的な目標は、複雑で長時間実行されるビジネスプロセスの「自動化」を可能にすることです。プロセスの自動化には、必要な人的資源を減らすだけではなく、次のような利点も含まれます。

- **スケーラブルなスループット** — 人員の数によってキャパシティが左右されることはありません。ビジータイム中の行き詰まりを回避できます。
- **整合性** — ビジネルルールは、プロセスの一部として形式化されると確実に順守されます。取引パートナーの合意 (TPA) の条項は強制的に適用され、会社のパフォーマンスが記録されます。
- **適合性** — プロセスは、予期しないボトルネックを自動的に検出して回避するように設計できます。
- **アップグレード機能** — プロセスは、ビジネス要件の変化に迅速に対応できるように調整できます。プロセスの個々のコンポーネントは、プロセス自身をすべて書き直すのではなく、修正または「変更」します。
- **強力な監査機能** — アクティビティ、プロセスインスタンス、およびビジネス単位を包括的に報告するときに、さまざまなソースからさまざまなデータセットを引き出す必要はありません。
- **顧客の要求に応答するよりよい機能** — プロセスは、顧客または取引パートナーによってリアルタイムで開始され、24 時間 365 日稼動方式で実行されます。ターンアラウンド時間は日数から時間に、または時間から分に短縮できます。
- **ビジネスプロセスを改善する新しい機会** — 強力な監査と BPM による報告機能によって、プロセスに関連した分析に基づいた新しいカテゴリを生成します。これにより、組織内で非能率なものと改良の機会が明らかになります。

プロセス設計とアプリケーション設計

プロセス設計とアプリケーション設計は異なる基準点から始まります。企業「アプリケーション」設計では、通常厳密に焦点を当てた問題に対するデータ中心の見方と、対応してスコープされたデータ指向の解決があります。一方、「プロセス」設計では、発明の特許取得、クレームの処理、および競売の実行などのビジネス目標を達成することを目的としています。プロセスへの入力は電話をかけることで、出力はトラックで 55 ガロン分のドラム缶である場合があります。プロセスの実行には、多くのタスクを完了することが必要になる場合があります。データ要求はタスクの連鎖によって大きく変化します。

プロセス設計は、単なる「データの出入り」ではなく、全体像について考える必要があります。これは、プロセスをモデル化しているときに使用する必要がある「アプリケーション」についてだけでなく、それらのアプリケーションを実行する必要がある時間の順序、プロセスを構築するアプリケーションによって作成された保証と責任、アプリケーションの可能な相互依存、アプリケーションが失敗していなくてもプロセスが途中で終了する場合のさまざまな可能性についても考える必要があります。

モジュール方式

モジュール方式の概念は、プロセスのモデル化において重要です。たとえば、次のようになります。

- 特定のビジネスプロセス(またはワークフローモデル)を構築するさまざまな構成アクティビティは、それ自体がプロセスとなり得ます。これは、「再帰的構成」と呼ばれることもあります。
- 特定のアクティビティは、(互いに無関係な) 複数のプロセスで役割を果たす場合があります。たとえば、プロセス A の「信用調査」アクティビティは、プロセス B およびプロセス C によっても使用される場合があります。これは「アクティビティの再使用」です。
- プロセスモデル自体に影響を与えずに、アクティビティの実装を変更できます。たとえば、会社の方針 (またはアルゴリズムの変更など) で変化を反映する新しいビジネス論理は、プロセスの「信用調査」アクティビティで実施されますが、プロセス自体を変更する必要はありません。

注記：カスタム構成の大きい作業単位を汎用的な小さい作業単位に分割する原理は、「ファクタリング」としてアプリケーション開発者によく知られています。ファクタリングの目的は高価なリソースの「再使用」を促進することです。

プロセスを構築するアクティビティにはパブリック用Webサービスが必要であったり、ローカルアプリケーションサーバで実行しているファイアウォール内のサービスに限定されたりする場合があります。外部の取引パートナーが参加者である場合または参加者でない場合や、プロセスがシステム内に多くの「コールバック」を持ち長時間実行されたり、比較的短期間で終了したりする場合 (つまり、ストレートスルー処理) があります。

単純なストレートスルー処理の例

自動化されたビジネスプロセスの例は、次の図のとおりです。

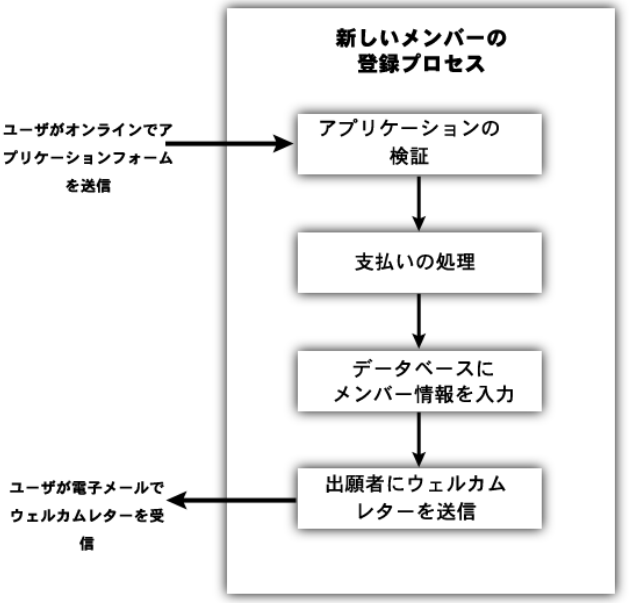


図 1-2

このシナリオでは、メンバー組織は、オンラインでメンバーの申し込みを受け取ります。申込者は、組織の Web サイトに HTML 形式のデータを送信することで、次の4つのアクティビティから構成される自動化されたプロセスをトリガします。

22 最初のアクティビティは、申し込みが完全であるかチェックし、おそらくデータベースを検索して申込者がすでにメンバーであるかどうかを確認します。

23 2 番目のアクティビティでは、ユーザの電子支払情報を処理します。

24 一度支払いを受け取ったら、新しいメンバーに関する情報がメインメンバーデータベースに入力されます。

25 最後のアクティビティは、パーソナライズされた「ようこそ」メッセージを電子メールで新しいメンバーに送信します。

この明らかに簡単な例では、新しいメンバーの登録プロセスの 4 つのコンポーネントアクティビティのいずれかが、それ自身の権利で自動化されたプロセスを表す場合があります。アクティビティ(Process Payment) のいずれかは、ビジネスパートナーによって提供される Web サービスに連携して動作する場合があります。他のアクティビティは、アプリケーションサーバのローカルアクティビティになります。

プロセス管理と新しい技術

ソフトウェアによって互いにリンクされたタスクに関して高レベルなビジネス機能をモデル化することは、特に Web サービスモデル、および一般的な分散型コンピュータ環境で動作する強力な例となります。XML、SOAP、WSDL、UDDI のような技術の出現によって、強力かつ強固で、洗練されたビジネスアプリケーションを設計して配備できるようになり、このビジネスアプリケーションは、それぞれの単位における実装の詳細に関係なく、「相互接続」された作業のさらに小さいタスク指向単位に調整された方法に従います (実装からのインタフェースの分離は、Web サービスアーキテクチャの主要な機能です)。

eXtend Composer Process Manager は、現在次のような最も重要な企業コンピュータ技術を多く活用しています。

- **XML** (eXtensible Markup Language)。データの移植。
- **SOAP** (Simple Object Access Protocol)。プラットフォームに依存しないペイロードの処理と、リモートプロシージャの呼び出し。
- **WSDL** (Web Services Definition Language)。サービスへのパブリックインタフェースの記述。
- **J2EE** (Java 2 Enterprise Edition)。相互運用、セキュリティ、スケーラビリティ、およびプラットフォームの非依存に対する標準。

また、Composer の Process Manager ランタイムエンジンは、指定された Web Services Flow Language (WSFL) 標準の主要な機能を利用します。

プロセスとサービスの違い

プロセスは、関係するアクティビティ間でルール型のデータのフローによって特徴付けられている、動的でステートフルなシステムです。入出力の観点から、プロセスは入力データを受け取り、データの変換または追加、あるいはその両方を行い、他のサービスと同じように出力データを作成します。また実際には、プロセスが (WSDL によって記述された)Web サービスとして公開された場合は、他の Web サービスのようになります。

プロセスと従来の Web サービスと異なる点は、プロセスでは特定のビジネス機能を達成する比較的大きな作業単位間の制御フローとデータを結び付けて組織化します。この場合、プロセスは他のサービスの相互動作を命令する「メタサービス」です (メタサービスに組織にとっての外部サービスが含まれる場合もあります)。

従来の Composer サービスと「プロセス」の重要な相違点については、次の表で簡単に説明します (詳細については、次の節を参照してください)。

表 1-1:

従来のサービス	プロセス
短期間	長時間実行
パフォーマンスが重要	通常、迅速な実行は重要ではない
実行は、起動しているサーバに依存	外部サービスに依存するプロセスでは、サーバのダウン中も実行の継続が可能
論理のシリアル実行 非同期プロセスにはあまり依存しない	アクティビティの非同期処理および並列実行が普及
予期しないデータの上書きはまれ	複数のアクティビティ出力は、同じターゲットメッセージ (またはメッセージ部分) にマップ可能。このため、上書きは潜在的なであり、「誰がいつどこで書き込む」という問題を扱うポリシーの明確な定義が必要
例外として制御フローの停止を処理	フローで閉塞ポイントが「回避される」場合あり。それ以外の場合には、タイムアウト / 再試行ポリシーが開始
データフローおよび制御フローが密接に結合	データと制御の結合は密接ではない
「スリープ」しないメカニズム	長時間実行されるプロセスは、アイドル時間中にスリープになる場合あり。プロセスのライフタイムでは、スリープと起動が繰り返される
簡単なテスト要件	制御フローパスは、テストするには多すぎ、ビジネスパートナーの詳細な調整が必要
管理の重点がパフォーマンスの調整および設定問題に置かれている	管理の重点が、ライフサイクルイベント、状態の監視に置かれている

大きな作業単位と小さな作業単位

「プロセス」での作業単位は、比較的大きくなります (アプリケーションまたはサービス全体が含まれます)。同様に、データに対する操作は、文書全体または文書集約のレベルで発生する (詳細ではなく) 粗く処理される傾向があります。詳細なデータ操作は、プロセスを構築するアクティビティの内部に発生します。

長時間実行とストレートスルー

(従来のアプリケーションまたはサービスに対して)「プロセス」で主に際立っている特徴は、通常は「長時間実行される」ことです。これは、プロセスがパートナーとの連携、定期的なバッチ処理、人為的な処理などに依存するため、実行に何時間、何日間、何週間もかかる場合があるからです。たとえば、契約者から見積りを入手するプロセスは、現実では、完了するまでに何週間もかかる場合があります。逆に、信用調査アプリケーションは、すぐに実行されることが必要とされています。信用調査タスクは、別々のスタンドアロンアプリケーションとして実装することをお勧めします。呼び出し元が応答を待っている間、ストレートスルー形式で情報を処理します。反対に、それぞれに独自の内部手順と制約を持つ多数の見積り発行者が含まれる RFP プロセスは、大規模で長時間実行されるプロセスを構成します。これにより、一体型で内蔵式の Web サービスとして確実に実装させることが難しかったり不可能だったりする場合があります。

待ち状態と持続性

ステータス情報の持続性は、回復面からだけではなくリソースを効果的に使用するためにも、自動化されたプロセスには必要です。長時間実行されるプロセスは、管理上必要なものであろうとハードウェアのダウンタイムであろうと (設定されていてもいなくても)、サービスの停止および再開を処理できるようにする必要があります。

Composer の Process Manager は、プロセスインスタンス情報をデータベースに格納するまで持続します。そのため、必要に応じてプロセスをスリープにして、(終了したアクティビティからのデータが到着したなどの) 適切なイベントに応答するために再度起動させます。このようにして、有益な RAM リソースおよび CPU リソースを長い待ち状態から解放します。

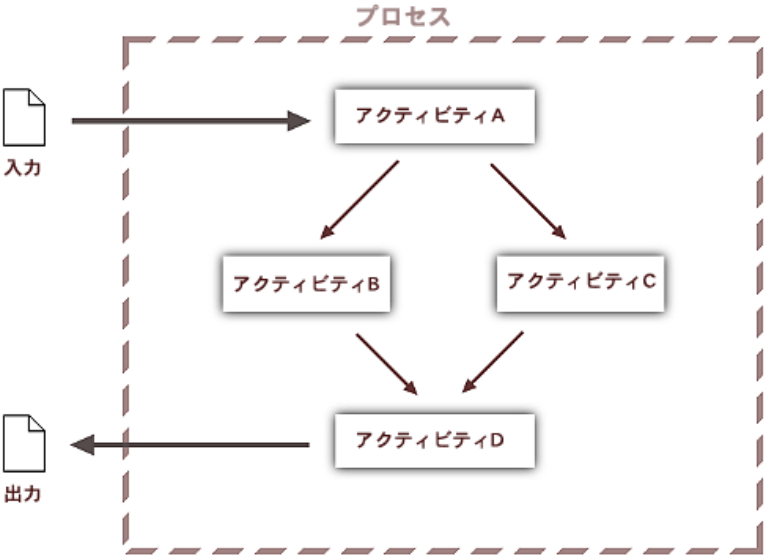
注記：ステータス情報は、(プロセスがスリープ状態になったときだけでなく) プロセス実行の「すべての」段階で持続されます。そのため、サーバが再起動しても混乱は起こりません。

並列実行

一般的に、プロセスは「ストレートスルー」処理を 1 つ以上呼び出します。たとえば、プロセスのそれぞれのタスクが完了まで 3 日間かかる場合、ストレートスルー実行チェーンは、プロセスが実行するには 9 日間必要とすることを意味します。これは、タスクが互いに直接には依存しない場合、非能率的になります。したがって、分割およびマージ (並列実行および再同期化) は、プロセス制御フローの共通の機能です。

これらの概念を理解するために例をあげます。次の図は、タスクの並列実行に依存するプロセスを示しています。

図 1-3



この例では、着信要求 (SOAP 要求、HTTP POST などを介して受け取った form データ) は、要求を処理するためのプロセスインスタンスをトリガします。アクティビティ A は初期プロセスを実行し、追加処理を行う 2 つ以上のアクティビティ (アクティビティ B およびアクティビティ C) を呼び出します。アクティビティ B およびアクティビティ C の出力は、アクティビティ D への入力を形成します。最終的に、アクティビティ D は、リクエスタに出力を送信します。

この図では、広範囲のシナリオを扱うことができます。たとえば、次の場合です。

- 要求がファイアウォールの内外から送信される場合。プロセスを「非同期的」に呼び出したり、プロセスが戻るまで待機 (同期的実行) したりすることができます。

- プロセスが、元のリクエスタに出力を送るように設計されている場合。実際には、別の場所に出力を命令する場合があります。
- アクティビティ A が、アクティビティ B またはアクティビティ C、およびその両方を非同期的または同期的に呼び出す場合。
- アクティビティ B が指定されたタイムアウト時間内に応答しなかった場合には、再試行が発生するようにプロセスを設計する場合。
- アクティビティ B およびアクティビティ C が、リモート環境にあるビジネスパートナーによって操作される Web サービスの場合。
- アクティビティ D が、アクティビティ B またはアクティビティ C が (どちらかが最初に) 終了すると実行するように設計されたり、アクティビティ B およびアクティビティ C の両方がデータを配信するまで、アクティビティ D を実行しないように要求されたりする場合。後の例では、アクティビティ B またはアクティビティ C のどちらのデータを選択するかを要求される場合があります。
- 図に示された4つのアクティビティのいずれかは、それぞれの権利で処理されます。つまり、このアクティビティは、Web サービスまたは Composer コンポーネント、またはこのいずれかの組み合わせになります。

プロセス管理の用語および概念

Composer の Process Manager の生産性を向上するためには、主要な用語および概念について理解しておく必要があります。この節では、Process Manager の操作時に最も理解しておく必要のある用語と概念について説明します。

後で説明するプロセスを自動化する慣用法のほとんどが、Composer の Process Manager によって実装されたように、Web Services Flow Language (WSFL) から直接読み込みます。主要な用語の詳細な説明については、以下の WSFL 仕様を確認してください。

http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf

Composer でのプロセスの自動化に使用される基本的な概念の多くは、WSDL (Web Services Description Language) に起因しています。WSDL の詳細な説明については、次を参照してください。

http://www.w3.org/TR/wsdl

注記： WSDL と Web サービスについてさらに理解すると、Composer でのビジネスプロセスの自動化について理解しやすくなります。

アクティビティ、メッセージ、およびリンク

Composer は、アクティビティおよびリンクの観点からプロセスを実装します。「アクティビティ」はプロセスの段階を実行する作業単位です。これらは Web サービス、アプリケーション、またはプロセスなどです。「リンク」は、アクティビティ間で実現可能な制御フローパスを確立します。データは、あるアクティビティから別のアクティビティに (全体または一部が) 渡される「メッセージ」によって、プロセスモデルを通じて移動します。

アクティビティ

アクティビティは、自動化プロセス内で作業の基本単位を表します。Process Manager プロセスモデルでは、アクティビティは次のとおりです。

- Composer コンポーネント
- Web サービス
- 別のプロセス

注記： Web サービスの概念では、実装に制限はありません。Web サービスは、WSDL で記述されるインタフェースを持つ限り、あらゆるプラットフォームで任意の言語で実装されます。

この場合、プロセスモデルで使用予定のコンポーネントおよびサービスは、すでにサーバに配備されています (または、すでに Web のどこかに存在しています)。つまり、プロセスは、単純に前から存在するサービスを「相互接続」する場合があります。それ以外の場合は、プロセスの要件を満たすために、最初からコンポーネントまたはサービスを開発します (すべてのアクティビティが完全に実装されるまでは、プロセスのテストや「配備」を行うことはできません)。

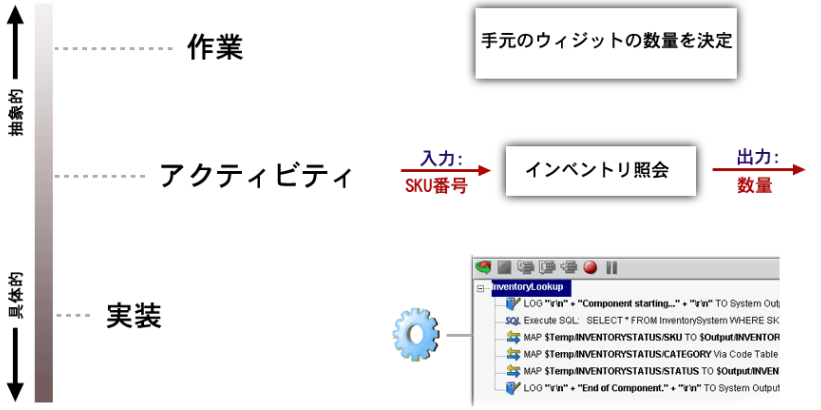
開始アクティビティおよび終了アクティビティ

プロセスに対するトリガは、1 つ以上の「開始アクティビティ」に (メッセージとして) マップできるデータを提供します。プロセスモデルの最後のアクティビティは、終了「アクティビティ」と呼ばれています (ワークフローに 1 つ以上の開始アクティビティと終了アクティビィが存在します)。開始アクティビティと終了アクティビティは他のアクティビティと類似していますが、開始アクティビティには着信制御リンク、終了アクティビティには送信制御リンクがありません。

開始アクティビティでも、送信制御リンクを持たない場合があります。たとえば、プロセスモデルの起動アクティビティの 1 つが、(Composer の JMS Connect を使用して作成された) JMS メッセージングコンポーネントの場合です。このコンポーネントは、通知をさまざまなキューに非同期的に送信します。ダウンストリームアクティビティが、メッセージングアクティビティからのデータに依存しない場合、メッセージングアクティビティは開始アクティビティと終了アクティビティの両方になります。

タスク、アクティビティ、および実装

Composer でのプロセスのモデル化では、タスク、アクティビティ、および実装で微小ながらも重要な違いがあります。



最も抽象的なレベルでは、「タスク」はビジネス機能です (実際には、ビジネス機能は「支払い履歴の取得」、「注文書の発行」、「入荷待ち情報の決定」などと呼ばれています)。ビジネスタスクには、人によって実行されるものや、自動化されるものがあります。20 世紀の企業コンピュータでは、主に、ビジネスタスクの自動化または半自動化を行う方法を見つけることに関心がありました。

アクティビティはタスクを「実行」します。「アクティビティ」という単語は、ソフトウェアで実現されるタスクを意味していますが、「実際の実装については何も意味していません」。アクティビティで必要な入力と出力について事前に分かっている場合がありますが、これは単にアクティビティの「インタフェース」の内容が分かっていることを意味します。基盤となる実装の詳細が知られていることは意味していません。

アクティビティの「実装」には、いくつかの形式があります。これは Web サービスでは重要なものです。共有化は、実装ではなくてインタフェースに依存しています。Composer の Process Manager は、サービスがプロセス内でアクティビティとして使用されるために、WSDL について記述されたインタフェースを持つことを許可する Web サービスモデルを活用しています。どのようにアクティビティを実装するかについては制限がありません。アクティビティには、ビジネスパートナーの Web サーバで実行する C++ プログラム、ユーザのサーバで実行するカスタム EJB、または Composer JDBC コンポーネントなどがあります。

メッセージ

次に、アクティビティは、論理パートを構成する「メッセージ」上で動作します。このパートは、スキーマで定義された「名前」と「タイプ」の特性を持ちます (このタイプは、XSD データの省略タイプ、またはカスタムスキーマで定義された複雑なタイプです)。メッセージパートは、アクティビティの入力データおよび出力データに対応していると考えることができます。

サービスの入力および出力を XML ドキュメントまたは DOM として考慮することを習慣としている場合、メッセージ例では、参加者間の共有化についての概念を含むために、入力ドキュメント / 出力ドキュメントの慣用法を単純に拡張します。メッセージは「インタフェース」(特殊なデータで事前に定義された操作) を意味します。大切なことは、決してデータの静的コンテナではないメッセージが、メッセージとメッセージパートに「名前が付けられている」ことから生じた暗黙の運用上の意味を持って実行されるということです。したがって、インタフェースに関連付けられた「操作」で指定されます。

注記：「メッセージのようなデータ」の概念は WSDL の基本です。メッセージ、メッセージパート、タイプなどを含む Web Service Description Language 概念に精通していない場合は、WSDL 仕様をご覧ください。

メッセージ例は強力です。なぜなら、実際には、アプリケーションがインタフェースの内容を指定する (そして相互運用を可能にする) には十分具体的ですが、参加者が、それぞれの実装の詳細については何かを知る必要がなくなるほど抽象的ではないからです。これは、異なるプログラマによって、異なる時間と場所で、完全に互いと無関係にアプリケーションを開発でき、必要が生じれば相互に運用することを意味しています。

メッセージ例の開発によって、Composer の Process Manager は、アクティビティの相互接続で高い柔軟性を実現するために、インタフェースおよび実装を分離させておくことができました。

メッセージパート

Composer の Process Manager は、アクティビティが WSDL に関連付けられている場合に、WSDL について記述されたサービスのメッセージ部分の意味を「認識」します。

アクティビティが通常の Composer コンポーネントを構成するスキーマ内で、メッセージパートを明示的に定義する必要はありません。Component の入力 DOM および出力 DOM は、(デフォルトでは) メッセージとして扱われます。

リンク

リンクはプロセスモデル内で許可される制御フローパスを定義します。アクティビティが「開始アクティビティ」でも「終了アクティビティ」でもない場合 (次を参照)、1 つ以上の着信リンクと、0 以上の送信リンクが必要です。

注記： リンクの単なる存在は、ランタイム時にリンクをたどることを意味するものではありません。ここでは、トランジション条件論理について決定します (次の説明を参照してください)。

運用上の点では、アクティビティが終了すると、リンクが Process Manager ランタイムエンジンに「次にすること」を通知します。

リンクは、アクティビティを表すボックスまたはアイコンを接続するラインまたは矢印として作成できるので、設計時環境でアクティビティ間の制御フローを視覚化するための例を提供します。

順序設定、時間調節、およびプロセスレベル論理

プロセスは、単なるリンクとアクティビティの集合ではありません。プロセスモデル内のリンクは、幹線道路網の道のようなものです。これらは、作成可能なパスをすべて定義しますが、「実際」には「どのように」作成されるかについては定義しません。実際には、道路網での交通の流れのパターンは、交通規則、高架道路での車両接触限界などに影響されます。同様に、プロセスモデルでの実行フローは、さまざな設計時規則とランタイム時に適用される規則に依存します。

ランタイムフローパターンに影響を与える要因は次のとおりです。

- **リンク作成論理** —「リンクトランジション条件」のレベルに適用されるルール (次を参照)。
- **同期論理** — 複数の着信リンクを持つアクティビティのトリガを管理するルール。この場合、「結合」アクティビティは、結合を評価する前に、すべての入力アクティビティの実行を終了させることができます。それ以外では、ターゲットアクティビティは、(どんな着信アクティビティからでも) 最初の入力が到着するとすぐに実行を開始するように設計されている場合があります。
- **再試行およびタイムアウトポリシー** — いくつかのビジネス操作は、詳細な試行 / タイムアウト / 再試行要件に従うことが要求されます。たとえば、次のようになります。「このベンダーを照会して、承認のために最大 2 時間お待ちください。照会を、合計で 3 回まで繰り返します。」すべてのアクティビティはタイムアウト / 再試行ポリシーを持つことができます (「必須」ではありません)。

これらのフローとその他のフローの制御要因については、次の節で説明しています。

制御フロー論理

制御フローは、プロセス内の 3 つの需要ポイント (リンクトランジション条件、アクティビティ終了条件、および結合条件) で適用できる論理によって実現されます。

リンクトランジション条件

ランタイム時に指定されたリンクを作成するかどうかを判断する論理を「トランジション条件」と呼びます。通常、トランジション論理は、リンクに着信するデータの検査に基づいて、「ブール」値を返します。トランジション条件が true と評価された場合はリンクが作成され、そうでない場合は作成されません。

リンクは、トランジション論理を持つことを「要求」されません。デフォルトでは、リンクはストレートスルーで作成されます。

前の例 (図 1-3 を参照) では、アクティビティ間の矢印が「リンク」を構成します。それぞれのリンクはトランジション条件 (XPath によって示される) に関連付けられています。アクティビティ A のデータは、受信したデータのタイプ、またはデータに含まれていた特定の値に従って、アクティビティ B をトリガしたり、しなかったりします。

アクティビティ終了条件

すべてのアクティビティは「終了条件」を持つことができます。終了条件はブール値を生成する論理式です。その値は、関連するアクティビティが正常に終了したかどうかを示します。終了条件がランタイム時に *true* を返すと、着信リンクが発生します。false の場合、元のアクティビティは再実行されますが、送信リンクは発生しません (アクティビティに送信リンクがない場合、終了条件はありません)。

結合条件

同じターゲットアクティビティで 2 つ以上のリンクが生じる場合、実行を続けるために論理を適用する必要があります。この論理は、マップポリシーに関連した「結合条件」の形式を使用します。

結合条件は、着信リンクの真の値を検討して true または false を返す式です (真の値は、リンク条件の最終値です)。

注記： 終了条件およびリンク条件が XPath に表される場合、AND、OR、NOT および (グループ化のための) 括弧を使用して、結合条件を簡単な形式で指定します。

アクティビティが結合条件を持つ場合、処理方法を決定するために結合論理を確認します。以下のシナリオを確認してください。

- 3 つの仕入先から見積りが請求されました。会社の方針は、残りのプロセスに取り組む前に、必ず 3 つの仕入先の「すべて」から見積りを受け取ることを要求しています。結合条件はリンク値間で論理 AND を指定します。このパターンは、AND 結合と呼ばれています。

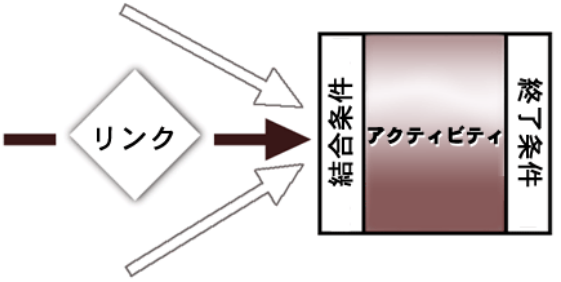
- 会社は、各従業員が 2 つの退職プランから選択できるようにしています。それぞれのプランは、従業員に対して適切な事務処理を作成するアクティビティに関連付けられています。事務処理には、結合アクティビティに渡されるデータがあります。結合条件は次のように指定します。

  ```
  (Plan1 AND NOT Plan2) OR (NOT Plan1 AND Plan2)
  ```

  これは、排他的 OR(つまり XOR) 結合と呼ばれています。

- アクティビティは複数のリンクのいずれかから入力を受け取ります。いずれかまたはすべての入力を使用できます。このパターンは OR 結合です。

さまざまなフロー論理間の関係を視覚化する最も簡単な方法は、結合条件をアクティビティの「入力側」の論理とし、終了条件を「出力側」の論理として考えることです。



結合条件は、OR/XOR/AND タイプの「同期論理」の実装を主な目的にしています。1 つ以上のアクティビティからのデータは検査され、次のアクティビティが実行するかどうかを決定する判断材料として使用されます。

終了条件は、(一度実行を終了した) 関連付けられたアクティビティが次のアクティビティによって使用されることに適しているデータを作成したかどうかを判断するための厳密なメカニズムです。終了条件が *true* を返す場合、アクティビティのデータ出力が任意の送信リンクへの継続として最小基準を満たすことを意味しています。終了条件が満たされた場合、「すべて」の送信リンクをたどることになります。終了条件が満たされない場合は、送信リンクをたどりません。

「トランジション条件」は、ソースアクティビティからの出力を使用して、次のアクティビティが入力されるかどうかを判断します。リンクが他のリンクターゲットに関して「認識する」方法がないので、トランジション論理は、比較的単純になる傾向があります (多くの場合は *true* にデフォルト設定されます)。

注記： 従来の「分岐」は、リンクレベルで実装できます。次の章の「分岐論理」にある説明を参照してください)。

Deferred モードと Immediate モード

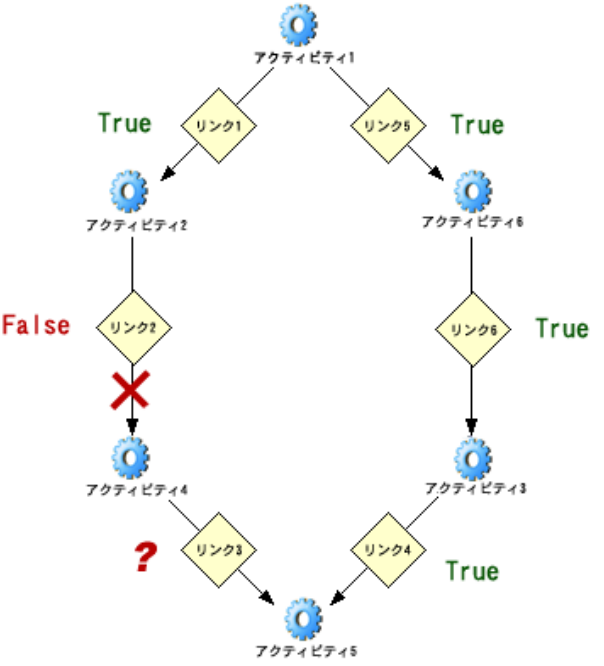
結合は完全に同期的 (たとえば、実行が終了した「すべて」のソースアクティビティに依存します)、または「非同期的」(任意の入力アクティビティからのデータが到着するとすぐに継続が許可されます) です。デフォルトでは、すべての結合は Deferred モードで発生します。結合条件を評価する前に、結合の入力アクティビティをすべて終了させる必要があるからです。このモードでは、結合条件は一度だけ評価されます。

すべてのソースアクティビティが完了する「前」に結合アクティビティを発生させる場合は、Immediate モードを使用します。このモードでは、結合条件は、ソースアクティビティが終了するたびに評価されます。結合への着信リンクが複数存在する場合、最初の「true」リンクが検出されるとすぐに結合条件が発生するように設定することができます。

Composer Process Manager によって、アクティビティごとに Deferred モードまたは Immediate モードを設定できます。

デッドリンクと同期障害

結合条件が着信リンクの真の値を待っていても、(アップストリームポイントでフローが停止したので) リンクの条件が返ってこない場合、結合はハングアップします。以下のシナリオを確認してください。



このフローグラフでは、アクティビティ 5 で結合が発生します。Deferred モードでは、結合条件は、リンク 3 とリンク 4 の true 条件が判断されるまで評価されません。ただし、アクティビティ 2 が正常に終了すると、リンク 2 のリンク条件が *false* に評価されると仮定します。この場合、アクティビティ 4 は決して発生しません。アクティビティ 4 が発生しないとリンク 3 が評価されません したがって、リンク 3 は「デッドリンク」になり、フローグラフのセグメントは「デッドパス」を作成します。最終的には、アクティビティ 5 での結合はハンブアップします。

この種の同期障害を避けるためには、条件式が false を返したときに Process Manager のランタイムエンジンがルックアヘッドを実行します。ルックアヘッドは次のように実行されます。

- falseリンク(またはfalse結合条件)で開始すると、「結合アクティビティ」と「終了アクティビティ」のどちらが先に発生しても、どちらかが到達するまで、エンジンはすべてのダウンストリームリンクを通過します。ここで、通過を停止します。
- 通過するパスのそれぞれのリンクは false に設定されています。
- 通過するパスが「結合」で終了すると、(他のリンクの true 値と結合モードに基づいて) 結合条件が評価されるかどうかをエンジンは決定します。その場合、*false* 値を持つ着信リンク (デッドリンク) がただちに評価されます。結合条件が *true* の場合は、結合は「維持されている」とみなされ、これ以上はデッドリンクの作成が発生しません。結合が false の場合は、送信リンクがデッドの状態になります。ステータスを false に「設定」する必要があり、ルックアヘッドは、このポイントからダウンストリームを続ける必要があります。

「デッドパスの排除」手順は、false 条件がダウンストリームの結合をハングアップできないようにしています。必要に応じて、ランタイムエンジンによって自動的に実行されます。

マップポリシーとデータのマージ

複数のアクティビティが単一のアクティビティに出力を命令する場合、ソースアクティビティがターゲットアクティビティへの入力で、それぞれの他のデータを上書きする場合があります。「マップポリシー」はマッピングの順序を指定して、衝突を解決するためにたどっていくポリシーを上書きします。

選択可能なポリシーは、次の 3 つです。

- **First writer wins (FWW)** — これは、アクティビティの入力テンプレートにマップされる最初のデータが、結合アクティビティへの入力として使用されることを意味しています。後続のメッセージは上書きできません。
- **Last writer wins (LWW)** — 前のデータのマップ方法に関係なく、到着する最後のデータはマップされます。

- **マップ順序** — データのマップは、着信時間に関係なくユーザ指定の順番で発生します。

後の「プロセスの作成とテスト」という章の「マップポリシー」の説明を参照してください。

タイムアウトと再試行

(結合の一部であるかどうかに関わらず) アクティビティはタイムアウト / 再試行動作を明確に定義します。つまり、アクティビティが指定されたタイムアウト時間内に使用可能な出力を作成しない場合、指定した回数まで再試行を実行します。再試行は、ユーザが指定した間隔で繰り返されます。

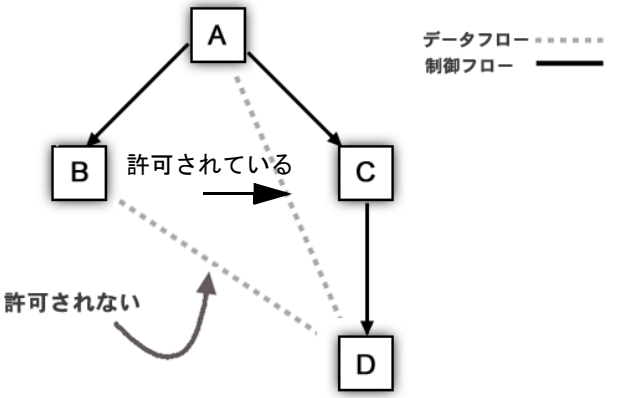
注記：タイムアウト / 再試行動作は、アクティビティ単位で使用できますが、全体ではオプションです。

タイムアウトおよび再試行は多くの標準のビジネス通信において重要な部分で、RosettaNet によって定義された Partner Interface Processes のように、業界標準で形式化される場合があります。Composer Process Manager によって、アクティビティごとにタイムアウトオプションおよび再試行オプションが設定されます。

データフローパターン

自動化されたビジネスプロセスでは、人が介入するプロセスのように、データフローが制御フローに結合されますが、常に密接ではありません。たとえば、いくつかのアクティビティは、正常な制御チェーンの外側からデータを要求します。実行する最後のアクティビティは、実行された最初のアクティビティからのデータを必要とする場合があります。このデータは多くの制御リンクに到達できません。他のアクティビティは、入力側のデータを要求しますが、「それ自身」の「出力データ」がありません (アクティビティの「出力」は、トラックに積まれた物理的な品物である場合があります)。実際には、データフローおよび制御フローが異なるパスを通る場合があります。

Composer の Process Manager によって、データおよび制御が独自のパスを追跡し、1 つの制限事項に順守することができます。アクティビティが、直接リンクしていないアクティビティからのデータを要求したとき、次の図に示すように、制御パスを通過して「アップストリームに動く」(ダウンストリームではない) 場合は、ソースアクティビティが到達可能でなければなりません。



前の例では、A から C、C から D へのパス (すべて一方向) 上に A があるので、データは A から C、C から D へのパスに流れて、D が A から直接データを取得することができます。ただし、D は、B からデータを取得することは許可されていません (B から A、A から C、C から D へのパスは、まず、最初の移動が B から A への「アップストリーム」、次に A から C、C から D へのへの「ダウンストリーム」になります)。

D が B からデータを取得するのは保証されていません。なぜなら、リンクトポロジは、D の前に B を完了することを保証していないからです。制御リンクによって作成された保証の 1 つに、ダウンストリームの終了でアクティビティを実行する「前」に、リンクの「アップストリーム」の末端で任意のアクティビティを実行する必要があります。前の例では、B が実行するのに 3 日かかります。しかし、C は数秒で実行するかもしれません。D が B からデータを取得する唯一の安全な方法は、2 つのアクティビティの間に制御リンクを作成することです。つまり、D を「結合」アクティビティにします。

データの伝達の詳細情報が多すぎることがないように、通過時に、Process Designer がアイコンと矢印で示す制御フロールールとは異なる独自のルールセットに、アクティビティ間のデータ転送 (またはマップ) が従う必要があります (フローグラフは、データフローではなく、「制御」フローを示す Designer に作成されます)。データルーティングは制御フローよりも簡単に理解できますが、独自の方法がいくつか適用されます。詳細については次の章を参照してください。

ライフサイクルイベント

プロセスはライフサイクルイベントのいずれかに応答できます。つまりプロセスの実行「全体」に影響すると言われています。

- **Spawn** — Spawn イベントは、「非同期」モードでプロセスを呼び出したり、インスタンス化したりします。呼び出しを行うエージェントは、プロセスが戻ってくるまで待機 (ブロック) しません。したがって、エージェントは、プロセスを起動した後、固有のプロセスインスタンスが正常に起動したことを示すインスタンスのデータセット (「受信」) を呼び出し元に返すとすぐに、元の場所に戻ります。その後、必要に応じて、呼び出し元はこのデータを使用して、ステータスの更新などにおいてプロセスを照会します。
- **Call** — エンティティがプロセスを「呼び出す」と、呼出元は、リアルタイムでプロセスからデータを受け取ります (つまり同期的)。Call はプロセスを呼び出し、プロセスは実行が完了すると戻ります。プロセスの出力は呼び出し元に直接戻ります。
- **Suspend** — Suspend が進行中のプロセスを「停止」しますが、破壊するわけではありません。制御フローは一時的に中断されます。ライフサイクルイベントのこのタイプは、通常、管理上のコンテキストで発生します。
- **Resume** — 「中断」の逆。前に停止したプロセスが動作を再開します。これは、再度、主に管理上重要なイベントになります。
- **Inquire** — ステータス情報に対してプロセスを照会します。
- **Terminate** — プロセスインスタンスを中止します。

Process Manager アーキテクチャ層

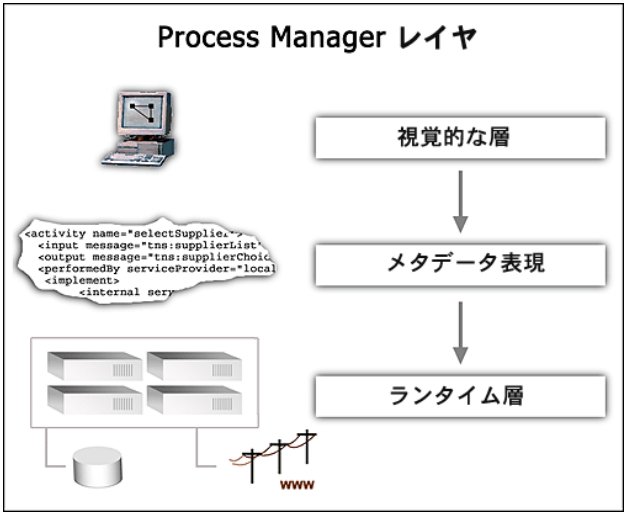
eXtend Composer Process Manager を使用して作成するプロセスは、3 つの異なるレベルで実装されて管理されます。

デザインレベル : 設計層には、プロセスの視覚的表現またはユーザインタフェース表現を管理する責任があります。この層では視覚的な設計ツールを使用して、アクティビティを定義し、リンクによってアクティビティを接続し、メッセージマップを決定し、論理をトランジションポイント (リンク、結合、および終了条件) に割り当てることができます。ここで作成するプロセスモデルは「メタデータによるプロセスの表現」(次を参照) の基礎になり、ランタイムエンジンはプロセスインスタンスを作成および管理するために使用します。

メタデータレベル : 非視覚的レベルでは、プロセスモデルは、アクティビティ、リンク、入力メッセージ、出力メッセージなどの XML の記述として、「メタデータ形式」で格納されます。このメタデータの記述は、ランタイム環境でプロセスをインスタンス化する必要のある、すべての情報を提供します。このレベルでは、プレゼンテーション関連の情報は必要ありません。

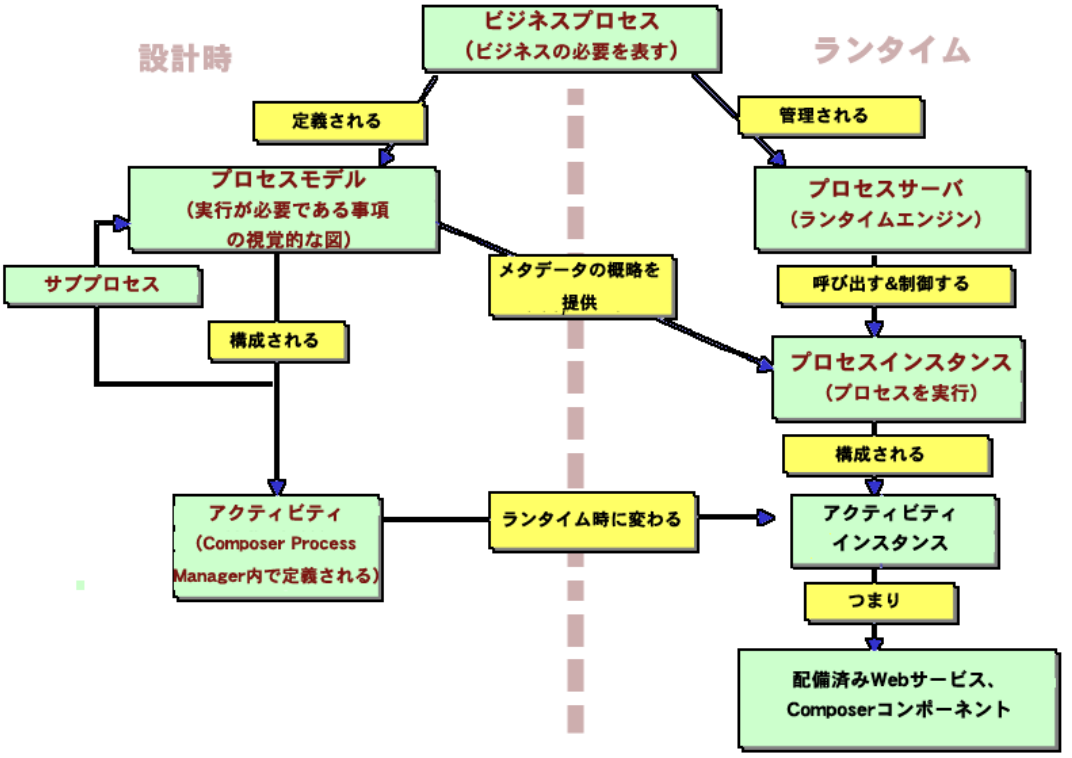
ランタイムレベル : ランタイム層は、プロセスインスタンスの実行を管理します。ランタイム層は、ステータス情報を管理し、ライフサイクルイベントを管理し、タイムアウト / 再試行動作を実装し、アクティビティ間のデータフローと制御フローを実現し、データベースでのインスタンスデータの持続性を含む日常業務タスクを実行します。プロセスへの管理アクセスは、この層を経由して発生します。

これらの層の関係は、次の図のとおりです。



層は、互いにトップダウンで作動します (ボトムアップでは作動しません)。たとえば、視覚的な層または設計層は、プロセスのメタデータによる表現を作成しますが、メタデータ層は、特定のプレゼンテーションを記述しません。同様に、メタデータ層は、ランタイム層に対してルールを設定しますが、ランタイムエンジンはメタデータを修正しません。メタデータはプロセスの概略を作成します。

層 (および構成要素) の設計時およびランタイムの責任については、次のグラフィックで詳しく説明しています。



プロセスは、最終的に、具体的なランタイム実装に対して、すでに配備されたサービスおよびコンポーネントの上に配備され、連携して動作します (これらのいくつかはリモート環境の Web サービスです)。あらかじめ作成されているサービスを使用するプロセスを配備すると、管理フレームワークを配備させたことができるので、管理フレームワークの唯一のジョブは特別なルールに従って既存のアプリケーションを呼び出します。

既存のアプリケーションは任意のプロセスで役割を果たす必要があります。たとえば、アクティビティ X、Y、および Z を使用するプロセス A と、アクティビティ X、Z、および Q を使用するプロセス B があると仮定します。アクティビティ Z をパブリック URL を持つ Web サービスにすると、実際には他の会社で使用されるリモートプロセスの役割を果たします。

Web サービスのプラグアンドプレイの性質は、プロセス管理に対して高い能力と柔軟性をもたらし、Composer の Process Manager を効果的に使用する方法について理解するために重要です。

Process Manager FAQ

これまでに、プロセスを Composer の Process Manager を使用して修正する方法や、プロセス設計時で何が制限されているかについて、多くの疑問が生じていることでしょう。この章より後の章で、その質問の多くの答えが明らかになりますが、ここでも、FAQ に対して迅速に答えています。

Process Manager 内で Composer コンポーネントを作成または編集することはできますか？

できます。つまり、Process Manager 設計時エディタは、完全に Composer 内で動作します。複数のコンポーネント、サービス、およびプロセスを同時に開いて、ウィンドウ間を自由に切り替えることができます。実際、アニメーションモードでは、プロセスアクティビティのステップオーバとステップインを実行でき、指定されたアクティビティの基本的な実装が Composer に作成された場合には、アクティビティの実装のアクションモデルを「ステップイン」して、プロセス自身に戻る前にステップスルーします。同じ環境で、アクションモデルとプロセスモデルをすべてデバッグする機能があります。

アクティビティを実装していなくても、プロセスの設計を開始することはできますか？

できます。プレースフォルダアクティビティアイコンをプロセスキャンバスの上に置いて、名前を付け、アイコン間にリンクを作成することができます。もちろん、便利なメッセージパートマップを実行するには、各アクティビティに対して実際のコンポーネントまたはサービスを指定する必要がありますが、その場合でもコンポーネントを完全に作成する必要はありません。アクティビティが Web サービスに設定された場合、WSDL がサービスのために存在する限り、サービスを構築する前でも、プロセスモデルにマップを指定できます。

テスト目的で、設計時環境でプロセスを実行することはできますか？

できます。アニメーションモードで Composer コンポーネントを実行するのと同じ方法で、アニメーションモードで Composer 内のプロセスを実行できます。*これは、ワークフローとプロセス自動化ツール間の独自の機能です。*他のワークフロー製品で、「骨組み」プロセスをすぐに作成することができる場合がありますが、通常、低レベルのプログラミングを行う設計時環境内でしか、アクティビティ層の機能を実装することはできません。また、アクティビティ層を実装したときは、元の設計環境でテストすることはできません。Composer Process Manager を使用すると、「プロセス」だけでなくアクティビティも、設計時環境内で設計、テスト、およびデバッグをすることができ、開発サイクルを短くすることができます。

インポート WSFL フローモデルを他の環境で作成することはできますか？

できません。Composer の Process Managert は他のソースからワークフローモデルをインポートするように設計されていません。この時点では、WSFL は未発達なので、ユーザが要求している機能をすべて提供することはできません。したがって、2 つのベンダーが互換性を持つ方法で 2 つの WSFL ソリューションを実装することはほとんどありません。さらに、Composer Process Manager のプレゼンテーション (グラフィック) 層は、メタデータ層からは直接実行されません。つまり、プロセスの特定のグラフィク表示は WSFL メタデータモデルにはありません。また、Process Designer には、グラフを表示する方法についての「先天的」な概念はありません。

XML エディタでプロセスモデルのメタデータを編集できますか？

Composer の Process Manager によって作成されたプロセスモデルのメタデータの記述を、手動では編集しないでください。メタデータを直接編集しないことをお勧めします。

Process Manager は並行分割、排他的選択、および他の分岐作成をサポートしていますか？

サポートしています。デザイナによって、ブール値を個々のリンクだけでなく、(結合条件および終了条件で) アクティビティのエントリと終了側に置くことができるので、WSFL は特別な構成要素を定義しないで、任意の複雑なフローパターンを設定することができます。つまり、(WSFL のリードに続く) Composer Process Manager は、特定の分岐または結合フロープリミティブを定義しません。ただし、適切なトランジション条件を使用すると、目的の分岐 / 結合動作を簡単に達成できます。

Process Manager はループをサポートしていますか？

しています。ただし、後方リンクは許可されていません。ダウンストリームアクティビティとソースアクティビティを接続するリンクは、「サイクリックグラフ」と呼ばれるものを作成します。これは、再入可能性の問題があるので、WSFL ではサポートされていません (これらの問題の詳細については、Process Manager で実際にサポートされているループの構成とともに、次の章で説明します)。

マニュアルルーティングとユーザエージェント機能に対して Process Manager を使用することはできますか？

人間用のアクティビティを持つキューベースワークフローは、Process Manager を使用して作成されます (「高度なヒント」の章を参照してください)。ワークアイテムを含むキューの概念、ワークアイテムの優先度、役割を持ったアドレス (個人)、タイムアウト、ロック、管理的な制御、およびキューの参照はすべて Process Manager によってサポートされています。また、これらの機能をサポートするさまざまなアクションは、Composer のすべてのコンポーネントタイプ (およびすべてのコンポーネントエディタ) で使用できます。

自動化プロセスはシステムに大量の要求をしますか？

要求しません。Composer の Process Server、プロセスを実行するシステムのロードおよびパフォーマンスの特徴は、プロセスを作成するアクティビティによって決定されます。Process Server の 1 つのインスタンスがたくさんの実行プロセスを制御しているので、Process Server「自体」に、非常に小さい処理オーバヘッドが発生します。また、通常、プロセスは長時間実行されるので、進行中のプロセスインスタンスの大部分が、ほとんどスリープ状態となります。これらの待機期間中、アクティビティは永続的な格納に存在するので、実際には CPU サイクルは必要ありません。

プロセスが実行しているときにサーバを起動および停止することができますか？

できます。なぜなら、プロセスのステータス情報は、それぞれのプロセスインスタンスが実施されている間は保存されます。また、一般的にプロセスは長時間実行されていますが、ほとんどは休止しています。プロセスインスタンスの実行の停止は、WSFL と Process Manager によってサポートされています。プロセスは、Process Server コンソールでいつでも停止できます。

すべてのアクティビティを Web サービスとして実装する必要がありますか？

いいえ。アクティビティは、Composer コンポーネントまたは Web サービスの形式です。

プロセスを Web サービスとして公開する必要がありますか？

いいえ。公開することもできますが、必須ではありません。

2 プロセスのモデリング準備

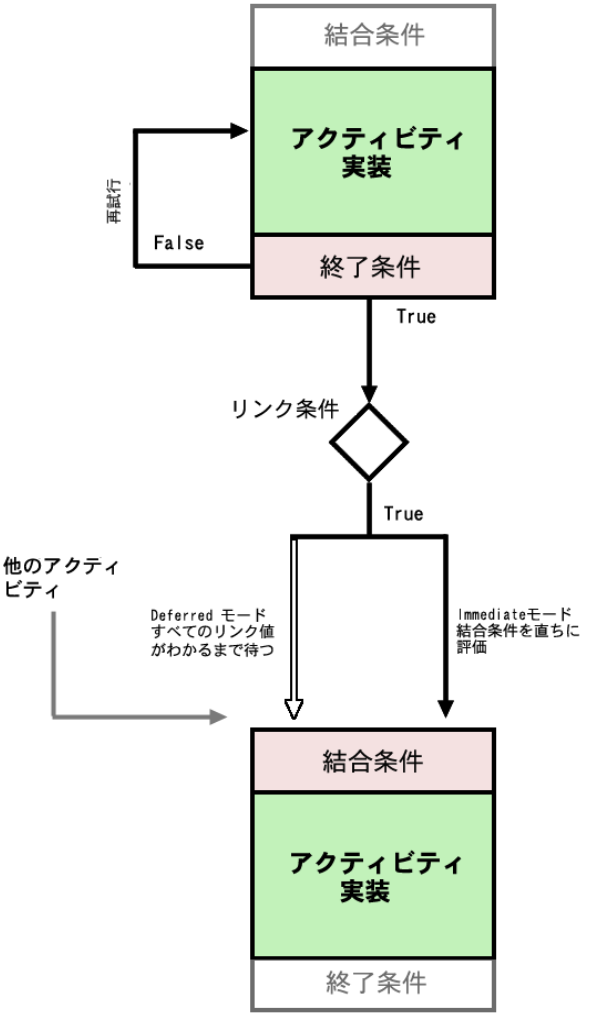
この章では、第 1 章の抽象的な概念をさらに明確にします。まず、ランタイムフローのしくみ (Composer Process Server によって実装されます) を検証し、次に、さまざまな使用ケースと設計パターンをProcess Designerでどのように実装できるかについて説明します。

Process Server 実行モデル

Process Server の基本的な実行アルゴリズムを理解することは、プロセスの設計方法を理解するために必要です。

Process Server (またはランタイムエンジン) では、次のようにしてプロセスインスタンスを実行します。

1. Spawn イベントからプロセスが非同期的に呼び出された場合、Process Server では、新しいプロセスをインスタンス化すると、ProcessID (「受信者反応」) を呼び出し元にただちに返します。そうでない場合は、Call によってプロセスが呼び出されると、呼び出し元ではそのプロセスが終了するまでブロックすることが想定されます。
2. Process Server では、「開始アクティビティ」を続行するプロセスモデルのアクティビティを判断します。
3. プロセス (1 つまたは複数の「メッセージパート」) に対する入力データが、開始アクティビティにマップされます。
4. 開始アクティビティが呼び出されます。
5. アクティビティ(開始アクティビティであるかどうかにかかわらず)が終了すると、Process Server では、アクティビティの「終了条件」を確認し、関連付けられている XPath 式を評価します。終了条件が *false* と評価された場合、アクティビティは再実行されます (前回と同じ入力を使用して)。タイムアウトまたは true という終了条件のいずれかの状態が発生するまで、実行が繰り返されます (次の図を参照)。



6 完了したアクティビティの終了条件が *true* の場合、Process Server では、アクティビティの発信側に接続されているコントロールリンク (存在する場合) を判断し、このようなコントロールリンクのトランジション条件を評価します。

7 データが次のアクティビティ (複数の場合もあります) にマップされます。

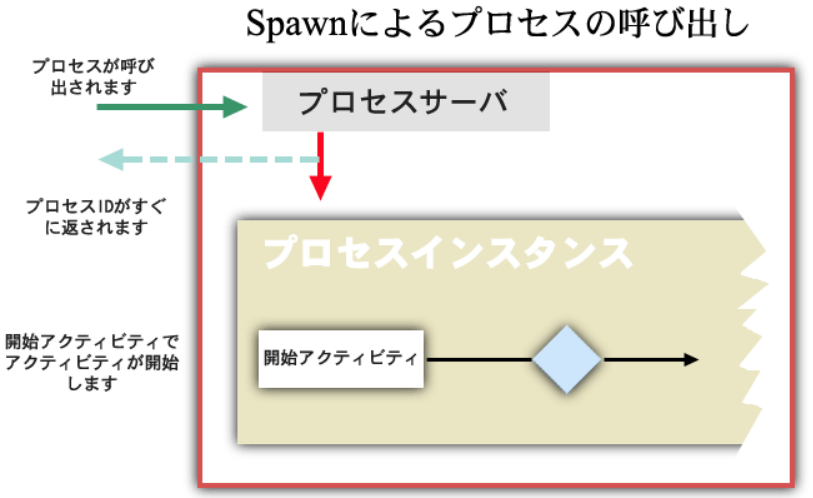
8 トランジション条件が *true* である各コントロールリンクに対して、Process Server では、リンクターゲットの結合条件を評価します。この評価は、結合が Deferred モード (デフォルト) になっている場合、すべてのリンク条件が評価された後で 1 回だけ行われます。結合モードが Immediate の場合、結合条件は複数回評価されます (リンクの真の値が計算されるたびに 1 回)。つまり、リンク条件が評価されると、値が false である場合でも、エンジンによってただちに結合条件が評価されます。

9 結合条件が true と評価された場合はターゲットアクティビティが起動され、そうでない場合は起動されません。

注記： 同期モード (Immediate/Deferred) にかかわらず、結合のターゲットは、結合条件がある時点で true になるまであるいは true にならない限り起動されません。

10 リンクのターゲットアクティビティで実行が完了すると、手順 5 からサイクルが再び開始します。実行は、何も行われなくなるまで続行します (つまり、終了アクティビティの終了条件すべての真の値がわかるまで)。

Spawn から呼び出されたプロセスインスタンスの一般的なプロセス起動のしくみは、次の図のとおりです (呼び出し元は、「実行後削除」するという方法でプロセスを呼び出すことを選択しています)。

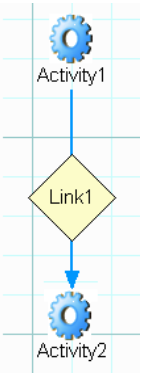


設計時ビュー

ランタイムエンジンでは、特定のプロセスモデルによって使用されるアクティビティ、これらのアクティビティがどのように互いにリンクされているか、これらのアクティビティ間にどのようなデータマッピングがあるかなどを把握している必要があります。これらの情報は、すべてプロセスグラフで設計時に指定されなければなりません。これを行うには、Process Designer を使用します。

Process Designer は、プロセスのグラフィック表示を作成し、プロセスのアクティビティ間のデータ関係を指定するための視覚的な編集環境です。これを実行できるようにするツールとして、マウス操作による「レイアウトツール」とテキストベースの「プロパティシート」(モードレスダイアログボックスのように機能します) の組み合わせがあります。Process Designer に対して固有なこれらの GUI 機能の他に、Composer のコンポーネントやサービスを作成する場合にも同じく使用する Composer の標準のメニューコマンド、ナビゲーションフレーム、マルチドキュメントコンテンツフレーム、および出力フレームがあります。つまり、Process Designer は、完全に Composer 内で動作します。

単純な 2 アクティビティのプロセスの Process Designer ビューは、次のとおりです。

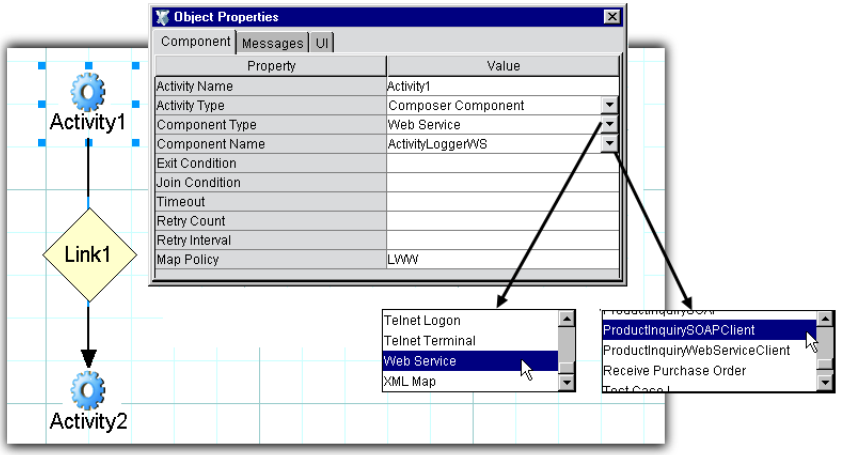


この場合、アクティビティアイコンは Composer コンポーネントを表しています。2 つのアクティビティはリンクによって接続されています。リンクアイコンがダイヤの形をしているということは、カスタムトランジション条件がリンクに対して指定されていることを意味します (カスタム条件のないリンクにはダイヤのアイコンは表示されず、単に「Link」という語が表示されます)。

この図における Activity1 はソースアクティビティ、Activity2 はターゲットアクティビティと呼ばれます。

このグラフを見ると、Activity1 にカスタム終了条件があるかどうか、再試行プロトコルがいずれかのアクティビティに適用するかどうか、マッピングポリシー (Last Writer Wins など) が Activity2 への入力に適用するかどうかなどが明確ではなく、メッセージパートがアクティビティ間で実際にどのようにマップされているか判断できません。制御フロー関係は、このグラフで明確率直に示されているので直感的に認識できますが、データ関連情報は隠れています。

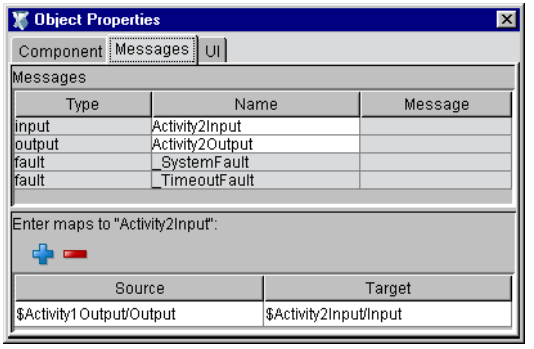
データリンク情報は、コンポーネントベース、メッセージベース、および UI ベースの情報に対するタブが含まれている、モードレス (およびドッキング可能) な［Object Properties］パレットから取得できます。



このグラフでは、Activity1 に焦点が置かれているため (周りにハンドルがあることから示されます)、［Object Properties］パネル (または「プロパティシート」) には、Activity1 に該当する情報が表示されています。Activity2 に焦点を置いた場合 (マウスで［Activity2］をクリックすることによって)、［Object Properties］パネルが更新され、このアクティビティに固有な情報が表示されます。同様に、［Link1］をクリックすると、パネルにはこのリンクに固有な情報が表示されます。これらのパネルはリアルタイムで自動的に更新されるため、情報はいつでもどんなグラフ要素に対しても表示されます。ただし、情報は単なる読み込み専用ではありません。［Object Properties］パネルのフィールドは、データ関連およびアクティビティレベルの属性値を指定する場所です。

［Component］タブ (前の図を参照) では、アクティビティレベルの情報を表示できます。このような情報には、アクティビティの名前とタイプ、コンポーネントのタイプ (この場合は Web サービス)、コンポーネントの名前、その終了条件と結合条件 (存在する場合)、再試行情報、およびマップポリシーが含まれます。これらの値の一部は、正しい選択肢がすでに入力されているドロップダウンメニューを使用して設定できます (前の図を参照)。他は、パネルに直接値を入力できるテキストフィールドです。

［Messages］タブをクリックすると、焦点が置かれているアクティビティのデータ関連情報を表示できます。



このパネルの上部には、アクティビティの入力メッセージと出力メッセージに対するプロセス固有の「名前」だけでなく、サービス (つまり、アクティビティの実装) の WSDL で指定された特定のタイプとメッセージの説明も表示されます。別の言い方をすると、［Type］および［Message］フィールドには、WSDL *portType* セグメントから取得された値が自動的に入力されます。

パネルの下部は、どのソースメッセージパートをどのターゲットメッセージパートに (XPath を使用して) マップするかを正確に指定できる場所です。直前の図は、以前に示したフローグラフの Activity2 に適用し、Activity2 に対する入力がどのように構成されるかが表示されています。ソース XPath では、この場合、Activity1 の出力メッセージパートが Activity2 の［Input］の部分に直接マップされることを指定します。これは、Activity2 が起動されると、その入力が Activity1 出力として使用されることを意味しています。これは明らかに単純なケースですが、Activity1 の出力メッセージパートから Activity2 の入力の部分への複雑な XPath マッピングが多数存在する場合もあります。

［Object Properties］パネルについては、後に詳しく説明します。ここでは、［Object Properties］パネルは次を指定できる場所であるということだけを理解していれば十分です。

- アクティビティ名
- アクティビティタイプ (Web Service Send、Web Service Receive、Composer Component、Subprocess、または Synchronize Subprocesses)
- アクティビティの終了条件
- アクティビティをトリガするための結合条件
- タイムアウトおよび再試行の設定
- 複数の着信ソースのデータが同じターゲットメッセージパートにマップされる状況においてのマップポリシー (または上書きポリシー)

- ソースメッセージパートからターゲットメッセージパートへのデータの XPath 間マッピング

フロー制御方法

WSFL モデルでは、リンクと結合のレベルでフロー論理を「細粒化」しようとするため (フロー意思決定を「XOR 分割」のような、さらに高レベルの構成要素に集約するのではなく)、WSFL ベースの方法 (Composer Process Manager が後に続く) を使用して条件付き分岐や他の一般的な制御フローパターンを指定する方法は必ずしも明確ではありません。しかし、Composer Process Designer を使用して想像できる各種フロー論理を仮想的にモデリングすることは可能です。

この節では、さらに一般的なフロースタイルの一部と、それらを Process Designer でどのように実装できるかについて説明します。

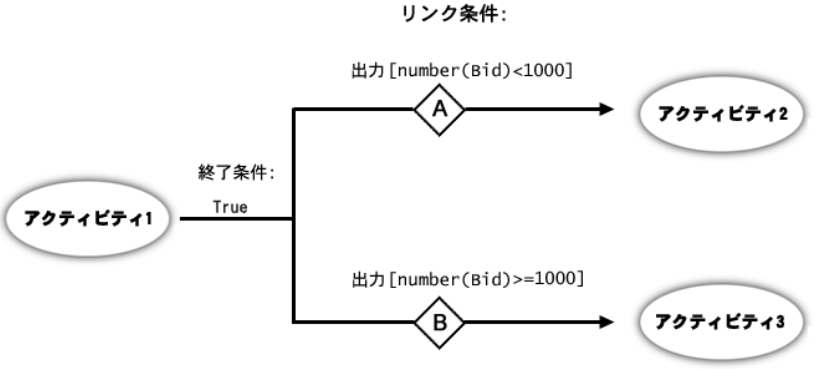
分岐論理

多くのワークフロー専門家は、分岐論理および結合論理を考慮することを習慣としています。この節ではまず分岐パターンについて説明し、結合パターンは次に説明します。

条件付き分岐 (XOR 分割)

WSFL には、条件付き分岐の概念が本来ありません。つまり、アクティビティでは、アクティビティの発信側に複数のリンクがある場合に使用するリンクを独自に決定することができません。代わりに、使用するリンクは、「リンク自体」によって決定されます。しかし、リンクでは、他のパラレルリンクのトランジション条件について把握することはできません。リンクで決定できることは、前のアクティビティの出力に基づいてそのパスを使用するかどうかということだけです。

しかし、このフロー制約メカニズムは、条件付き分岐をモデリングするために十分です。例は次のとおりです。



このシナリオでは、Activity1 によって、ある会社の見積もり情報を含む出力が生成されます。A のリンク条件は、/Bid が 1000 未満の場合にリンク A が使用されるということです。また、B の条件は、/Bid が 1000 以上の場合にリンク B が使用されるということです。当然のことながら、1 つのリンクが使用されたら、もう 1 つのリンクは使用されないため、これは排他的 OR 分割 (または XOR 分割) という条件付き分岐を表しています。

AND 分割

AND 分割ケース (「すべて」の送信リンクが常に使用される) は、Process Manager モデルのデフォルトの動作を表します。

ここで定義されている AND 分割は、各送信リンクによってそのターゲットアクティビティが「起動」される場合です。これは、各リンクに *true* という値が含まれていることを単に意味しています。

非排他的 OR 分割

複数の送信リンクがあるアクティビティでは、(出力データに基づいて) 任意の数のターゲットアクティビティを起動できる場合があります。この例として、「何かが true の場合は Activity1 を起動し、別の何かが true の場合は Activity2 を起動し、さらに別の何かが true の場合は Activity3 を起動する」というような条件が指定されている場合が挙げられます。ランタイム時に実際に起動できるアクティビティの数は、0、1、2、または 3 つです。

このケースも、配布されているリンク論理によって簡単に処理されます。各リンクでは、ソースアクティビティの出力を「参照」し、起動するかしないかについての決定に達するために適切な XPath 論理を適用することができます。そして、最後には、適切な数のターゲットアクティビティが起動されます。

複合分岐論理

「リンク A、B、および C は常に使用し、D は条件的に使用し、E は D が使用されなかった場合に使用する」というような複雑な場合を処理するために、前のパターンを組み合わせることが可能です。このケースを実装するには、次を行います。

- リンク A、B、および C を *true* に接続します。
- 「このノード値が特定の何かである場合にターゲットアクティビティを起動する」というような条件を (適切な XPath を使用して) D に設定します。
- 「[D でテストされたノードと同じノード] が特定の何かでない場合にターゲットを起動する」というような条件を E に設定します。

擬似コードでは、最終的な結果 (ターゲットアクティビティを起動するリンクに関して) は次のようになります。

```
(A AND B AND C) AND (D OR E)
```

他の複雑なケースを実行することも可能です。ただし、複合分岐方法に深入りする前に、このような複雑な構成要素はできるだけ使用するべきではない理由について、一歩下がって理解することが重要です。

プログラミングでは、複雑さは、プロシージャまたはコードブロックがさらに小さな論理単位に分解される必要があるという印です。Java コードでは、多くの AND や OR が条件にリンクされているということはまれです。これは、必要なアクションを単一の switch/case ブロックまたは一連の if/else で単純な条件とともに実行できるためです。データの従属性が複雑すぎて、これを実行できない場合、論理が単純になるよう従属性自体を準備する必要があります。データのもつれでは、入り組んだ論理を決定付けることができないようにしなければなりません。

米国取得税法には、複雑なデータの従属性に関して入り組んだ論理の適例が多数あります。しかし、内国税歳入局では、誰でも毎年正しく入力できる納税申告用紙を作成しなければなりません。納税申告用紙は、まず、主な従属性を独自の専用用紙 (または「表」) があるサブジェクトグループに分解することから作成されています。各専用用紙内には、計算をグループ化する主な区分 (部分) があります。この主な部分は、さらに単純な if/else ステートメントに分けられます。if/else ステートメントの一部は、if/else が評価される前に完了しなければならない「表」を指しています (これらの「表」は、それぞれが、部分にグループ化された一連の if/else ステートメントです)。当然のことながら、各納税申告用紙では、理論上、用紙の上部の「単一の複合式」に if/else 論理をすべて表すことができます。しかし、このような単一ステートメントプロシージャは納税者が読み込むことはできません。

複雑な分岐の必要性は、作成しているモデルがさらに単純な論理単位に分解される必要があるという印です。

結合論理

リンク論理では、ターゲットアクティビティが実際に起動されるかどうかについてではなく、ターゲットアクティビティを起動できるかどうかを決定します。アクティビティが起動されるかについての最終的な決定は、結合条件次第です。

Deferred モード (デフォルト) では、結合条件は評価されません。このため、結合アクティビティは、アクティビティの着信リンクすべての「真の値」がわかるまで起動できません。すべてのリンク値が把握されると、結合条件が評価されます。結合条件が true の場合にのみ、ターゲットアクティビティを起動できます。

Immediate モードでは、結合条件は、リンクの真の値がランタイムエンジンによって決定されるたびに評価されます。このため、1 つのターゲットアクティビティに対してインバウンドリンクが 4 つある場合は、アクティビティの結合条件が 4 回別々に評価されることが可能です。結合条件が true であること (および変更できないこと) が明らかになるとすぐに、ターゲットアクティビティが呼び出されます。

注記： 前の章で説明したとおり、結合論理は、XPath が使用されない、プロセスモデルでの唯一の論理接触点です。リンク条件と終了条件では、アップストリームデータにアクセスでき、XPath 評価における意思決定の基礎を形成します。結合条件では着信リンクの真の値のみを把握します。また、XPath を使用することはなく、データ駆動型でもありません。

結合条件は、次のようになる傾向があります。

```
(Link1 OR Link2)
```

これは、単純な非排他的 OR 条件で、1 つのリンクまたは両方のリンクが true の場合に結合が true であることを意味します。

排他的 OR 条件 (つまり、1 つの (1 つだけの) リンクが true の場合に条件が true になる) は、次のようになります。

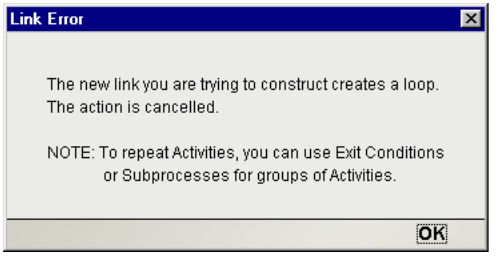
```
(Link1 AND NOT Link2) OR (NOT Link1 AND Link2)
```

この場合、「いずれ」のリンクでもアクティビティを起動することができますが、両方のリンクが true の場合、アクティビティは起動されません。この条件を目的どおりに機能させるには、結合モードを Deferred にする必要があります。

結合条件は、ターゲットアクティビティを起動するのに必要なリンク値の数 (および正確な組み合わせ) を決定するためのメカニズムと見なすことができます。リンクでは、他の (兄弟) リンクが存在しているかどうかについてや、*true* と評価される兄弟についてを知る方法はないため、これは重要です。この知識は、結合にのみ存在します。

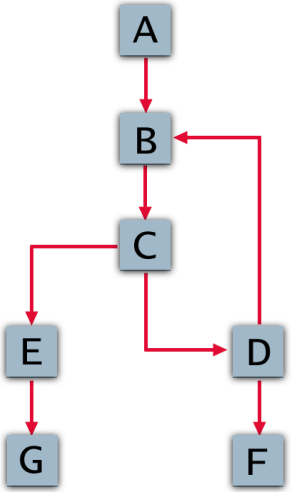
ループ

場合によっては、特定の条件を満たすまで、あるアクティビティにおいて繰り返す必要があります。たとえば、ある種のバッチ処理が存在するとします。この場合、一般的な傾向としては、ターゲットからソースの 1 つにリンクを再び描画します。しかし、この種類の制御フロー (再入可能フロー) は、WSFL や Process Designer では許可されていません。このため、サイクリックグラフを作成しようとすると、次の警告が表示されます。



アクティビティの繰り返しは、プロセス論理レベルにおいてではなく、アクティビティの実装 (または、呼び出しアクティビティの実装) によって行われる必要があります。

プロセスモデルでループが許可されていない理由は、管理するのが困難であるあいまいな状況に門戸を開いてしまうためです。たとえば、D から B に戻されたループリンクのある次の制御フローグラフについて考えてみるとします。この場合、ランタイムエンジンでは、難しい決断を下さなければなりません。



- B は、A と D という 2 つのアクティビティの入力を持つ結合ノードです。Deferred モードの場合、B では、結合条件が評価される前に「両方」の着信リンクが真の値を持つまで待機しなければなりません。しかし、D では、B と C が事前に実行されることを必要とするため、B が実行されない限り D を実行することはできません。D が実行されるのを B が待機している間、モデルはハングします。B での結合は、Immediate モードでのみ機能します。
- C は、B-C-D ループの一部として、複数回実行します。C が実行するたびに、C から E へのリンク (および C から D へのリンク) が使用され、E は、そのリンクが true と評価された場合に繰り返し起動されます。このため、E は、無意識にループの一部になることがあります。これを避けるには、E の入力リンクに対するリンク論理がループの繰り返しステータスについて「把握」している必要があります。
- E は、複数回実行されると、G を複数回トリガすることがあります。このため、G も、ループに関して把握している必要があります (また、E のアクティビティの全ダウンストリームに対しても同様)。
- E が最初の呼び出しから戻る前にループを C に戻す場合、E の新しいインスタンスを生成すべきかどうか。
- ループを通して D が実行するたびに、各サイクルで F を実行するかどうか。

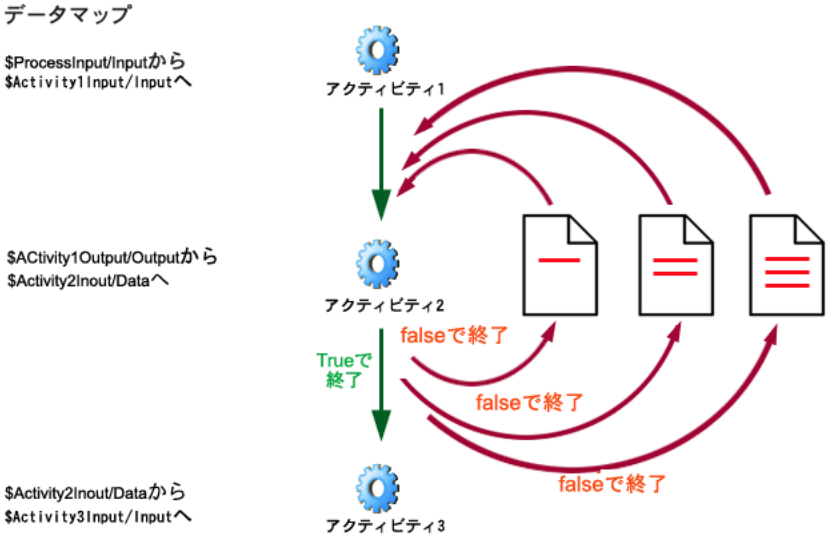
これらの問題が解決しても、このモデルの「テスト」(および、その後のデバッグ) は困難である場合があります。

ループの実現における安全性

Process Manager では、多くのループ実例が考えられます。WSFL に特有の「終了条件が false の場合に再試行する」メカニズムに依存するものもあれば、アクティビティ実装にループを委任するものもありますが、非同期ファンアウトに役立つ特別な Process Manager アクティビティがあります。

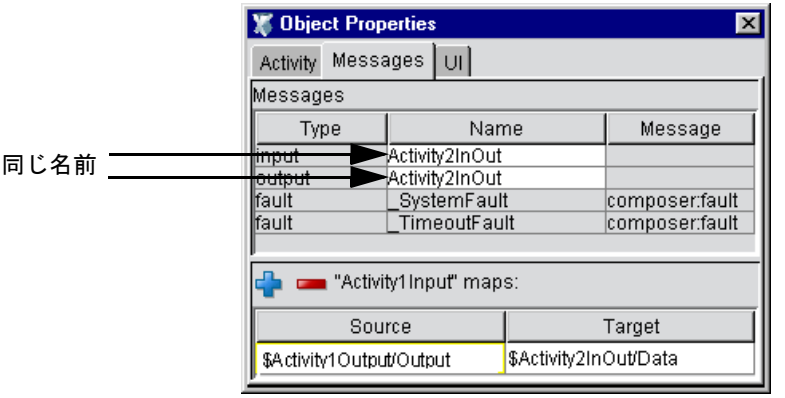
アクティビティのそれ自体へのマップ

Process Manager では、コントロールリンクをループに使用することは許可していませんが、アクティビティの「出力」をデータソースとして指定し、再試行のイベントの「入力」に使用することは許可しています。アクティビティの終了条件が false である場合の WSFL および Process Manager の標準の動作は、アクティビティを試そうとすることであるため、これによりループの 1 タイプが提供されます。出力を入力に再びマップすることによって、アクティビティは、終了条件が true になるまで、必要に応じて独自の出力をループできます。終了条件が true になると、ループは停止し、制御は配信リンクを下へと順に進みます。このシナリオは、図式的に次のように表すことができます。



Activity2 には、*Activity2InOut* という名前の入力メッセージの他に、入力メッセージと「同じ名前」の出力メッセージがあります。Activity2 は、*false* という終了条件で終了した場合、*Activity2InOut* を使用して再実行します。しかし、*Activity2InOut* のデータは、ループの一部としてアクティビティの最初の実行で変更されています (データベースルックアップの新しい情報が Data ドキュメントに追加されている可能性があります)。どのような場合においても、Activity2 の終了条件は、出力 DOM のフラグ値を検査する XPath 式になる可能性があります。フラグ値は、ループを再び繰り返す必要性またはループから抜け出す必要性のいずれかに対する信号を送信します。

この種類のマッピングを実装するには、ループアクティビティの出力メッセージと入力メッセージに同じ名前を付ける必要があります (次を参照)。



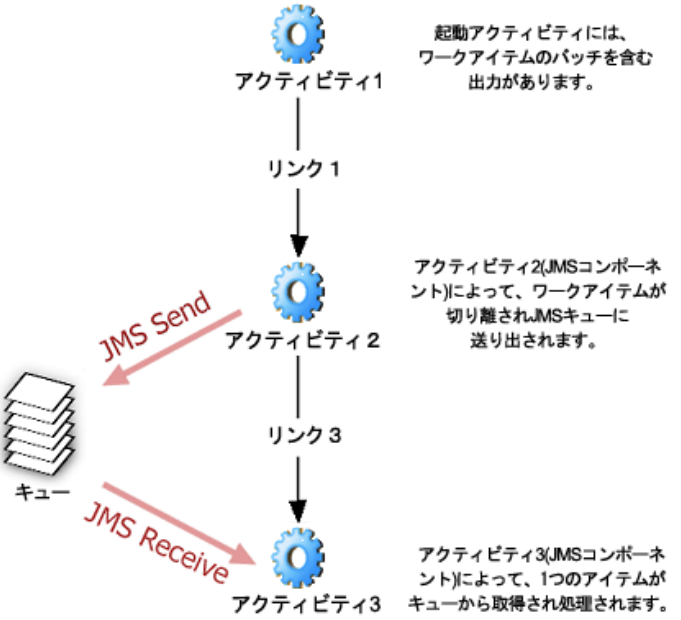
Activity2 の最初の呼び出しでは、*Activity2InOut* に *Activity1Output/Output* のデータが入力されます。false という終了条件のために Activity2 が再実行すると、*Activity2InOut* (データがすでに入力されている) は単にアクティビティに返されます。

注記： この種類のマッピングをセットアップする際には、ループアクティビティに終了条件 (ループを正常に終了する条件) を適用するのを忘れないでください。この適用を忘れた場合、無限ループが発生します。

外部データストアに対する繰り返し

前に説明したループのタイプは、ループ結果を含む出力ドキュメントが段階的に作成され、ループを通した各トリップで新しいデータが出力に追加される必要がある場合に便利です。しかし、ワークアイテムは単にキューからプルされ、一度に 1 つずつ (ループを通したトリップごとに 1 つのワークアイテム)、最終出力ドキュメントに対するデータが「それ自体」統合されることなく、処理されなければならない場合もあります。

出力としてワークアイテムのバッチを生成する開始アクティビティがプロセスにあるとします。各アイテムは、この目的のために設計された特定のアプリケーションによって「個々に」処理される必要があります。これは、処理アプリケーションが複数回 (ワークアイテムごとに 1 回) 呼び出されなければならないことを意味します。考えられるプロセスモデルは、次の図のとおりです。



このシナリオでは、Activities2 および Activities3 は Composer JMS コンポーネントですが、キューの代わりにデータベースを使用する JDBC コンポーネントの場合もあります。同じ概念は、他の外部ストアにも適合します。その目的は、Activity2 が入力としてデータのバッチ (WSDL メッセージとしてパッケージ化された) を受信することです。Activity2 は、入力の切り離し (および、可能性としてある種の事前処理の実行を) 行います。また、各ワークアイテムをメッセージキューにプッシュします。Activity3 は、そのキューを検査します。

この例では、Activity2 からの出力に (メッセージパート内の要素の) *JMSDestination* が含まれます。これは、Activity3 が操作すべき対象のキューの場所を表します。「ワークアイテムカウント」は Activity3 に渡される必要はありません (Activity3 は、1 つのワークアイテムを単に取得して処理するように設計されています)。

Activity3 は、次を実行します。

1 Activity3 は、1 つの JMS Receive アクションを実行します。このアクションでは、待機中のメッセージをキューからプルすることに成功するか、またはキューが空であることを検出します。

2 メッセージがキューから正常にプルされると、そのデータが処理され、*false* という終了条件で Activity3 が終了し、このアクティビティは再び実行します。

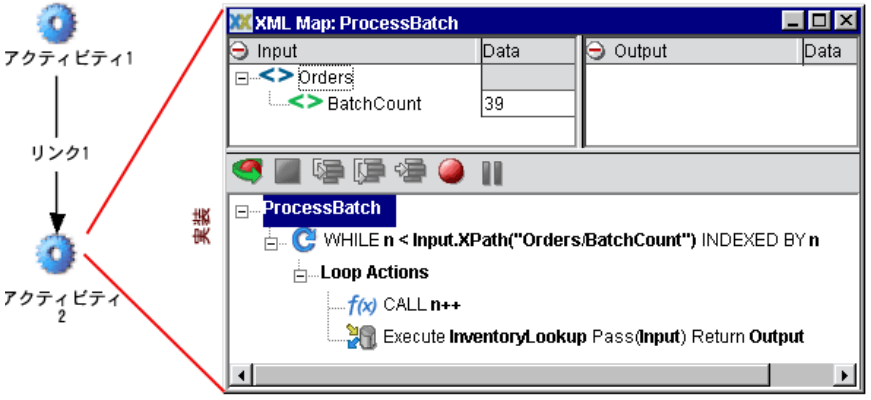
3 メッセージが取得されなかった場合 (つまり、*JMSMessageID* が空の状態で返された場合)、*true* という条件で Activity3 が終了します。

Activity3 の終了条件は、JMS Receive が成功したかどうかに基づきます。メッセージが処理された場合、終了は *false* になり、Activity3 が再び実行します。メッセージが処理されなかった場合 (つまり、キューが空であるためにアクティビティでは何もすることがない)、条件は *true* になり、Activity3 はプロセスフローの次のアクティビティに制御を渡します。

このパターンはサイクリックグラフを必要とせず、後方リンクに対する WSFL の制限に違反することもしません。これは、「間違った方向を指す」コントロールリンク、つまり再入可能性は存在しないためです。Activity3 の実行は繰り返されます。これは、特定の条件を満たすまでその「終了条件」は false になるためです。「JMS Send」および「JMS Receive」というラベルの付いた矢印は、プロセスモデル外のデータフローを表しています。データは、どのメッセージマップにも入りません。

アクティビティ実装へのループ動作の委任

前の方法の代替は、アクティビティの実装内でループ動作を非表示にすることです。たとえば、While ループで特定のコンポーネントを繰り返し呼び出すために Execute Component アクションと Repeat While アクションを使用する Composer サービスを作成するような場合があります。この方法では、Process Server でループ繰り返しのすべての部分を管理する必要はありません。

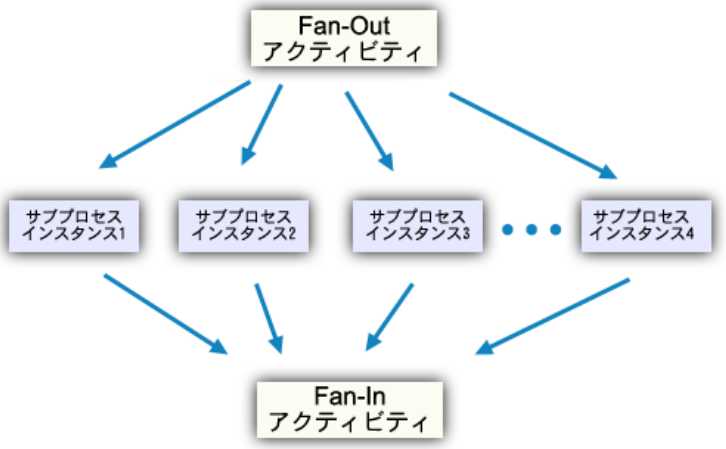


このフローグラフは、*InventoryLookup* という Composer コンポーネントがプロセスエンジンではなく Activity2 のアクションモデルによって呼び出されるということ以外は、前の場合と同じです。

ファンアウト

ワークアイテムは、一度に 1 つずつ処理するのではなく、「パラレルに (並行して)」処理すると効果的な場合があります。同時処理では、パフォーマンスが大幅に向上することもあります。

複数の同時プロセスを生成することは、「ファンアウト」と呼ばれます。設計パターンは、次のとおりです。



Fan-Out アクティビティは、ワークのバッチを受信すると、ワークアイテムを切り離し、適切なターゲットサブプロセスの複数のインスタンス (ワークアイテムごとに 1 つのインスタンス) を生成します。サブプロセスは「DetermineQuantityOnHand」のように呼ばれることがあり、バッチは SKU メンバーの集合である場合があります。*DetermineQuantityOnHand* の 1 インスタンスは、各 SKU 番号に対して作成されます。

サブプロセスの各インスタンスは、WSFL 定義の *Spawn* メカニズムによって呼び出されます。このメカニズムは、各インスタンスが独自のスレッドで実行されることを意味します (つまり、パラレルインスタンスは同時に実行され、終了します)。

このシナリオを機能させるためには、サブプロセスの各インスタンスから出力を収集し、プロセスグラフの次のアクティビティに制御が渡される前にすべてのインスタンスが終了するまで待機する「Fan-In」アクティビティが必要です。これを行うアクティビティは、前の図で Consolidator アクティビティとして表示されています。

サブプロセスの各インスタンスは、終了すると、データをマージコンポーネントに手渡します。マージコンポーネントでは、手渡されたデータを収集して、1 つの最終ドキュメントに (通常は) マージします。サブプロセスのインスタンスがすべて取得されると、マージコンポーネントの終了条件は true になり、グラフの次の点に制御を渡します。

このパターンをフローグラフに実装する場合には、問題点が 2 つあります。1 つ目の問題点は、ランタイム時に任意の数のアウトバウンドリンクを作成するためのネイティブメカニズムが WSFL によって提供されないということです。2 つ目の問題点は、ランタイム時にのみ認識される数のリンクをたとえ作成できたとしても、同期の処理に必要となる遅延バインディング結合論理の種類を指定するためのネイティブ規定がないことです。

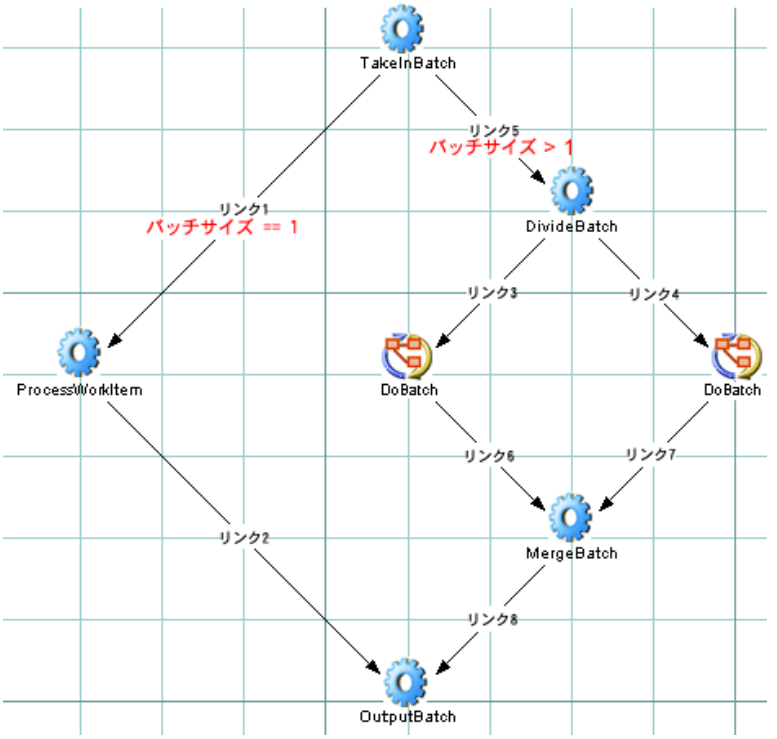
幸い、これらの問題は解決することができます。

コンポーネント制御のファンアウト / ファンイン

1 つの方法は、コンポーネントの実装でファンアウトを非表示にすることです。コンポーネントのアクションモデルにバッチに繰り返す Repeat While ループが含まれており、各ワークアイテムで Process Execute アクションを呼び出すとします。Process Execute アクションで「Spawn」の実行メソッドを指定すると、各プロセスは独自のスレッドで起動され、実行後削除されます。役立つプロセスインスタンス (「fanee プロセス」) は、これらのインスタンスによって結果がデータベース、JMS メッセージキュー、または他の外部ストアにポストされるように設計できます。同期は、2 番目のコンポーネント (「consolidator」) によって処理されます。「同期コンポーネント」は、リスナ例または定期ポーリング例のいずれかに従うことができます。後者のパターンを使用する場合は、ポーリングを連続ループまたは時間ベース (CPU への負荷が少ない) で実行できます。一方、リスナ例は、JMS サービスを使用して簡単に実装できます。

再帰的なファンアウト / ファンイン

ファンアウトは、再帰的グラフを使用して WSFL にモデリングできます。つまり、ファンアウトは、適切な数の「fanee」アクティビティが呼び出されるまで (呼び出されると、結果が結合によって累積されます)「それ自体を呼び出す」プロセスとしてモデリングできます。再帰的なファンアウトプロセスは、図式的に次のように表されます。



アルゴリズムは、次のように要約できます。

1 ワークのバッチを取り込みます。「バッチ」に複数のワークアイテムが含まれている場合は、さらに小さな 2 つのバッチに分割され (つまり、前の図に示されている Link5 が使用される)、これらの小さなバッチを入力として使用して新しいインスタンスが起動されます (つまり、Link3 と Link4 が使用され、*DoBatch* の 2 つの新規インスタンスが起動されます)。*DoBatch* の新しいインスタンスのこの再帰的な呼び出しは、着信バッチが分割できなくなるまで続行します。

2 着信バッチに「1 つ」のワークアイテムが含まれている場合は、Link1 を使用します。Link1 のターゲットは、このワークアイテムを実際に処理するコンポーネント (*ProcessWorkItem*) です。

3 *ProcessWorkItem* アクティビティの出力は、その後必要となる処理や戻りを行う *OutputBatch* に手渡されます。

4. *DoBatch* インスタンスの戻りが再帰的に呼び出される場合は、その送信リンク (Link6 または Link7 のうち該当する方) が使用されます。その後、遅延結合が *MergeBatch* で行われます。

5. *MergeBatch* コンポーネントでは、Link6 および Link7 から到着したデータを、*OutputBatch* に送信される 1 つの出力メッセージに累積します。戻り / マージ / 戻り / マージのサイクルは、1 つにまとめられたメッセージ (ドキュメント) に処理済みのワークアイテムがすべて累積されるまで続行します。

6. 最後に、*DoBatch* の一番上のインスタンスにより、まとめられたドキュメントが返されます。

グラフの一番上にあるリンク論理は、*TakeInBatch* から生成された分割を効果的に排他的 OR 分割にします (これにより、Link2 および Link8 も互いに排他的になります)。アルゴリズムは、基本的に「分割またはワーク」のうちの 1 つです。*ProcessWorkItem* アクティビティには、バッチが個々のワークアイテムに分割されるまで到達しません (このアクティビティに到達すると、対応する数の *ProcessWorkItem* インスタンスが起動されます)。出力ドキュメントは、最初に 2x2、次に 4x4 のように、プロセスの最終出力が 1 つのまとめられたドキュメントになるまでマージされます。

これは、動的にサイズ化されたタスクを実現するために単純な個別の操作を使用する、十分に分解された設計の例です。フローは、通常の WSFL 構成要素を使用して明示的に図解でき、アクティビティレベルの実装では (ビジネス論理以外は) すべて表示されます。すべてのデータは、通常のデータリンクを通して移動します (外部ストアを通した特殊な「グラフ外」の通信はありません)。処理はすべて同時で、結合はすべて同期です。

Synchronize Subprocesses アクティビティ

Composer Process Manager のネイティブアクティビティタイプの 1 つは、*Synchronize Subprocesses* と呼ばれる専用のアクティビティです。このアクティビティは、「ファンイン」機能 (非同期サブプロセスからの戻りの同期および統合) を提供するために存在しています。

Synchronize Subprocesses アクティビティを使用するグラフでは、次のパターンが実装されます。

Activity1 は、コンポーネントの実装の一環として Repeat While ループ内に複数のサブプロセスインスタンスを生成することによってファンアウトを実行する Composer コンポーネントです (これは、*Spawn* モードと *Call* モードのある Process Execute アクションによって実行できます)。Activity1 はファンアウトを作成し、(独自のアイコンを持つ) Synchronize Subprocesses アクティビティである Activity2 はファンインを作成します。

サブプロセスが Spawn によって呼び出されると、Process Server では、フローインスタンス ID を呼び出し元に返します。Activity1 は、呼び出した各サブプロセスのフローインスタンス ID を収集して、Synchronize Subprocesses アクティビティに ID を渡します。

Synchronize Subprocesses アクティビティの実装は、Composer XML Map、JDBC、または他のコンポーネントから構成されます。このコンポーネントでは、前に触れたフローインスタンス ID のリストと照合ドキュメントを入力として受信します。後者は、「fanee」サブプロセスによって返されたデータを累積するために、ランタイム時に使用されます。これに対する Process Server のしくみについては後の章で説明しますが、主要な概念は、fanee が戻るたびにファンインコンポーネント (ランタイムエンジンによって呼び出される) が通知を受け、関連付けられているワークアイテムデータが照合ドキュメントに追加されるということです。fanee がすべて取得されると、コンポーネントは true という条件で終了し、その出力 (ワークの完了したバッチ) はモデルの次のアクティビティ (複数の場合もあります) に前方へとマップされます。

検討中のプロセスアーキテクチャ

主要な概念の簡潔な要約は、次のとおりです。Process Designer でモデルを作成する際には、これらの概念を考慮するようにしてください。

アクティビティには、次の 5 つのタイプがあります。

- Composer Component
- Subprocess
- Web Service Receive
- Web Service Send
- Synchronize Subprocesses

Web Service アクティビティには、2 つの種類があります。Web Service *Send* タイプでは、WSDL の依頼 - 応答パターンと通知パターンを処理し、Web Service *Receive* タイプでは、WSDL のリクエスト - 応答操作と一方向操作を処理します。

Synchronize Subprocesses は、ファンアウトの同期に対して提供されている (Process Manager に固有な) アクティビティの専用のタイプです。

Subprocess は、さらに大きなプロセス内でアクティビティとして使用される、単なる任意の Composer プロセスです。アクティビティとしてプロセスを使用すると、ビジネスワークフローの「階層型モデリング」が可能になります。

プロセスモデルでは、データの流れを調整し、ローカルおよびオフサイトのアプリケーション (ビジネスパートナーによって管理される Web サービスを含む) の異種混合間を制御します。

開始アクティビティには、インバウンドリンクはありません。また、終了アクティビティには、送信リンクはありません。他のすべてのアクティビティには、1 つまたは複数の着信リンクがあり、送信リンクは存在する場合と存在しない場合があります。

アクティビティ間のデータの従属性は、「データリンク」により実装されます。Process Designer では、データリンクはユーザによって描画されません。これらのリンクは、あるアクティビティの出力メッセージパートを別のアクティビティの入力メッセージパートにマップすると自動的に作成されます。

アクティビティ間の時間順の従属性は、「コントロールリンク」により実行されます。コントロールリンクは、ソースアクティビティとターゲットアクティビティを接続します。リンクは、ターゲットがソースよりも先に実行されないということを保証します。この結果、サイクリックグラフパターン (ダウンストリームアクティビティにアップストリームアクティビティを指しているリンクがある) は許可されなくなります。

ワークの同期は、「結合」によって実現されます。

データの条件付きフローは、「リンク論理」(トランジション条件) によって制御されます。

「フィーダ」アクティビティの完了に基づくアクティビティの条件付きトリガは、「結合論理」(結合条件) によって制御されます。Deferred モードでは、結合条件は、着信リンクすべての真の値がわかるまで評価されません。Immediate モードでは、結合条件は、ソースリンクが評価されるたびに評価されます。

データ上書きは、「マップポリシー」を使用することによって制御できます (2 つのソースアクティビティが次のアクティビティに対する入力で同じ XPath の場所をターゲットにする可能性がある場合)。ポリシーは、Last Writer Wins (着信順の上書き)、First Writer Wins (最初のマッピングは常時で、遅延データは無視)、およびリテラルマップ順のいずれかになります。

再試行の動作は、アクティビティの「終了条件」またはタイムアウト値、あるいはその両方によって制御されます。終了条件が false の場合、アクティビティは元の入力を使用して再実行します。アクティビティは、true という終了条件またはタイムアウトのいずれかの状態が発生するまで、再実行し続けます。

リンク条件および終了条件は、XPath を使用して指定される必要があります。結合条件では、XPath を使用できません。このため、リンクの真の値に関するブール式によって指定されます。

Process Server は、プロセスインスタンスの実行を管理するランタイムエンジンで、プロセスのライフサイクルのあらゆる場所における状態の情報やインスタンスのデータなどを保持します。

プロセスは、Process Server コンソールから監視および管理 (一時停止、再開など) できます。

WSFL によって定義される (および Process Manager によってサポートされる) ライフサイクルイベントは、*Spawn*、*Call*、*Suspend*、*Resume*、*Enquire*、および *Terminate* です。

Suspend、*Resume*、および *Terminate* イベントは、Process Server コンソールから管理的に制御できます。*Enquire* イベント (ステータスクエリに対する) は、コンソールでは Enquire イベントというラベルが付けられていません。代わりに、完了ステータス情報がプロセスステータスビューにいつでも表示されます。*Spawn* および *Call* は、プロセスイニシエータによって制御されます。イニシエータは、リクエストに応答する SOAP サーバや、Process Execute アクションを実行したコンポーネントである場合などがあります。

最良実施方法の使用

WSFL ベースのプロセスモデルの主要な特徴は、互いの実装の詳細について何も知らないのに、把握しているインタフェースに基づいて協力的に通信できる作業の単位に依存していることです。このタイプのシステムの場合、作業の単位 (アクティビティ) には、これらが使用されているコンテキストについての知識はありません (また、このような知識を持つべきではありません)。アクティビティで把握している必要のあるものは、アクティビティに対する「入力メッセージ」にすべて含まれています。

最適なプロセスモデルでは、実装からインタフェースを分離するというこの原則を利用しています。これは、効率的なコードを再使用できるようにするだけでなく、技術、プラットフォーム、パートナーなどの間で相互運用を可能にします。また、トラブルシューティング、テスト、および保守を簡単にします。

適切に設計されたプロセスモデルの特徴には、次が含まれます。

- 十分に分解されたアクティビティ層。単一のアクティビティでは、「やり過ぎ」になることはありません。また、機能的な要件において一体型であるアクティビティはありません。
- すべてのアクティビティは、隣接するアクティビティに関する特別な知識なしで、スタンドアロンで実行するように設計されています。
- すべてのアクティビティには、明確なデータ入力の必要性と、それに対応する明確なデータ出力の責任があります。
- アクティビティ間のデータの従属性は、メッセージマッピングで明示的に記述されます。
- ビジネス論理は、アクティビティ実装内で完全に非表示になっています。メッセージマップに使用されるビジネス論理はありません。

  注記：アクティビティ間のメッセージマッピングでは、ある程度の詳細が表示されるべきです。基盤となる XML の要素レベルの変換 (つまり、非常に詳細なドキュメントの操作) は、プロセスモデルではなく、アクティビティ実装内で実行される必要があります。

- フローグラフは、わかりやすく読みやすいものです。グラフで処理できるアクティビティまたは結合の数を超え始めたら、関連アクティビティをサブプロセスに分解することが推奨されます。分割、結合、アクティビティなどが多数含まれているモデルは、テストやデバッグが非常に困難になる場合がありますが、同じモデルでも 3、4 つのサブプロセスに分解し、各サブプロセスに 3 つまたは 4 つのアクティビティを含めることができる場合は、それらのサブプロセスをスタンドアロンでテストし、最終的に統合された 1 つの強力なモデルに結合することが可能です。

3 Process Designer のユーザインタフェース

この章では、プロセスモデルを作成するための設計時環境である、Composer Process Designer のユーザインタフェース機能について説明します。

メイン機能

Process Designer は、矢印の図で示したプロセスモデルを作成するための視覚的な編集環境です。この環境では、(アイコンで表される) アクティビティの作成と配置、2 つのアクティビティ間のリンクの作成、2 つ以上のアクティビティ間で行うデータマップやリンク条件の設定などを簡単に指定できます。描画環境ではポイントアンドクリックを使用できるため、フローグラフを簡単に作成できます。

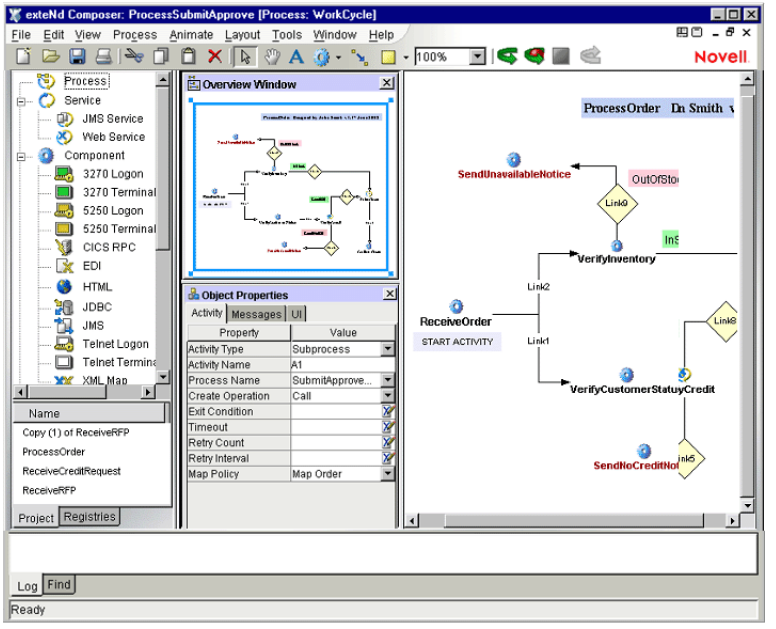
Process Designer には、設計時に任意のプロセスをアニメーションモードで実行できる重要な機能があります。これにより、複数のプロセスモデルを 1 つのセッションで設計、テスト、およびデバッグすることができます。アニメーションモードでは、アクティビティのステップインまたはステップオーバー、ブレークポイントの設定、発生したデータ変換の監視、ログメッセージの表示、分割動作および結合動作の監視などをすべてリアルタイムで実行できます。また、このセッション内では、アクティビティの実装におけるドリルダウン、コンポーネントでのアクションモデルの変更、メッセージマップやドキュメントの編集、リンクまたは結合論理の修正などをインタラクティブに行うこともできます。この機能によって、開発スピードを大幅に向上できます。

Process Designer ウィンドウ

Process Designer は (他のコンポーネントエディタと共に) Composer 内部で動作するため、Composer ユーザに親しみやすい環境にする必要があります (次の図を参照)。

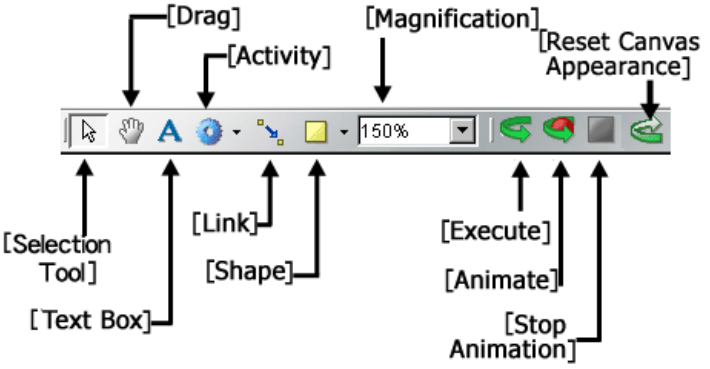
Process Designer がフロントエディタの場合は、3 つの新しいペインが表示されます。

- ［*Process Model*］ペイン (キャンバスと呼ばれることもあります) では、プロセスモデルグラフを作成します。このペインは最も大きなペインです。
- ［*Object Properties*］ペインでは、プロセスモデルのさまざまな要素 (アクティビティ、リンク、テキストラベル、形状など) に対して、プロパティを指定することができます。
- ［*Overview*］ペインには、メインキャンバスの［bombsight］ビューがあります。青い長方形の上でマウスボタンを押したまま小さなウィンドウ内にドラッグすることによって、視覚焦点が特定領域に即座に設定されるので、スクロールバーを使用しなくてもメインキャンバスを超えて操作できます。

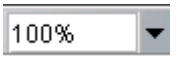


ツールバー

Composer Process Designer では、複数の新しいツールが Composer ツールバーに追加されます (次を参照)。



ツール	アイコン	使用方法
Selection Tool		このポインタでアイテムをクリックすると選択できます。アイテムをいったん選択すると、キャンバスの任意の場所にドラッグできます (<Shift> を押してクリックすると、複数の要素を個別に選択できます)。
Drag		［Drag］ツールを使用するとキャンバス全体を移動できるため、キャンバスの「画面全体を移動」して関心のある領域を表示できます。
Text Box		このツールでキャンバスをクリックすると、長方形の中にテキストラベルを作成できます。
Activity		このツールを使用すると、キャンバスに新しいアクティビティ要素を配置できます。
Link		このツールを使用すると、制御フローのリンクを表す矢印で2つのアクティビティを接続できます。
Shape		このツールを使用すると、キャンバス内の長方形や楕円のサイズを変更できます。

ツール	アイコン	使用方法
Magnification		ドロップダウンメニューを使用すると、あらかじめ設定されている拡大表示から選択できます。また、カスタムの拡大要素をドロップダウン横のテキストフィールドに入力することもできます。
Execute、Animate、Stop Animation、および Reset	 	これらのボタンを使用すると、設計時環境でプロセス (テスト目的のプロセス) を開始または停止できます。(左下にある) [Reset] ボタンは、アニメーションの実行が終了するまでグレー表示されます。[Reset] ボタンを押すと、フローダイアグラムの表示がリセットされます。

グラフ要素

グラフ要素には、アクティビティ、リンク、テキストボックス、形状 (長方形と楕円) があります。これらの要素の作成ツールには、[Composer] のメインメニューバーにある [Process] メニューや、メインツールバーのツールアイコンからアクセスできます。作成ツールはポイントアンドクリック形式で使用できます。

注記 : 次で説明しているグラフ要素の表示の特性 (色、境界、テキストの行端揃え) は、[Object Properties] ペインの [UI] タブにあるコントロールを使用して調整できます ([View] メニューの [Object Properties] コマンドを使用すると、ペインの表示レベルを切り替える事ができます)。

アクティビティ

アクティビティには、上で説明した 5 つのタイプがあります。それぞれのアクティビティについて簡単に説明します。

アクティビティタイプ	説明
Component アクティビティ	Component アクティビティは、1 つ以上 Composer コンポーネントを使用して 1 つ以上の外部システムと対話するために、コンポーネントまたはサービスに対してランタイム対話を提供します (たとえば、JDBC、3270、5250、CICS、RPC、JMS、HTML、Telnet、EDI、または XML Map と、Composer JMS サービスまたは Composer Web サービスとの対話)。コンポーネントアクティビティでドリルダウンして、Composer コンポーネントのアクションモデルを表示および編集できます。
Web Service Receive アクティビティ	Web Service Receive アクティビティは、公開された Web サービスに対してランタイム対話を提供し、着信メッセージを現在のプロセスインスタンスに関連付けます。
Web Service Send アクティビティ	Web Service Send アクティビティは、公開された Web サービスに対してランタイム対話を提供します。ランタイム対話によって、Process Manager ユーザは、Web サービスの WSDL リソース、サービス名、結合、操作、エンドポイントロケータ、および接続を選択できます。これは、Composer 3.0 導入された WS Interchange アクションと似ています。
Subprocess アクティビティ	Subprocess アクティビティは、Process Designer で作成したプロセスを表します。つまり、プロセスにより他のプロセスを呼び出すことができます。これにより、階層的なフローアーキテクチャが作成されます。Subprocess アクティビティをダブルクリックして、サブプロセスグラフを表示および編集します。
Synchronize Subprocesses	これは、繰り返し実行したサブプロセスから返ってきた情報をマージできる特別なアクティビティタイプです。

グラフ内でこれらのアクティビティタイプのうち 1 つだけインスタンスを作成するには、メインツールバー (またはアイコンの下のフライアウトアイコンリスト) から対応するツールアイコンを選択して、メインキャンバス内の任意の場所をクリックします。

リンク

［Link］ツールを使用すると、有向辺 (矢印) を持ったアクティビティを接続できます。この操作は簡単です。まず、ツールバーからツールを選択します。その後、アクティビティをクリックします。アクティビティはリンクのソースアクティビティになります。マウスを押したまま、目的のターゲットアクティビティにラインを引きます (ラインは、実際にはアクティビティではなくて、アクティビティアイコンの中心に伸ばしてください)。マウスを放すと、リンクの「ターゲットエンド」に矢印が表示され、2 つのアクティビティが制御フローによってリンクされます。ここで選択カーソルを使用して、キャンバスにあるいずれかのアクティビティをドラッグすると、必要に応じてリンクが自動的に拡張または再設定、あるいはその両方が行われ、両方のアクティビティの接続が維持されます。

テキストボックス

［Text］ツールを使用すると、キャンバスにテキストボックスを配置できます。キャンバスをクリックすると、「Untitled」と表示された長方形が表示されます。次に、バックグラウンドやアウトラインなどの色を設定し、［Object Properties］ペインの［UI］タブに適切な設定を入力して、ボックス内のテキストを変更します。

テキストボックスは、アクティビティの特性や制御フローの目的などを文書化したり、タイトル、著者情報、更新日などを示したりするために、キャンバスのさまざまな場所で使用できる単純な任意のテキストラベルです。テキストボックスの位置は (ドラッグして) いつでも変更できます。制御フローへの影響はありません。これらは任意で使用できます。

［Object Properties］ペインの［UI］タブでコントロールを使用すると、テキストボックスの表示を変更できます (色、サイズ変更、余白、センタリングなどだけではなく、テキストサイズ、フォント、スタイルも変更できます)。

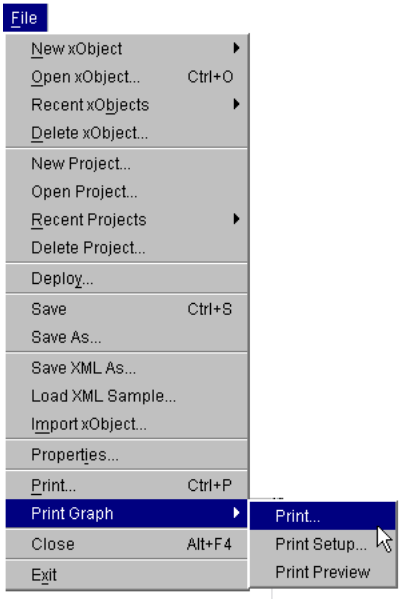
形状

［Shapes］ツールを使用すると、長方形、楕円、自分で作成した .jpg または .gif グラフィックをキャンバスの任意の場所に置くことができます。厳密には、これらの要素は飾りです。プロセスランタイムダイナミクスには影響ありません。

メニューコマンド

Composer で、Process Manager がフロントエディタの場合、Composer メニューにプロセス固有のメニューコマンドがいくつか表示されます。［**File**］メニュー、［**View**］メニュー、［**Process**］メニュー、［**Layout**］メニューの構造について、図とともに説明します。

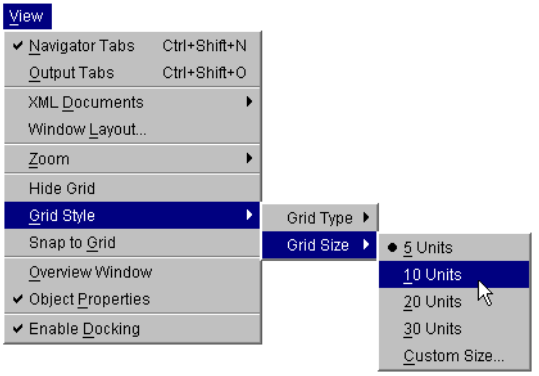
［File］メニュー:



Composer の File メニューへの追加は、［**Print Graph**］コマンドだけです。

メニュー	サブメニュー	コマンド	説明
File	Print Graph	Print	これを選択すると、すべてのグラフおよび説明、または選択されたグラフおよび説明を表示できます。
File	Print Graph	Print Setup	これを選択すると、プロセスのどの部分を印刷するかを指定できます(詳細については次のダイアログを参照)。
File	Print Graph	Print Preview	これを選択すると、選択した項目を印刷する前にプレビューできます。

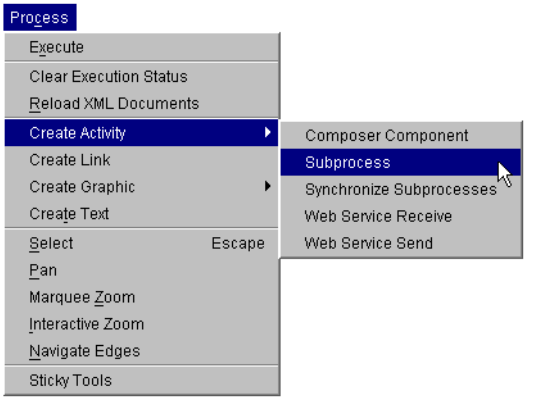
［View］メニュー:



メニュー	サブメニュー	コマンド	説明
View	Show XML Documents		プロセスで使用される XML ドキュメントを表示するかどうかを指定します。
View	Zoom	Many	［Zoom］ツールを使用すると、キャンバスの拡大表示 (%) を設定できます。プルダウンメニューを使用すると、あらかじめ設定されている値を指定できます。また、［Custom Zoom］を選択すると、任意の拡大率を指定できます。
View	Show/Hide Grid		グリッドの表示レベルを切り替えます (次を参照)。
View	Grid	Grid Type	空白のバックグラウンドか、(グリッドサイズオプションと) グリッドビューのいずれかを選択することができます。デフォルト値は［None］です。
View	Grid	Grid Size	グリッドビューモードの場合、このコマンドはライン間またはドット間の空白を設定できます。
View	Overview Window		レイアウトの作成または編集時の［Overview］ペイン (［bombsight］ビュー) の表示レベルを切り替えます。
View	Object Properties		レイアウトの作成または編集時の［Object Properties］ペインの表示レベルを切り替えます。このペインは、データマップ (メッセージ) が指定されるペインです。

View	Enable Docking		上で説明したモダールウィンドウをグラフの端に持ってくると、ドッキングします。デフォルトは［On］です。

［Process］メニュー:



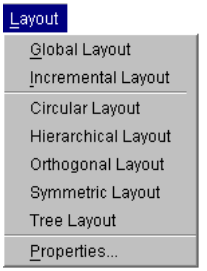
次に、Process Menu コマンドについて説明します。

メニュー	サブメニュー	コマンド	説明
Process	Execute		開始から終了まで、プロセスを実行します。
Process	Clear Execution Status		このメニューコマンドはツールバーの右端の［Reset］ボタンの機能と同じで、アイコン、リンクなどのグラフィック状態をすべて元のアニメーション状態にリセットします。
Process	Reload XML Documents		コンポーネントエディタ内で、Component メニュー項目の［Reload］と同じ機能を実行します。
Process	Create Activity	Composer Component	アクティビティツールを［Activity］ツールに変更して設定します。キャンバスをクリックすると、新しいコンポーネントアクティビティが作成されます。

Process	Create Activity	Subprocess	アクティビティツールを［Activity］ツールに変更して設定します。キャンバスをクリックすると、新しいサブプロセスが作成されます。
Process	Create Activity	Synchronize Subprocesses	アクティビティツールを［Activity］ツールに変更して設定します。キャンバスをクリックすると、新しいSynchronize Subprocessesアクティビティが作成されます。
Process	Create Activity	Web Service Receive	アクティビティツールを［Activity］ツールに変更して設定します。キャンバスをクリックすると、新しいWeb Service Receive アクティビティが作成されます。
Process	Create Activity	Web Service Send	アクティビティツールを［Activity］ツールに変更して、Composer コンポーネントモードにツールを配置します。キャンバスをクリックすると、新しい Web Service Send アクティビティが作成されます。
Process	Create Link		アクティビティツールを［Link］ツールに変更します。
Process	Create Graphic	Rectangle	アクティビティツールを[Graphics]ツールに変更して設定します。キャンバスをクリックすると、サイズが変更できる長方形が作成されます。
Process	Create Graphic	Oval	アクティビティツールを[Graphics]ツールに変更して設定します。キャンバスをクリックすると、サイズが変更できる楕円が作成されます。
Process	Create Graphic	Rounded Rectangle	アクティビティツールを[Graphics]ツールに変更して設定します。キャンバスをクリックすると、サイズが変更できる丸みのある長方形が作成されます。
Process	Create Graphic	Diamond	アクティビティツールを[Graphics]ツールに変更して設定します。キャンバスをクリックすると、サイズが変更できるひし形が作成されます。

Process	Create Graphic	Picture	アクティビティツールを［Graphics］ツールに変更して設定します。キャンバスをクリックすると、イメージファイル (.jpg または .gif) が配置されます。キャンバスのプロパティシートの［UI］タブでイメージファイルを指定できます (この章の最後を参照)。
Process	Create Text		アクティビティツールを［Text］ツールに変更します。
Process	Select		(グラフィック項目を選択するために) 現在のツールを矢印カーソルに変更します。
Process	Pan		現在のツールをハンドツールに変更すると、キャンバスで、大きなグラフを迅速に移動することができます。
Process	Marquee Zoom		このオプションは、［Overview］ウィンドウ (［**View**］>［**Overview Window**］の順にクリック) が表示されている場合に役に立ちます。このオプションがアクティブの場合は、青いマーキーボックスの外側をクリックして、キャンバスを拡大表示できます。
Process	Interactive Zoom		上のオプションと似ていますが、これを選択するとマーキーボックス (青いボックス) のコーナーハンドルをドラッグして「ビューのサイズを変更」できます。
Process	Navigate Edges		任意のアクティビティをクリックしてアクティビティツールを変更し、リンクパスと (任意のアクティビティを実行しない) グラフのアニメーションを表示できます。実行可能ファイルは実行しません。
Process	Sticky Tools		ツールを一度選択すると、選択された状態が維持されます。そのため複数のアクティビティをキャンバスにドロップしたり、［Link］ツールを何度も選択することなく複数のリンクを作成したりできます。

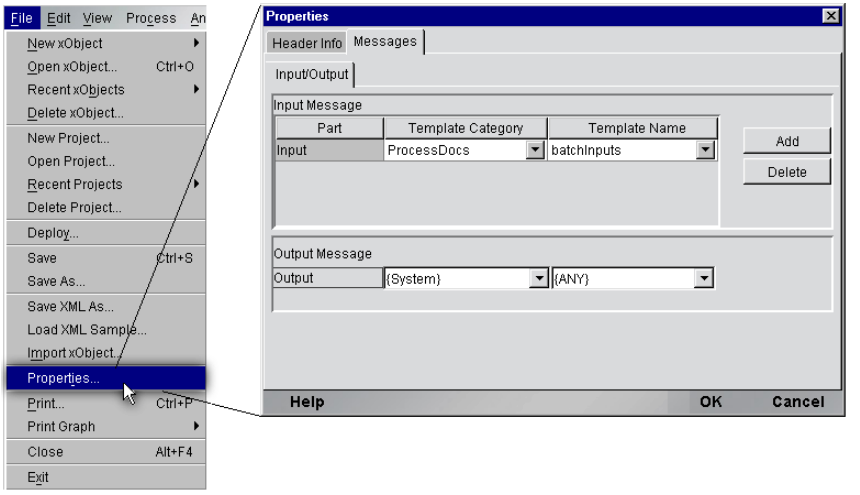
［Layout］メニュー:



メニュー	コマンド	説明
Layout	Global Layout	デフォルトのレイアウトモード。グラフ全体が設計時にメモリ内にキャッシュされます。
Layout	Incremental Layout	メモリをより効率的に使用するオプションのレイアウトモード。
Layout	Circular Layout	ハブとスポークのようにノードを配置します。「Layout Properties」章の説明を参照してください。
Layout	Hierarchical Layout	トップダウンの関係を強調する、従来の作図スタイルである「組織図」を適用します。
Layout	Orthogonal Layout	ノードを制限して、行と列のモチーフにリンクします。
Layout	Symmetric Layout	「対称的な関係」が強調されるように、縁の重なりが最小化され、ノードの配置が統一されます。
Layout	Tree Layout	従来の「ファミリツリー」レイアウトになります。このレイアウトは上で説明した階層スタイルと類似していますが、リンクが並列ではないので完全に水平または垂直になることはほとんどありません。
Layout	Properties	上の設定を微調整するための［Preferences］ダイアログボックスを表示します。

プロセスプロパティ

プロセス全体の一般情報は、［**File**］>［**Properties**］の順にクリックしてアクセスできます。ダイアログボックスには、［Header Info］と［Messages］の 2 つのタブがあります。［Header Info］タブには、該当するプロセスに関する「名前」と「コメントタイプの情報」があります。［Messages］タブには、プロセスの入力メッセージと出力メッセージに対する XML テンプレート情報があります。



Object Properties

Process Designer で作成した矢印の図の説明に出てきたオブジェクトのそれぞれのタイプには、それ自身のプロパティセットがあります。プロパティは状況依存型です。そのため、キャンバスで選択したオブジェクトのタイプによって変化します。オブジェクトの現在のプロパティを表示するには、(ポインタツールを使用してクリックして) オブジェクトを選択し、(［Object Properties］パレットが表示されていない場合は)［**View**］メニューで［**Object Properties**］を切り替えます。

［Object Properties］パレット (オブジェクトの「プロパティシート」と呼ばれています) では、次のような重要なアクティビティの属性を指定できます。

- 終了条件
- 結合条件
- タイムアウト
- 再試行回数
- 再試行間隔

- マップポリシー
- その他 (次を参照)。

次の節では、オブジェクトの適切なタイプについて作業する場合に、プロセスのさまざまな要素のプロパティシートがどのように表示されるかについて説明します。

アクティビティプロパティ

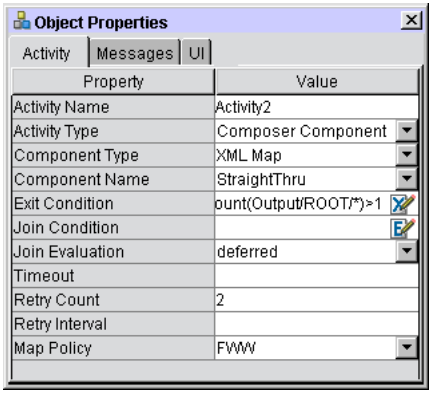
Process Manager では 5 つのアクティビティタイプがサポートされており、それぞれ個別にオブジェクトのプロパティセットを設定できます。サポートしているアクティビティのタイプは、Composer Component、Web Service Send、Web Service Receive、Subprocess、または Synchronize Subprocesses です。それぞれのプロパティシートは、次で詳しく説明します。

Composer Component

Component アクティビティの［Object Properties］パネルには、［Activity］、［Messages］、および［UI］の 3 つのタブがあります。これらの表示を図に示し、機能を次の表で説明します。

［Object Properties］タブとパネルは、「状況に対応」します。これらの内容はキャンバスで選択したアクティビティの属性を反映して、自動的に更新されます。同様に、該当するフィールドの作業を終えると、プロパティ設定で行った変更はすべて、リアルタイムでただちに有効になります (変更を有効にするには、プロパティフィールドの外側をクリックする必要があります)。

［Activity］タブ

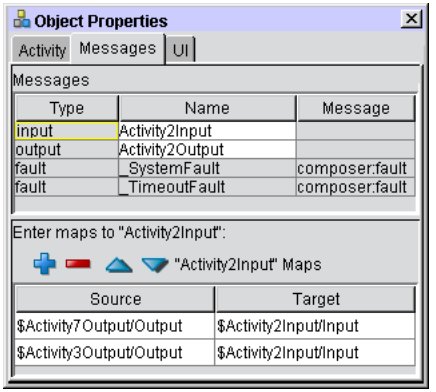


プロパティ	コントロールタイプ	使用方法
Activity Name	テキストフィールド	キャンバスのアクティビティアイコンの下に表示される名前。
Activity Type	ドロップダウン	ドロップダウンリストを使用すると、現在選択しているアクティビティのアクティビティタイプを変更できます。ドロップダウンは、アクティビティタイプの 5 つのカテゴリを表示します。
Component Type	ドロップダウン	使用できる Composer コンポーネントタイプ (XML Map1、Web サービス、JDBC コンポーネントなど) のリストを表示します。
Component Name	ドロップダウン	現在のプロジェクトですでに作成している［Component Type］(前のプロパティ) で選択したコンポーネントに対応した［Component Names］のリストを表示します。

Exit Condition	テキスト フィールド	［Exit Condition］はブール値の Xpath 式です。この式は、アクティビティが正常に終了したかどうかを判断するためのものです。 ［Exit Condition］の式は、アクティビティの出力メッセージまたは同じ制御パスでアクティビティの前に実行したアクティビティの出力を参照します。 Exit Condition の値が True (真) と評価されると、アクティビティは「完了」として扱われます。アクティビティが完了すると、プロセスはコントロールの通常フローに戻ります。完了しない場合、アクティビティは再実行されます。 アクティビティは、X 回実行されます (X は後で定義されている［Retry Count］の値)。 ［Retry Interval］は、再実行する間隔を定義します。
Join Condition **(結合ターゲットの場合だけ表示される)**	テキスト フィールド	［Join Condition］は単純な OR/AND/NOT 構文のブール式です。この式は、着信リンクの真の値に基づいて、並列な作業を同期化するためのものです。 着信リンクが 2 つ以上ある場合、アクティビティは結合アクティビティと呼ばれます。［Join Condition］が True (真) の場合に限り、結合アクティビティが実行されます。条件を明示的に指定しない場合、デフォルトは True (真) です。
Join Evaluation **(結合ターゲットの場合だけ表示される)**	ドロップダウン	選択肢は［Deferred］および［Immediate］です。これらのオプションの意味については、第 1 章を参照してください。

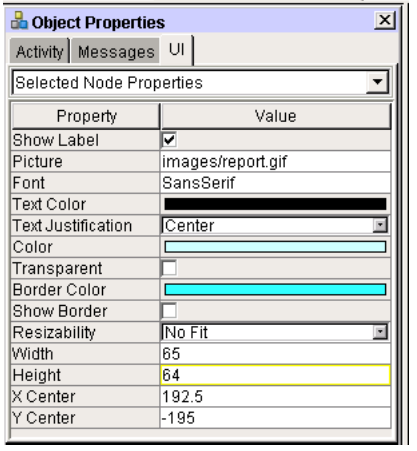
Timeout	テキスト フィールド	［Timeout］属性は、時間間隔を定義します。アクティビティは、この時間内に動作を終了させる必要があります。タイムアウトになると、［Retry Count］が指定されている場合は適用され、アクティビティが再実行されます。 タイムアウト後、Process サーバは次の再試行を開始する前に、一定時間（［Retry Interval］で指定した時間）待機します。次で説明する［Retry Interval］は、アクティビティが再試行するまでの待ち時間を定義します。 ［Timeout］および［Retry］の設定は任意です。［Retry］および［Retry Interval］ともに、デフォルトは 0 です。
Retry Count	テキスト フィールド	アクティビティを再試行する回数。
Retry Interval	テキスト フィールド	再試行が必要な場合は、再試行するまでの待ち時間。
Map Policy	テキスト フィールド	［*Last Writer Wins*］、［*First Writer Wins*］、または［*Map Order*］です。2 つのアクティビティが互いのデータを上書きする場合（たとえば、2 つのソースアクティビティがターゲットアクティビティの入力メッセージ内で同じ XPath の場所を求めて争う場合）に限り、この値は重要です。

［Messages］タブ



プロパティ	コントロールタイプ	使用方法
Messages	列は次の 3 つです。 • [Type]（編集不可） • [Name] • [Message](編集不可)	WSDL が存在する場合、[Type] と [Message] は WSDL の Port Type Operations Input 要素と Port Type Operations Output 要素から抽出されます。[Name] は [Type] にデフォルトの [Activity Name] が追加された名前がデフォルトになります (*Activity2Output* など)。
Maps:	[+] アイコンおよび [-] アイコン	アクティビティの最後の出力から現在の入力までの割り当てを追加および削除します。
Source	ボタン	(グラフ内で前のアクティビティの出力に適用した) [Source] の XPath 式。
Target	ボタン	(現在選択されているアクティビティの入力に適用した) [Target] の XPath 式。

[UI] タブ



プロパティ	コントロールタイプ	使用方法
Show Label	チェックボックス	テキストラベル (名前) を現在選択しているアクティビティオブジェクトの下に表示するかどうかを指定します。
Picture	テキストフィールド	現在選択しているアクティビティオブジェクトを表示する場合に使用されるイメージ (Gif または JPEG) へのパス。必要に応じて、これを使用して、カスタムのアイコンアートをポイントします (これは、設計時に使用します。アートは任意の jar ファイル内に配置されます)。
Font	ダイアログボックス表示	[Value] フィールドをクリックすると、[Choose Font] ダイアログボックスが表示されます。このダイアログボックスには、フォント、スタイル (プレーン、ボールド、イタリック、またはボールドイタリック)、ポイントサイズを選択する 3 つのドロップダウンがあります。
Text Color	カラーピッカー	現在のオブジェクトに関連付けられているテキストで使用される色を表示します。このバーをクリックすると、[Color Picker] ダイアログボックスが表示されます。
Text Justification	ドロップダウンメニュー	左 中央 (デフォルト) 右
Color	カラーピッカー	これは、選択したオブジェクトのバックグラウンドカラーです。このバーをクリックすると、[Color Picker] ダイアログボックスが表示されます。
Transparent	チェックボックス	このチェックボックスをオンにすると、オブジェクトは透明になり、オフにすると不透明になります。
Border Color	カラーピッカー	選択しているオブジェクトの境界線カラー このバーをクリックすると、[Color Picker] ダイアログボックスが表示されます。
Show Border	チェックボックス	このチェックボックスをオンにすると境界線が表示され、オフにすると表示されません。

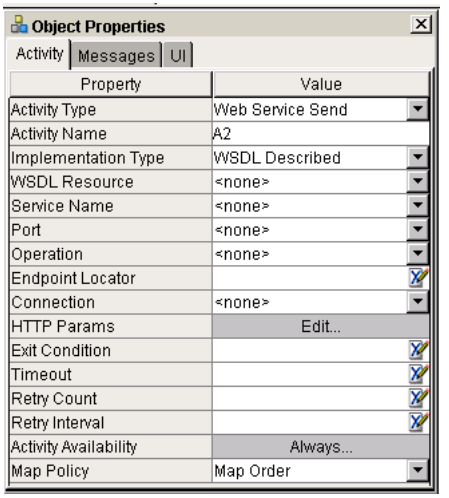
Resizability	ドロップダウン	選択肢は次のとおりです。 No Fit Tight Fit Tight Width Tight Height Tight Fit Preserve Aspect Preserve Aspect
Width	テキストフィールド	アイテムの幅。デフォルト値は 40.0 です。
Height	テキストフィールド	アイテムの高さ。デフォルト値は 32.0 です。
X Center	テキストフィールド	X 座標。
Y Center	テキストフィールド	Y 座標。

Web Service Send

Web Service Send アクティビティには、［Object Properties］パネルの［Activity］タブに反映される独自のオブジェクトプロパティがあります。

注記： このアクティビティの［Message］タブおよび［UI］タブは、すでに説明したコンポーネントアクティビティの［Message］タブおよび［UI］タブと同じです。ここでは［Activity］タブについてのみ説明します。

Web Service Send の［Activity］タブ



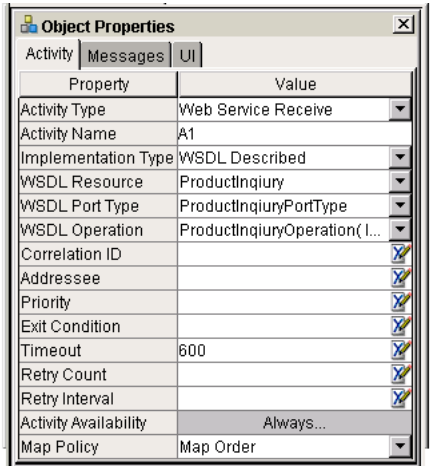
プロパティ	コントロールタイプ	使用方法
Activity Type	ドロップダウン	アクティビティタイプのドロップダウンリスト。
Activity Name	Activity1 ～ n (デフォルト)	アクティビティの名前です。
WSDL Resource	ドロップダウン	Composer プロジェクト内で使用できる WSDL リソースのドロップダウンリスト。
Service Name	ドロップダウン	WSDL リソース内で使用できる Web サービスのドロップダウンリスト。
Service Port or Binding	ドロップダウン	WSDL リソース内で使用できるバインド名のドロップダウンリスト。
Operation	ドロップダウン	WSDL リソース内で使用できる操作名のドロップダウンリスト。
Endpoint Locator	XPath 式	使用する Web サービスの［Endpoint Location］(通常はサーブレットを指す URL) を引用符で囲んで入力します (または、ランタイム時に［Endpoint Location］を評価する XPath 式を入力します)。
Connection	接続	接続のドロップダウンリスト。
HTTP Params	プッシュボタン	コンテンツの長さおよび他の共通 HTTP パラメータを指定できる、「HTTP ヘッダパラメータ」が表示されます。

Exit Condition	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Join Condition (存在する場合)	テキストフィールド	86 ページ「Join Condition」の説明を参照してください。
Join Evaluation	ドロップダウン	［Join Condition］と同様に、ターゲットアクティビティが結合アクティビティの場合に限り、このフィールドは表示されます。ドロップダウンの選択肢 (［Immediate］または［Deferred］) は、結合の評価モードを指定します。
Timeout	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Retry Count	数値フィールド	87 ページ「Retry Count」の説明を参照してください。
Retry Interval	テキストフィールド	87 ページ「Retry Interval」の説明を参照してください。
Map Policy	テキストフィールド	87 ページ「Map Policy」の説明を参照してください。

Web Service Send の［Messages］タブおよび［UI］タブ

これらのタブの設定は、コンポーネントアクティビティですでに説明した設定と同じように動作します。

Web Service Receive



Web Service Receive の［Activity］タブ

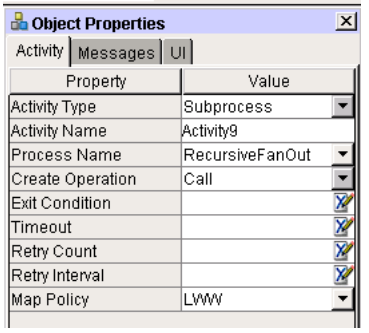
プロパティ	コントロールタイプ	使用方法
Activity Type	ドロップダウン	使用できるアクティビティタイプのドロップダウンリスト。
Activity Name	Activity1 ～ n (デフォルト)	アクティビティの名前です。
Implementation Type	ドロップダウン	次のいずれかになります。Web サービス、JMS サービス、または外部。
WSDL Resource	ドロップダウン	Composer プロジェクト内で使用できる WSDL リソースのドロップダウンリスト。
WSDL Port Type	ドロップダウン	このサービスのポートタイプ。
WSDL Operation	ドロップダウン	WSDL リソース内で使用できる操作名のドロップダウンリスト。
Correlation ID	テキストフィールド	トランザクションを個別に認識するために使用される任意のユーザ定義値。
Addressee	テキストフィールド	通常、この特定のトランザクションまたはアクティビティに関連付けられている「所有者」(個人名) を定義する任意の文字列ラベル。

Priority	テキストフィールド	通常、このアクティビティまたはワークアイテムの重要性に関連するいくつかの任意の値。
Exit Condition	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Join Condition (存在する場合)	テキストフィールド	86 ページ「Join Condition」の説明を参照してください。
Timeout	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Retry Count	数値フィールド	87 ページ「Retry Count」の説明を参照してください。
Retry Interval	テキストフィールド	87 ページ「Retry Interval」の説明を参照してください。
Map Policy	テキストフィールド	87 ページ「Map Policy」の説明を参照してください。

Web Service Receicve の［Messages］タブおよび［UI］タブ

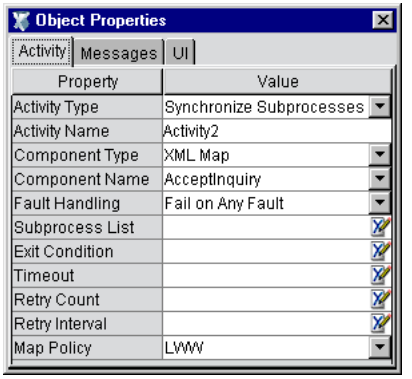
これらのタブの設定は、Component アクティビティですでに説明した設定と同じように動作します (前を参照)。

Subprocess



Subprocess の［Object Properties］パネルにあるすべてのタブのプロパティは、すべて Composer Component のプロパティの名前と同じです (［*Create Operation*］プロパティは除く)。［*Create Operation* ］プロパティは、サブプロセスが非同期 (「実行後削除」) または同期 (応答があるまでポーリングする) に呼び出される必要があるかどうかを反映して、［*spawn*］または［*call*］のいずれかに設定されます。

Synchronize Subprocesses



Synchronize Subprocesses アクティビティは、ファンアウトサブプロセスによって生じた複数のファンインを調整する特別なアクティビティです。この章の後半の「Synchronize Subprocesses アクティビティ」の説明を参照してください。

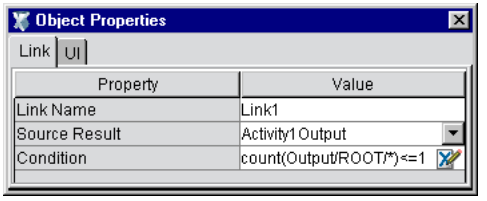
プロパティ	コントロールタイプ	使用方法
Activity Type	ドロップダウン	使用できるアクティビティタイプのドロップダウンリスト。
Activity Name	Activity1 ～ n (デフォルト)	アクティビティの名前です。
Component Type	ドロップダウン	この Composer プロジェクトで使用できるコンポーネントのリスト。
Fault Handling	ドロップダウン	選択肢は［Fail on Any Fault］と［Fail If All Fail］です。
Subprocess List	テキストフィールド (XPath)	ファンアウトサブプロセスに関する ProcessInfo の Xpath の場所。
Exit Condition	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Join Condition (存在する場合)	テキストフィールド	86 ページ「Join Condition」の説明を参照してください。
Timeout	テキストフィールド	86 ページ「Exit Condition」の説明を参照してください。
Retry Count	数値フィールド	87 ページ「Retry Count」の説明を参照してください。

Retry Interval	テキストフィールド	87 ページ「Retry Interval」の説明を参照してください。
Map Policy	テキストフィールド	87 ページ「Map Policy」の説明を参照してください。

リンク

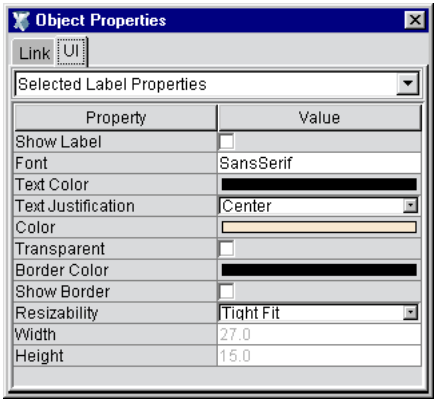
リンクの［Object Properties］パネルには ［Link］および［UI］の 2 つのタブがあります。

［Link］タブ



プロパティ	コントロール	使用方法
Link Name	テキストフィールド	リンクの名前。この名前は、結合条件式でも使用されています。
Source Result	ドロップダウン	リンクのソースアクティビティを指定します。
Condition	テキストフィールド	リンクの XPath 条件を指定します。

リンクの［UI］タブ



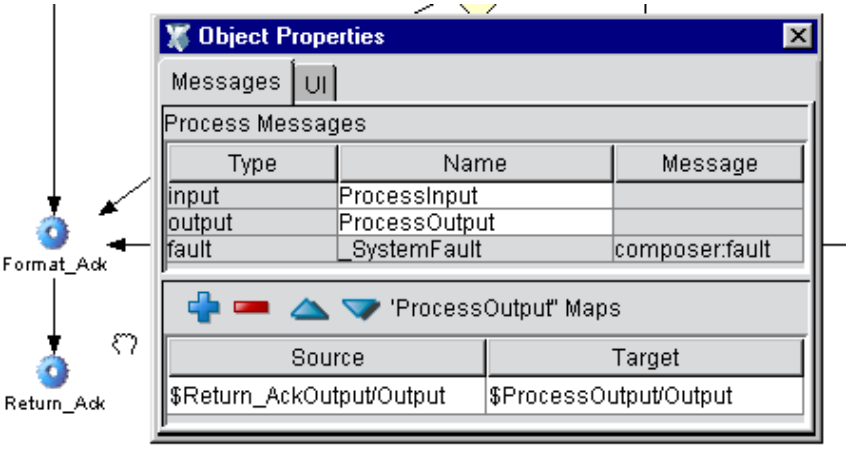
プロパティ	コントロール	使用方法
Show Label	チェックボックス	リンク名の表示レベルを切り替えます。
Font	テキストフィールド	このフィールドをクリックするとダイアログボックスが表示されます。ダイアログボックスでは、さまざまなフォントプロパティを設定できます。
Text Color	カラーピッカー	リンクに関連付けられたテキスト (リンク名) の色を設定できます。
Text Justification	ドロップダウンメニュー	Center、Left Justify、Right Justify
Transparent	チェックボックス	リンクの透明度のオンとオフを切り替えます。
Border Color	カラーピッカー	リンクのアウトラインの色を選択できます。
Show Border	チェックボックス	境界線(表示または非表示)を切り替えます。
Resizability	ドロップダウンメニュー	さまざまなリンク作成ポリシーを指定できます。
Width	テキストフィールド	リンク全体の幅を指定できます。
Height	テキストフィールド	リンク全体の高さを指定できます。

グラフの［Object Properties］

プロセスオブジェクト (またはグラフ) のプロパティシートには、［Messages］タブと［UI］タブがあります。グラフのプロパティを表示するには、キャンバスの何も表示されていない任意の場所をクリックして、［Object Properties］パレットを表示します (［**View**］メニューの［**Object Properties**］コマンドを使用します)。このウィンドウを使用して、プロセス全体の入力、出力、および障害のメッセージマップを設定し、グラフの表示をカスタマイズします。

［Process Messages］タブ

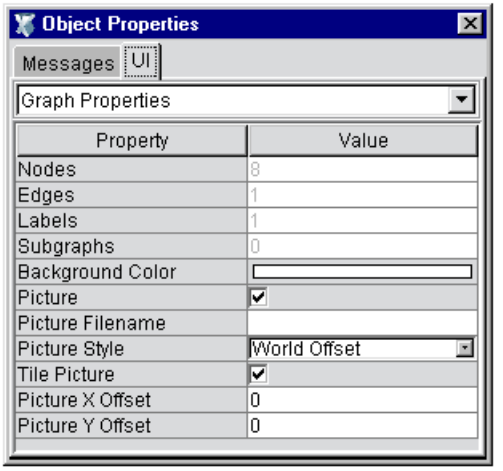
［Messages］タブでは、通常、出力データマップを処理する終了アクティビティを指定します。たとえば、グラフの Activity4 がプロセスの「終了アクティビティ」(実行する最後のアクティビティ) で、Activity4 の出力を含むメッセージを返すプロセスが必要な場合、このタブで ProcessOutput マップを指定します。次に例を示します。



上の例では、アクティビティ「*Return_Ack*」がこのプロセスの終了アクティビティになります。［Object Properties］ペインの下部に表示されているように、出力は `$ProcessOutput/Output` にマップされます。このペインを設定するには、最初にキャンバスの何も表示されていない場所をクリックします (これにより、すべてのアクティビティ、リンク、および他のグラフィック要素の選択が解除されます)。これにより、［Object Properties］ペインに「プロセス全体」のプロパティが反映されます (入力メッセージおよび出力メッセージは「*ProcessInput*」と「*ProcessOutput*」です)。

グラフの［UI］タブ

グラフの［UI］タブを使用すると、グラフ全体で表示についてカスタム設定を定義したり、グラフ上のノード数、リンクの数、ラベルの数に関する集計情報を提供したりできます。



キャンバスの何も表示されていない場所をクリックした場合に限り、プロパティセットにアクセスできます。

注記： グラフの表示をカスタマイズする方法の詳細については、次の「Layout Properties」の章を参照してください。

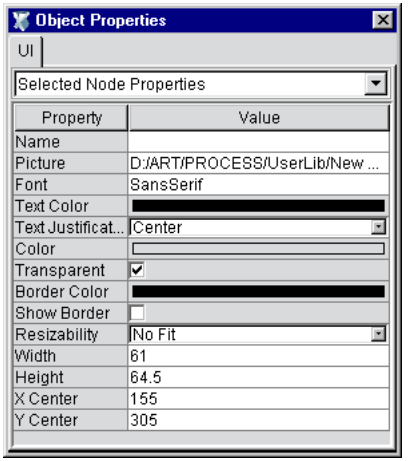
次の表で、プロセスの［Object Properties］パネルの［UI］で使用可能なプロパティについて説明します。

属性	値	説明
Nodes	0 ～ n	このフィールドは編集できません。グラフのノード数を提供します。
Edges	0 ～ n	このフィールドは編集できません。グラフの辺の数値を提供します。
Labels	0 ～ n	このフィールドは編集できません。グラフのラベルの数を提供します。
Subgraphs	0 ～ n	このフィールドは編集できません。グラフのサブグラフの数を提供します。

Background Color	色自体を表示します。デフォルトは白です。	これは、バックグラウンドの色です。
Picture	チェックボックス	このチェックボックスをオンにすると、グラフに画像を表示します。オフにすると表示されません。
Picture Filename	画像のファイル名	画像ファイル完全パス名。画像ファイルは JPEG または GIF です。
Picture Style	「World Offset」または「Device Offset」	「World Offset」を選択すると、図の真中に画像を表示します。「Device Offset」を選択すると、［Picture X Offset］と［Picture Y Offset］で定義したオフセットに画像を表示します。
Tile Picture	チェックボックス	このチェックボックスをオンにすると、画像を並べて表示します。オフにするとオフセットに画像を表示します。
Picture X Offset	0.0	画面に表示する画像の X オフセットを変更するために使用されます。
Picture Y Offset	0.0	画面に表示する画像の Y オフセットを変更するために使用されます。

［UI］タブの［Selected Node Properties］

［UI］タブの［Selected Node Properties］では、グラフに表示されているオブジェクトの「表示属性」を検査および設定します。オブジェクトをシングルクリックして選択してから、［Object Properties］ペインの［UI］タブをクリックします。次に、すタブの上部のドロップダウンメニューコントロールから「dropdown menu」を選択します。次の図を参照してください。



［UI］タブ (［Selected Node Properties］)

属性	値	説明
Name	Activity1 ～ Activityn (デフォルト)。	これはアクティビティの名前です。デフォルトは Activity1 ～ Activityn になります。
Font	ダイアログ • SanSerif (デフォルト) • Serif • MonoSpaced • DialogInput	［Value］フィールドをクリックすると、［Choose Font］ダイアログボックスが表示されます。このダイアログボックスには、フォント、フォントスタイル (標準、太字、斜体、または太字の斜体)、フォントサイズを選択する 3 つのドロップダウンがあります。
Text Color	色自体を表示します。デフォルトは黒です。	［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。
Text Justification	左 中央 (デフォルト) 右	このフィールドはドロップダウンです。
Color	色自体を表示します。デフォルトは黄色です。	これは、バックグラウンド色です。［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。

Transparent	チェックボックス	このチェックボックスをオンにすると透明になり、オフにすると不透明になります。
Border Color	色自体を表示します。デフォルトは黒です。	［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。
Show Border	チェックボックス	このチェックボックスをオンにすると境界線が表示され、オフにすると表示されません。
Resizability	ドロップダウン • No Fit • Tight Fit • Tight Width • Tight Height • Tight Fit Preserve Aspect • Preserve Aspect	
Width	テキストフィールド	デフォルトは 40.0 です。
Height	テキストフィールド	デフォルトは 40.0 です。
X Center	テキストフィールド	X 座標。
Y Center	テキストフィールド	Y 座標。

テキストの［Object Properties］

［Text］オブジェクト、［Shapes］などの［UI］タブには、上で説明したものと同じような属性を持つ［Selected Node Properties］ペインがあります。次の表で、プロパティに関する詳細を説明します。

［UI］タブ

属性	値	説明
Name	Untitled	テキストオブジェクトの名前。これは、テキスト / キャプション / ラベルそのものと同じ名前です。
Margin Width	3.0 (デフォルト)	これは、テキストの左右の余白幅です。
Margin Height	1.0 (デフォルト)	これは、テキストの上下の余白幅です。
Font	ダイアログ • SanSerif (デフォルト) • Serif • MonoSpaced • DialogInput	［Value］フィールドをクリックすると、［Choose Font］ダイアログボックスが表示されます。このダイアログボックスには、フォント、フォントスタイル (プレーン、ボールド、イタリック、またはボールドイタリック) 、フォントサイズを選択する 3 つのドロップダウンがあります。
Text Color	色自体を表示します。デフォルトは黒です。	［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。
Text Justification	左 センター (デフォルト) 右	このフィールドはドロップダウンです。
Color	色自体を表示します。デフォルトは白です。	これは、バックグラウンド色です。［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。
Transparent	チェックボックス	このチェックボックスをオンにすると透明になり、オフにすると不透明になります。
Border Color	色自体を表示します。デフォルトは黒です。	［Value］フィールドをクリックすると、［Choose Color］ダイアログボックスが表示されます。
Show Border	チェックボックス	このチェックボックスをオンにすると境界線が表示され、オフにすると表示されません。

Resizability	ドロップダウン ◆ No Fit ◆ Tight Fit ◆ Tight Width ◆ Tight Height ◆ Tight Fit & Preserve Aspect ◆ Preserve Aspect	
Width	テキストフィールド	デフォルト値は 48.0 です。このフィールドは有効になっていません。テキストがデフォルトの「untitled」から変更されたり、余白の幅とフォントが変更されたりすると、幅が変更します。
Height	テキストフィールド	デフォルト値は 19.0 です。このフィールドは有効になっていません。余白の高さとフォントが変更されると、高さが変更します。
X Center	テキストフィールド	X 座標。
Y Center	テキストフィールド	Y 座標。

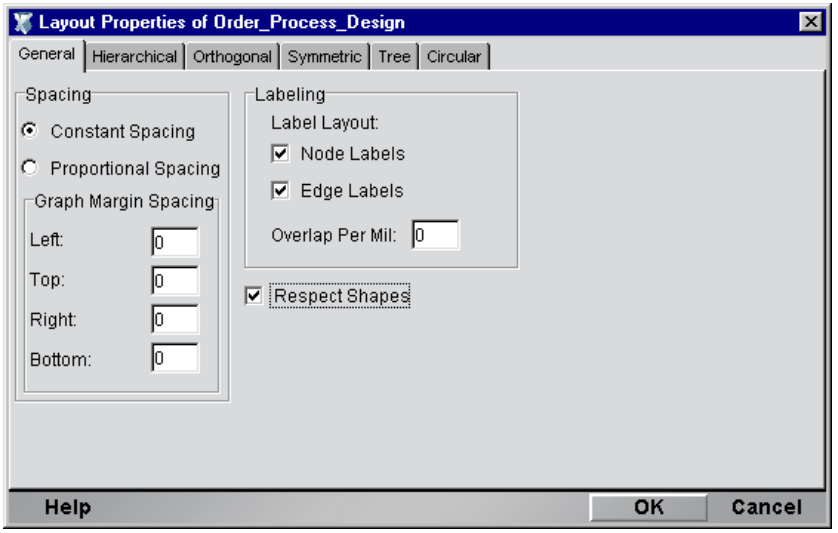
Layout Properties

(必要に応じて)Process Designer は 5 つのフロー図アルゴリズムのいずれかに従って、グラフを自動的に再フォーマットします。

- ［**Circular**］— できるだけ、同じ長さのスポークを使用して、ハブとスポークのようにノードを配置します。このレイアウトのタイプは、［*clustering*］が有力なアークテクチャの機能である場合 (たとえば、LAN または WAN レイアウトで説明したような場合) に適切です。
- ［**Hierarchical**］— トップダウンの関係を強調する、従来の作図スタイルである「組織図」(ただし、このレイアウトオプションを設定すると、左から右、または右から左に表示されます)。このレイアウトオプションは、グラフ内で「階層関係」が強調される必要がある場合に適しています。
- ［**Orthogonal**］— このスタイルは、ノードを制限して行と列のモチーフにリンクします。リンクは X 軸および Y 軸に平行になるように制限されています。また、複数の着信リンクを持ったノードは、他のノードに関連する概観で拡大します。このレイアウト方法は、階層関係ではなくて、要素間の［*grid*］または［*lattice*］の関係を強調する場合に適しています。

- ［**Symmetric**］— 「対称的な関係」が強調されるように、縁の重なりが最小化され、ノードの配置が統一されます。
- ［**Tree**］— この作図スタイルは、親 / 子関係を示したい場合に適してします。これは従来の「ファミリツリー」レイアウトを使用します。このレイアウトは上で説明した階層スタイルとよく似ていますが、リンクが並列ではないので完全に水平または垂直になることはほとんどありません。

上記のすべての作図スタイルは、［**Layout**］>［**Properties**］で表示した設定を使用して、大幅にカスタマイズされる可能性があります。このダイアログボックスを表示するには、Composer のメインメニューバーで、［**Layout**］メニューの［**Properties**］の順にクリックします。



このダイアログボックスには、次の 6 つのタブがあります。［General Preferences］タブと、これまでに説明した 5 つの自由レイアウトスタイルに対応した 5 つのタブです。たくさんのコントロールと設定があるそれぞれのタブを使用すると、特定のグラフスタイルを特徴づける多くの制約の中で、細かいコントロールを実行できます。

一般的なレイアウトのヒント

次のヒントは、Process Designer を使用した生産性を最大限引き出す場合を想定しています。

「スナップ」および「グリッド」動作

- デフォルトでは、非表示の 5 ピクセル*5 ピクセルのグリッドにスナップを書いたり置いたりします。ただし、<Alt> キーを押しているときは、この動作が上書きされます (［**View**］、［**Grid Style**］、［**Grid Size**］、［**Custom Size**］の順にクリックして、［Grid Size］を 1 に設定すると、この動作は完全に上書きされます)。
- ［**View**］メニューの［**Hide Grid**］コマンドまたは［**Show Grid**］コマンドを使用すると、グリッドの表示レベルを切り替えることができます (コマンドは実際には 1 つしかありません。コマンドの名前は、入力したモードによって動的に変化します)。［View］メニューコマンドを使用すれば、いつでも［Grid Size］と［Style］(ドットまたはライン) を設定できます。
- ［**View**］>［**Snap to Grid**］を使用すると、即座にグラフノードをグリッドに整列させることができます。グラフアイテムは、最も近いグリッドラインに突然「ジャンプ」します。

［Undo］の連続使用

［Undo］または［Redo］の連続使用は、すべてのレイアウト操作で実行できます。

Sticky Tools

通常、ツールは一度使用すると矢印カーソルに戻ります。たとえば、［Activity］ツールを選択し、キャンバスをクリックして新しいアクティビティアイコンを配置した場合、マウスを放すとツールはすぐに矢印 (もしくは選択ツール) に戻ります。［Sticky Tools］オプションを使用すると、この動作を上書きして、ツールモードがマウスのクリックを持続させます。**Sticky Tools** のプロセスメニューを参照してください。

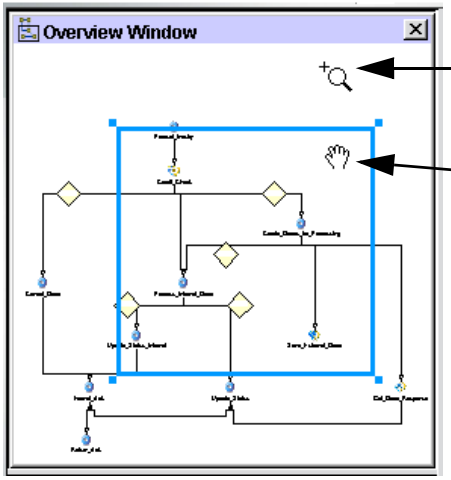
Overview Window

［Overview Window］を使用すると、操作とズームに対する例外的なコントロールを持つことができます (次を参照)。［**View**］、［**Overview Window**］の順にクリックして、このペインの表示レベルを切り替えます。

Overview Window から 2 つの動作を使用できます。

- 青いビューポートを Overview Window にドラッグして、リアルタイムでキャンバスを操作できます。
- ビューポートの外側をクリックアンドドラッグして、インタラクティブに、キャンバスを大きいサイズまたは小さいサイズにズームします。

カーソルは、マウスの位置 (ビューポートの内外) に従って、表示を変更します。



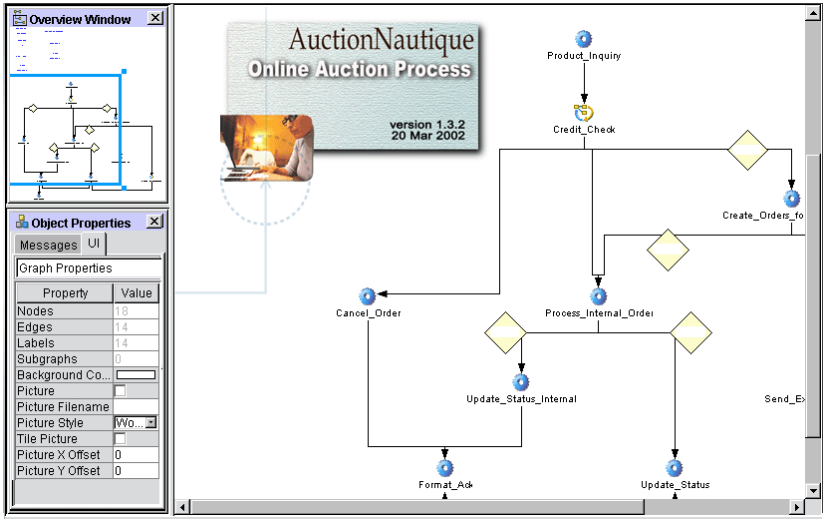
キャンバスのカスタマイズ

さまざまな方法で、キャンバスをカスタマイズできます。たとえば、バックグラウンドイメージを指定したり、カスタムイメージを使用するために任意のアクティビティの表示を変更したり、いくつかの装飾用のイメージまたはロゴをキャンバスに追加したり、[Send to Back] や [Bring to Front] を使用して、任意の順序でイメージを「スタック」したりできます。これらの機能を使用すると、会議やデモンストレーションなどで使用するために、プレゼンテーション用のプロセスグラフを作成することができます。

注記： キャンバスのプロパティにアクセスするには、キャンバスの任意の場所をクリックしてから、[Object Properties] パネルの [UI] タブを選択します。

カスタムバックグラウンドの使用

キャンバスのカスタマイズに、「**.gif**」イメージまたは「**.jpg**」イメージを構成するカスタムバックグラウンドを追加する方法もあります。「**.jpg**」バックグラウンドを含むキャンバスは、次の図のとおりです。



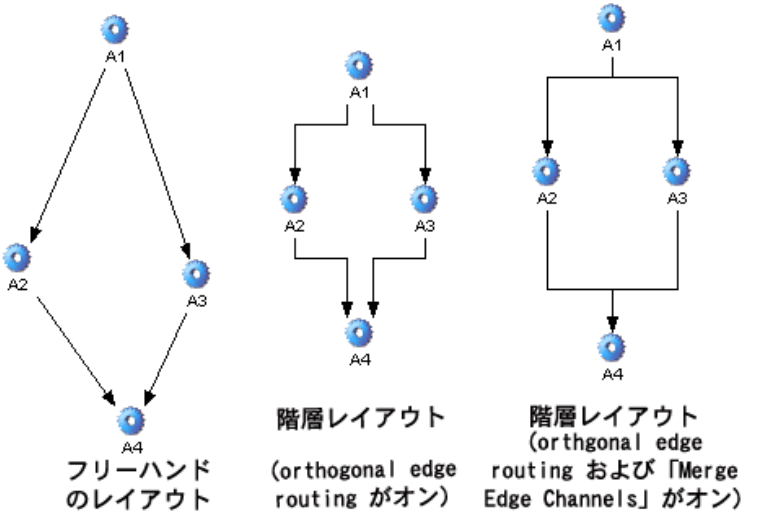
➢ キャンバスに画像を追加する

1. キャンバスで何も表示されていない任意の場所をクリックします。
2. ［**Object Properties**］パネルをビューに切り替えます。
3. ［**UI Tab**］を選択します。
4. ［**Picture Filename**］の右側の白い領域をクリックします。ナビゲーションダイアログボックスが表示されます。
5. ハードディスクまたはネットワークを移動して、グラフのバックグラウンドの画像として使用する **.jpg** ファイルまたは **.gif** ファイルを検索します。
6. ［UI］タブでは、［**Picture**］チェックボックスをオンにして、イメージをキャンバスに適用します。
7. オプションで、イメージを持ったキャンバスを並べて表示する場合は、［**Tile Picture**］チェックボックスをオンにします。
8. ［**Picture Style**］の隣にドロップダウンメニューがあります。このメニューで使用できる 2 つの選択肢のいずれかを選択します。
   - ［**World Offset**］— このオプションを選択すると、イメージは、異なるズーム設定を選択するとキャンバスが増減され、キャンバス内の他のオブジェクトとの相対的な位置を維持します。これは、すべての Process Designer グラフィックで通常の動作です。
   - ［**Device Offset**］— このオプションを選択すると、イメージを移動またはズームしても、イメージの位置が増減したり変更したりすることはありません。

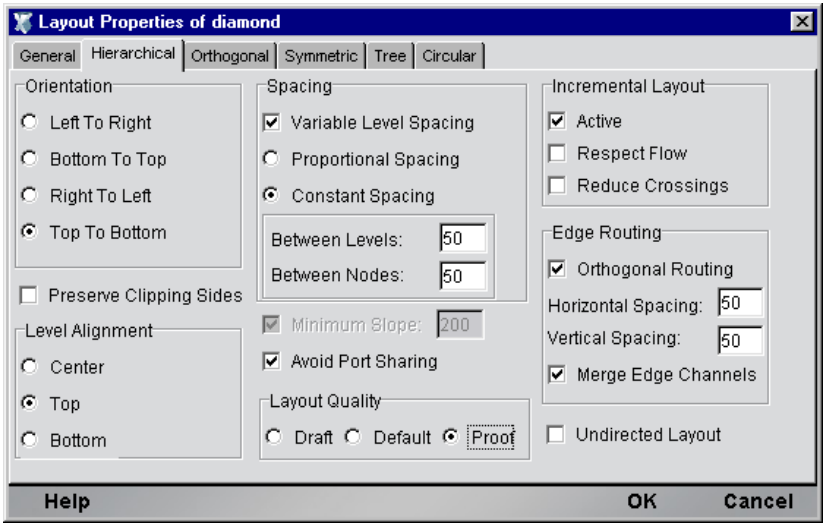
9 オプションで、［**Picture X-Offset**］値または［**Picture Y-Offset**］値、あるいはその両方の値を調整して、グラフ上で配置したい場所に正確に画像を配置します (必要に応じて、正の値または負の値を入力します)。

自動レイアウトオプション

これまでに説明したように、Process Designer は、必要に応じて、さまざまな作図アルゴリズムに従ってグラフを再フォーマットします。使用する可能性が最も高い自動作図オプションは、［Hierarchical］レイアウトオプションです。このオプション (［**Layout**］、［**Hierarchical Layout**］の順にクリック) は、グラフをトップダウン (もしくは左から右へ) の階層ビューに再フォーマットします (そのとき、リンクの X/Y 位置、および並列リンクのマージはあり、またはない状態でフォーマットされます)。



制約に関するさまざまなオプションが［Hierarchical Layout］で使用できます (他の自動レイアウトモードでも使用できます)。この設定にアクセスするには、［**Layout**］メニューの［**Properties**］コマンドを使用します。これにより、［Layout Properties］ダイアログボックスが表示されます。



右下の［Edge Routing］コントロールグループに特に注意してください。X 軸と Y 軸を調整する場合は、［**Orthogonal Routing**］チェックボックスをオンにする必要があります。シングルステムとして表された (共通ノードの内外に送られてくる) パラレルリンクのステムが必要な場合は、［**Merge Edge Channels**］チェックボックスをオンにする必要があります。

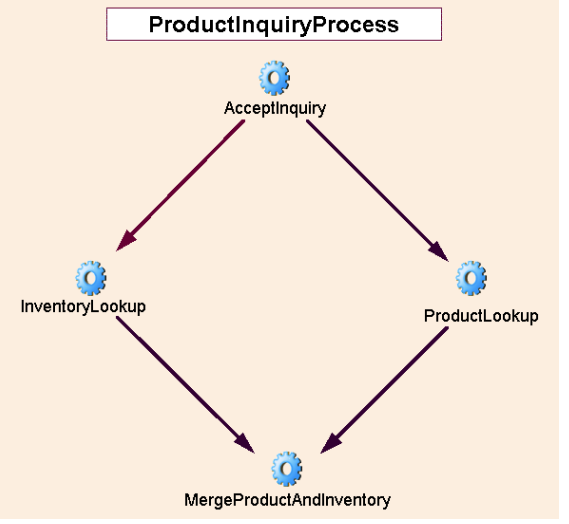
4 プロセスの作成とテスト

この章では、プロセスの作成、データマップの指定、リンクへの論理の適用、結合の制御、設計時環境でのプロセスのアニメーション表示 (ステップスルーまたは実行) の方法について説明します。

Process Designer を初めて使用する場合は、最初のプロセスを作成する前にこの章を必ずお読みください。

例：単純なストレートスルー処理

Composer Process Designer を使用していかに早くプロセスモデルを作成しテストできるかを確認するために、次の図に示されている単純なストレートスルー処理の作成を 1 つ 1 つ行います。



説明

ProductInquiryProcess モデルは、製品情報の要求を処理します。プロセスへの入力は、SKU (製品 ID) 番号を含む XML DOM です。プロセスは該当する製品に関する詳細な情報を含む XML DOM を出力します。必要な製品情報は、JDBC を通じて 2 つのソース (2 つのデータベース) から取り出されます。

プロセスでは 4 つのアクティビティを使用し、それらはすべて Composer コンポーネントです。アクティビティの役割と責任は次のとおりです。

AcceptInquiry (XML Map コンポーネント) — SKU 情報を含む入力 DOM を取り込み、単純にその情報をトラッキング番号とともに出力 DOM に書き込みます。

InventoryLookup (JDBC コンポーネント) — *AcceptInquiry* の出力を使用して、このコンポーネントは在庫管理システムに対してデータベースの検索を実行し、SKU 番号が渡された製品のカテゴリおよびステータス情報を取得します。

ProductLookup (JDBC コンポーネント) — *AcceptInquiry* の出力を使用して、このコンポーネントはマーケティングデータベースに対してデータベースの検索を実行し、価格、色、説明などの製品情報を取得します。

MergeProductAndInventory (XML Map コンポーネント) — このコンポーネントは、2 つの JDBC コンポーネントから受け取ったデータをマージします。出力はプロセス出力全体を構成します。

注記： Composer チュートリアルについて理解している場合、上記のコンポーネント (このプロセスに特有の AcceptInquiry を除く) は Composer チュートリアルで使用されているものと同じです。

プロセス作成の基礎

Process Designer をよく理解するまでは、まず次に示すようなプロセスモデルを作成されることをお勧めします。

➢ プロセスモデルを作成する

1. 新規の空白のプロセスグラフを作成します。
2. すべてのアクティビティアイコンを作成し、配置します。
3. リンクでアクティビティを接続します。
4. アクティビティの間にメッセージマッピングを作成します。
5. プロセスの種々の点で適用される「リンク条件」を指定します。
6. 適用される任意の「終了条件」を指定します。
7. 適用される任意の「結合条件」を指定します。

8 プロセスモデルのアクティビティに適用されるその他の属性(タイムアウト/再試行の値、マップポリシーなど) を設定します。

9 すべての独立したアクティビティの「実装」(すなわち、アクティビティの実行プログラムを構成する基礎の Composer コンポーネント、Web サービス、またはサブプロセス) の作成、テスト、およびデバッグをまだ行っていない場合は、行います。

10 最後に、モデルをテストします。

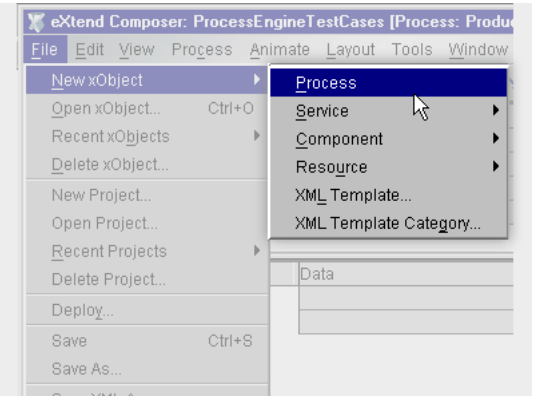
新しいプロセスの作成

Composer コンポーネントおよびサービスを以前に作成している場合、新しいプロセスの作成手順には類似した点が多くあります。

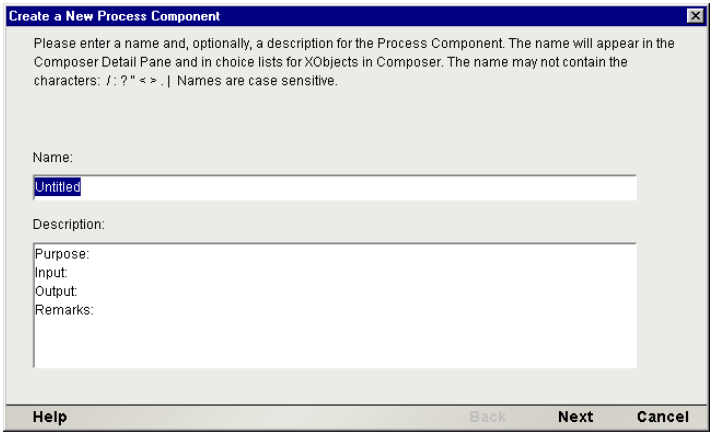
➢ 新しいプロセスを作成する

1 Composer が起動していない場合は起動します。

2 [**File**] メニューから、[**New xObject**]、[**Process**] の順に選択します (次の図を参照)。

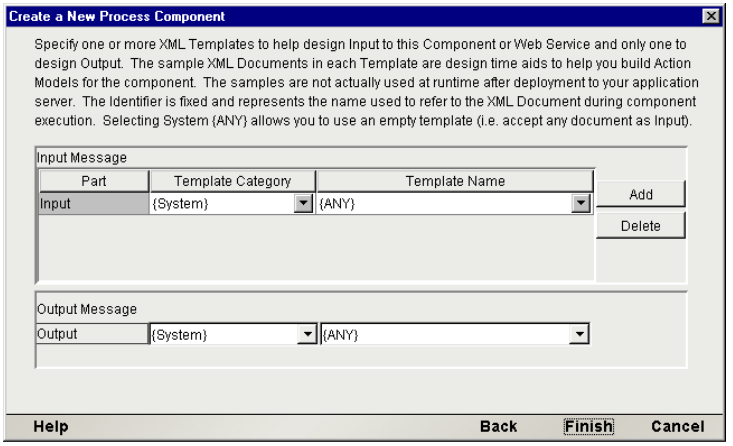


ダイアログボックスが表示され、プロセス名を入力するように指示されます。



3 ［**Name**］にプロセス名を入力します。オプションで、このプロセスに関連付ける追加の説明文を入力します。

4 ［**Next**］をクリックします。新しいダイアログボックスが表示されます。

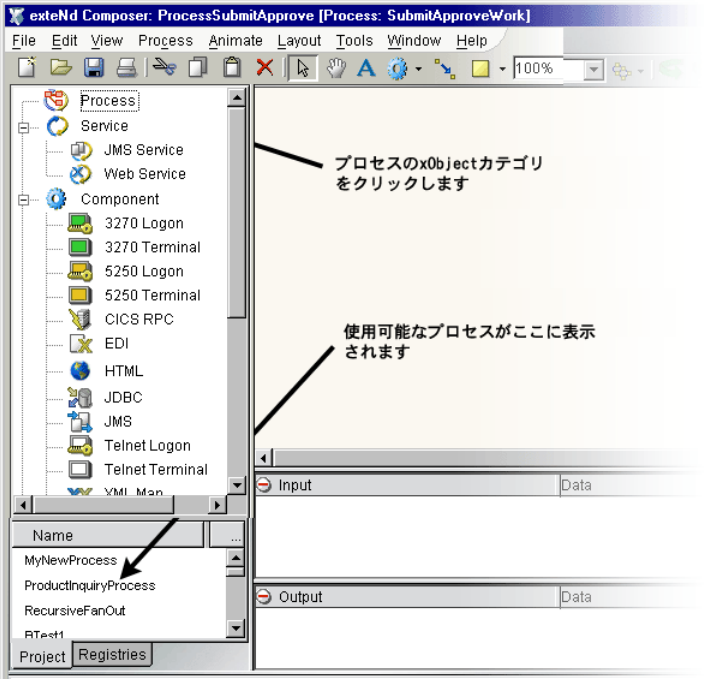


5 プロセスの設計時テストの補助として、プロセスの入出力に使用する「XML テンプレート」を追加します (サーバにプロセスを配備した後は、これらのドキュメントは実際に使用されません。設計時の補助にのみ使用されます)。

注記： XML テンプレートは、Composer のほぼすべてのコンポーネントタイプで使用されます。テンプレートの作成および使用に慣れていない場合は、『Composer ユーザガイド』の XML テンプレートに関する章を参照してください。

通常、ここで指定するテンプレートはプロセスのアクティビティ (Composer コンポーネントの開始アクティビティと想定します) を開始するために使用されるものと同一です。この場合、プロセスにサンプル入力データを提供する能力のある 1 つまたは複数のテンプレートが必要になります。

6 ［**Finish**］をクリックします。Process Designer のウィンドウが空のキャンバスで開きます (次の図を参照)。

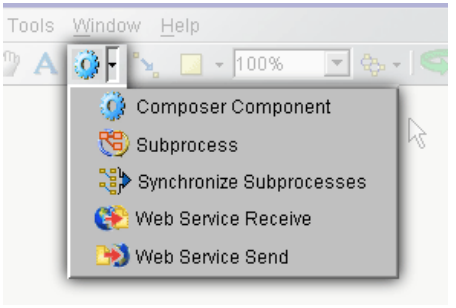


アクティビティの作成

アクティビティの実装 (Composer コンポーネント、サブプロセスなど) がまだ作成されていない場合でも、いつでもアクティビティアイコンの適用を開始できます。この場合、アクティビティが事前に作成された Composer コンポーネントから成り立っていると想定します。

➢ アクティビティを作成する

1 ツールバーから該当する［**Activity Tool**］を選択します。フライアウトアイコンリストを確認するために、現在のアクティビティツールの隣の小さな三角形をクリックします。



2 キャンバスを「**クリック**」します。新しいアクティビティが表示されます。

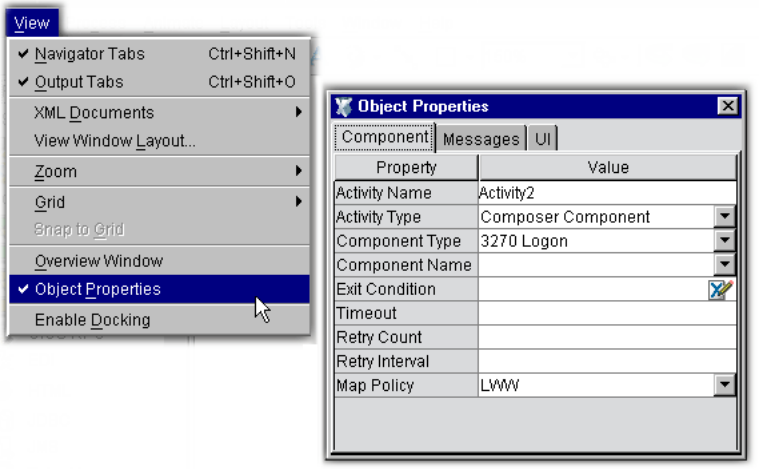
注記： キャンバス上でアクティビティアイコンの位置を変更するには、最初に［**Pointer Tool**］を選択し、アクティビティアイコンをクリックしてドラッグします。Composer の［View］メニューの下の［Grid］サブメニューを使用して、グリッドに吸着する動作を制御できます。

3 必要に応じて、このステップを繰り返し、プロセスモデルの追加アクティビティを作成します。

➢ アクティビティに実装を関連付ける

1 アイコンで 1 度クリックして、アクティビティを「**選択**」します。

2 Composer の［View］メニューの下で、［**Object Properties**］を選択して、［Object Properties］パネルを表示します。次の図を参照してください。



3 ［Object Properties］パネルの［**Activity**］**タブ** (またはアクティビティの種類に応じて、適宜、［Subprocess］タブなど) を選択します。

4 表示されているタイプが該当するものでない場合、指定されたプルダウンメニューから適切な「**アクティビティタイプ**」(Composer コンポーネント、サブプロセスなど) を選択します。

注記： この議論のためには、アクティビティの実装が Composer コンポーネントであることを前提としています。

5 ［**Component Type**］で、コンポーネントの該当するタイプ (JDBC、XML Map、または適宜、適用されるもの) を選択します。

6 ［**Component Name**］の隣のプルダウンメニューを使用して、前の手順で指定されたコンポーネントタイプと一致する現在のプロジェクトですでに作成されているコンポーネントから選択します。(現在の Composer プロジェクトに 4 つの XML Map コンポーネントがあり、ステップ 4 のコンポーネントタイプとして XML Map を選択している場合、プルダウンメニューに XML Map コンポーネントの名前が表示されます。)

➢ アクティビティの名前を変更する

1 ポインタツールを使用してアクティビティを「**選択**」(クリック) します。サイズ変更ハンドル (小さな青の四角形) が、アイコンがフォーカスをもつことを示すアクティビティアイコンの周りに表示されます。

2 アクティビティの「**名前**」を直接、クリックします。テキスト入力フィールドが表示され、アクティビティ名がハイライトされます。

3 アクティビティの新しい名前を「**入力**」します。

4 アクティビティを選択解除 (フォーカスを削除) するために、端から外れた場所を「クリック」します。

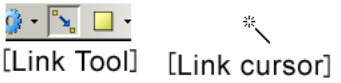
注記： アクティビティには、適用されている実装とは別の名前が付けられています。

リンクの作成

アクティビティがキャンバスで実行されると、制御リンクを通じてそれらを接続することが必要になります。第 1 章で説明したとおり、制御リンクはプロセス実行フローを制御します。データフローについては「メッセージのマップ」の章で詳しく説明します。

➢ リンクを作成する

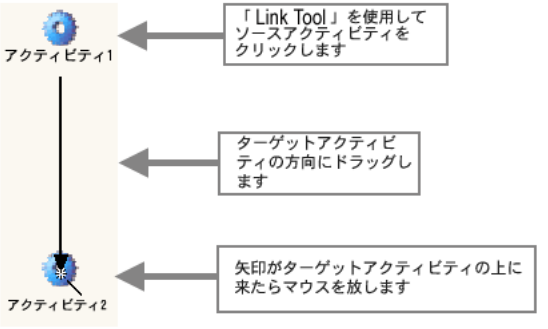
1 Process Designer ツールバーの［**Link Tool**］を選択します。



2 アクティビティを「**クリック**」します。リンクの「ソース」としてアクティビティが指定されます。

3 マウスボタンを離さずに、カーソルをソースアクティビティから「ターゲット」アクティビティに指定する任意のアクティビティまで「**ドラッグ**」します。キャンバス上をドラッグして回ると、リンクの矢印はマウスを追跡しながら「ゴムバンド」のように伸縮します。

4 カーソルをターゲットアクティビティの上に直接置き、マウスボタンを「**放し**」ます。リンクのカラーは変更され、すぐに再描画されて、2 つのアクティビティを囲むボックスの間の接続が示されます。



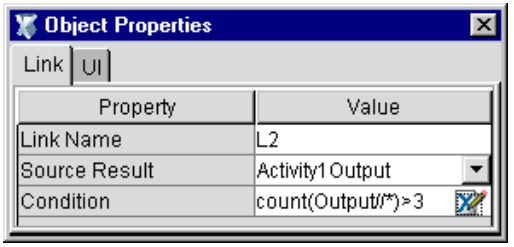
トランザクションロジックのリンク

リンク条件は XPath で指定され、ソースアクティビティの出力メッセージ構造の知識を必要とするため、すべてのデータマッピングが指定されるまで、通常はリンク条件をしないことが最適です (後の説明を参照)。しかし、ソースとターゲットアクティビティの間のデータの関係をすでに理解している場合は、いつでもリンク条件を指定することができます。

➢ リンク条件を指定するため

1 リンクをクリックして「**選択**」します。

2 ［**Object Properties**］パネルが表示されていない場合は、それを表示します。(Composer の［View］メニューの下の［Object Properties］メニューを使用して表示します。)

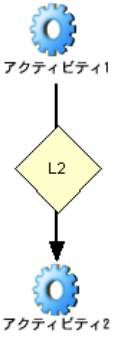


3 ［**Link**］タブをまだ選択していない場合は、それを選択します。

4 ［**Condition**］の隣で、ブール値を評価する XPath 式を入力します。

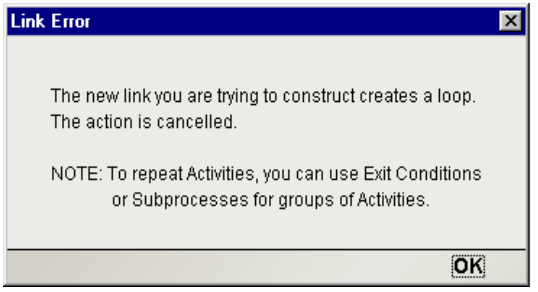
注記： 何も入力しない場合、ランタイムエンジンはリンクの値がデフォルトの *true* であるということを前提とします。

5 希望に応じて、［Object Properties］パネルを閉じます。リンクのオンスクリーン表示は、XPath ロジックがそのリンクと関連付けられていることを示すよう、菱形に変わります。



「間違った方向」を指しているリンク

ターゲットアクティビティを上位にあるソースに接続するリンク (後向きのリンクなど) を描こうとすると、エラーが発生します。



サイクリックグラフパターン (再入可能ループ) は、Composer Process Manager ではサポートされていません。詳細については、第 2 章 (56 ページ「ループ」) を参照してください。

メッセージのマップ

1 つのアクティビティから別のアクティビティへのデータの転送は、「データリンク」を通して行われます。制御リンクと異なり、データリンクは「描画可能」ではありません。プロセスグラフでは視覚的な記号表示がありません。代わりに、データリンクは「メッセージマップ」を通して設定されます。これらのマップは単純に、ソースアクティビティの出力とターゲットアクティビティの入力における XPath 間の相互関係です。言い換えると、通常の Composer XML Map コンポーネントとほぼ同様の方法で定義されます。

メッセージの名前付け

Composer Process Manager は、メッセージのソースおよびターゲットにデフォルトの名前付けスキームを使用します。最初のアクティビティを新しいキャンバスに配置すると、Process Manager はアクティビティに *Activity1* のデフォルト名を割り当てます (その後のアクティビティは、*Activity2*、*Activity3* などという名前が付けられます)。したがって、Process Manager は *Activity1Input* のデフォルト名を *Activity1* の入力メッセージに割り当て、名前 *Activity1Output* をアクティビティの出力メッセージに割り当てます。後で *Activity1* の名前を *CodeRedFireAlarm* に変更した場合でも、その入力および出力の「メッセージ」の名前は、手動で変更しない限り変更することはありません (次の手順を参照)。引き続き *Activity1Input* および *Activity1Output* のデフォルト名を持ちます。

DOM はメッセージに関連付けられ、DOM 名 (Input、Input1、Temp、Output など) は、メッセージ名から参照されます。その後、通常の XPath ルールが適用されます。例 :

```
Activity1Output/Output/PRODUCTREQUEST/SKU
```

は、*Activity1Output* という名前のメッセージの出力 DOM 上の XPath ノード /PRODUCTREQUEST/SKU を意味します。その後の例とスクリーンショットでこれがどのように動作するかを確認できます。

メッセージのマップの定義方法

ソースアクティビティからターゲットアクティビティにデータを送信するには、少なくとも 1 つのメッセージマップを定義する必要があります。

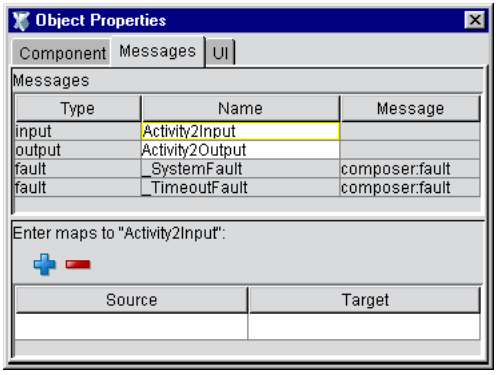
注記 : すべてのメッセージマップは「ターゲットアクティビティ」(受信データの「受信者」) として定義されます。

➢ メッセージのマップを定義する

1 アクティビティを選択 (クリック) します。これは、指定するデータソースをもつアクティビティです。

2 ［**Object Properties**］パネルが表示されていない場合は、それを表示します。(Composer の［View］メニューの下の［Object Properties］メニューを使用して表示します。)

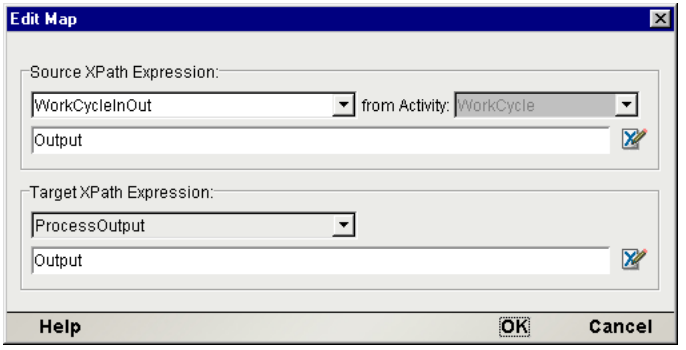
3 ［**Messages**］タブを選択します。タブはこれと同じように表示されています。



4 パネルの上半分には［**Type**］、［**Name**］、および［**Message**］情報が表示され、入力、出力、および障害メッセージのためのデフォルト名が示されています。適宜、この時点でメッセージに新しい名前を入力できます。

注記： 障害メッセージについては、この章の後で説明します。

5 パネルの下半分では、XPath を使用してソースからターゲットへのメッセージのマップを定義できます。［**Plus**］アイコンをクリックして、マップに追加します。ダイアログボックスが表示されます。

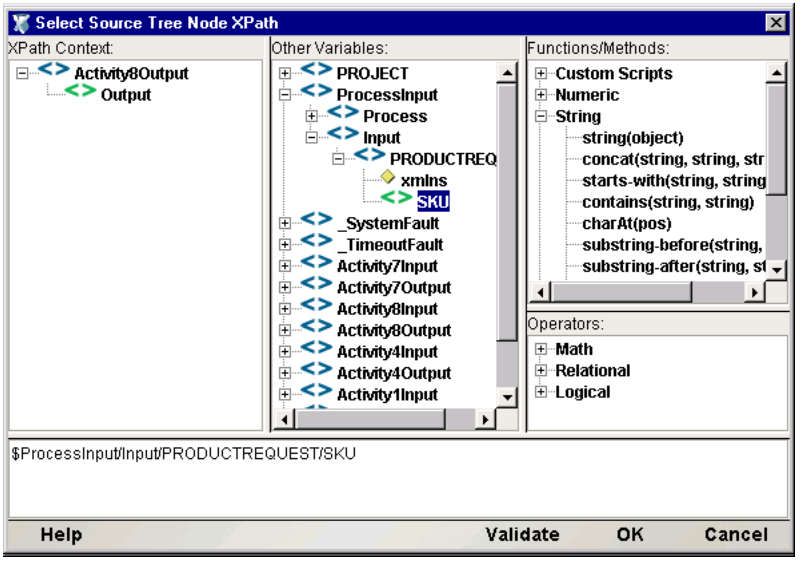


6 ［**Source XPath Expression**］のすぐ下のプルダウンメニューを使用して、このマッピングのデータソースとして使用するメッセージを選択します。追加されたリストは、すべての選択可能な (正当な) データソースからの出力メッセージ名を含みます。(言い換えると、) リンクを逆にたどって到達することができるアクティビティからのデータをマップするように選択できます。これには、*ProcessInput* メッセージ自体が含まれます。)

7 ［**Source XPath Expression**］テキストフィールド領域で、［from Activity］プルダウンメニューで示されているアクティビティからソース要素、ノードツリー、DOM 全体などを指定するために、該当する XPath ステートメントを入力してください (このメニューは、現在のアクティビティに 1 つの着信リンクしかない場合にはグレイ表示されます。ターゲット結合時にこのメニューにはすべての選択可能な着信メッセージの名前が追加されます)。

注記： ここで示されているように、単独の着信メッセージ部分として「出力」(ソースアクティビティの出力 DOM) を指定することは一般的です。

8 (オプション) Composer の Expression Builder を使用して XPath を生成する場合、テキストフィールドの右側の「鉛筆と X」のアイコンをクリックしてください。これにより、XPath Expression Builder ウィンドウが呼び出されます。

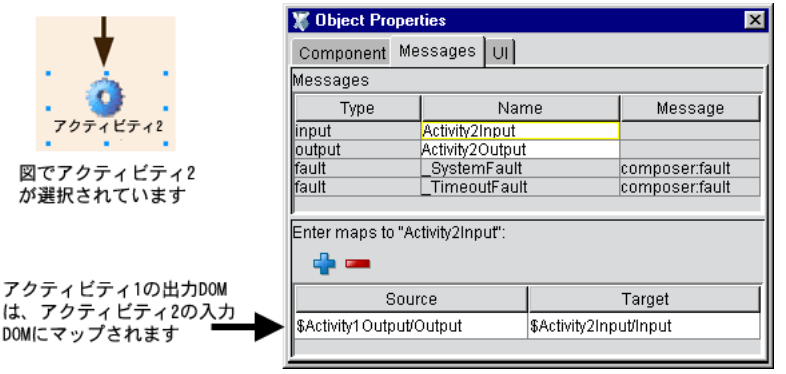


このエディタウィンドウの上のペインはあらかじめメッセージツリー、XPath ネイティブスクリプトメソッドなどが XPath 式の作成のために便宜上、追加されています。ツリーの任意のノードをダブルクリックして、正しい従属式が表示されるようにします。［OK］をクリックして、［Edit Map］ダイアログボックスに戻ります。

9 ［**Target XPath Expression**］テキストフィールド領域に、着信メッセージからデータを受信するターゲットを指定するために任意の該当する XPath ステートメントを入力します。

注記： ターゲットメッセージ部分として、「Input」(ターゲットアクティビティの入力 DOM) を指定するのが一般的です。これは、アクティビティの入力 DOM にソースデータをマップするのと同じです。

10 ［**OK**］をクリックして、ダイアログボックスを閉じます。



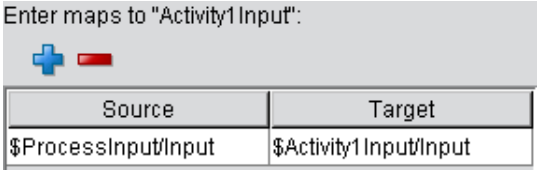
指定したばかりのソースおよびターゲットマップ情報は、上に示すとおり［Object Properties］ウィンドウの［Messages］タブで表示されます。(情報のサマリ表示は、このフィールドの上にマウスを移動させると、ロールオーバーツールヒントでも選択可能になります。)

この類のマップは、プロセスグラフで示された一連のアクティビティを通じて続行されます。このマップ手順はデータを受信するアクティビティごとに少なくとも一度は実行する必要があります。

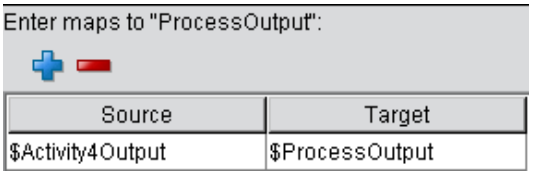
開始および終了アクティビティのためのデータマップ

プロセスの (1 つまたは複数の) 開始アクティビティに入力を指定するためには、開始アクティビティをクリックし、［Object Properties］パネルを表示し、上に示した手順を使用して［*ProcessInput*］から開始アクティビティへののマップを指定するだけです。

開始アクティビティが Activity1 という名前の場合、結果のマップ指定は次のようになります。



「終了」アクティビティから *ProcessOutput* へのマップを指定するには、生のキャンバスの任意の場所をクリックし、［Object Properties］パネルを表示し、終了アクティビティの出力メッセージから *ProcessOutput* メッセージへのマップを指定します。結果は次のようになります。



プロセス入力テンプレートの選択

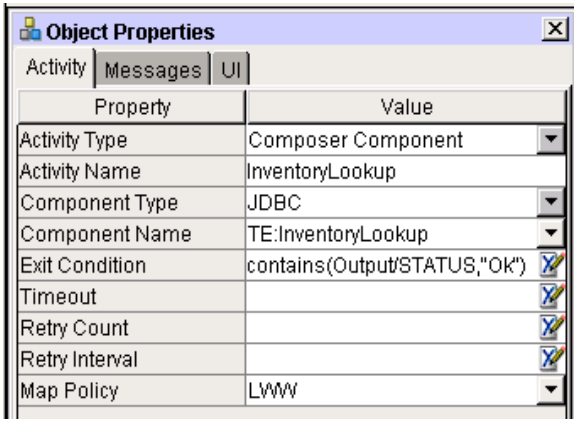
前に説明したように (新しいプロセスの作成方法の説明)、プロセスを初めて作成するときに、*ProcessInput* (設計時テスティングの目的) の入力テンプレートドキュメントを指定できます。その時、XML テンプレートを指定しない場合、あるいは別のテンプレートを使用する場合は、［File］メニューに進み、［**Properties...**］コマンドを選択します。ダイアログボックスが表示されます。そのダイアログ内の［**Messages**］タブを選択します。そこで、適宜、テンプレートを追加または削除できます。

アクティビティレベルでのフローロジックの適用

アクティビティフローロジック (結合条件および終了条件) を［Object Properties］パネルで指定できます。この条件はオプションです。デフォルトで、ランタイムエンジンは、空の条件が *true* であることを前提とします。

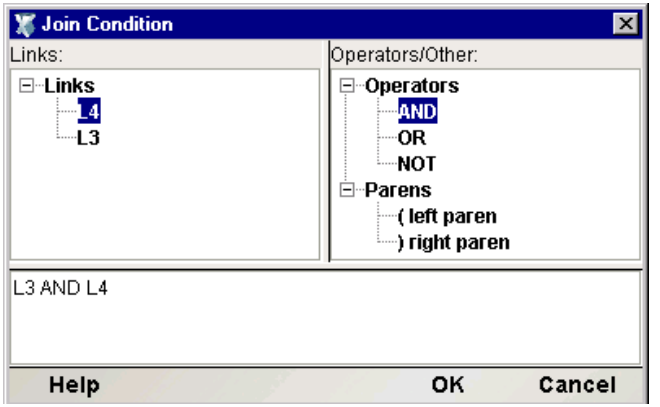
➢ **終了条件を指定する**

1. アクティビティを「**選択**」(クリック) します。
2. ［**Object Properties**］パネルが表示されていない場合は、それを表示します (Composer の［View］メニューの下の［Object Properties］メニューを使用して表示します)。
3. ［**Activity**］**タブ** (または、選択したアクティビティにより適宜、［Subprocess］など) を選択します。
4. ［**Exit Condition**］の隣で、XPath 式を入力します。この条件はランタイム時に *true* または *false* に評価されなければなりません。false に評価された場合、アクティビティは元の入力データを使用してもう一度実行されます (第 2 章の説明を参照)。このアクティビティは、終了条件が true であるか、タイムアウトが起こるまで再実行し続けます。



➢ 結合条件を指定する

1. 結合アクティビティを「**選択**」(クリック) します。つまり、複数の着信リンクをもつアクティビティです。
2. ［**Object Properties**］パネルが表示されていない場合は、それを表示します (Composer の［View］メニューの下の［Object Properties］メニューを使用して表示します)。
3. ［**Activity**］***タブ*** (または、選択したアクティビティにより適宜、［Subprocess］など) を選択します。
4. ［**Join Condition**］の隣に、着信リンクの true の値に基づいた結合式を入力します。この条件はランタイム時に *true* または *false* に評価されなければなりません。オプションで、**Expression Builder** を使用して、結合条件を作成できます。テキストフィールドの右端の青のアイコンをクリックします。［Expression Builder］ダイアログボックスが表示されます。



5 左上のリンクツリーは、選択可能な着信リンクの名前が追加されます。リンク式の構文ヘルパが右上に表示されます。任意のツリーのリーフノードをダブルクリックして、下のテキストフィールドに式を作成します。ダイアログボックスを閉じます。

結合条件が true になるまで、結合アクティビティを開始できないことに注意してください。Deferred モード (デフォルト) では、結合条件は、着信リンクすべての true の値がわかった時点で正確に一度だけ評価されます。Immediate モード (［Object Properties］パネルで選択できる) では、結合条件は true の値がわかると評価され、それが true になると、結合条件はすべてのソースアクティビティの実行を終了したかどうかにかかわらず開始されます。

注記：設計セッション中に、アクティビティに結合条件を割り当て、後で 1 個または複数の着信リンクを削除した場合、結合ロジックは意図通りに機能しなくなることがあります。結合への着信リンクが削除されたり、置換されたりしたときはいつでも、結合条件を忘れずに更新してください。

タイムアウトと再試行

Process Manager ではタイムアウト、再試行回数、および再試行間隔パラメータがサポートされ、単純な要求 / 応答シナリオおよび依頼 / 応答シナリオでは表現されないタイミングと往復の対話を含むパートナー間の複雑な処理ができるようになります。

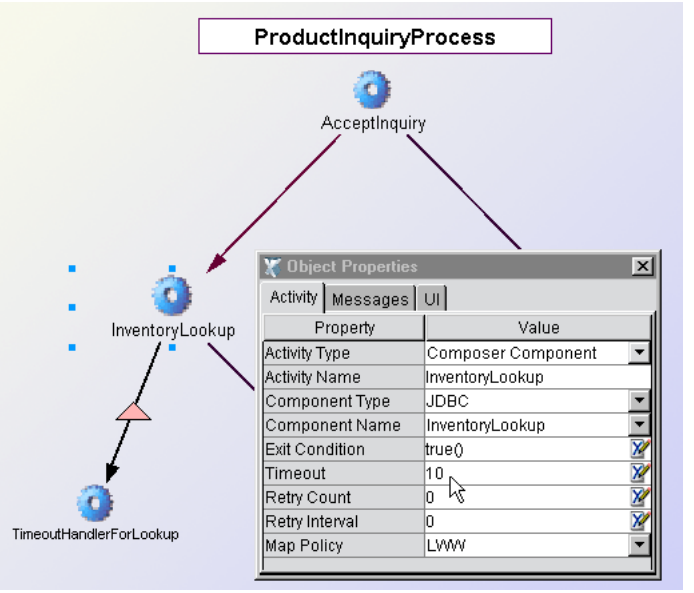
注記：再試行回数と再試行間隔は、タイムアウト値が指定されたときにのみ有効になるので、注意することが重要です。タイムアウト値が指定されていない場合、再試行回数と再試行間隔は無視されます。

タイムアウト障害が発生する条件およびその発生方法の詳細については、この後の障害トラップに関する説明を参照してください。

➢ タイムアウトおよび再試行パラメータを指定する

1 タイムアウトおよび再試行パラメータを設定するアクティビティをクリックします。

2 ［**Object Properties**］パネルが表示されていない場合は、それを表示します (Composer の［View］メニューから［Object Properties］メニューを使用して表示します)。



3 ［**Activity**］タブを選択します。

4 ［**Timeout**］の隣に、日数、分数、または秒数の値 (適宜、「d」、「m」、または「s」を接尾語として入力する) を入力します。例 : 7 日を指定するには、「7d」と入力します。

注記 : 単位の指定子を使用する場合、値全体を疑問符で囲んでください。単位なしの値を (疑問符を付けずに) 入力すると、「秒」として解読されます。

5 タイムアウト値を入力すると、［**Retry Count**］の隣のテキストフィールドに数字をオプションで入力できます。最初の試行がタイムアウトした後にアクティビティを再試行する回数です。*数字を入力しないと、再試行は行われません。*

6 ［**Retry Interval**］の隣のテキストフィールドには、再試行の間の待ち時間を表す値を入力します (秒数)。デフォルトは「ゼロ」で、アクティビティがタイムアウトするとすぐに待ち時間なしで再試行が行われることを示します。再試行間隔がゼロでない場合、Process Manager はタイムアウトから再試行まで指定された時間待ちます。

マップポリシー

マップポリシーは複数のデータソース (着信メッセージ) がターゲットアクティビティの入力メッセージの同じ場所にマップするパートを持つ場合に有効です。たとえば、Activity1 および Activity2 が結合アクティビティ Activity3 にリンクしているプロセスを考えてみます。Activity1Output の *Output/ShipmentMode* が *Activity3Input/Input/ShipVia* に、Activity2Output の *Output/Carrier* が *Activity3Input/Input/ShipVia* にマップされるとき、競合が潜在的に存在します。結果は最後に到着するデータを維持するか (適宜、発生するたびに上書きを許可する)、*最初に*到着するデータのみを維持するかによって決まります。これを指定するためには、マップポリシーを適宜、LWW (最後の書き込みが優勢) または FWW (最初の書き込みが優勢) に設定することが必要です。

注記： 複数のアップストリームソースからデータを受け取るために、*アクティビティには結合アクティビティがない*ことに注意してください。したがって、ターゲットアクティビティに 1 個の着信コントロールリンクしかないときでも、マップポリシーが有効になることがあります。

LWW、FWW、およびマップ順序

マップポリシーの選択肢には、［LWW］(Last Writer Wins) または、［FWW］(First Writer Wins) あるいは［Map Order］があります。2 つの選択肢の意味は、文字通りの意味を表しています。マップ順序については、次に説明します。

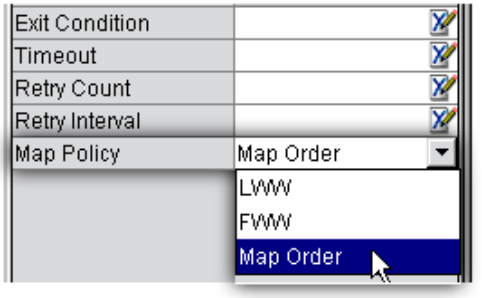
マップ順序は、着信メッセージのタイムスタンプにかかわらず、アクティビティのプロパティシートの［Message］タブで上から下の方向にマップが指定される順番に従って XPath から XPath へのマップが実行されることを意味します。つまり、タイムスタンプは無視されます。メッセージは到着時にキャッシュされ、その後、マップが実行されるときに、マップを指定した文字通りの順番に従って各メッセージ部分がマップされます。

一般的に、このオプションは、メッセージが発信される場所が実際の到着順位よりも重要である場合に、上書きを処理する方法として使用します。たとえば、いくつかのアクティビティ結合に送られ、1 個の特定のソースアクティビティが常に他のフィーダアクティビティよりも書き込みの際に優先されなければならない場合、マップ順序を使用して、他のものよりもこのソースを優先して上書きすることができます。

➢ マップポリシーを設定する

1. マップポリシーを設定するアクティビティをクリックします。
2. ［**Object Properties**］パネルが表示されていない場合は、それを表示します (Composer の［View］メニューの下の［Object Properties］メニューを使用して表示します)。
3. ［**Activity**］**タブ**を選択します。

4 ［**Map Policy**］の隣のプルダウンメニューを使用して、［**Last Writer Wins**］(LWW)、［**First Writer Wins**］(FWW)、または［**Map Policy**］を指定します。



障害メッセージと障害の処理

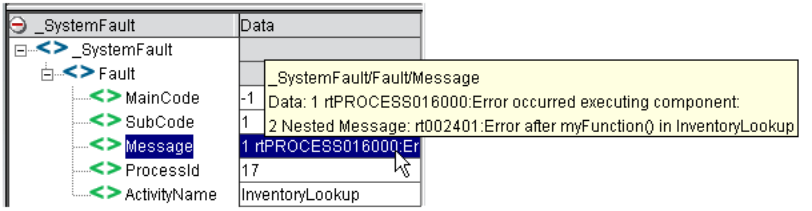
Process Manager により生成される障害には、一般的に、システムとタイムアウトの 2 種類があります。どちらも特別なメッセージとして生成されます。障害メッセージは、障害が発生したアクティビティの Activity Output メッセージの代わりに出力されます。つまり、障害のあるアクティビティの実装は、出力を生成したとは見なされません。したがって、アクティビティは出力メッセージまたは障害メッセージのいずれかを生成し、両方を生成することはありません。

システム障害

ランタイムエンジンでは、次の状況で「システム障害」が発生します。

- アクティビティの実装が、未処理例外を生成するとき
- サブプロセスアクティビティが障害メッセージを返すとき
- ランタイムエンジンが、処理方法のわからないメッセージまたはメッセージタイプを見つけたとき
- タイムアウト障害が発生し、その目的で設計されたアクティビティにより処理されなかったとき (この場合、実際には 1 個のタイムアウト障害および 1 個のシステム障害の 2 個の障害が生成されます)

システム障害が発生すると、プロセスインスタンスは *_SystemFault* と呼ばれるメッセージに同様に *_SystemFault* と呼ばれるパート名を付けてメッセージを作成します。メッセージの DOM 表示は次のとおりです。



すべてのシステム障害は、MainCode、SubCode、Message、ProcessID、および ActivityName 要素を含みます。各要素の内容は、Message 要素の上に示されるロールオーバーツールヒントに表示されます。Fault/Message 要素には、ネストされたメッセージが含まれるため注意してください。このネストされたメッセージの値は、Log または Raise Error アクションで入力したいずれかのカスタム文字列の値です。

原因にかかわらず、障害が「その目的で設計されたアクティビティで処理されない限り」、障害 (どの種でも) では実行中のプロセスが終了されます。この観点では、障害は例外に似ています。ハンドラが存在しない場合、障害はプロセスエンジンに「バブルアップ」し、プロセスはプロセスインスタンスが障害メッセージでのみ終了することを許可します。未処理の障害時に存在するアクティビティインスタンスは中断されます。

障害コード

現在実装されている MainCode 値は次のとおりです。

SYSTEM_FAULT_MAINCODE	-1
TIMEOUT_FAULT_MAINCODE	-2

現在実装されている SubCode 値は次のとおりです。

COMPONENT_FAULT_SUBCODE	1
UNHANDLED_MESSAGE_SUBCODE	2

タイムアウト障害

ランタイムエンジンは、Timeout 値がアクティビティに存在するときには、次の動作を強制します。

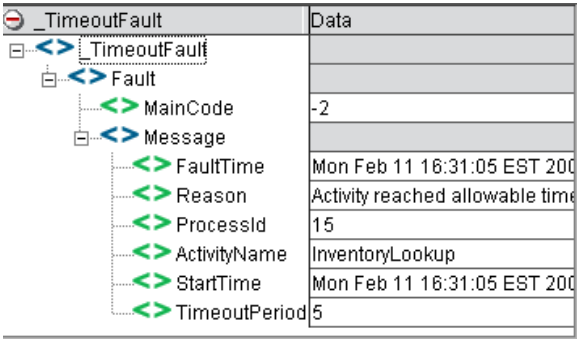
- アクティビティが起動すると、タイマが開始されます。
- アクティビティがタイムアウト期間の前に *true* の終了値で終了した場合、出力リンクに制御を渡します。

- アクティビティがタイムアウト期間の前に *false* の終了値で終了した場合、アクティビティは直ちに再実行されます (false の終了条件で終了したすべてのアクティビティの標準アクション)。

- タイムアウトに到達したときにアクティビティの実行が終了しなかった場合、ランタイムエンジンはアクティビティを中断し、再試行回数パラメータを調べます。再試行回数がゼロでない場合、再試行間隔パラメータ (該当する場合) を調べ、ランタイムエンジンは再試行間隔で指定された時間を待ちます。その後、タイムアウトクロックをリセットし、元のデータマップを使用してアクティビティを再実行します。この実行 - 待ち - 再試行サイクルは、タイムアウト障害が発生する再試行回数に達するまで繰り返されます。

タイムアウト障害がタイムアウトアクティビティにより処理されない場合、ランタイムエンジンはプロセスを終了します。

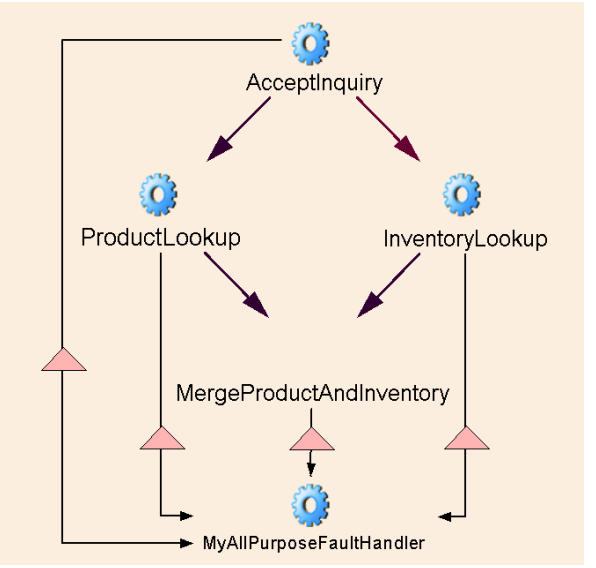
タイムアウト障害メッセージは、ツリー表示では、次のようになります。



メッセージ要素には、文字通りの意味があります。`MainCode` 値はタイムアウトに対して -2 です (図を参照)。

障害の処理

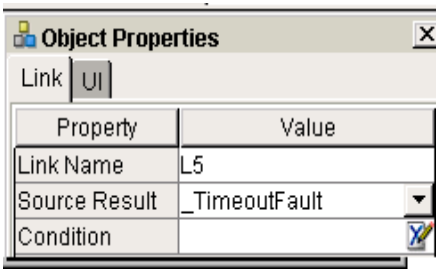
カスタム設計アクティビティ (代わりに Web サービス、Composer Component などとして実装できる) により処理できるため、障害ロジックは必要な程度高度に設定できます。障害ハンドラを必要とする各アクティビティに 1 個の障害ハンドラアクティビティを指定できます (実装は各事例の同じ Composer コンポーネントから構成されている場合もあります)。または、プロセス全体のすべての障害を処理する単独の障害処理アクティビティを持つことができます。後者の例は下のグラフに示されます。各アクティビティはプロセス全体の障害を処理する *MyAllPurposeFaultHandler* にリンクされます。



各リンクの三角形は、障害フローを処理するためにリンクが設計されていることを示します。次に示す手順は、障害を処理するための必要な制御とデータリンクの作成方法を示します。

➢ アクティビティに障害ハンドラを添付する

1. 「障害ハンドラアクティビティ」の実装をまだ作成していない場合は、それを作成します。(このアクティビティは一般的にアプリケーションサーバでローカルに使用されるため、Composer Component として実装する方が正当です。)
2. 障害ハンドラのアクティビティアイコンをプロセスグラフ上に配置します。
3. 適切な「ソース」アクティビティ(障害を生成するアクティビティ) から障害ハンドラアクティビティへのリンクを描きます。
4. 描いたばかりのリンクをクリックし、選択します。
5. ［**Object Properties**］パネルが表示されていない場合は、それを表示します。
6. ［**Link**］**タブ**をクリックします。次に類似するものが表示されます。



7 ［**Source Result**］の隣のプルダウンメニューを使用して、該当する障害タイプを選択します。この場合、`_TimeoutFault` が選択されました。

注記： グラフで、リンクはこの時点で三角形アイコンを取得します。

8 特別な条件がリンクに適用される場合、該当する **XPath** 式を入力します。

9 作業を保存します。

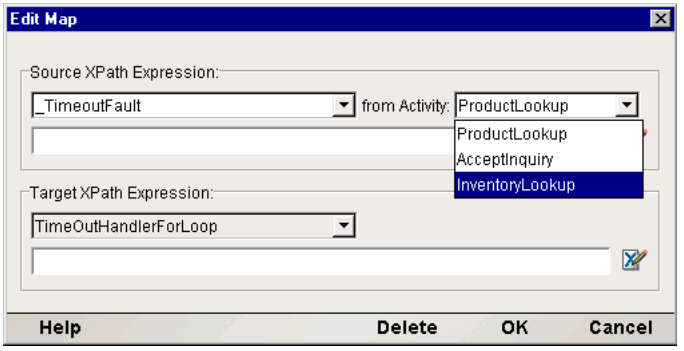
➢ 障害ハンドラへのデータマップを作成する

1 障害ハンドラアクティビティをクリックして選択します。

2 ［**Object Properties**］パネルを表示します。

3 ［**Messages**］タブをクリックします。

4 タブの一番下の「プラス記号」をクリックして、メッセージを追加します。次のダイアログボックスが表示されます。



5 適宜、［**SystemFault**］または［**TimeoutFault**］を［**Source XPath Expression**］の左上部分にあるプルダウンメニューから選択します。

6 ［**from Activity**］の隣のプルダウンメニューで、このメッセージのソースアクティビティを選択します。

注記： すべてのソースアクティビティの候補がリストされます。つまり、1 方向の逆方向リンクで到達可能なアップストリームアクティビティです。これらのアクティビティのいずれか 1 つを選択すると、アクティビティから障害ハンドラへのデータリンクを作成することになります。このリストから選択するソースアクティビティは制御リンクによりターゲットアクティビティに直接、接続される必要はありませんが、ほとんどの場合、データリンクはそれ自体で障害ハンドラを発するには十分でないため、フロー制御接続などが必要になります。ボトムライン：障害メッセージをアクティビティ入力にデータマップする場合、ソースアクティビティから障害アクティビティへも「制御リンク」を描き、障害アクティビティが実際に実行されるようにしてください。

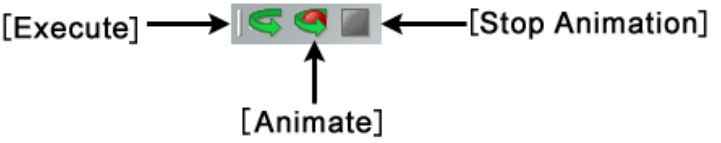
7 ［**OK**］をクリックしてダイアログボックスを閉じます。

8 この障害ハンドラに送る各アクティビティについて、ステップ 4 からステップ 7 までを繰り返してください。

アニメーションとテスト

Process Manager の固有の強力な機能は、「設計時環境」でプロセスの実行およびデバッグ (ステップ実行またはアクティビティに対してなど) を許可することです。Process Designer は Composer 自体の中で実行されるため、アクティビティの実装を構成する Composer Component に直接ステップインできます。コンポーネントの中に入ると、コンポーネント設計セッション中と同様にアクションモデルをステップ実行し、リアルタイムで DOM の変化を監視し、ブレークポイントを設定することなどができます。プロセスのテストおよびデバッグを行うのと同時にアクティビティをデバッグできます。

この目的のための特別なツールバーボタンを使用してプロセスをアニメーション表示または実行できます。



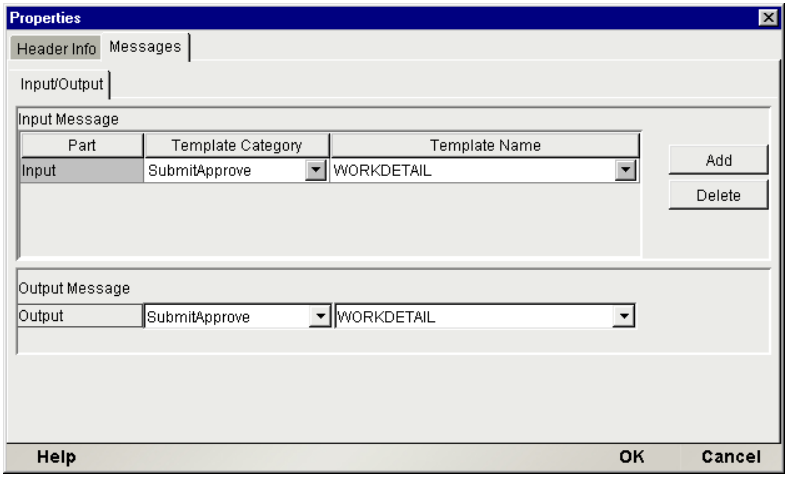
実行とアニメーション表示の違いは、実行はプロセスを中断なしで最初から最後まで実行することで、アニメーション表示はプロセスをステップ実行できるということです。

Process Designer はアニメーション中に貴重なビジュアルフィードバックを与えます。プロセスを実行するか、アニメーション表示するかにかかわらず、1 つのハイライトされたアクティビティから次のアクティビティに渡すときに個々の制御リンクがハイライトされる (太く表示される) のを確認でき、リンクをたどることができない場合 (条件が false であるため)、リンクの表示は実践から破線に変わります。したがって、どのリンクをたどることができ、どのアクティビティが実行中であるかを一目で容易に確認できます。

➢ プロセスをアニメーション表示する

1. (オプション) Composer メインウィンドウの一番下にある出力ペイン (システムメッセージが表示される場所) をクリアします。このためには、ペインの内側をクリックし、<Control>+<A> (Select All) を入力し、<Backspace> キーを押します。

2. `ProcessInput` データテンプレートをテストの目的でプロセスにまだ割り当てていない場合、［**File**］メニューから［**Properties**］を選択し、［**Message**］タブをクリックします。すでに割り当てている場合は、手順 6 に進みます。

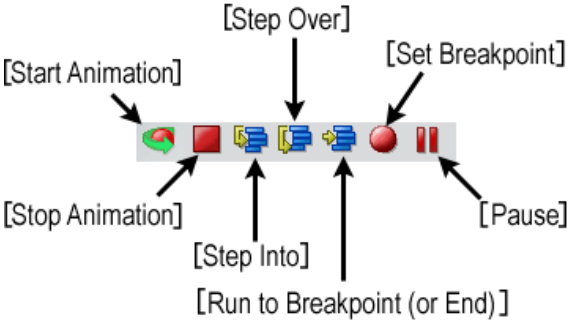


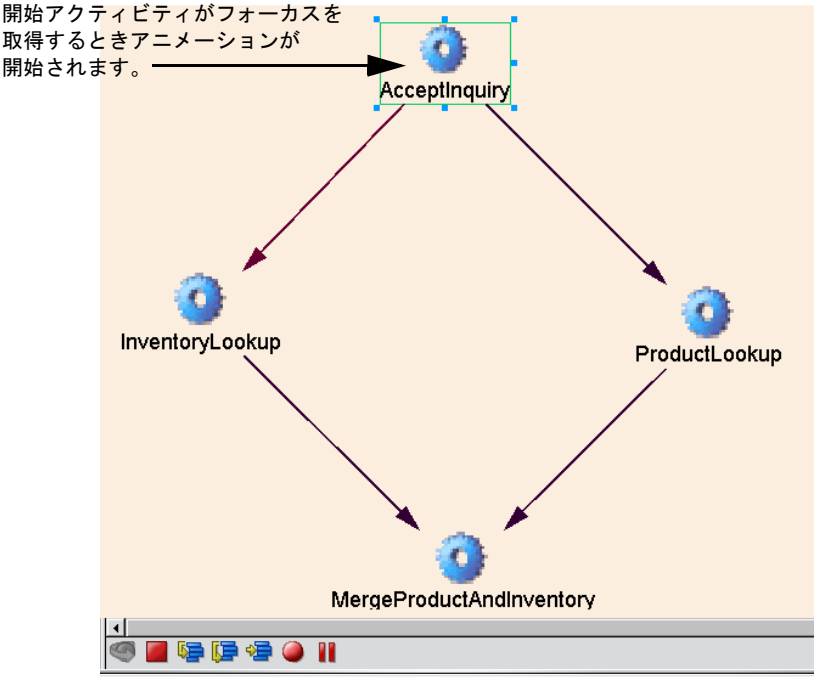
3. ダイアログボックスの上半分にあるプルダウンメニューを使用して、選択可能な XML テンプレートから選択し、入力メッセージを選択または「**追加**」するか、その両方を行います。同様に、必要に応じて、出力テンプレートを選択します。

4. ［**OK**］をクリックして［Properties］ダイアログボックスを閉じます。

5. ［*ProcessInput*］と開始アクティビティの間のデータマップを作成します (これは以前、説明した手順です。この章の「開始および終了アクティビティのためのデータマップ」を参照してください)。

6. Process Designer ツールバーの［**Animate**］ボタンをクリックします。開始アクティビティがハイライトされます (次の図を参照)。新しいツールバーがグラフィックウィンドウの一番下に表示されることに注意してください。アイコンは左から右へ［Animate］(アクティブのときに淡色表示される)、［**Stop**］、［**Step Into**］、［**Step Over**］、［**Run to Breakpoint**］、［**Set Breakpoint**］、および［**Pause**］です。



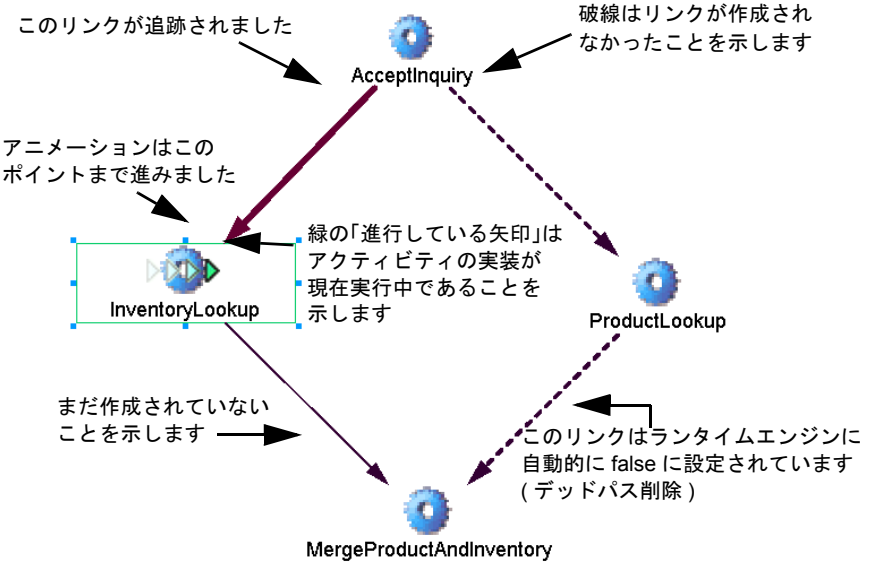


7 アクティビティの実装にステップインする場合、[**Step Into**] ボタンをクリックします。これにより、Composer 内の適切なコンポーネントエディタ環境でアクティビティの下のコンポーネントが開きます。Composer の場合と同様、コンポーネントのアクションリストをステップ実行できます。アクションリストの最後のアクションをステップ実行した後、プロセスグラフウィンドウが再表示されます。

8 現在ハイライトされているアクティビティをステップオーバーする場合、［**Step Over**］ボタンをクリックします。該当するリンクが表示され、リンクは二重の太さの実線 (true のリンクの場合) または破線 (false のリンクの場合) で表示されます。実行はターゲットアクティビティで停止します。その後、ステップオーバーまたはステップインを再び使用することなどが可能です。

9 プロセスの最後まで実行するには、［**Run to End**］をクリックします。プロセスが実行を終了すると、小さな警告ダイアログが表示され、プロセスが通常に終了されたか、または何らかのエラーが発生したかを説明します。

連続する制御をステップ実行すると、種々のリンクがハイライトされ、実行中にたどる実際のパスを反映するように表示が変わります。たとえば、次に示すグラフでは、アニメーションがフローの開始アクティビティから次のアクティビティに進んだことが示されています。開始アクティビティからの 2 つの送信リンクの 1 つを追跡しています (左に暗い、実線のリンクが示されます)。他のリンクは移行条件が false であったため、追跡していません (右に破線が表示されています)。



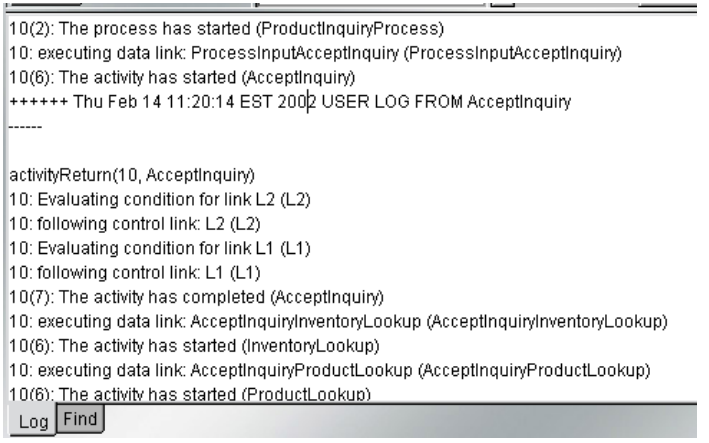
AcceptInquiry から *ProductLookup* へのリンクは追跡されなかったため、*ProductLookup* から *MergeProductAndInventory* へのリンクもまた、デッドパス削除によって、(実行がこのポイントまで進んでいない場合でも) 破線で示されていることに注意してください。プロセスエンジンは、*AcceptInquiry* から *ProductLookup* へのリンクが false である場合、*ProductLookup* から *MergeProductAndInventory* へのリンクを追跡することはできないことを認識しています。したがって、ダウンストリームリンクも false に設定されます (実際、false に設定される必要があります)。これは、*MergeProductAndInventory* の結合条件は、評価されない *feeder link* の true の値を待っている場合に決して評価されることがないためです (「デッドリンクと同期障害」の説明については、第 1 章を参照してください)。

デバッグの支援

Process Designer には、プロセスのステップごとの実行をモニタするためのたくさんの方法があります。たとえば、貴重なリアルタイムフィードバックが Composer ウィンドウの［Log］ペインで (プレインテキスト形式で) 指定され、障害メッセージの DOM 表示とともにアクティビティの入力または出力 DOM を調べ、フローの種々のポイントで作成されるデータ値がどれであるかを正確に確認できます。

アニメーション時のシステムメッセージの監視

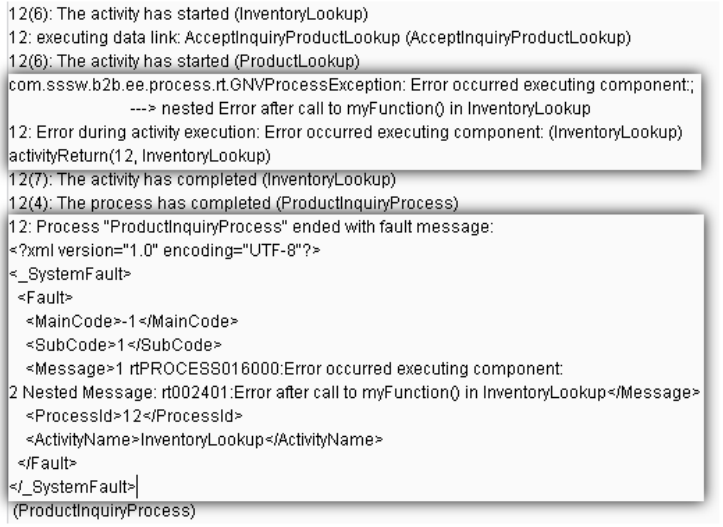
Process Designer でプロセスを実行またはアニメーション表示する場合はいつでも、システムメッセージが Composer のメインウィンドウの一番下の［Log］ペインに表示されます (次の図を参照)。



このペインには非常に詳細な情報が表示されます。すべてのアクティビティの開始、リンク評価、結合評価、アクティビティの終了、アクティビティエラー、つまりすべてのイベントがログされるので、連続するイベントを戻って、何が、いつどの時点で実行され、何が失敗したか、またその理由について正確に調べることができます。

注記： 2 つの数字 (そのうち 1 つが括弧で囲まれている) がすべてのメッセージの先頭に付きます。最初の番号は、現在のインスタンスの ProcessID です。括弧で囲まれている 2 番目の番号は、該当するイベントのイベントコードです (アクティビティの開始は 6、アクティビティの終了は 7 など)。

障害が発生する場合、妨害しているアクティビティを容易に識別し、完全な障害メッセージを (XML 形式で) 確認することもできます。



アクティビティが Composer コンポーネントまたはサブプロセスとして実装された場合、該当するアクティビティ (プロセスグラフの右) をダブルクリックすると、コンポーネントは適切なコンポーネントエディタで開きます。その後、コンポーネントを変更し、保存し、別のアニメーションセッションのために Process Designer に戻ります。

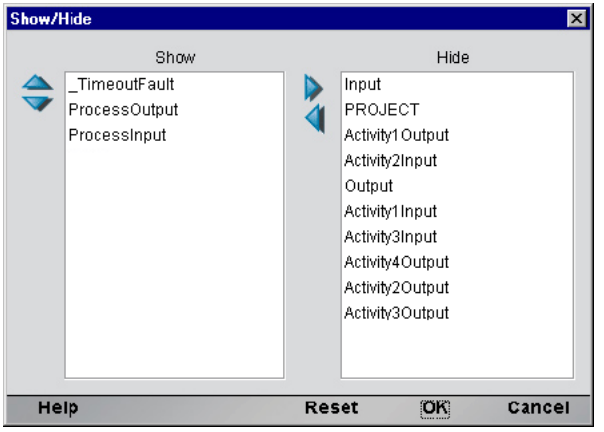
注記： アクションモデルを変更したときは、プロセスを再実行する前に必ずコンポーネントを保存してください (変更を保存)。保存しないと、同じエラーが発生します。

メッセージの検査

プロセスの任意の時点で作成されたメッセージは、固有のペインの DOM ビュー、テキストビュー、または様式化されたビューに表示できます。これには、ProcessInput、ProcessOutput、_TimeoutFault、および _SystemFault メッセージだけではなく、すべての入力および出力メッセージが含まれます。

➢ メッセージを表示する (または既存のメッセージを隠す)

1 メインメニューバーから、［**View**］>［**XML Documents**］>［**Show/Hide**］の順に選択します。



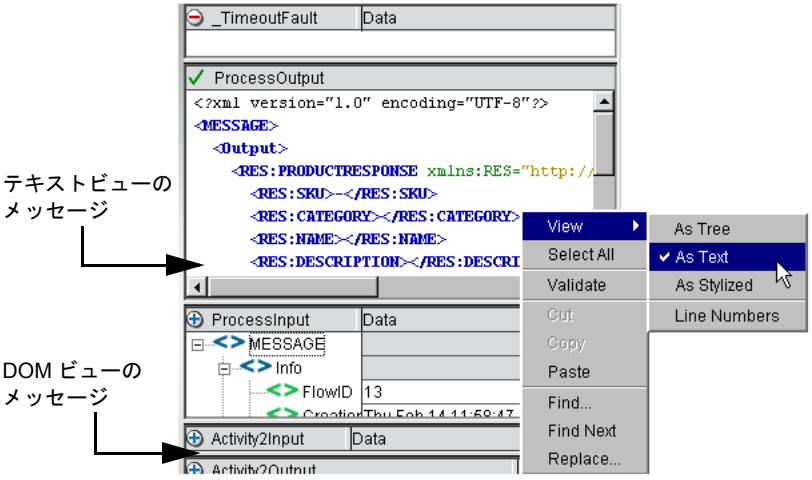
2 表示されるダイアログボックスで、「左右の矢印」ボタンを使用して必要に応じて列を表示または隠します。

注記 : 右に追加されたリストには、プロセスの実行で実際に作成または使用されたメッセージの名前のみが含まれます。プロセスが早く終了した場合、一部のアクティビティのメッセージが表示されない場合があります。

3 (オプション)「上下の矢印」ボタンを使用して、適宜、表示アイテムの順番を変更します。

4 ［**OK**］をクリックしてダイアログボックスを閉じます。

5 ［Show］の下で指定されたメッセージは、固有のデータペインに表示されるようになります (次の図を参照)。



該当する DOM を右マウスクリックすることにより、DOM の異なる表示を取得でき、その後、コンテキストメニューから［**View**］>［**As Text**］または［As Tree］/［As Stylized］) を選択します (図を参照)。

5 高度なトピック

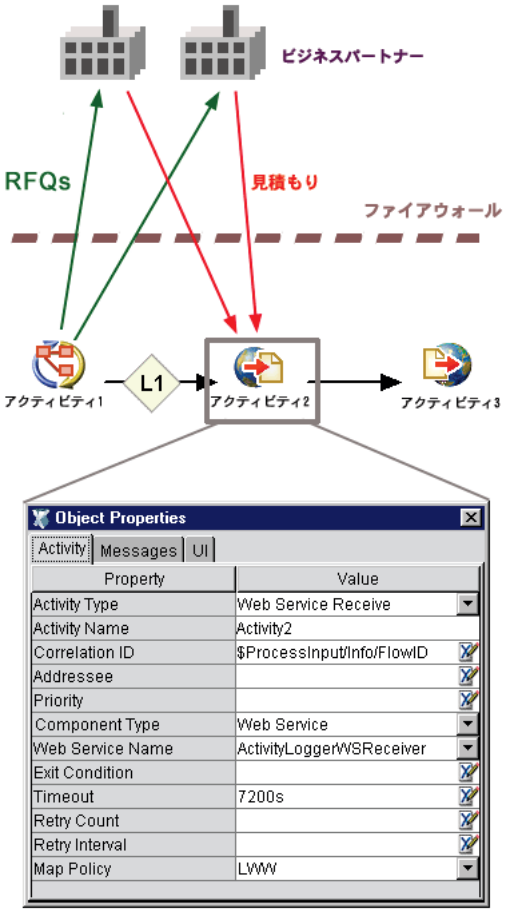
この章では、これまで説明してきた単純な「ストレートスルー処理」使用ケースの範囲を超えた概念と事例について取り扱います。特に、Web Service Receive アクティビティおよび Synchronize Subprocesses アクティビティについて考察します。Web Service Receive アクティビティは、進行中のプロセスの一部として受信の通知またはリクエストに依存する設計パターンを実装する場合に便利です。一方、Synchronize Subprocesses アクティビティは、アップストリームアクティビティによる以前のファンアウト (または複数のサブプロセスインスタンスに対するワークフローのパラレル部分) から結果を収集したり再同期化させたりする場合に便利です。

この章を最大限に活用するには、WSDL ベースの Web サービス、Composer のアクションモデル、およびメッセージマップ、障害メッセージ / リンク論理に関する概念に精通している必要があります。

Web Service Receive

Web Service Receive アクティビティタイプでは、WSDL のリクエスト - 応答および一方向のポートタイプパターンを実装できます。これらは、受信リクエストによってトリガされるまで「エンドポイント」アクティビティ (起動する Web サービスを示す) では何も実行されないパターンです。つまり、ターゲットアクティビティの実装は、通知および依頼 - 応答のシナリオ (この場合、基本となるサービスは「リクエストされる側」ではなく、「リクエスタ」) とは異なり、能動的になります。

Web Service Receive アクティビティは、プロセスモデル内のアクティビティに対する通常の規則をすべて遵守する必要があります。つまり、タイムアウトおよび再試行の動作や、障害の動作などが可能な、リンクターゲットとして機能できなければなりません。



この例では、アクティビティ 1 (サブプロセスアクティビティ) によって、アクティビティ 2 (Web Service Receive アクティビティ) がリンク 1 を介して「起動」されます。アクティビティ 2 が *true* という条件で終了する場合、(アクティビティ 3 への) 送信リンクをたどることになりますが、終了するまでこれは行われません。アクティビティ 2 が *true* という終了条件でタイムアウト時間内 (この場合、7200 秒つまり 2 時間) に戻らない場合、アクティビティ 2 により `_TimeoutFault` が生成されます。

理解するべき重要な概念は、ランタイムエンジンでは、アクティビティ 2 でサービスを「実行」しないということです。このエンジンでは、(アクティビティと同様に) 単に適切な時間に適切な入力メッセージを送信し、適切な時間に出力メッセージを収集します。「サーバへの受信リクエスト」により、Web Service Receive の「実装」は呼び出されるか、または (サーブレット、JMS リスナなどに関する) 適切なトリガメカニズムを通して、プロセスエンジンからは独立して実行されます。つまり、WSR アクティビティの基礎となる Web サービスアプリケーションは、他の Web サービスと同様、サーバ上にある単なる Web サービスであり、その URL はいつでもヒットされる可能性がありますが、プロセスエンジンでは、「特定のプロセスのコンテキスト内で」、すべてのタイムアウトの制限付きで Web サービスを処理 (および応答する) ことのみを行います。WSR アクティビティが有効でないときにビジネスパートナーが URL をヒットした場合、そのパートナーには SOAP 障害メッセージが返されるのが一般的です。

WSR の通常の使用では、さまざまなベンダにリクエストを送信し、最初の有効な応答を収集して、ある種の処理を続行するためにプロセスが作成されている場合があります。前の図のパターンを使用した場合の往復通信のシナリオは、次のとおりです。

- 「**アクティビティ 1**」の実装には、リクエストを処理するために設計された Web サービスを使用している 1 つまたは複数の外部ビジネスパートナーに対し、RFQ (見積もりのリクエスト) を「通知」を通して発行するアプリケーションが考えられます。

- 「**アクティビティ 2**」は、通知を受けたビジネスパートナーが見積もりの返信に 2 時間を使用できるよう設定できます。(いかなるパートナーからも) 2 時間以内に返信がなかった場合、このアクティビティによりタイムアウトの障害が生成されます。

- 「**アクティビティ 2**」の基盤となる実装には、有効な見積もりを受信するとただちにプロセスエンジンを起動させる Web サービスが考えられます。

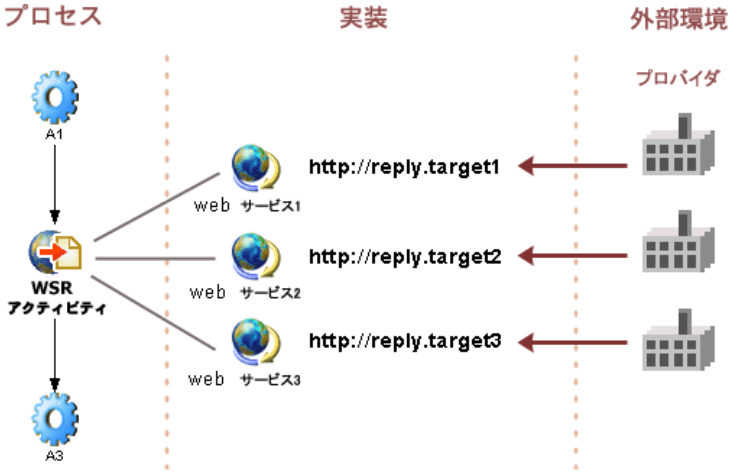
- 「**アクティビティ 2**」は、返信が処理されると (次のアクティビティに制御を転送して) ただちに終了するか、2 時間後にタイムアウトの障害を生成して終了します (この例では、システムの障害ケースについて取り扱いません)。

- 「**アクティビティ 3**」では、担当者または担当部署 (または別のアプリケーションなど) に、あるパートナーから見積もりを受信したことが通知されます。

これは、ファンアウト / ファンイン型のシナリオではなく、「早い者勝ち」型のシナリオであるということに注意してください。複数のパートナーに通知を送信し、複数の応答を収集する場合は、Synchronize Subprocesses アクティビティ (後の節で説明) を使用します。

単一の WSR アクティビティに対する複数の実装

単一の Web Service Receive アクティビティの「実装」として機能する Web サービスを複数含めることは、可能です。これは、Web Service Receive アクティビティが、何かを受信することを待機する (「ヒット」されることを待機する)Web サービスの上に作成されるためです。



前の図では、Web Service Receive アクティビティ (A1 と A2 という 2 つのアクティビティの間に存在する) は、アクティビティに対する実装として配備されている 3 つの異なる Web サービスのいずれにも応答することができます。WSR アクティビティが「起動」されると、プロセスでは、URL の **reply.target1**、**reply.target2**、および **reply.target3** で示された 3 つの Web サービスの中の 1 つを待機し、ビジネスパートナーからの入力を受信します。各 Web サービスは、*Find Waiting Activity* アクション (次の章で詳細に説明) を含む Composer アプリケーションです。Web サービスのうちの 1 つによって Find Waiting Activity アクションが実行され、Release Waiting Activity アクションが次に実行されると、プロセスでは、障害が発生していないものとして次のアクティビティである A3 に移行します。

注記： WSR アクティビティが有効でない (たとえば、すでに起動されている、または起動されたがタイムアウトになっている) 場合に、ビジナスパートナーが 3 つの Web サービスのうちの 1 つを「ヒット」すると、このパートナーはある種のエラーメッセージを受信します。多くの場合、これは SOAP 障害です。

実装の独立性

Process Manager では、Web Service Receive アクティビティの実装に関して制限を設けることはありません。これは、Web サービス全般に対して言えることです。WSDL の作成者は Web サービスが実装される方法について制限を設けることはなく、また、使用される転送メカニズムについても制限はありません。たとえば、Web サービスでは HTTP を使用する必要はありません。また、ペイロードも SOAP で渡される必要はありません。

Process Manager では、Composer Web サービス、JMS サービス、または外部 (任意の実装で、Composer に組み込まれていない) といったさまざまな形式で Web Service Receive アクティビティを実装することもできます。これらの選択肢は、［Object Properties］ダイアログボックスの［Activity］タブにあるプルダウンメニューコントロールに表示されます (次の図を参照)。

Web サービスを真の Web サービスとして機能させるには、当然のことながら WSDL 定義を関連付ける必要があります。Web Service Receive アクティビティのメッセージマップの処理方法を決定する際、Composer では WSDL と通信します。また、これは Web Service *Receive* アクティビティであるため、基礎となるサービスでは、WSDL の一方向またはリクエスト - 応答ポートタイプのシナリオのいずれかを実装する必要があります (この 2 つのパターンの際立った特徴は、それらを実装しているサービスがトランザクションの「イニシエータ」にはならないということです。サービスは「受信者」であるため、ポートで「リスニング」していると考えることができます)。

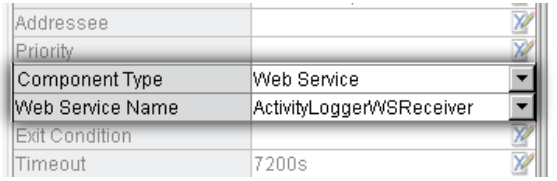
➢ Web Service Receive アクティビティを使用する

1. アクティビティの実装として機能する Web サービスを設計し、実装します。この Web サービスには、独自の WSDL リソースが必要です (Composer でサービスおよび WSDL リソースを作成する方法については、『eXtend Composer ユーザガイド』を参照してください)。
2. 別のプロジェクト内で Web サービスを作成した場合、この Web サービスとそのリソースを現在のプロジェクト、つまりプロセスを含むプロジェクトにインポートできます。
3. Web Service Receive アクティビティを使用するプロセスを作成するか、開きます。
4. (Process Designer ツールバーにある) アクティビティツールの Web Service Receive アクティビティバリアントを使用して、Web Service Receive アクティビティのアイコンをプロセスグラフに配置します。
5. その他の種類のアクティビティの場合と同様に、Web Service Receive アクティビティへのリンクおよび Web Service Receive アクティビティからのリンクを作成します。

6 (必要に応じて［**View**］>［**Object Properties**］を選択して)［**Object Properties**］ペインを表示します。

7 ［**Object Properties**］ペインで、［**Activity**］タブをクリックします。

8 プロパティリストの［**Component Type**］の隣で、プルダウンメニューを使用して、［**External**］、［**JMS Service**］、または［**Web Service**］のいずれかを選択します (次の図を参照)。



9 ［**Web Service Name**］の隣で、プルダウンメニューを使用して、手順 1 で作成した Web サービスを選択します (このリストには、現在のプロジェクトにあるすべての Web サービスの名前が表示されます)。

10 ［Activity］タブで指定する他のプロパティを設定します。

11 ［**Messages**］**タブ**に切り替えます。

12 「**プラス記号**」のアイコンを使用して、追加するデータマッピングを追加します。

13 作業を「**保存**」します。

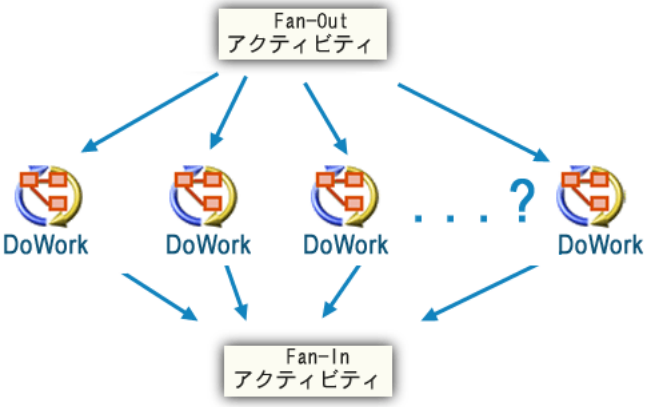
プロセスで Web Service Receive アクティビティを使用するためには、WSDL に記述された一方向またはリクエスト - 応答のパターンを通して通信する Web サービスで構成された、基盤となる「実装」がなければなりません。一方、このサービスでは、「完了」の状態 (ほとんどの場合、ある種の XML データ) をプロセスエンジンに通信し直すことができなければなりません。このような通信には、次の章で説明されているように、サービスのアクションモデルの Find Waiting Activity アクションおよび Release Waiting Activity アクションを使用する必要があります。

注記： Web Service Receive **アクティビティタイプを使用する場合、次の章で説明する** *Find Waiting Activity* **アクションおよび** *Release Waiting Activity* **アクションについての節を読むようにしてください。**

Synchronize Subprocesses アクティビティ

Synchronize Subprocesses アクティビティは、実装が受信データを能動的に待機し、最終的に終了する前に何度も呼び出されることができるとみなすという点において、Web Service Receive アクティビティに類似しています。Web Service Receive アクティビティと異なる点は、Synchronize Subprocesses アクティビティでは、Composer コンポーネント (例 : XML Map コンポーネント) を実装として使用する必要があるということです。

Synchronize Subprocesses アクティビティの目的は、ランタイムになるまで入力数を特定できない場合に、多数の入力アクティビティからのデータを単一のアクティビティとして収集できるようにすることです。つまり、この場合、(リンク数が不特定なため) プロセスグラフで描画できないシナリオになります。このようなシナリオは「ファンアウト / ファンイン」シナリオとも呼ばれます。



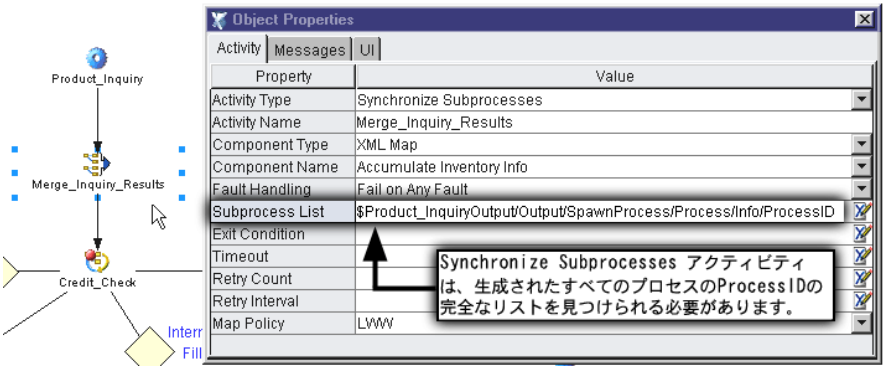
この図の Fan-Out アクティビティは、ワークアイテムのバッチを受信するプロセスにおける開始アクティビティを示します。ただし、ワークアイテムの数はランタイムになるまで特定されません。たとえば、*DoWork* というサブプロセスでは、1 つのワークアイテムのみを処理し、次のアクティビティに渡すことができると仮定します。開始アクティビティで *N* 個のワークアイテムを *N* 個の *DoWork* インスタンスにファンアウトし、それらのインスタンスを平行して実行した後、さまざまな *DoWork* インスタンスのすべての結果を図のような中央の Fan-In アクティビティで収集できるようにするのが理想です。

問題となるのは、このパターンは、*DoWork* の可能な最大インスタンス数 (最大バッチサイズ) があらかじめ特定されている場合にのみ描画できるということです。たとえば、バッチにワークアイテムを 13 個以上含むことができないということをあらかじめ特定できる場合は、*DoWork* の起動可能なインスタンス 12 個を示すアクティビティアイコンを 12 個グラフに配置して、Fan-Out アクティビティから *DoWork* の各インスタンスへのリンク (および各 DoWork から Fan-In アクティビティへの送信リンク) を接続することができます。各リンクの単純な XPath の条件では、(Fan-Out アクティビティからの出力をチェックすることによって) 該当するソース XPath にデータが含まれているかどうかに基づいて特定のリンクが起動されるかどうかを決定することができます。

これまでに説明したタイプの明示的なグラフは、目的の機能を果たします。外観はあまり美しくなく、データマッピングのスペルアウトにも時間がかかるので厄介ですが、確実に機能します。ここでの問題は、たとえば 6ヶ月後に、最大バッチサイズを 12 ではなく 200 にする必要があると判断されたり、バッチサイズの制限がなくなる可能性があるということです。このような場合にはどうすればいいでしょうか？

Synchronize Subprocesses アクティビティは、ファンアウトの結果の再同期化を処理するために設計されています。プロセスエンジンでは、Synchronize Subprocesses アクティビティに代わって特定のサービスを実行し、アクティビティの実装は、特定のランタイム動作を考慮して設計される必要があります。考慮すべき重要な点は、次のとおりです。

- Fan-Out アクティビティ(Process Manager アクティビティの標準タイプのいずれか) では、「サブプロセスアクティビティ」の *N* 個のインスタンスを呼び出します。インスタンスは、Fan-Out のアクションモデル内の **Process Execute** アクションから、ループの一部として生成されます。
- 「ワークアクティビティ」は「サブプロセス」であり、(同期的に呼び出されたのではなく)「生成される」ものであるため、各サブプロセスでは、Fan-Out アクティビティに ProcessID をただちに返します。
- Fan-Out アクティビティの実装では、`Output` 内の既知の XPath で、ProcessID を収集する必要があります。一方、XPath は、次の図に示すように、Synchronize Subprocessesアクティビティのプロパティシートに指定されている必要があります。



- Synchronize Subprocesses アクティビティの基盤となる実装を提供するコンポーネントでは、ProcessID のリストを認識する必要はありません。ランタイムエンジンでは、実装コンポーネントをこのリストに基づいて適切な回数呼び出し、(各サブプロセスが完了すると) 障害の条件がない場合、制御を次のリンクまたはチェーン内のリンクに渡します。このため、実装コンポーネントでは、ループの一部として使用されていることを認識する必要はありません。
- ファンインの実装を起動するたびに、`Input` メッセージパートには、完了したサブプロセスからの出力が含まれます。新しく取得されたデータを必要に応じて処理するかどうかは、Synchronize Subprocesses の実装 (ファンインコンポーネント) によって異なります。通常は、これは `Output` に累積されることを意味します (次に説明する理由のため)。
- すべてのサブプロセスが戻ると、(障害の条件がない場合) アクティビティが戻り、ペアレントプロセスでは通常の制御チェーンを続行します。

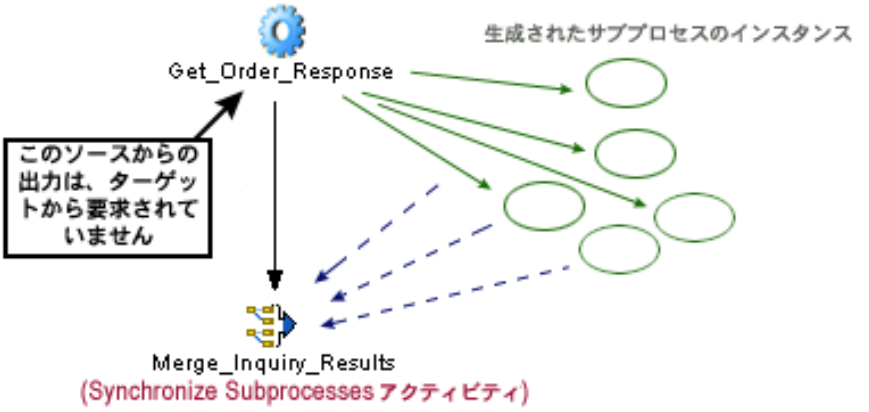
Synchronize Subprocesses アクティビティ内のデータマッピング

Synchronize Subprocesses アクティビティには、常に `Input`、`Input1`、および `Output` という 3 つのメッセージパートが少なくとも必要です。アクティビティの「実装」には、これらのパート名に対応する DOM もありますが、パートには独自の役割があり、実装は、これらの役割を十分考慮して設計する必要があります。

Input

実装の面から考えると、*Input* メッセージパートは、「サブプロセスの出力」が受信されるところです。生成されたサブプロセスが戻るたびに、その出力は、マージコンポーネントの `Input` に渡されます (この場合の「マージコンポーネント」とは、XML Map コンポーネント、JDBC コンポーネント、またはその他の Synchronize Subprocesses アクティビティの「実装」を意味します)。

他のほとんどのアクティビティタイプでは、以前のアクティビティの `Output` からのデータがターゲット実装の `Input` DOM に渡されます。しかし、Synchronize Subprocesses の場合は、Synchronize Subprocesses アクティビティを起動するアクティビティが該当するデータソースではないため、これは当てはまりません (次を参照)。



Synchronize Subprocesses アクティビティの実装 (または「マージ」コンポーネント) では、生成された「サブプロセスのインスタンス」によって提供されるデータを必要とし、そのデータを探すために `Input` を検索します。マージコンポーネントでは、起動されるたびに、`Input` から単一ワークアイテム分のデータを探します。

Input1

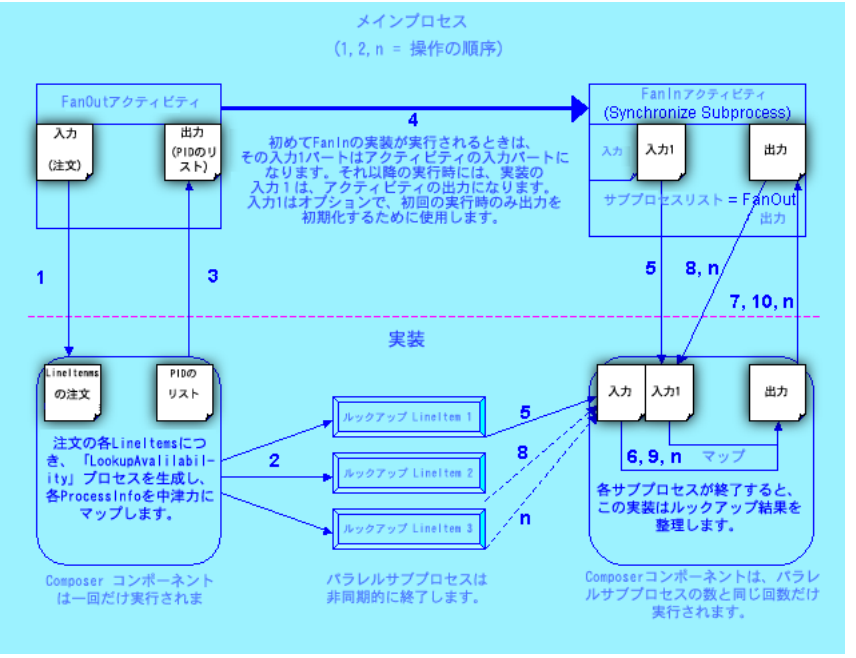
Synchronize Subprocesses アクティビティの実装では、通常、まず最初に `Input1` DOM を `Output` に直接マップします。つまり、通常は、実装のアクションモデルの上に、次の図のような XML Map アクションがあります。

これは、マージコンポーネントの `Output` の部分が、その後のコンポーネントの各呼び出しで `Input1` に「戻される」ためです (次の説明を参照)。

Output

Synchronize Subprocesses アクティビティの実装で、段階的に作成された DOM にサブプロセスの戻りを 1 つずつ追加して「ワークアイテム」を単一ドキュメントに累積または統合するには、Synchronize Subprocesses アクティビティで、実装の `Output` を `Input1` で再利用します。つまり、呼び出し *N* において、実装では、呼び出し *N–1* からの `Output` を `Input1` で受信します (呼び出し 0 では、`Input1` は空です)。

これについては、次の図を参照してください。



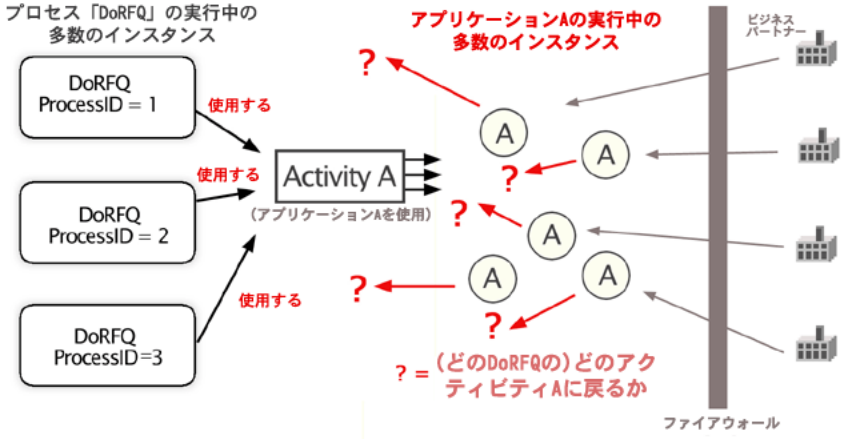
障害の処理

Synchronize Subprocesses アクティビティでは、*Fail on Any Fault* または *Fail if All Fail* という 2 つのポリシーのいずれかに従って、障害のメッセージを生成させることを選択できます。最初のインスタンスでは、フィーダアクティビティ (データを生成するサブプロセス) のいずれかが障害を起こすと、Synchronize Subprocesses アクティビティはただちに障害を起こします。2 つ目のケースでは、生成されたすべてのサブプロセスは、障害が生成される前に戻る必要があります。いずれの場合も、Synchronize Subprocesses アクティビティで障害が起こると、その障害が (通常の方法で) 処理されるまで、一貫となるプロセスは終了します。したがって、たった 1 つのファンアウトされたサブプロセスのインスタンスによってプロセス全体が失敗することを回避するため、確実な障害処理スキームについて「熟慮」する必要があります。

待機中のアクティビティ

アクティビティ (サブプロセスや Web Service Receive アクティビティなど) が、別のアクティビティによって非同期的に行われたリクエストに対する応答を受信することを待機しているため、「待機の状態」にある場合、このアクティビティは「待機中のアクティビティ」と呼ばれます。待機の状態では、アクティビティは、その言葉のとおり「実行中」ではなく、メモリにも存在しません。アクティビティの実装は、WSDL のリクエスト - 応答または一方向ポートタイプに従って動作する Web サービスである場合があります。この実装は、リクエストが HTTP を通してサーバに送信されたり、JMS メッセージリスナに送信されたメッセージを通してサーバに送信されたりすると起動されます。サービスが完了すると、その実装のアクティビティ (待機中のアクティビティ) は「起動」し、適切な「プロセスインスタンス」で該当するフローパターンの実行を続行できるようプロセスエンジンに通知する必要があります。

しかし、ある種の単なるアプリケーションまたはサービスであるアクティビティの実装では、ステートフルなプロセスで使用されていることを必ずしも認識しているわけではありません (認識している必要もありません)。アプリケーション (アクティビティの実装) は、一般的で再利用可能な、外部クライアントおよびローカルアプリケーションによって呼び出される複数の役割を果たすアプリケーションまたはコンポーネントである場合があります。また、いくつかの異なるプロセスモデルの一部である場合もあります。いかなる場合も、コンポーネントをアクティビティの実装として使用しているプロセスインスタンスがたくさん存在する可能性があります。コンポーネントのインスタンスの起動時、そのインスタンスでは、呼び出し元および呼び出された理由について認識せず、実行中のプロセスでアクティビティの実装として使用されていることを認識することもありません (次の図を参照)。



アクティビティが出力を生成するために基盤となる実装を待機する場合、基盤となるサービスまたはコンポーネントには、正しいプロセスインスタンスに戻るための何らかの方法が必要です。これは、(異なるプロセスモデルに属している可能性のある) 多数のプロセスインスタンスによって同じ実装が使用されている場合があるためです。実装によってアクティビティが待機の状態から脱却し、適切なプロセスインスタンスでナビゲーションが再開できるよう、何らかの相関値が待機中のアクティビティの実装に渡される必要があります。

注記：相関値をいつどのように指定するかについての詳細は、次の章で取り扱います。

ここでのシナリオは、次のようになります。

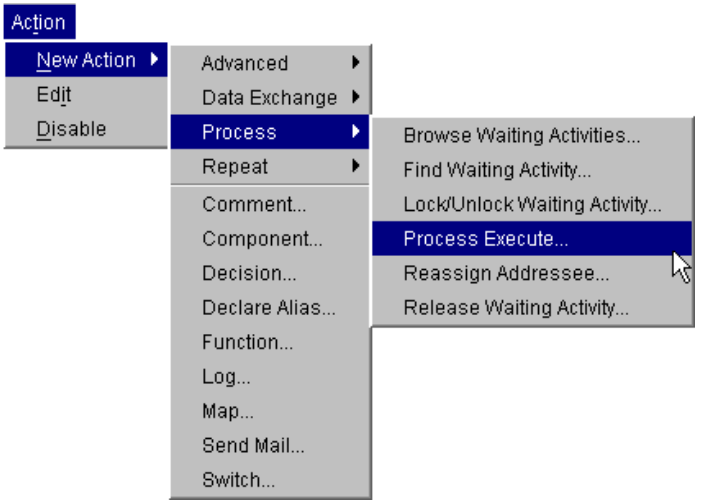
- 外部のサービスまたはビジネスパートナーへの元の送信通知を作成するアクティビティでは、サービスに「相関値」を渡す必要があります。これは、プロセス名に関連したカスタム CorrelationID、またはアクティビティ名に関連した ProcessID である場合があります (詳細については、次の章を参照してください)。
- Web Service Receiveアクティビティの実装として機能するWebサービスでは、外部サービス (ビジネスパートナー) から相関値を取り戻す必要があります。
- Web サービス (WSR の実装) は、*Find Waiting Activity* アクション (［**New Action**］、［**Progress**］、［**Find Waiting Activity**］の順に選択) を含むアクションモデルを持つ Composer サービスでなければなりません。相関値は、該当するプロセスの中で適切な待機中のアクティビティを検索する方法として、このアクションで使用されます。
- Find Waiting Activityアクションが正常に実行されると、*Release Waiting Activity* アクション (［**New Action**］、［**Process**］、［**Release Waiting Activity**］の順に選択) が次に続く必要があります。

「Waiting Activity」アクション

Composer のインストールの一部として Process Manager がインストールされている場合、あらゆるコンポーネントタイプ (JDBC、XML Map、JMS、Telnet など) に対するすべてのコンポーネントエディタには、任意のアクションモデルで使用できるプロセス関連アクションが 6 つあります。

- Browse Waiting Activities
- Find Waiting Activity
- Lock/Unlock Waiting Activity
- Process Execute
- Reassign Addressee
- Release Waiting Activity

これらのアクションは、［New Action］メニューの［Process］サブメニューから選択できます。Composer コンポーネントまたはサービスの任意のタイプ (XML Map コンポーネント、JDBC コンポーネントなど) のアクションモデルでこれらのアクションを使用することができますが、コンポーネントで使用する場合は、そのコンポーネントが Composer の「Web サービス」内でラップされる必要があります。



6 つのアクティビティのうち、5 つは待機中のアクティビティの機能と関係があります。これらのすべての機能では、実装において「一方向」または「リクエスト - 応答」タイプの通信パターンに従うアクティビティが存在することを想定しています。このようなパターンは、外部のリクエストを Web サービスが受動的に待機するパターンです。

Find Waiting Activity および Release Waiting Activity アクションは、関連付けられているビジネスタスクの性質に関係なく、待機中のアクティビティが関係するほとんどのシナリオで一緒に使用されます。その理由は、Web Service Receive アクティビティを有効にすると、「起動」するために両者が必要となるためです。

ランタイム時にプロセスフローが WSR アクティビティに到達すると、プロセスはスリープ状態になり、次の場合にのみ再び起動されます。

- Composer Web サービスにより、WSR アクティビティを対象とする Release Waiting Activity アクションが実行された場合
- WSR アクティビティがタイムアウトになった場合

別の言い方をすると、WSR アクティビティとその基盤となる実装の結合はかなり緩いものです。Web Service Release アクティビティは、プロセスが目覚まし時計によって起動されるまで (アクティビティがタイムアウトになることを意味する) あるいはプロセスを再び起動する方法を認識している Web サービスによって起動されるまでスリープ状態になる、プロセスフローの単なる一段階だと考えることができます。

Waiting Activity アクションおよび使用方法の詳細については、次の章を参照してください。

待機中のアクティビティおよびオペレータによる操作

Browse Waiting Activity、*Lock/Unclock Waiting Activity*、および *Reassign Addressee* のアクションにより、オプションの機能を追加できます。このような機能は、アクティビティによる通知に応答するタスクを人間のオペレータが実行する場合のように、オペレータにより介入されるタイプのワークフローで待機中のアクティビティを使用できるように設計されています。このタイプのフローでは、通知は、待機中のアクティビティからプロセスに作業を最終的に戻すオペレータに送信されることが一般的です。このタイプのシナリオについては、次の章で詳しく説明します。

Addressee の概念は、一部の「待機中のアクティビティ」アクションのシナリオの中で使用されます。これにより、ワークアイテム (メッセージパート、またはパート内のノードの分岐) を、実行中のプロセスの一部として、役割に応じて特定の個人に割り当てることができます。対象となる個人は、その目的のために設計されたアクティビティから着信する作業の通知を受けることができます。そして、プロセスインスタンスにより Web Service Receive アクティビティ (またはその他の「待機中のアクティビティ」) を呼び出して、システムでさまざまな個人の作業を受信できます。

ワークアイテム「*Priority*」の通知もこのシステムで使用されます。

注記 : Addressee および Priority は、Web Service Receive アクティビティの［Object Properties］パネルで最初に指定します (Address および Priority のプロパティは、その他のアクティビティタイプでは表示されません。これらのプロパティを表示するには、Web Service Receive アクティビティを選択する必要があります)。

Lock/Unlock Waiting Activity アクションを使用すると、ある個人が専用に使用するために、ワークアイテムに「*locked*」というマークをプログラムで付けることができます。

Reassign Addressee アクションを使用すると、ワークアイテムを別の個人に「再割り当て」することができます。

また、Browse Waiting Activity アクションを使用すると、特定の個人のワークキューを表す待機中のアクティビティを「参照」または記録することも可能です。

これらのアクションを活用すると、ワークキュー、さまざまな優先度を持つワークアイテム、役割のあるオペレータなどが関連する効率的 (かつ安定性がありテストも容易) なワークフローシステムを開発できます。

6 Waiting Activity と Addressee

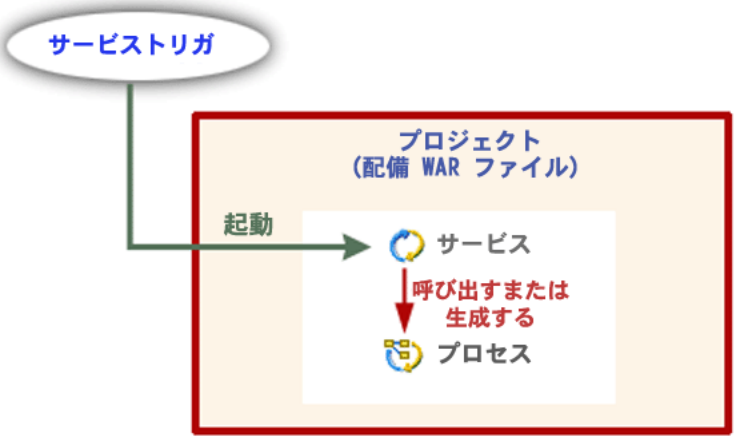
この章では、プロセスおよびアクティビティの呼び出しとコントロールに関連する様々な項目について説明します。プロセスおよびアクティビティには、作業リストなども含めて「人間が介入するアクション」のシナリオを実現するためにコンポーネント内部で利用可能な各種のアクションが含まれます。この章を深く理解するためには、すでに Composer プロジェクトの配備、標準 J2EE パッケージング、および EAR/WAR ファイル、*web.inf* ファイル、コンテキスト、サーブレットなどの配備の構造について精通している必要があります。また、基本的 Composer サービスのトリガタイプについても熟知していなければなりません。これらの項目の詳細については、必ず、ご使用のアプリケーションサーバ環境 (WebSphere,、Weblogic、SilverStream) に合った『Composer Server ユーザガイド』を参照してください。

プロセスがトリガされる仕組みの理解

プロセスが呼び出されるためには、プロセスが、呼び出す側のコンポーネントの *Process Execute* アクションと関連付けられている必要があります (詳細は「Process Execute アクション」を参照)。呼び出す側のコンポーネントには、任意の有効な Composer コンポーネントタイプ (XML Map、JDBC、HTML、Telnet その他) を使用できます。ただし、コンポーネントはサービスによってコンポーネント自身が呼び出されるものでなければならず、またサービスは標準の Composer サービストリガタイプのいずれかによってトリガされる必要があります。起動の順番は、上から次のとおりです。

- HTTP/SOAP リクエストにより、(通常) サーブレットが起動されます。
- サーブレット (サービストリガ) により、Composer サービスが起動されます (Web サービス)。
- Composer サービスにより、Composer コンポーネントが起動されます。
- Composer コンポーネントにより、プロセスが起動されます。
- アクティビティが開始され、そしてプロセスインスタンスの実行期間内に終了します。

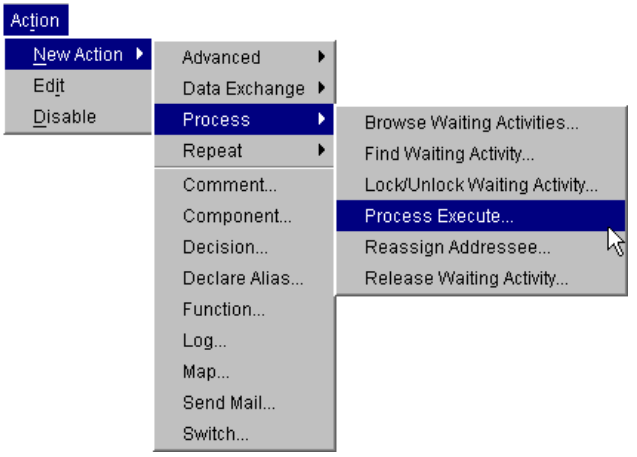
これを最も簡単な形で表すと、起動の連携は次のようになります。



この図では、Composer サービスがプロセスを直接呼び出し、あるいは生成しているように示されています。しかし、既に説明したとおり、任意のコンポーネントタイプ (XML Map、JDBC、その他) でもプロセスを呼び出したり生成したりできます。この方法は、Process Execute アクションを介して実行されます (後で詳細に説明します)。

プロセス関連アクション

Process Manager では、コンポーネントエディタメニューに 6 つのプロセス関連アクションが追加されました。



これらのアクションは、次の 3 つの基本機能をサポートするものと考えることができます。

- **アクションによるプロセスの呼び出し**: これは、Process Execute アクションによって実現できます。指定されたプロジェクト内では、Composer コンポーネントあるいはサービスは (プロセスの内部であっても外部であっても) このアクションを通じてそのプロジェクトに任意のプロセスを起動させることができます。
- **プロセスへのリエントリ** : Find Waiting Activity アクションと Release Waiting Activity アクションによって、Web Service Receive アクティビティを実装するサービスは、アクションの実行が終了した後プロセスエンジンを起動することが可能となります。
- **人間がアクセス可能なワークキュー**: Browse Waiting Activity アクション、Lock/Unlock Waiting Activities アクション、Reassign Addressee アクションは、組織内における個人の作業負担の軽減などを含む各種のシナリオをサポートします。

Process Execute アクション

Process Execute アクションでは、指定したランタイムの入力と出力を使用してプロセスを起動することができます。コンポーネントのアクションモデルでこのアクションを利用すると、現在のプロジェクトで任意のプロセスを呼び出せます。

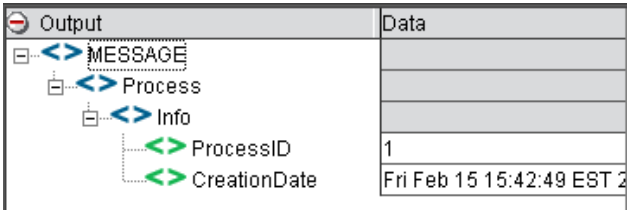
Process Execute アクションは、Composer の通常の Component アクション (コンポーネントを起動) と類似していますが、それ以外に、実行に際して 2 つのメソッド (Call と Spawn) が可能であることと、さらに呼び出されたあるいは生成されたプロセスをペアレントプロセスのサブプロセスとして登録できることが特徴です。

プロセスが *Call* によって開始された場合、Process Execute アクション (ソースコンポーネント) を含むアクションモデルは、呼び出されたプロセスが戻るまでその他のアクションの処理を停止します。これは通常のコンポーネントアクションと同じ動作です。

プロセスが *Spawn* によって開始された場合、プロセスは「実行後削除」モードで実行され、生成した側では生成されたプロセスが戻るのを待ちません。その代わりに、生成されたプロセスは直ちにプロセスとタイムスタンプの固有の識別子から構成される「お知らせ」を戻します (次の図を参照してください)。この情報は、Browse Waiting Activities アクションあるいは Find Waiting Activity アクションなど他のプロセスアクションで使用できます。

生成されたプロセスによって戻されるデータ

Spawn によってプロセスを生成すると、プロセスは「実行後削除」するという方法で非同期に呼び出されます。生成されたプロセスは、生成されたプロセスインスタンスに関する特定の情報を含むメッセージを戻します (この情報は後で役立ちます)。生成されたプロセスによって戻された「受信者反応」情報は、次の図のようになります。

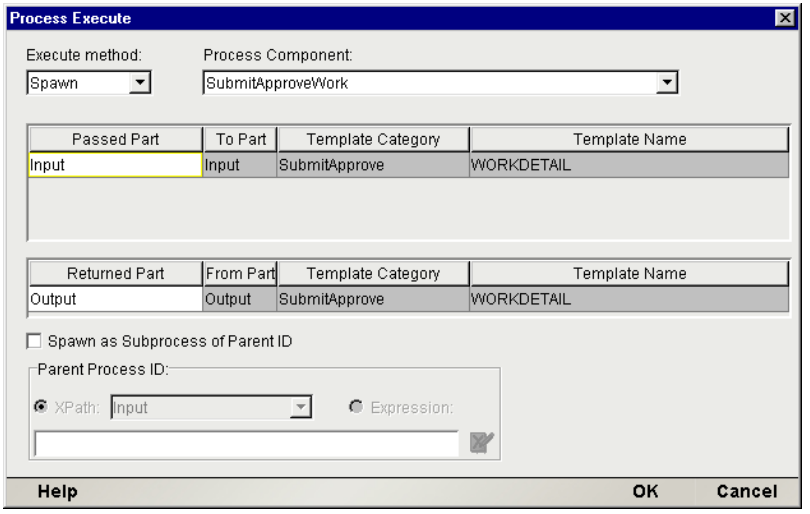


戻された情報には、`Process/Info/CreationDate` の中に、さきほど開始されたばかりの特定のプロセスインスタンスと関連付けられた `ProcessID` とプロセスインスタンスの発生日が含まれています。

Process Execute アクションの作成方法

➢ **Process Execute アクションを作成する**

1. プロセスを呼び出す Composer コンポーネントまたは Composer サービスを開きます。アクションを挿入する場所でアクションモデル内をクリックします。
2. ［**Action**］メニューから、［**New Action**］、［**Process**］、［**Process Execute**］の順に選択します (アクションペインでマウスを右クリックして、コンテキストメニューからこのコマンドを利用することもできます)。ダイアログボックスが表示されます。



3 ダイアログボックスの左上にあるプルダウンメニューを使って、[Execute Method] について、[**Spawn**] (実行後削除) または [**Call**] (結果が戻されるまでブロック) を選択します。

4 [**Process Component**] の下のプルダウンメニューで、呼び出したいプロセスを選択します。メニューには、現在のプロジェクトに存在するすべてのプロセス xObjects の名前が表示されます。

5 [**Passed Part**] で、(このフィールドをクリックすると表示されるドロップダウンリストから) プロセスのデータソースになるコンポーネント DOM の名前を選択します。

6 [**Returned Part**] で、(表示されるドロップダウンリストから) プロセスから戻される情報を受け取る DOM の名前を選択します。Spawn アクションの場合は、次の説明を参照してください。

7 ファンアウトの一部としてこのプロセスを生成する場合は (後で Synchronize Subprocesses アクティビティを使用して同期バックアップ取る場合のことを考慮)、[**Spawn as Subprocess...**] チェックボックスをチェックして、XPath 式によってペアレントプロセスの ProcessID のある場所を提示します。

注記： これは、主に Synchronize Subprocesses アクティビティタイプを指定して作業する場合に役立つ高度なオプションです。現在のコンポーネントがプロセス用のアクティビティの実装で、同一プロセスの別の場所で Synchronize Subprocesses アクティビティを使用している場合以外は、このチェックボックはチェックしないでください。

8 [**OK**] をクリックして、ダイアログボックスを閉じます。

Process Execute ダイアログボックスの詳細

［*Passed Part*］は、ProcessInput メッセージとしてターゲットプロセスに受け渡されるソースコンポーネントパーツのランタイム名を表します。実行するプロセスをドロップダウンリストボックスから選択すると、入力 XML テンプレートによる定義どおりのパーツが表示されます。したがって、受け渡す現在のコンポーネントパーツをプロセスの対応パーツと照合するだけです。［Passed Parts］は、必ずしもテンプレートパーツと名前が一致する必要はありません。ただし、必要なデータをすべてプロセスで受け取っていることを確認するため、受け渡されたパーツの数は必要なパーツの数と同じでなければなりません。

Process Execute アクションの出力は、指定した現在のコンポーネントの「Part」に戻されます。Call によってプロセスが実行される場合、プロセス出力は Returned Part に置かれます。Spawn によってプロセスが実行される場合、Process Info のお知らせは Returned Part に置かれます。

ペアレント ID のサブプロセスとして生成

Process Execute ダイアログボックスの［Returned Part］セクションの下にある［Spawn as a subprocess of Parent ID］チェックボックスは、実行メソッドとして Spawn が選択されているときにだけ表示されます。チェックボックスのすぐ下にあるコントロールもまた Spawn モードが選択されている場合にのみ表示されます。これらのコントロールを使用すると、生成されたプロセスと指定されたペアレントプロセスとの相関関係を築くことができるため、プロセスエンジンはサブプロセスの戻りを追跡することができます。このことは Synchronize Subprocesses アクティビティ実装のコンテキストにおいてのみ重要になります。

注記： Synchronize Subprocesses アクティビティで「ファンアウト / ファンイン」タイプのシナリオを実装しない場合には、この説明について考慮する必要はありません。

Process Execute アクションによってプロセスが生成されるとき各プロセスにペアレント Process ID を付与することによって、エンジンは生成された各プロセスの結果を正しいペアレントプロセスに戻すことができます。同時に多数のペアレントプロセスのインスタンスが実行される可能性があるため、このメカニズムを利用することによって、あるペアレントプロセスのインスタンスが別のインスタンスの結果を受け取るのを防止することができます。

Synchronize Subprocesses アクティビティを使用する「ファンアウト / ファンイン」シナリオの詳細については、後続の章の「Synchronize Subprocesses アクティビティ」を参照してください。

配備と Process Execute アクション

すべての Composer プロジェクトにおける配備の単位は、Web Service xObject です。このように、他のすべての Composer コンポーネント (JDBC、EDI、XML Map、3270 他) と同様に、ビジネスパートナーに提示したいプロセスはすべて Web サービスコンポーネントの内部から実行される必要があります。

もっとも単純なケースでは、Web サービスの内部に単一の Process Execute アクションを配置することによってプロセスを配備できます。その他は通常の配備とまったく同じです。Web サービスの入力メッセージ (およびその構成パーツ) がプロセスの入力と合致することを確認するだけでよいのです。次に、Process Execute アクションを介してプロセスを呼び出し、パーツに渡すのは簡単なことです。

もっと複雑な配備では、Process Execute アクションは、最初のプロセスメッセージを準備する、あるいはプロセスだけでなくその他の複数のコンポーネントを実行する大きなアクションモデルの一部である場合があります。

Find Waiting Activity アクション

Find Waiting Activity アクションは、通常 Web Service Receive アクティビティ実装の内部で使用されます (普通はその後に *Release Waiting Activity* アクションが続きます。次の説明を参照)。*Find Waiting Activity* アクションでは、(たとえば) ビジネスパートナーから起動されるのを待機している Web Service Receive アクティビティのランタイム情報をプロセスエンジンから取得することができます。ビジネスパートナーのメッセージと一緒に取得された情報は、次にアクティビティの出力メッセージを生成するために利用されます。

Release Waiting Activity アクションは、通常すべての *Find Waiting Activity* アクションの後に続きます。*Release Waiting Activity* アクションによって、Web サービスアクティビティに渡される出力とプロセスエンジンに渡される「終了準備」の信号が作成され、これによって引き続きプロセスを継続することができます。

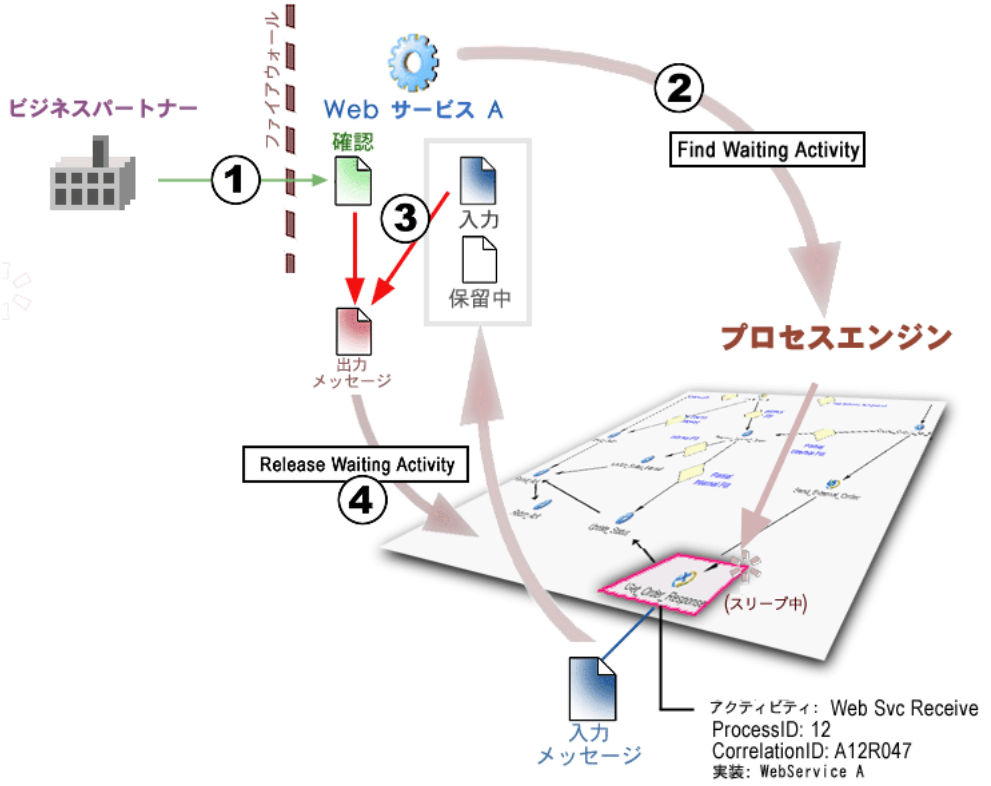
Process Manager で、*Web Service Receive* アクティビティは、Web サービスがプロセスフローを継続するのに必要な情報を受け取るのを待機するため、プロセス (またはプロセスの分岐)でそのアクティビティの制御フローを停止できることを思い出してください。

Web Service Receive を実装するということは、通常ビジネスパートナーに対して公開されている WSDL エンドポイント使った exteNd Composer Web サービスにコンタクトすることとなります。コンタクトした後、この Web サービスは正しいプロセスインスタンスに関連付けられた Web Service Receive アクティビティを検索して、ビジネスパートナーのメッセージを渡し、そしてアクティビティが完了したことを通知する方法が必要となります (後に説明のある *Release Waiting Activity* を参照)。*Find Waiting Activity* アクションは、適切な Web Service Receive アクティビティを配置するニーズを満たします。

注記： 説明全体を通して、アクティビティとアクティビティ実装は同じでないことを念頭に入れておいてください。「アクティビティ」は、プロセス環境に対してのみ意味のある特定の属性と状態を有する抽象的なエンティティです。「アクティビティ実装」は、ソフトウェアによってタスクを実行するビジネスアプリケーションです。アクティビティにはそれに関連付けられた特定のプロパティがあります。プロパティは、［Object Properties］パネルのタブのプロパティシートに示されています。しかし通常、アクティビティで指定されたビジネスタスクを完了するのに必要な実際の作業を実行する実装およびその基盤になるアプリケーションについては認識されないのが普通です。逆に、実装でもこれがプロセスで使用されていることは認識されません。

シナリオ

次のシナリオを想定してください。注文を申込み、注文の最終確認書をビジネスパートナーに送付し、ビジネスパートナーからの連絡を待つというプロセスを定義しました。パートナーは注文番号を確認するメッセージを戻します。この時点では、注文プロセスは継続中です。*Web Service Receive activity* は、ビジネスパートナーからの連絡を待つというプロセスで使用されます (次の図の No. 1 を参照)。このアクティビティの実装は通常、標準の Web サービスです。次に Web サービスは、すべてをインスタンスにバインドしておくために *Correlation ID* を使用します。ビジネスパートナーに対し、プロセスのアップストリームアクティビティ (パートナーを照会したアクティビティ) からこの ID が通知されます。パートナーが最終的な回答を受け取るまでの流れを次の図に示します。



ビジネスパートナーが Web Service A (図中では Web Service Receive アクティビティを実装) に確認メッセージを送信すると、Web Service A は関連アクションとプロセスを検索してこれを起動する必要があります。幸いなことに Web Service A にはこれを正確に行うことのできる Find Waiting Activity アクションが含まれています (図の 2 を参照)。Find Waiting Activity アクションを使用すると、Web サービスではビジネスパートナーからの応答を待機している *Web Service Receive* アクティビティが検索されます。そのアクティビティの入力メッセージと PendingActivity ドキュメントは、アクティビティの出力メッセージを作成するときに利用されます (図の 3 を参照)。PendingActivity ドキュメントを使用すると、Web サービスは *Release Waiting Activity* アクションを実行し、出力を Web Service Receive アクティビティに戻します (図の 4 を参照)、これによって処理が終了しプロセスを継続することができます。

待機中のアクティビティの検索

待機中のアクティビティは次の 2 つのメソッドのいずれかを使用すると検索できます。1 つのメソッドでは Process Name と Correlation ID を組み合わせて使用し、もう 1 つのメソッドでは Activity Name と ProcessID を使用します。

Correlation ID メソッドは、ファイアウォールの反対側にいるビジネスパートナー (たとえば、2 つの別々の会社) とのビジネス対話にもっとも一般的な方法です。Correlation ID は、以前のプロセスでビジネスパートナーと通信したときに作成された任意の固有な値であれば何でも構いません。たとえば、タイムスタンプ、注文番号、確認番号などが使用できます。*Find Waiting Activity* アクションは、指定した入力ドキュメントの場所から CorrelationID を抽出し、次にこの ID をプロセスエンジンに検索用として渡します。

2 番目の検索メソッドは、Activity Name (コンポーネントが一部である Web Service Receive アクティビティの名前など) と問題となっているプロセスインスタンスの ProcessID を組み合わせて作成した固有キーを検索のベースとして使用します。Find Waiting Activity ダイアログボックス (次の図) で、この情報を入力できます。

リクエスト側と応答側が両方とも共通のファイヤウォール内にある場合は、2 番目の検索メソッドの方が一般的です。

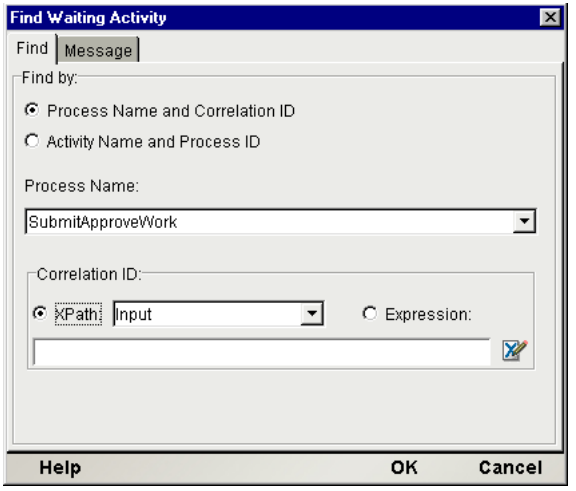
いずれかの検索メソッドを使用して、ビジネスパートナーは入力メッセージに必要な ID 情報を Web サービスに提供することが重要になります。その ID 情報は、Correlation ID または Process ID と Activity Name の組み合わせのいずれかから構成されます。

Find Waiting Activity ダイアログボックス

Find Waiting Activity ダイアログボックスは、2 つのコントロールタブから構成されます。［Find］タブでは、待機中のアクティビティを検索するのに使用する条件を指定します。［Message］タブでは、検索するアクティビティに関してプロセスエンジンから戻された情報を配置する場所を指定できます。

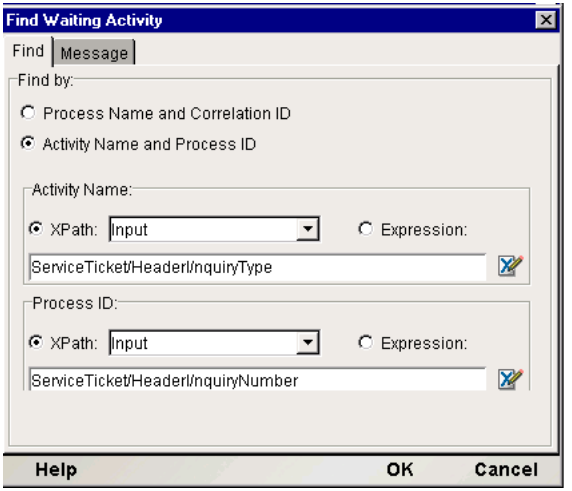
［Find］タブ

［Find］タブは、ダイアログボックスの一番最初で 2 つのラジオボタンのどちらを選択するかによって表示が異なります。［Process Name and Correlation ID］を選択すると、ダイアログボックスは次のように表示されます。



［**Process Name**］と［**Correlation ID**］によってアクティビティを検索する場合は、ドロップダウンリストからプロセス名を選択します。次に、ビジネスパートナーから受け取ったメッセージで、Composer が CorrelationID を検索するための Xpath 式を指定します (または、Expression ラジオボタンをクリックして、必要な ID を評価する ECMAScript 式を指定します)。通常、ビジネスパートナーのメッセージを含むメッセージパートが入力になりますが、それ以外でも可能です。CorrelationID を含む要素の名前は必ずしも「CorrelationID」である必要はないことに注意してください。有効な Xpath 式は次のようになります :

`PurchaseOrder/Header/POID`。

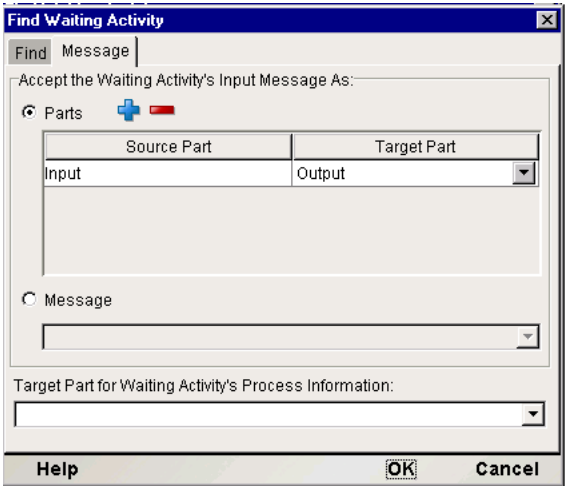


Activity Name と **Process ID** (「Activity Name and Process ID」のラベルが付いているラジオボタンを使用)によってアクティビティを検索する場合は、ビジネスパートナーから受け取ったメッセージで、Composer が Activity Name (Web Service Receive アクティビティの名前など) を検索するための Xpath 式を指定します。通常、ビジネスパートナーのメッセージを含むメッセージパートが入力になりますが、それ以外でも可能です。Activity Name を含む要素の名前は必ずしも「Activity Name」である必要はないことに注意してください。有効な Xpath 式は次のようになります : `ServiceTicket/Header/InquiryType`。同じように、ProcessID の場所を指定します。

[メッセージ] タブ

アクティビティが見つかると、プロセスエンジンは *Find Waiting Activity* アクションを発行した Web サービスに 2 つの XML ドキュメントを戻します。1 つめのドキュメントは、ビジネスパートナーからの連絡を待機する前の、Web Service Receive アクティビティに対する元の入力メッセージです (1 つ以上のパートから構成)。これによって、アクティビティの実装はアクティビティメッセージで使用したり、あるいはビジネスパートナーから受け取ったメッセージとともに参照として使用できるようになります。

プロセスエンジンから戻された 2 つめのドキュメントは、待機中のアクティビティに関するランタイム情報です (次の詳細を参照)。



[Message] タブの最初のセクションでは、Web サービスで作業ができるようにするためアクティビティの元の入力メッセージを Web サービスにマップできます。2 つのラジオボタンは使用可能なオプションを制御します。

- ［**Parts**］ラジオボタンでは、アクティビティの入力メッセージの各パートを Web サービスのパートにマップできます。ほとんどのアプリケーションではこれを選択します。
- ［**Message**］ラジオボタンでは、アクティビティの入力メッセージ全体 (すべてのパートを含む) を Web サービスの単一パートにマップできます。使用できるパートがない場合は、Temp ドキュメントを Web サービスに追加する必要があります。

［Message］タブの 2 番目のセクションでは、プロセスエンジンから戻される待機中のアクティビティのプロセス情報を Web サービスのどのパートで受け取るかを指定できます。ドロップダウンリストからパートを選択します。使用できるパートがない場合は、Temp ドキュメントを Web サービスに追加する必要があります。

PendingActivity ドキュメント

Find Waiting Activity アクションによって戻される 2 番目のドキュメントは、*Release Waiting Activity* アクションによって使用され、Web Service Receive アクティビティの完了を示す信号を送り、引き続きプロセスを実行できるようにします。プロセスエンジンによって戻された待機中のアクティビティについて記述されているドキュメントには、*PendingActivity* と呼ばれるルート要素が含まれます。

PendingActivity ドキュメントには、次のチャイルド要素が含まれます。

- **ProcessID** — Web Service Receive アクティビティが存在する「プロセスインスタンス」に関連付けられた固有の番号です。これは Release Waiting Activity アクションによって使用され、待機中のアクティビティを再開します。
- **QueueDate** — Web Service Receive アクティビティがビジネスパートナーからの連絡を待ち始めたときを示す日付 / 時間のスタンプです。
- **ActivityName** — 待機中の Web Service Receive アクティビティの名前。
- **ProcessName** — Web Service Receive アクティビティに属するプロセスの名前。
- **CorrelationID** — Web Service Receive アクティビティを識別したり検索したりするために使用される固有キー。値は Web Service Receive アクティビティのプロパティとして指定され、アクティビティが実行されて待機を開始したときにプロセスによって設定されます。
- **Addressee** — この Web Service Receive アクティビティのコンタクト先となっているユーザの名前。このデータは通常、長時間実行されるプロセスでユーザの介入 / 操作を含むワークキューアプリケーションで作業する人々に対し作業を割り当てるときに使用されます。値は Web Service Receive アクティビティのプロパティとして指定され、アクティビティが実行されて待機を開始したときにプロセスによって設定されます。このデータは通常、完全にファイヤウォールの後ろで実行されるプロセスの中で使用されます。

- **Priority** — このデータは通常、ワークキューアプリケーションで作業する人々に作業を割り当てるときに使用され、これによってプロセスエンジンを照会するアプリケーションは相対的な重要度によって待機中のアクティビティを分類するできます。値は Web Service Receive アクティビティのプロパティとして指定され、アクティビティが実行されて待機を開始したときにプロセスによって設定されます。このデータは通常、完全にファイヤウォールの後ろで実行されるプロセスの中で使用されます。
- **LockedBy** — このラベルは通常、ワークキューの中でそのアクティビティを単独でロックし、排他的に行うようフラグが付けられたユーザ個人の名前に対して付けられます。実際のロックは作成されておらず、プロセスを照会するワークキューアプリケーションに対する信号あるいはフラグ値となります。値は *Lock/Unlock Waiting Activity* アクションによって設定されます。
- **LockedUntil** — ロック解除の時期を示す日付値です。値は *Lock/Unlock Waiting Activity* アクションによって設定されます。

Release Waiting Activity アクション

Release Waiting Activity アクションは、Web Service Receive アクティビティの実装内部 (通常は Web サービス) で使用されます。このアクションの前には通常、実装のアクションモデルのある時点で、*Find Waiting Activity* アクションが実行されます。アクションはデータを待機中の Web Service Receive アクティビティに渡します。そしてそのデータは出力メッセージとなり、プロセスエンジンにアクティビティの完了を示す信号の役割をします。渡されたデータは通常 Web サービスにコンタクトしたビジネスパートナーからの情報です。このように、*Release Waiting Activity* アクションは、Web サービスで使用されるコールバックのメカニズムとなっており、プロセス内で Web Service Receive アクティビティの出力メッセージを生成し、アクティビティの完了を示す信号の役割をします。

アクティビティを解放する前に、解放したいプロセスおよび Web Service Receive アクティビティを示した *PendingActivity* ドキュメントを保持するメッセージパートを生成する必要があります。これは、*Find Waiting Activity* アクションにより行うことができます (前の節を参照)。さらに、`output` メッセージとして待機中の Web Service Receive アクティビティに戻される 1 つ以上のパートが必要です。PendingActivity ドキュメントとアクティビティの出力としての以上のパートが準備できたら、アクティビティを解放することができます。

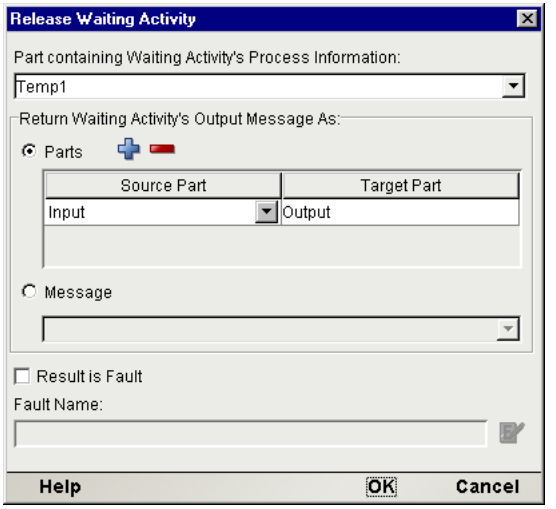
Release Waiting Activity ダイアログボックス

Release Waiting Activity ダイアログボックスには 3 つのセクションがあります。

最初のセクションでは、解放したいプロセスおよびアクティビティについて記述した PendingActivity ドキュメントを保持する Web Service のパートを指定できます。このパートは、*Find Waiting Activity* アクションによってすでに生成されている必要があります。

2 番目のセクションでは、Web サービスのパートを待機中の Web Service Receive アクティビティの出力メッセージパートにマップできます。

3 番目のセクションはオプションで、出力をアクティビティに戻すことができますが、障害のあるリンクを追跡するためプロセスフローが Web Service Receive アクティビティから外れてしまったエラーメッセージとしてフラグが付けられます。



［Part containing Waiting Activity’s Process Information］の下は、ドロップダウンリストからパート名を選択するだけです。

注記：リストにパートが表示されるようにするには、パートに *PendingActivity* ドキュメントが含まれている必要があります。

ダイアログ 2 番目のセクションでは、待機中のアクティビティの出力メッセージになる Web サービスからのデータを指定できます。［**Parts**］セクションでは、Web サービスの 1 つまたは複数のパートを Web Service Receive アクティビティの出力メッセージの 1 つまたは複数のパートにマップできます。ほとんどのアプリケーションではこれを選択します。入力した［Target Part］名は、アクティビティの出力メッセージの中で作成されます。［**Message**］オプションでは、Web サービスの単一パートをアクティビティの出力メッセージ全体 (すべてのパートを含む) としてマップできます。

ダイアログ 3 番目のセクションでは、戻されたデータをプロセス内でエラーのフラグを付けるよう指定できます (オプション)。Expression Builder を使用して、［Messages］タブで Web Service Receive アクティビティ用に定義された名前と対応しているエラーメッセージ名を指定します。

プロセスへの人間の参加

ほとんどのプロセスでは、プロセスの最初の起動以外に何も目的がない場合以外、何らかのオペレータの介入が必要となります。シナリオによっては、特定のビジネスタスクのすべての局面において人間の参加が必須な場合があります。注文書に個人の承認が必要な場合、製品の問い合わせに個人的な電子メールによる回答あるいは電話が必要な場合、大規模なトランザクションで特定の個人に対するエスカレーションが必要な場合などが考えられます。

Composer の Process Manager を使用して人間を中心とした高度なワークフローを実装することができます。Process Manager には次のことを簡単に行えるようにする機能が備わっています。

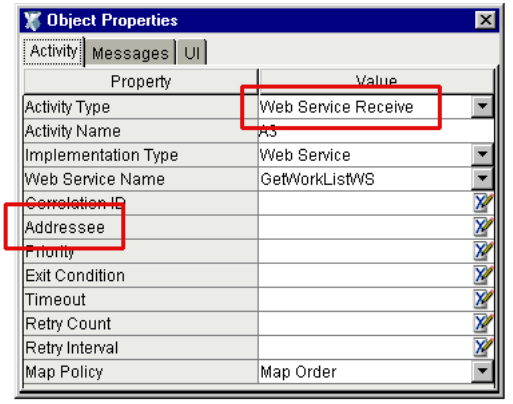
- 個人への作業の割り当て (再割り当てまたは再ルーティング)
- ワークアイテムの優先順位割り当て
- 個人による排他的使用を行えるようにするためワークアイテムロックまたはロック解除のマークを付与
- 個人別にフィルタリングされた、ワークリストのプロセスの表示
- 個人別ワークアイテムの取得
- バックエンドシステムをワークフローに統合
- JSP または HTML を介して簡単なフロントエンドアクセスをワークリストに統合

プロセスへの人間のエントリポイントを作成するには、通常 Web Service Receive アクティビティを使用して外部向けアプリケーションを公開します (Web Service Receive アクティビティの実装など)。ユーザ向けのサービスは通常 JSP あるいは HTML ページを通じて公開されますが、他の方法でも構いません。

ユーザ向けのサービスは、ユーザがワークキューの表示、ワークアイテムの検索とロック、ワークアイテムのロック解除、作業の再割り当て、あるいはワークアイテムをシステムに戻すことなどを実行できるように設計できます。これらの操作を可能にするためのアクションには、Find Waiting Activity アクション、Release Waiting Activity アクション、Browse Waiting Activities アクション、Lock/Unlock Waiting Activity アクションそして Reassign Addressee アクティビティが含まれます。

Addressees

プロセスに人間が参加することは、Process Manager で *Addressees* として知られています。ランタイムエンジンは、Web Service Receive アクティビティタイプの Web Service Receive パネルで Addressee プロパティを使って、Addressee を特定のドキュメントあるいはワークアイテムに関連付けます。



注記： Addressee は Web Service Receive アクティビティだけのプロパティです。他のアクティビティタイプの Object Properties パネルではこのフィールドは表示されません。

Addressee の値は、Xpath 式として指定されます。これによって、Addresse を渡されたメッセージパートの中に入れることができるため、ランタイム時に動的に Addressee を決定したり、あるいは特定のストリング値にハードコード化したりできるため、大幅な柔軟性が出てきます。このように、ユーザは任意の次の共通シナリオを設定できます。

- 注文書の処理を行う販売員は、オンラインで注文が入った時点で JSP スクリプレットまたは EJB によって決定されます。ProcessInput メッセージには、プロセスの呼び出し時にすでに必要な Addressee 名が含まれています。
- Web 経由で注文が入り、プロセスが起動されます。Addressee は、コンポーネントの事前処理の際にビジネスロジックによってジャストインタイムで動的に決定され、アクティビティの出力メッセージに表示されます。
- すべての注文は最終的に John Smith によって承認される必要があります。したがって、Web Service Receive アクティビティは「John Smith.」の Addressee 値にハードコード化されます。

Web Service Receive アクティビティの役割

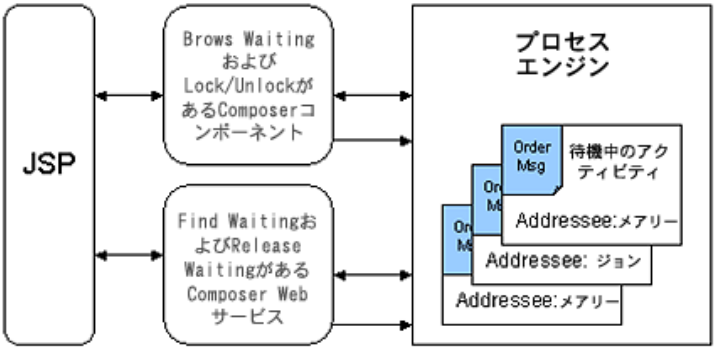
Web Service Receive アクティビティは、Process Manager によって作成された自動化プロセスに対するオペレータ入力の主要な接点となります。

Web Service Receiveアクティビティが起動されると、次の3つのことが発生します。

- Addressee プロパティはストリング値と関連付けられます (必須ではありませんが、通常は実在の人物の名前を表します)。
- そのアクティビティの基盤になる実装 (すなわち、WSDL のリクエスト - 応答または一方向ポートタイプに従って動作する Web サービス) が、操作可能になります。
- プロセスエンジンは、Web Service Receive アクティビティ (実装ではありません) を待機中の状態にします。

その人 (Addressee すなわち作業員あるいは管理者など) は、次に JSP に実装されているワークグループアプリケーションを使って Composer コンポーネントあるいはサービスを実行し、次のことを行うことができます。

1. 特定の人物にアドレス指定されたすべての待機中のアクティビティを表示します (Browse Waiting Activities アクションを使用)。
2. 管理者が待機中のアクティビティをレビューしている間は、他のユーザが注文を承認してしまうのを防ぐため待機中のアクティビティをロックします (Lock/Unlock Waiting Activity アクションを使用)。
3. 注文をロック解除します (Browse Waiting Activities アクションを使用)。
4. 注文を取得して承認済みの印を付けます(Browse Waiting Activitiesアクションを使用)。
5. 承認を完了し、プロセスが次のアクティビティを処理できるようにします(最終的には Release Waiting Activity アクションを呼び出す)。



待機中のアクティビティだけが実際に Web Service Receive アクティビティを完了 (つまり終了) させることができることに注意してください。したがって定義によって、Web Service Receive アクティビティ (たとえば Web サービス) には必ず Release Waiting Activity アクションを含めておく必要があります。その他のアクション (Browse Waiting Activities、Lock/Unlock Waiting Activity、Reassign Addressee および Find Waiting Activity) も、プロセスに直接的に接続されていない様々なコンポーネントで使用できます。このようなコンポーネントは、これらのアクションを使用して外部のアプリケーションに監視および参照機能を追加することができ、ユーザに待機中のアクティビティを表示したり管理したりする方法を提供できます。

Browse Waiting Activities Action

Browse Waiting Activities アクションは、プロセスおよびサービス / コンポーネントが Composer プロジェクトのパートとして配備されている限り、(たとえプロセスの外部であっても) 任意のサービスあるいはコンポーネントで使用できます。Browse Waiting Activities アクションの唯一の目的は、Addressee によってフィルタされた、保留中のリスト (1 つまたは複数) をアプリケーションで取得できるようにすることです。

Browse Waiting Activities アクションを実行する場合、ユーザは Process Server の名前あるいは名前のリストを指定するだけです。Process Server は、すべてのプロセスのすべてのインスタンスにおけるすべての Web Service Receive アクティビティを検査して、今問題となっている個人あるいは組織によってアクティブになるのを待機しているアクティビティのリストを返します。

ブラウザに応答して Process Server が戻したリストは、*PendingActivity* ドキュメントです。このドキュメントには、保留中のアクティビティの ProcessID と Activity Name が他の情報と一緒に含まれており、これらの情報は待機中のアクティビティに代わって検索や作業を行うときに使用することができます。(PendingActivity ドキュメントの構造の詳細については、「PendingActivity ドキュメント」の説明を参照してください。)

Browse Waiting Activities アクションを使用する場面

Browse Waiting Activities アクションは通常、JSP などの他のアプリケーションを介して実際の人間による介入や対話が必要な、長時間実行されるプロセスのシナリオの中で利用されます (詳細については、『eXtend Composer Silverstream Server Guide』の「Creating a JSP that calls a Composer Service」の節を参照してください)。たとえば、注文を一度に同時処理する非常に負荷の高いプロセスがあったとします。プロセスには、単一の注文を受け取り Web Service Receive アクティビティに渡すアクティビティが含まれており、その間プロセスは停止し、注文が特定の人物によって承認されるのを待ちます。そのオペレータは、代わりに JSP (Java Server Page) を使って承認を入力しておきます。このようなシナリオにおいて、Addressee (注文を承認する人物) は処理すべき作業を検索 (すなわち発見) し、その処理が終わったら直ちに作業をシステムに戻すことができる必要があります。発見のパートは、JSP によっても起動可能なサービスの Browse Waiting Activities アクション (プロセスの内部である必要なありません) を介して完了できます。データ入力のパートは、Web Service Receive アクティビティタイプを実装するサービスによって行うことができます。このサービスは、Find Waiting Activities アクションを使って個別のワークアイテムを検索し、Release Waiting Activity アクションによって「プッシュ」を実行します。

動作

待機中のアクティビティ(どのプロセスに含まれているかに関わらず) は、Browse Waiting Activity アクションによって、Addressee だけが指定できます。しかしこれが動作するためには、関連する Web Service Receive アクティビティに空でない Addressee プロパティが含まれている必要があります。Addressee の値を指定するには、Process Designer のプロセスグラフを開いて、問題となっている Web Service Receive アクティビティをクリックし、［Object Properties］パネルをビューに移動して、Addressee の隣に有効な Xpath の値を入力します (「Addressees」の下のスクリーンショットを参照)。XPath は、Addressee の文字列またはハードコード化された文字列値を含む入力メッセージパートのいずれかをポイントしていなければなりません。

Priority と呼ばれる 2 番目の Web Service Receive アクティビティプロパティもまた、Object Properties パネルで設定することができます。［Priority］は、ユーザに表示する前に、取得したワークアイテムをアプリケーションで分類したりフィルタリングしたりできるようにするための任意の数値です。任意の数値を割り当てたり、または値を指定しないで空のままにしておくこともできます。

ほとんどのアプリケーションにおいて、Browse Waiting Activities の後には Lock/Unlock Waiting Activities、Reassign Addressee、または Find (あるいは Release) Waiting Activity、あるいはこれらすべてのプロセスが続きます。たとえば、考えられる一つのシナリオは次のようになります。あるワークグループの管理者は、ユーザのグループに対して複数の待機中のアクティビティを選択します (Browse Waiting Activities アクションを使用)。管理者は、グループのユーザがレビューをしている間、他のユーザが作業中のワークアイテムを更新してしまうのを防ぐため選択したすべてのアクティビティにロックをかけます (Lock/Unlock Waiting Activities アクションを使用)。管理者はユーザにいくつかのワークアイテムの再割り当てをし (Reassign Addressee アクションを使用)、優先度の高いワークアイテムを検索して作業を行い (Find Waiting Activities アクションを使用)、作業を完了して (Release Waiting Activity アクションを使用)、次に作業の対象となっていないアクティビティのロックを解除します (Lock/Unlock Waiting Activities アクションを使用)。

表示と検索の比較

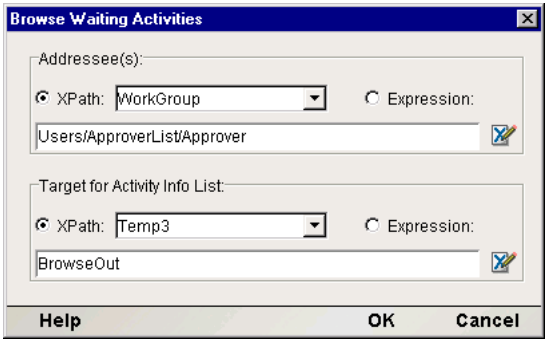
Browse Waiting Activities アクションは、次の点において Find Waiting Activities アクションと異なります。

- Browse Waiting Activities アクションでは Addressee のみが待機中のアクティビティを検索できるのに対して、Find Waiting Activities アクションでは Process Name/CorrelationID または Activity Name/ProcessID によってのみ検索できます。
- Browse Waiting Activities アクションは複数のアクティビティに情報を返すことができるのに対して、Find Waiting Activities アクションでは単一の待機中のアクティビティに対してのみ情報を返します。
- Browse Waiting Activities アクションは結果をメッセージパートまたは Xpath ロケーションにマップしますが、Find Waiting Activities アクションはパートに対してのみ結果をマップします。
- Browse Waiting Activities アクションは検索されたアクティビティに入力メッセージを返しませんが、Find Waiting Activities アクションは待機中のアクティビティの入力メッセージを返します。このように Browse Waiting Activities アクションと Find Waiting Activities アクションを組み合わせて利用することにより、管理者は実際の注文を検索するなど待機中のアクティビティの詳細について調べることができます。

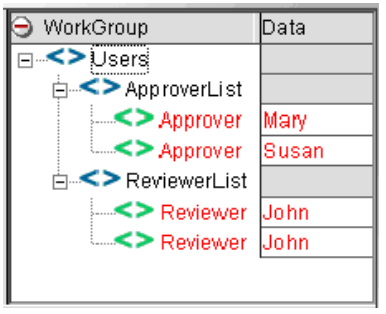
注記 : Find Waiting Activities アクションと Find Waiting Activities アクションの両方とも「非破壊」です。Release Waiting Activity アクションが呼び出されるまで、待機中のアクティビティはいずれも完了したことにはなりません。

Browse Waiting Activities アクションの作成

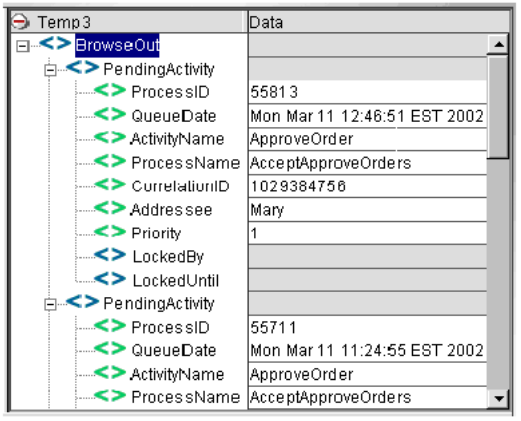
Browse Waiting Activities アクションを作成するには、コンポーネントに移動してアクションモデルで右クリックします。次にコンテキストメニューから［**New Action**］、［**Process**］、［**Browse Waiting Activities**］の順に選択します。ダイアログボックスが表示されます。



Browse Waiting Activity ダイアログボックスには、基本的な 2 つのコントロールグループがあります。最初のコントロールグループは、コンポーネントのメッセージパートの Addressee のリストをポイントする方法を提供します。ノードリストに含まれる値は、待機中のアクティビティを検索するための検索キーとして使用されます。たとえば、上のダイアログボックスの場合、［Addressee Xpath］は以下に示されている［Approvers］のリストをポイントします。



2 番目のコントロールグループでは、表示の結果を配置する場所を指定できます。パート名 (たとえば Temp) とパート内の Xpath ロケーションを指定します。表示の結果は、指定した Xpath のチャイルド要素として配置されます。



表示が成功すると 1 つまたは複数の *PendingActivity* ドキュメントを返し (上図を参照)、各ドキュメントには待機中のアクティビティについて記したチャイルド要素が含まれています。待機中のアクティビティがない場合は、指定した Xpath だけが作成され、その下には PendingActivity のチャイルド要素はありません。

注記： Find Waiting Activity アクションとは異なり、Browse Waiting Activities アクションは見つかったアクティビティの入力メッセージは返しません。入力メッセージを取得するには、Repeat for Element アクションを使って各 PendingActivity 要素をループし、気になるアクティビティに対して Find Waiting Activity アクションを実行する必要があります。(PendingActivity の分岐には、指定されたアクティビティを取得するために Find Waiting Activity で必要なすべての情報が含まれています。)

Lock/Unlock Waiting Activity

Lock/Unlock Waiting Activity アクションは、待機中の Web Service Receive アクティビティに使用中であることを示すフラグを設定したり、あるいはフラグをクリアして、待機中のアクティビティが使用可能になったことを示したりします。フラグは、2 つの要素 `/LockedBy` と `/LockedUntil` から構成され、*PendingActivity* ドキュメントと待機中の Web Service Receive アクティビティが関連付けられます。`LockedBy` 要素がヌル値でない場合、待機中のアクティビティはアクティビティが使用可能でないことを示します。ロックのフラグが設定されている場合、`LockedUntil` 要素にはロックまたはフラグがプロセスエンジンによって自動的に解除されるときを示す日付と時間のスタンプが含まれています。

重要な点として、Lock/Unlock Waiting Activity は待機中のアクティビティに物理的にロックしているわけではないことに注意してください。ただ単にアクティビティが使用中であることを示しているだけです。使用中であることを示すフラグが設定されていても、一連の Find Waiting Activity/Release Waiting Activity アクションは動作可能で、アクティビティを完了できます。ロックの重要性についてはワークグループアプリケーションの設計者次第です。

注記：ロックされているアクティビティは Browse の結果から排除されません。Browse から戻された *PendingActivity* 情報には、ロックが設定されているものとロックされていないものの両方を含む適用可能なすべてのアクティビティが表示されています。

Lock/Unlock Waiting Activity アクションは通常、JSP などの他のアプリケーションを介して実際の人間による介入や対話が必要な、長時間実行されるプロセスのアプリケーションの中で利用されます。(詳細については、『Composer Silverstream Server Guide』の「Creating a JSP that calls a Composer Service」の節を参照してください。) これらのワークグループは、共通して Addressee にワークアイテムが割り当てられているワークキューを使用します。

アクティビティのロック / ロック解除の前提条件

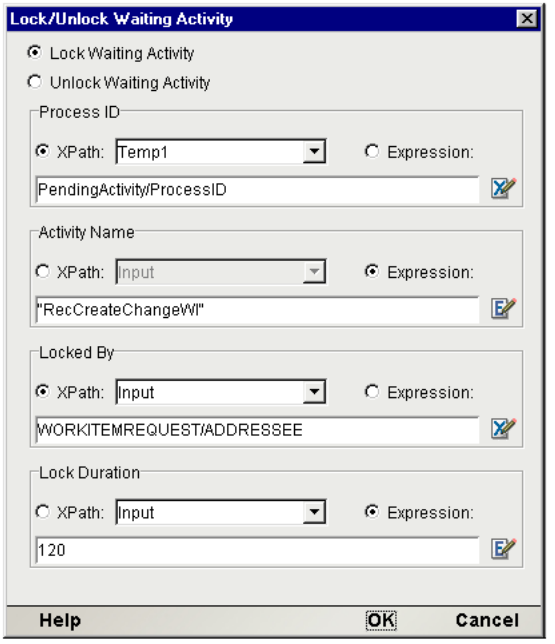
待機中のアクティビティをロックまたはロック解除する前に、待機中のアクティビティを検索するための *ProcessID* と *Activity Name* を把握しておく必要があります。ロック / ロック解除アクティビティは通常、アクションモデルの中でその前に Browse Waiting Activities アクションあるいは Find Waiting Activity アクションが正しく終了しています。いずれかのアクションの結果が *PendingActivity* ドキュメントで、そこから必要な ProcessID と Activity Name を参照できます。

ロックまたはロック解除しようとしているアクティビティがもはやプロセスエンジン内に存在しない場合、Composer は例外を出します。したがってロックについて練習しておくことはとても良いことです (たとえば、Try/On Error アクションの中で Lock/Unlock Waiting Activity アクションを設定してみるなど)。

Lock/Unlock Waiting Activity アクションが正しく終了した場合は (例外が出されない)、何も戻されず、アクションモデル内で次のアクションが実行されます。

Lock/Unlock Waiting Activity アクションの作成

Lock/Unlock Waiting Activity アクションを作成するには、コンポーネントに移動してアクションモデルで右クリックします。次にコンテキストメニューから［**New Action**］、［**Process**］、［**Lock/Unlock Waiting Activities**］の順に選択します。ダイアログボックスが表示されます。

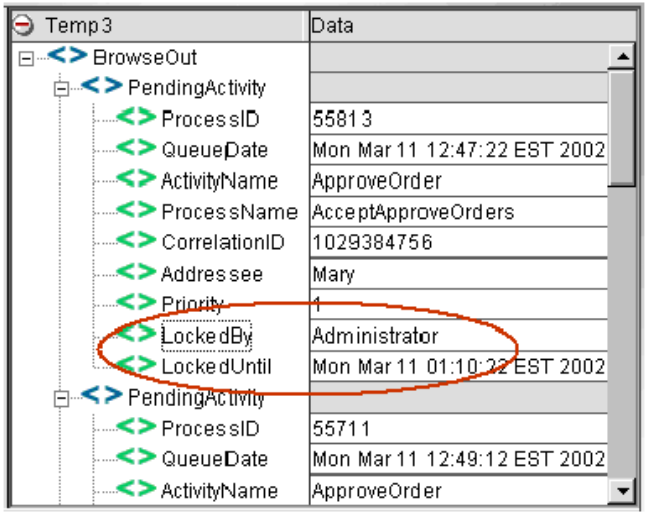


Lock/Unlock Waiting Activity ダイアログボックスには、5 つのセットのコントロールがあります。

- 最初のコントロールグループには、アクションにロックを設定するかあるいは既存のロックをクリアするのかを決定する2つのラジオボタンがあります。
- 2 番目のコントロールでは、ターゲットの［**ProcessID**］を指定します。ProcessID 要素の下の［*PendingActivity*］ドキュメント内に Xpath を指定します。
- 3 番目のコントロールでは、ターゲットの［**Activity Name**］を指定します。Activity Name 要素の下の［*PendingActivity*］ドキュメント内に Xpath を指定します。
- 4 番目のコントロールは、［**LockedBy**］フラグです。ロックのフラグが設定されている間に待機中のアクティビティに対して Browse または Find が実行されたとき、ロック設定したのは誰であるか人間またはプロセスで予測できるような意味のある値を指定します。
- 5 番目のコントロールは、**Lock Duration** です。*PendingActivity* ドキュメントの待機中のアクティビティに関連付けられた LockedUntil エレメントに配置する日付 / 時間のスタンプを計算するのに使われる間隔を指定します。時間間隔を測るデフォルトの単位は秒です。したがって、テキスト 60 を入力するとロックのフラグは 60 秒間設定され、その後自動的にフラグは解除されます。その他の計測の単位には、分 (単一引用符で囲み '60m' と指定)、時 (単一引用符で囲み '60h' と指定)、日 (単一引用符で囲み '60d' と指定) があります。

注記： ［Lock Waiting Activity］ラジオボタンを選択すると、すべてのコントロールの値が必要となります。［Unlock Waiting Activity］ラジオボタンを選択すると、［Process ID］と［Activity Name］のコントロールの値だけが必要となります。

他のユーザ (たとえば、May) が待機中のアクティビティを表示すると、スクリーンショットに表示される Lock ダイアログボックスの設定は次のようになります。



Reassign Addressee アクション

Reassign Addressee アクションでは、待機中の Web Service Receive アクティビティに割り当てられた Addressee 属性の値を変更できます。ほとんどの場合、Addressee の元の値は、Web Service Receive アクティビティが待機状態に入ったときに Process Manager によって設定されます。いったん待機状態に入ると、Reassign Addressee アクションによって現在の Addressee の値を変更することができます。また、オプションで 1 つの Web Service Receive アクティビティの現在の Addressee、あるいはその人物の Web Service Receive アクティビティのすべての Addressee を再割り当てすることができます (たとえば、Mary が病気で休みのとき、Mary のすべての作業を Joe に再割り当てできます)。

Addressee はオプション属性で、Web Service Receive アクティビティに割り当てることができます。Addressee の値が指定されていてもいなくても、外部のワークグループアプリケーションに対するフラグあるいはタグという点を除いては、Web Service Receive アクティビティの処理に影響を与えることはありません。

Reassign Addressee アクションは通常、JSP によって起動されるアプリケーションあるいはフォームなど他のアプリケーションを介して実際の人間による介入や対話が必要な、長時間実行される Composer プロセスのアプリケーションの中で利用されます。ワークグループアプリケーションでは、Addressee にワークアイテムが割り当てられているワークキューを共通で使用することができます。

Addressee の再割り当て

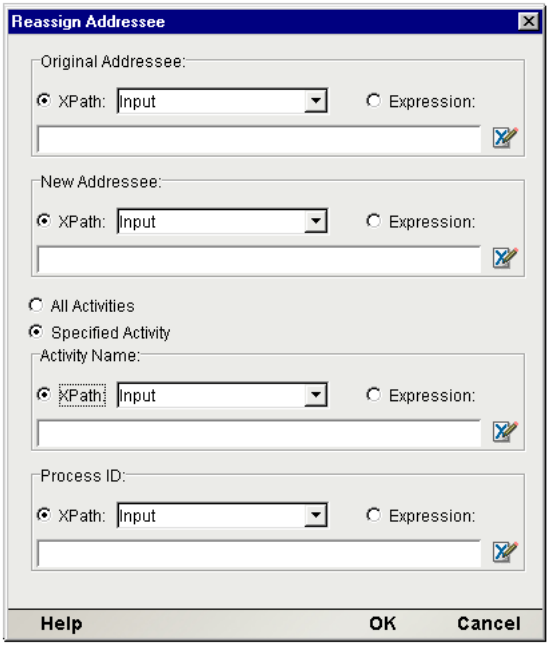
Addressee を待機中のアクティビティに再割り当てする前に、特定の Addressee に関連付けられている 1 つのアクティビティ、あるいはすべてのアクティビティを再割り当てするのかを決めておく必要があります。アクションにすべてのアクティビティを再割り当てさせたい場合は、そのアクション用の次の 2 つのパラメータを定義するだけです。Xpath あるいは ECMAScript は現在の Addressee を識別し、そして Xpath あるいは ECMASCript は新規の Addressee を識別します。

現在の Addressee の特定の 1 つのアクティビティだけを再割り当てしたい場合は、ProcessID と Activity Name も指定する必要があります。これを行うためには、Reassign Addressee アクションの前に、アクションモデルの中で Browse Waiting Activities アクションあるいは Find Waiting Activity アクションが正しく終了していなければなりません。Browse Waiting Activities アクションあるいは Find Waiting Activity アクションの結果が PendingActivity ドキュメントとなり、そこから必要な ProcessID と Activity Name を参照できます。

Reassign Addressee アクションが実行された後、それが正しく実行されても失敗しても、何も返されません。アクションが正しく実行されたことを確認するには、Browse を実行します。

Reassign Addressee アクションの作成

Reassign Addressee アクションを作成するには、コンポーネントに移動してアクションモデルで右クリックします。次にコンテキストメニューから［**New Action**］、［**Process**］、［**Reassign Addressee**］の順に選択します。ダイアログボックスが表示されます。



Reassign Addressee ダイアログボックスには、複数のコントロールグループがあります。最初のコントロールは、現在の Addressee (作業が再割り当てされる人物) を識別し、2 番目のコントロールは新規の Addressee を識別します。それぞれについて、現在のアクションモデルのパートから Xpath ロケーションを入力するか、あるいは正しい Addressee 名を解決する ECMAScript 式を入力します。それぞれの値は通常、Reassign Addressee アクションが使用されているコンポーネントに渡されます)。

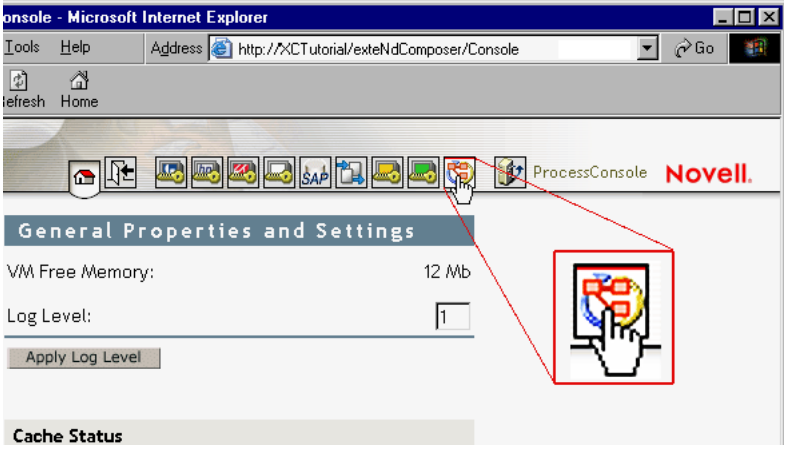
ダイアログボックスの真中の［**All Activities**］ラジオボタンと［**Specified Activity**］ラジオボタンは、現在の Addressee のすべてのアクティビティを再割り当てするのか (Mary のすべての作業を Joe に再割り当てするケースのような場合) 、あるいは特定の 1 つのアクティビティだけを再割り当てするのかを決めます。［Specified Activity］を選択した場合、［**Activity Name**］および［**Process ID**］と呼ばれるコントロールグループが有効となり、再割り当てする特定の［**Activity**］と、そのアクティビティが含む特定の［**ProcessID**］を識別する Xpath 式または ECMAScript 式を入力する必要があります。これらの値を指定するためには、通常 Browse Waiting Activities アクションあるいは Find Waiting Activity アクションを実行します。

7 プロセスのランタイム管理

この章では、Process Server コンソールを使用して、配備済みプロセスを管理する方法について説明します。

Server コンソールの使用法

Composer Enterprise Server のメインコンソールページ (次の図を参照) から、最上部のボタンの行にある［Process Console］アイコンをクリックします。



注記： これらのコンソールにアクセスする前に、アプリケーションサーバに Composer Enterprise Server と Process Server がインストールおよび実行されている必要があります。

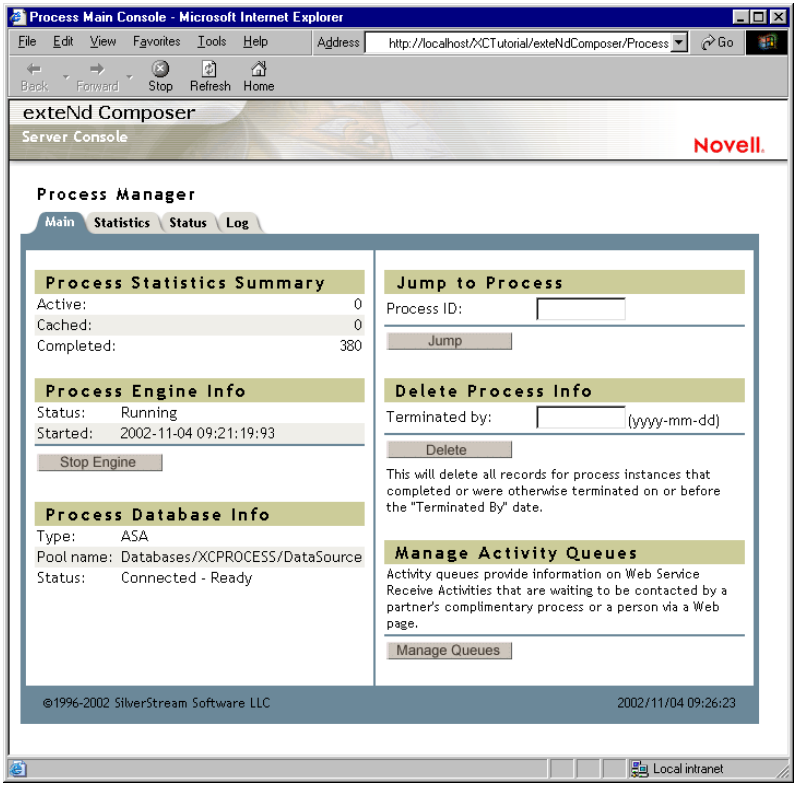
［Process Console］ボタンをクリックすると、次の節に示すように、新しい画面が新しいブラウザウィンドウに表示されます。4 つのタブ (［**Main**］、［**Statistics**］、［**Status**］、および［**Log**］) が存在することに注意してください。次の節では、これらのタブについて説明します。

Process Manager コンソール：［Main］タブ

コンソールの［Main］タブをクリックすると、次のセクションで構成される画面が表示されます。

- ［Process Statistics Summary］
- ［Process Engine Info］
- ［Process Database Info］
- ［Jump to Process］
- ［Delete Process Info］
- ［Manage Activity Queue］

次に、これらの各セクションについて説明します。



[Process Statistics Summary]

[Process Statistics Summary] セクションには、[Active]、[Cached]、および [Completed] の状態のプロセスの数が表示されます。最後の [Completed] は、Process Server の起動以降 (つまり、次の節で説明する起動日以降) に実行されたプロセスインスタンスの数を、成功したか、それとも何らかの障害が発生したかに関係なく、参照します。

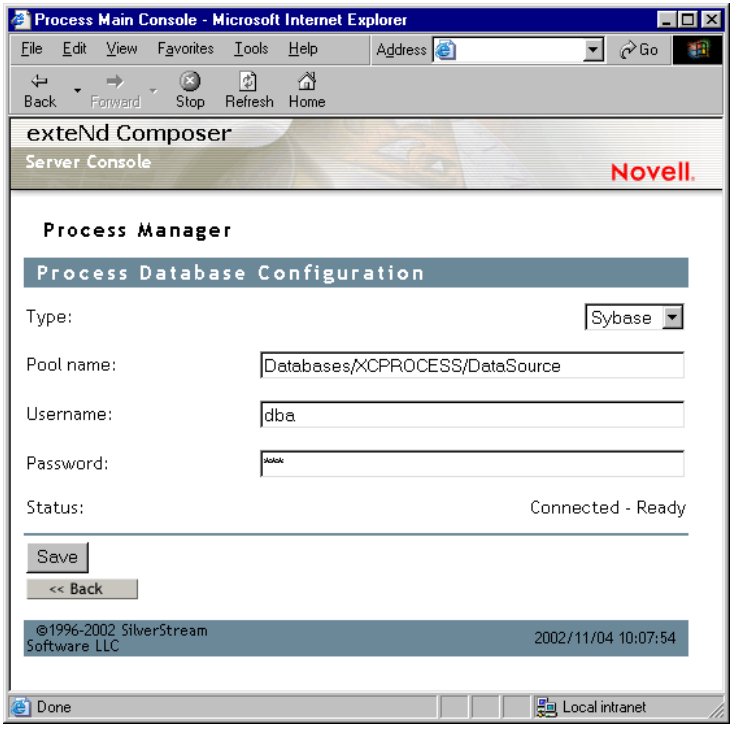
[Process Engine Info]

[Process Engine Info] セクションには、プロセスエンジンが実行中かどうかが表示され、実行中の場合は、プロセスエンジンが起動された日付と時刻も表示されます。プロセスエンジンが実行中の場合、[Process Engine Status] は「Running」になり、下のボタンに「Stop Engine」というラベルが付きます。プロセスエンジンが実行中で「ない」場合、[Process Engine Status] は「Suspended」になり、下のボタンに「Start Engine」というラベルが付きます。

[Process Database Info]

[Process Database Info] セクションには、プロセスデータベースに関する一般的な情報が表示されます (このデータベースの設定の詳細については、このガイドの最初の数ページの他に、製品リリースノートも参照してください)。プロセスデータベースは、長時間実行されるプロセスの「状態データ」を維持するために Process Manager が使用するデータベースです。

- [**Type**] — データベースのタイプ (Oracle、DB2、ASA など)。
- [**Pool Name**] — 接続プールの名前。
- [**Status**] — ステータスは、次のいずれかの場合があります。
  - *Not Connected to Database*
  - *Can't Connect to Database*
  - *Connected—Not Initialized*
  - *Connected—Ready*
- [**Configure**] — [Configure] ボタンは、Process Engine が停止している場合にのみ表示されます。[Process Database Info] セクションの [**Configure**] ボタンをクリックすると、[Process Database Configuration] ページが表示され、このページからデータベースを設定できます (次の図を参照してください)。



- データベースを設定するには、ドロップダウンリストからデータベースタイプ (Oracle、DB2、ASA など) を選択して、プール名を入力します。[**Save**] ボタンをクリックして、設定を保存できます。設定を保存すると、[**Initialize Database**] ボタンをクリックして、データベースを初期化できます。[Initialize Database] ボタンは、ステータスが「*Connected—Not*」の場合にのみ表示されます。

[Jump to Process]

メインコンソールの [Jump to Process] セクションでは、プロセスの ID を入力して [Jump] ボタンをクリックすることで、特定のプロセスのステータスを表示できます。

[Delete Process Info]

[Delete Process Info] セクションを使用して、プロセスレコードを完全に削除できます。特定の日付までに終了したプロセスインスタンス (完了または終了したプロセスインスタンス) の情報をすべて削除できます。このためには、[Terminated By] に日付を入力して、[**Delete**] ボタンをクリックします。たとえば、「2002-02-01」と入力して [**Delete**] ボタンをクリックすると、2002 年 2 月 1 日までに完了または終了したプロセスインスタンスのすべてのレコードが完全に削除されます。

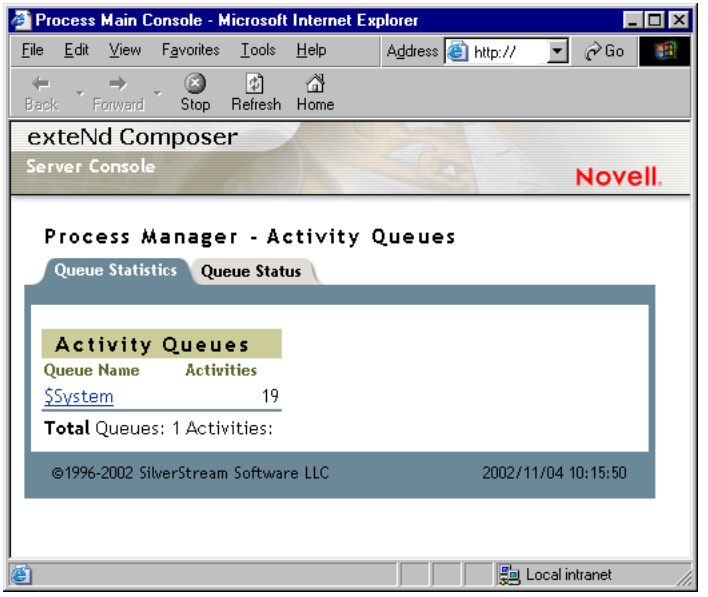
注記： プロセスの実行が完了した場合、またはプロセスを手動で終了した場合は、プロセスインスタンスの入力ドキュメントおよび出力ドキュメントのみが維持されます。プロセスインスタンスによって作成される中間ドキュメントは、プロセスインスタンスの終了時に自動的にパージされます。

［Manage Activity Queues］

［Main］タブの［**Manage Activity Queues**］ボタンをクリックして、アクティビティキューを管理できます。このボタンをクリックすると、キューの統計情報とステータスを提供する 2 つのタブの付いたページが表示されます。

［Queue Statistics］

［Queue Statistics］タブには、アクティビティキュー内の Addressee のソート済みリスト、およびその Addressee に割り当てられているワークアイテムの数を含むテーブルが表示されます。これらの統計情報は、自動的に 60 秒おきに更新されます。

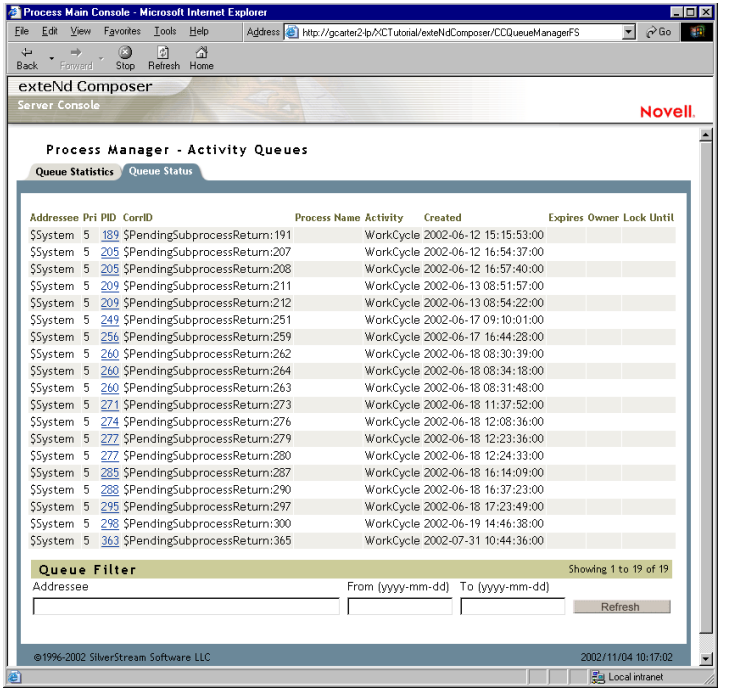


［Queue Status］

［Queue Status］タブ (次の図を参照) には、次の列で構成されるテーブルが表示されます。

- ［**Addressee**］— Addressee の名前
- ［**Priority**］— 優先度
- ［**PID**］— プロセス ID

- ［**Corr ID**］— 相関 ID
- ［**Process Name**］— プロセスの名前
- ［**Activity**］— アクティビティの名前
- ［**Created**］— アクティビティのインスタンスの作成日
- ［**Expires**］— アクティビティのインスタンスの有効期限
- ［**Owner**］— ロックの所有者
- ［**Lock Until**］— ロックの有効期限



［PID］の列には、プロセスインスタンスのプロセス詳細情報へのホットリンクが含まれます (リンクは、新しいブラウザウィンドウで開きます)。［Process Detail］ウィンドウについては、後の節で説明します。

アクティビティにタイムアウトが設定されていない場合、［Expires］の列は空白です。

Lock Waiting Activity アクションを使用してアクティビティがロックされている場合、［Owner］列には、ロックされているアクティビティの所有者の名前が表示され、［Lock Until］列には、アクティビティのロックが継続される日付が表示されます。アクティビティにロックが設定されていない場合、［Owner］および［Lock Until］フィールドは空白です。

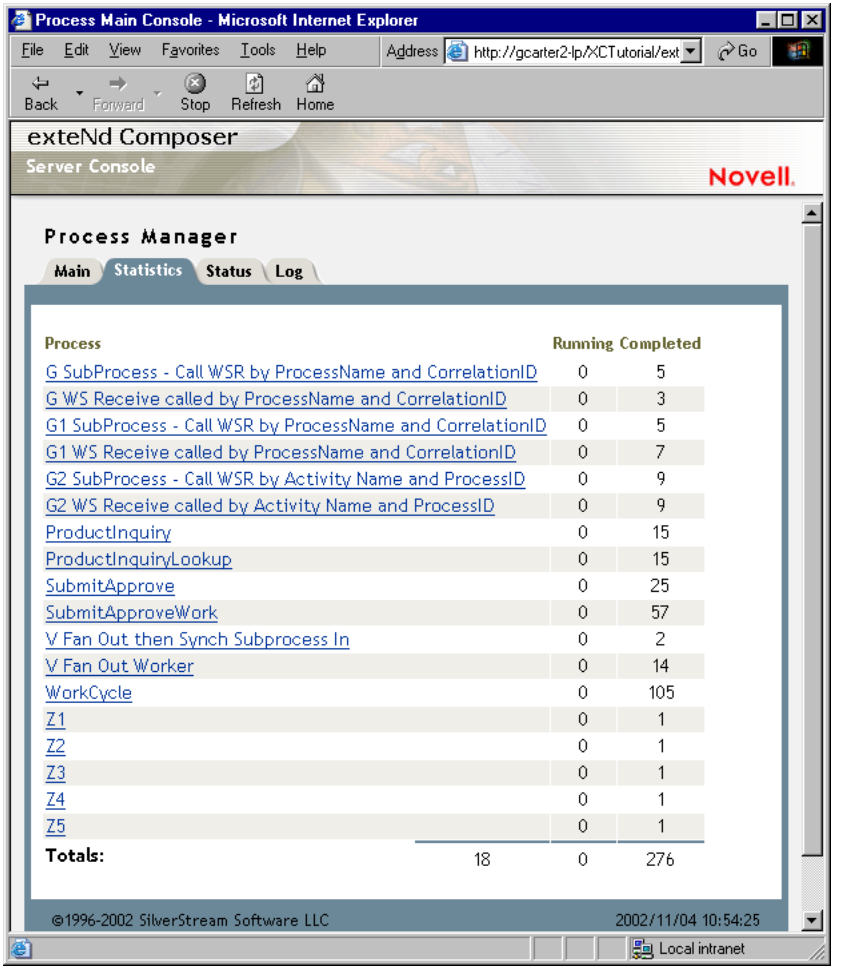
オプションで、Addressee の名前を入力して、［Queue Status］を Addressee を基準にフィルタできます。また、［From］および［To］フィールドに日付の範囲を入力して、特定の範囲内の日付にキューに入れられたアクティビティのみを表示するよう選択することもできます。

ナビゲーション

［Queue Status］タブには、Composer Process Manager コンソールの他のページと同様に、1 ページに最大 20 レコードが表示されます。20 を超えるレコードがある場合は、ページ下部にあるコントロールを使用して、最初のページ、前のページ、次のページ、または最後のページに移動できます。

Process Manager コンソール：［Statistics］タブ

Process Manger コンソールの［Statistics］タブでは、すべてのプロセスのリストと、各プロセスの「Running」および「Completed」プロセスインスタンスの数が提供されます (次の図を参照してください)。下部の［Totals］行には、左から右へ、「**プロセス**」(プロセス「インスタンス」ではなく、異なるプロセスモデル)、「**Running**」プロセスインスタンス、および「**Completed**」プロセスインスタンスの合計が表示されます。プロセスは、左側に名前のアルファベット順に表示されます。各名前はホットリンクになっており、名前をクリックすると、プロセス名でフィルタされたステータステーブルのリストを表示する［Status］ページが表示されます。

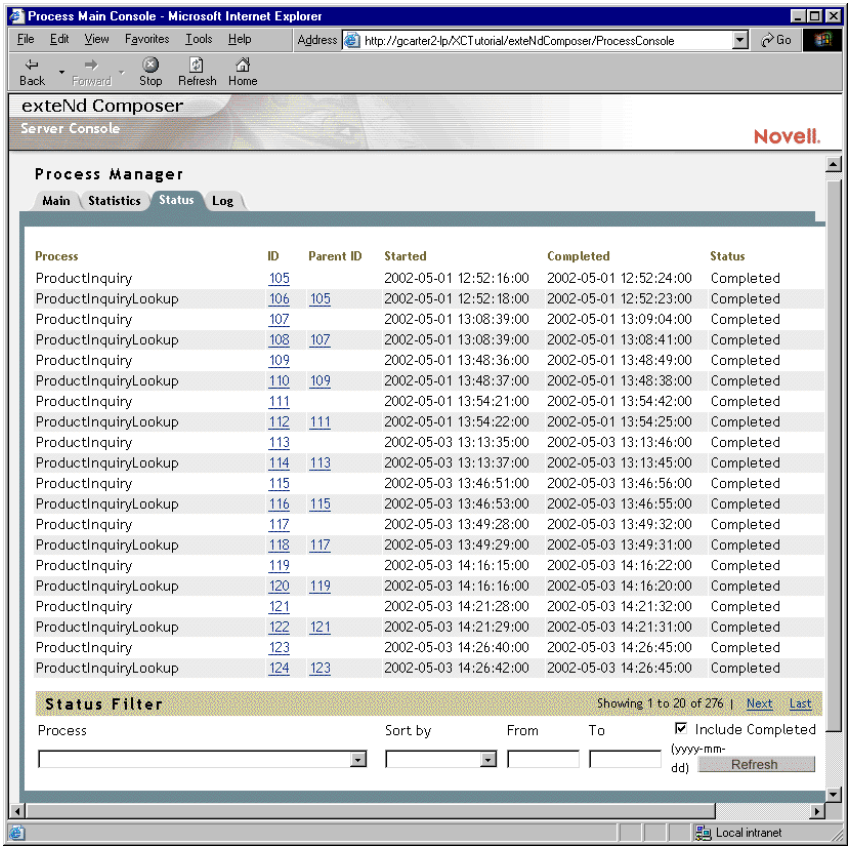


Process Manager コンソール：［Status］タブ

Process Manager コンソールの［Status］タブでは、すべてのプロセスの全体的な実行ステータスのビューが提供されます。このビューは、プロセス名と日付範囲を基準にフィルタ可能で、どのフィールドをソートの基準にするかのコントロールが用意されています。フィルタコントロールは、ページの下部にあります。

他のビューと同様に、一度に 20 の結果行が表示されます。使用可能な結果のページを移動するには、ページの右下の方の角にある［**First**］、［**Prev**］、［**Next**］、または［**Last**］のリンクをクリックします。

このビューはリアルタイムに更新されないため、右下角の近くに［**Refresh**］ボタンが用意されています。



［**Process ID**］および［**Parent ID**］の数字は、クリックできるリンクです。該当する［Process ID］のリンクをクリックして、特定のプロセスインスタンスを「ドリルダウン」できます。リンクをクリックすると、新しいブラウザウィンドウが開き、後で説明する［Process Detail］ページが表示されます。

［Status Filter］

［**Status Filter**］コントロールグループ (ページの下部) を使用して、［Status］タブビューにプロセスを表示する方法を制御できます。たとえば、［**Process**］ドロップダウンコントロールを使用して、特定の名前のプロセスのプロセスインスタンスを表示するよう選択できます。

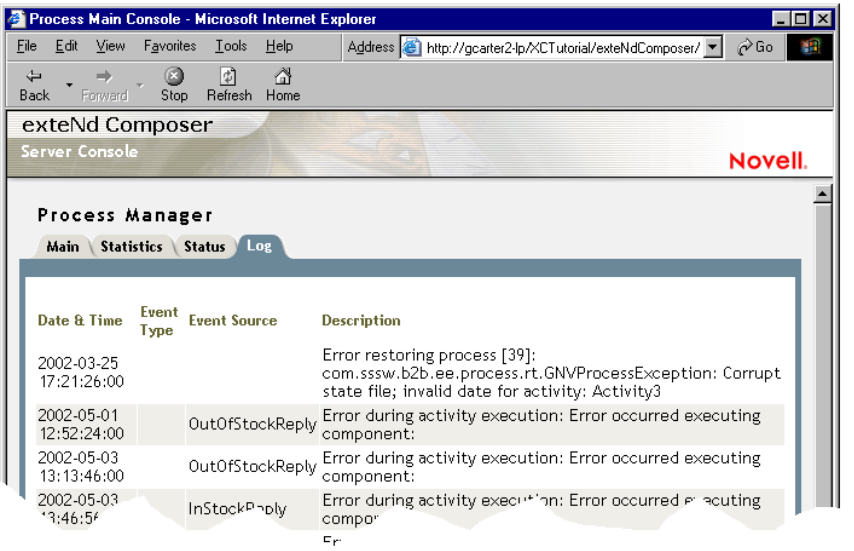
［**Sort By**］コントロールを使用して、表示されるプロセスのリストを、［Process Name］、［Process ID］、［Create Date/Time］、［Modify Date/Time］、または［Status］を基準にソートできます。

［**From**］および［**To**］フィールドに日付を入力して、指定した日付範囲内に起動したプロセスと完了したプロセス (オプション) を表示できます。完了したプロセスのレコードを表示する場合は、［**Include Completed**］チェックボックスをオンにします。

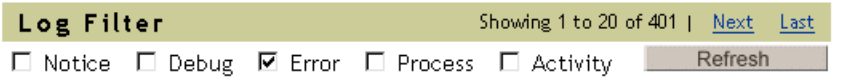
フィルタオプションを選択した後、［**Refresh**］ボタンをクリックして、現在のフィルタおよびソートの設定に基づいて、プロセスの新しいリストを表示します。

Process Manager コンソール：［Log］タブ

Process Manager コンソールの［Log］タブには、日付と時刻を基準にしてソートされた、次の形式のログメッセージが表示されます。



ページの下部にある次のチェックボックスをオンにして、ログのビューをフィルタできます。

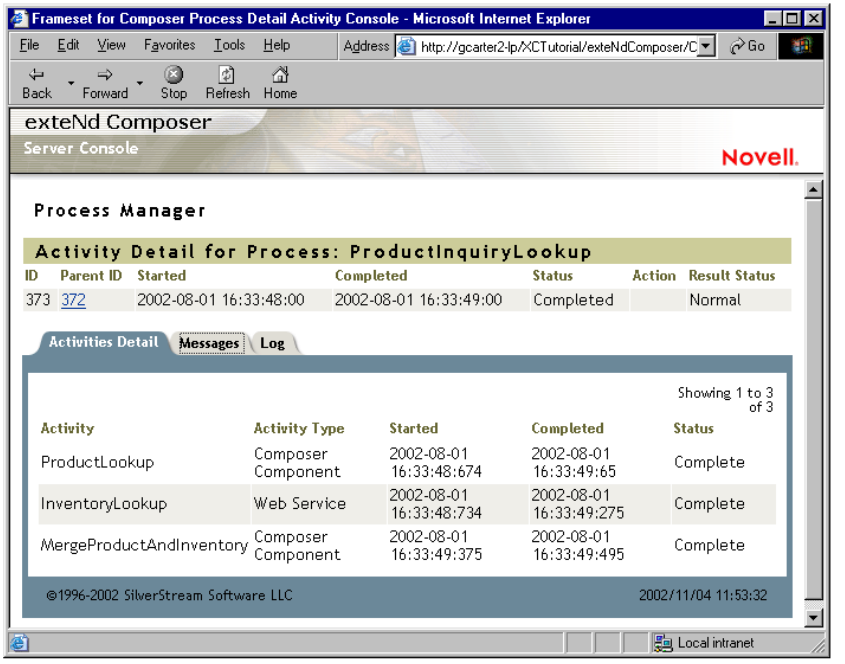


必要な項目を選択した後に、［Refresh］ボタンをクリックします。

プロセスインスタンスの詳細ビュー

プロセスインスタンスのリンク (メインプロセスコンソールの［**Status**］**タブ**の［**ID**］列のリンクなど) をクリックすると、新しいブラウザウィンドウに、そのプロセスインスタンスの［Activity Detail］ビューが表示されます (複数のブラウザウィンドウを使用して、複数のプロセスインスタンスを同時に監視できます)。

プロセスインスタンスの詳細ビューには、［**Activities Detail**］、［**Messages**］、および［**Log**］の 3 つのタブがあります。



プロセスの詳細 :［Activities Detail］タブ

［Activities Detail］は、詳細ページの最初のタブで、ウィンドウが最初に開いたときのデフォルトのビューです。

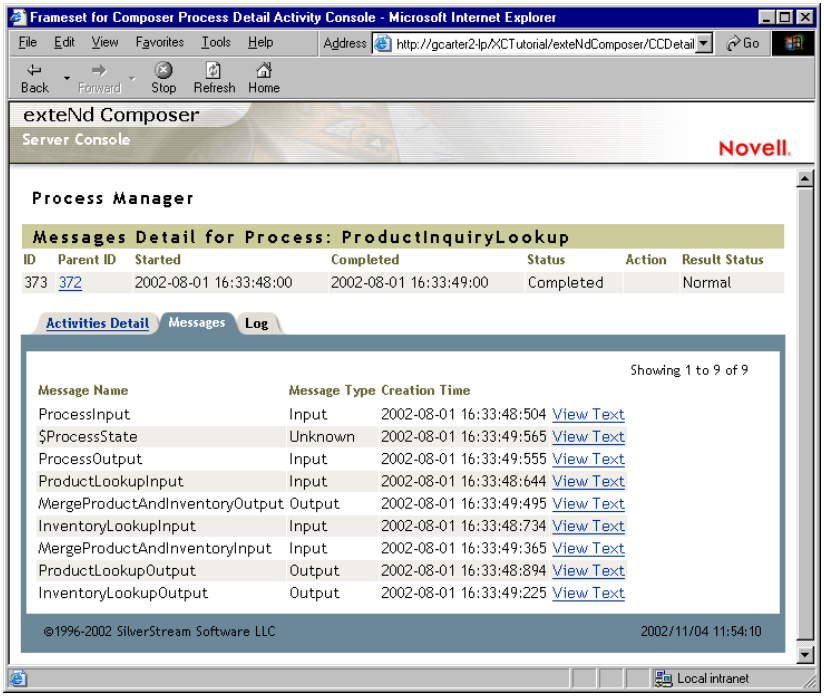
このタブには、プロセスインスタンスを構成する個々のアクティビティの名前、ID、開始日 / 時刻、完了日 / 時刻、およびステータスが表示されます。プロセスインスタンスは、「Running」または「Complete」のいずれかとして表示されます。プロセスが実行中の場合は、実行中のプロセスインスタンスを中断または終了できるボタンが表示されます。

［Activities Detail］タブビューの列には、次の意味があります。

［Activity］	アクティビティの名前です。アクティビティの名前は、［Activity Data Monitor］へのハイパーリンクになっています。［Activity Data Monitor］には、入力および出力ドキュメントと、それらのデータ値が表示されます。	
［Activity Type］	アクティビティのタイプで、「Web Service」、「Subprocess」、「Composer Component」、または「End Point」になります。	
［Started］	アクティビティが起動された日付と時刻です。	
［Completed］	アクティビティが完了した日付と時刻です。	
［Status］	この列には、アクティビティのステータスが表示されます。	
	「Completed」	関連付けられている処理が完了した後に、アクティビティが続行するかどうかは、終了条件によって決まります。終了条件が false に評価された場合、アクティビティは、関連付けられている処理に応じて「Enabled」または「Running」で続行することによって反復されます。終了条件が true に評価された場合は、「Completed」の状態になります。
	「Running」	アクティビティが起動された後の状態。
	「Terminated」	アクティビティよりも先にプロセスが完了した場合のアクティビティのステータス。
	「Enabled」	フローエンジンは、このアクティビティのインスタンスを現在実行できることを判断して、「Enabled」の状態にします。アクティビティの性質および関連付けられている処理に応じて、アクティビティは、明示的なリクエストによって起動されるまでその状態に留まるか (たとえば、*in* または *in-out* 処理など)、またはフローエンジンによってただちに起動される場合があります (たとえば、*out* または *out-in* リクエストなど)。

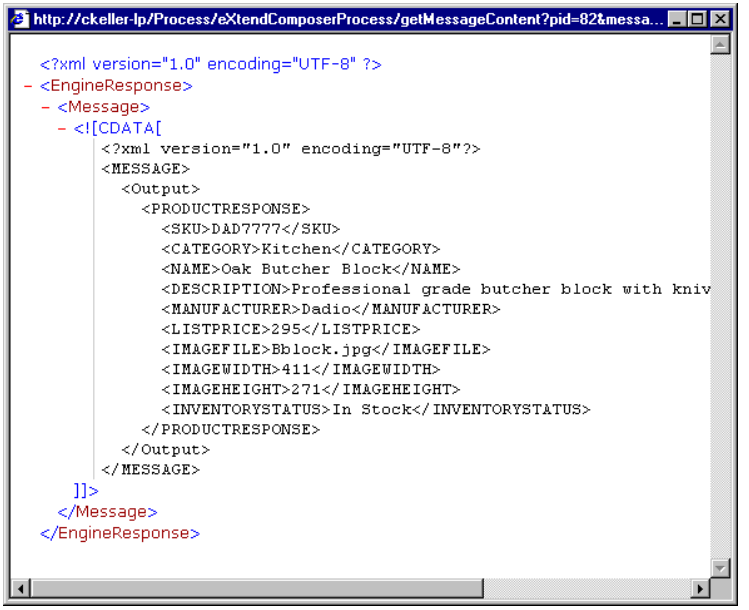
プロセスの詳細 :［Messages］

［Messages］タブでは、プロセスインスタンスのメッセージ (入力および出力ドキュメント) のビューが提供されます。メッセージは、名前を基準にソートされます。



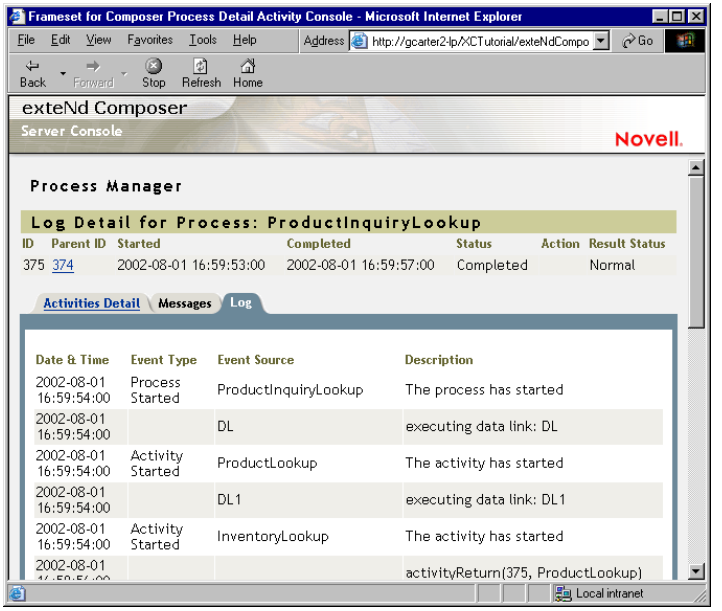
［**Messages**］タブには、次の情報が表示されます。

［Message Name］	メッセージの名前。
［Message Type］	メッセージタイプは、「Input」または「Output」のいずれかの場合があります。
［Creation Time］	メッセージが作成された時刻。
［View Text］	［View Text］リンクをクリックすると、次のように、新しいブラウザウィンドウにメッセージが表示されます。



プロセスの詳細：[Log]

［Log］タブには、さまざまなタイプの、ログに記録されたイベントの要約が表示されます。次の図を参照してください。



このウィンドウの下部には、次のチェックボックスがあります (必要に応じてスクロールします)。これらのチェックボックスを使用して、テーブルに要約を表示するイベントのタイプを制御できます。

目的のチェックボックスをオンにし、[**Refresh**] ボタンをクリックすると、ログに記録されたイベントのリストが、適切にフィルタされて表示されます。

A テスト

設計時のテストとサーバでのテストの環境的相違

Composer でのアニメーションベースの「ステップスルー」テストとサーバ側 (配備) のテストでは、環境が大きく異なっています。もちろん、作成したプロセスやサービスを検証するためには、両方のタイプのテストが必要です。認識しておく必要がある環境的相違の一部は、次の表で詳しく説明されています。

要件	Composer でのテスト	サーバでのテスト
コンソールビューおよび管理用監視	配備されていない設計時環境では使用不可	管理コンソールは使用可能
ログ出力	メッセージは Composer メインウィンドウの［Output］ペインに送信される	メッセージはコンソールの［Log］タブで表示可能
長時間実行されるプロセスのテスト	設計時の設定では実用的ではない (一部のプロセスでは、数日から数週間かかることもある)	実行可能 (ここでの実行が推奨される)
データの持続性	データベースは必要ない	プロセスサーバの使用に対してデータベースを設定する必要がある
実行されているプロセスの状態の視覚的表示	アニメーション時に可能 (プロセスが実行されるとキャンバスビューが更新される)	このリリースではキャンバス (グラフ) ビューはない

要件	Composer でのテスト	サーバでのテスト
プロセスインスタンス情報	プロセス ID は、各設計セッションの開始時に 1 から始まり、新しいプロセスインスタンスが実行されるごとに増分する。Process Designer を起動するたびに、プロセス ID 番号は再び 1 にリセットされる。	プロセス ID は、継続的に生成され、1 にリセットされることはない。
次に対するランタイム時の変数 * 接続名 * クライアントのアカウント情報 * ログファイルのパス * DTD URI * XSL URI * メール送信サーバ * XML 交換 URI の変更	多くの場合、設計およびテストのために、ローカルマシン上の場所を指す	プロダクションサーバおよび Web 上の場所を指すよう設定する必要がある
トリガ	プロセスは、アニメーションコンポーネント内の Process Execute アクションから、またはアニメーションツールバーボタンのうちの 1 つから直接実行できる	すべてのプロセスは、それらのプロセスを開始できるサービスで配備されなければならない

B パフォーマンスの調整

設定オプション

プロセスサーバのパフォーマンスは、さまざまな方法で調整できます。必要な調整は、**xc_process_config.xml** ファイルを編集することによって実行できます。SilverStream アプリケーションサーバをインストールするためには、このファイルをたとえば **D:\Silverstream3.7\eXtendComposer\lib** に保存します。

キャッシュ

プロセスサーバのキャッシュは、<PROCESS_CACHE> の値を変更することによって管理されます。

スリープ時間

<SLEEP> 要素の値では、メモリ内プロセスが <CUTOFF> 時間を超過していないかどうかをチェックする前に、プロセスサーバが遅延ループで待機する秒数が制御されます (次を参照)。

カットオフ時間

<CUTOFF> 要素の値では、アクティビティなしでプロセスがメモリ内に存在できる時間の最大秒数が制御されます。メモリ内プロセスの <CUTOFF> が、<SLEEP> 時間を超過した場合は、プロセスがメモリからパージされます。ただし、そのプロセスはデータベース内に維持され、メモリ内にある場合よりは遅くなりますが、実行可能な状態に戻ることもできます。

メモリ内プロセスインスタンスの合計

<SIZE> 要素の値では、スワップが生じるまで、メモリ内に存在できるプロセスの最大数が制御されます。

C プロセス管理用語集

Addressee

Addressee プロパティ (Web Service Receive アクティビティタイプにのみ存在する) では、アクティビティのインスタンスにラベルを付ける方法を指定します。通常このラベルは、組織における個人の名前と対応しています。

BPML

ビジネスプロセスモデル化言語 (Business Process Modeling Language)。これは、Business Process Management Initiative (http://www.bpmi.org) が作成および管理するワークフローを記述するための XML 文法です。スコープの面で、WSFL とほぼ同等です。Process Manager では、BPML ではなく、WSFL を使用します。

Call

Call イベントとは、プロセスのインスタンスを呼び出すことのできる 2 つのライフサイクルエベントのうちの 1 つを指します (もう 1 つのイベントは *Spawn* と呼ばれます。後の項目を参照)。直ちにインスタンス ID を返す生成されたプロセスとは異なり、呼び出されたプロセスはプロセスフローが完了するまで戻りません。呼び出し操作は、同期的に処理が行われることを意味しますが、生成操作は「実行後削除」と同様です。

Fan-Out

N 個の独立した作業アイテムの集合により、その作業アイテムで動作するよう設計された特定プロセスの *N* 個のインスタンスが非同期的に呼び出される、実行パターンのタイプ。

FlowInstanceID

Spawn 操作により呼び出される WSFL プロセスは、固有の FlowInstanceID を直ちにコーラに返す必要があります。この ID はタイムスタンプまたは任意の文字列である場合がありますが、実行中のプロセスの特定インスタンスを一意に特定していなければなりません。この値は、他のライフサイクル操作 (たとえば、*Enquire* など。後の「ライフサイクルインタフェース」を参照) の入力値として使用されます。

PIP®

RosettaNet の Partner Interface Processes のこと。これは、取引パートナー間のビジネス通信パターンを定義する、「事実上」の業界標準です。通信パターンには、一般的なビジネストランザクションのさまざまな種類に関する実行順序やタイムアウトの規則が含まれます。こうしたパターンは、実行順序 (タイムドメイン) の条件が複雑であるため、しばしば「コレオグラフィ」と呼ばれます。

ProcessID

ランタイム時に、プロセスサーバ内のプロセスインスタンス (実行中のプロセス) を特定する固有の番号。

RosettaNet

「IT (情報技術)、EC (電子部品)、および SM (半導体製造) のサプライチェーン間で行われる電子商取引において、開かれたコンテンツとトランザクションの標準を採用、推進すること」を目的とした非営利業界団体 (詳細については、**http://www.rosettanet.org/** を参照してください)。

SOAP

Simplified Object Access Protocol の略称。分散された環境で情報を交換するための簡易な XML ベースプロトコル。プロトコルの定義は、メッセージの内容と処理方法を記述するためのフレームワークを定義するエンベロープ、アプリケーション定義のデータタイプを指定するための文法、およびリモートプロシージャの呼び出しと応答を表す文法、の 3 つの部分で構成されています。

Spawn

Spawn は、プロセスの一方向の (非同期的な) 呼び出しを可能にする WSFL で定義されたライフサイクル操作です (対応する「同期化された」起動イベントは、*Call* イベントです。「Call」を参照)。プロセスが生成されると、直ちに結果 (プロセス ID) が返されます。

Synchronize Subprocesses アクティビティ

Synchronize Subprocesses アクティビティは、Process Manager の基本的なアクティビティタイプの 1 つです。これは、コンポーネントで生成された複数のインスタンスからのデータの集合と照合を円滑にするために使用する、特別なアクティビティタイプです。このタイプは通常、受信データをマージするため、アクティビティへの実装は「マージコンポーネント」と呼ばれます。つまり、Synchronize Subprocesses アクティビティは、「ファンアウト / ファンイン」シナリオの「ファンイン」の部分に相当します。

UDDI

http://www.uddi.org の管理する Universal Description, Discovery and Integration の仕様。ビジネスサービスを Web ベースレジストリを介して使用するためのスキーム。

Web Service Receive

Web Service Receive アクティビティは、Process Manager の基本アクティビティタイプの 1 つです。能動的な「受信する」アクティビティタイプであり、WSDL で記述されたリクエスト - 応答または一方向のトランザクションパターンを実装するために使用します。

WSDL

Web Services Description Language の略称。Web サービスをメッセージにおけるエンドポイント操作のセットとして記述するための XML 形式。操作とメッセージは抽象的に記述された後で、エンドポイントを定義する具体的なネットワークプロトコルとメッセージ形式を割り当てられます。関連付けられた具体的なエンドポイントは、抽象的なエンドポイント (サービス) に結合されます。このようにしてサービスは、6 種類の主な要素 (「タイプ」、「メッセージ」、「ポートタイプ」、「関連付け」、「ポート」、および「サービス」) で定義されます。

WSFL

Web Services Flow Language の略称。ワークフロープロセスをリンクしたアクティビティとして記述するための XML 形式。アクティビティは Web サービスまたはその他のワークフロープロセスである場合があります。

アクティビティ

アクティビティとは、ビジネスタスクとして機能するプロセスモデル内の作業単位を指します。操作レベルでは、アクティビティは、操作に関連付けられた入力、出力、および発生する可能性のある障害を指定する署名を持つ、名前の付けられた操作です。アクティビティは実装からは独立しています。実装 (Composer のコンポーネントタイプまたはコンポーネントサービス) は、アクティビティに代わってタスクを実行します。

一方向

一方向操作は、エンドポイントがメッセージを受信する (ただし、エンドポイントはイニシエータに返信しない) Web サービスの実行パターンです。一方向の Web サービスは、受動的な受信者です (「通知」も参照)。

依頼 - 応答

依頼 - 応答操作は、サービスが積極的にメッセージを「送信」し、次に応答を受信する Web サービスの実行パターンです。このシナリオでは、Web サービスは、トランザクションの「イニシエータ」です。パターンの一部として参加者からの応答が必要であるため、このタイプの Web サービスは同期的に実行されると見なされます。つまり、メッセージを送信すると、Web サービスは応答メッセージが送信されるまで遮断されます (「リクエスト - 応答」も参照)。

グラフ

ノードのシステムを抽象的に視覚表現したもの。Process Manager の用語では、プロセスの「グラフ」とは、Process Designer のキャンバスで作成したものを指します。

結合条件

2 つまたはそれ以上のアクティビティが、同じ後続のアクティビティをターゲットとする場合、後続のアクティビティを呼び出すかどうかは、アップストリームアクティビティの実行が終了している場合にのみ評価されるファクタによって決まる場合があります。これは、ランタイムエンジンにより、「結合条件」でのユーザ定義の論理に基づいて決定されます。結合条件では、受信リンクのそれぞれのブール値 (または「真の値」) を入力として使用します。結合条件は、リンク値でユーザ定義の論理操作のセットを実行し、true または false を返します。true の条件では、結合のターゲットが呼び出されます。false の場合、制御フローは結合の時点で終了します。リンクと終了の論理 (どちらも XPath を使用) とは異なり、結合条件は擬似コードのような単純なブール論理で表されます。つまり結合条件は、メッセージやメッセージパート (または一切のデータなど) を認識しません。リンクブール値のみを認識します。

コレオグラフィ

順序付けられた操作の特定のセットは、ビジネスプロセスのコンテキストで、しばしば口語的にコレオグラフィと呼ばれます (前の「*PIP*」を参照)。

コントロールリンク

コントロールリンクは、1 つのアクティビティから別のアクティビティへのコントロールフローにおける単一の手順を定義する、WSFL による生成物です。これにより、「アクティビティが通過する順序」が指定されるため、ワークフローエンジンは特定のアクティビティから次のアクティビティへ移行する方法を認識できるようになります。

サービスプロバイダ

サービスプロバイダは、ビジネスプロセス内における特定のアクティビティの実行を担う組織です。

サービスプロバイダのタイプ

WSFL では、ビジネスプロセスとその実装の定義を独立させておくため、アクティビティを (後でタイプにマップできる) 特定のサービスプロバイダではなく、抽象的なサービスプロバイダの「タイプ」によって実装されていると定義します。サービスプロバイダのタイプと関連付けられているインタフェースは、WSDL ドキュメントによって定義されます。サービスプロバイダは、ビジネスプロセスで特定のアクティビティを処理するために、Web サービスのインタフェースを適切に実装する必要があります。

サイクリックグラフ

「サイクリックグラフ」は、ループを形成してダウンストリームノードからアップストリームノードへのリンクを許可するグラフです。Process Manager では、このようなグラフのパターンは使用できません。

サブプロセス

別のプロセスから呼び出されたサブプロセス。

システムの障害

アクティビティの実装が未処理の例外を生成する、サブプロセスアクティビティが障害メッセージを返す、ランタイムエンジンで処理方法の不明なメッセージまたはメッセージタイプが発生する、またはタイムアウトの障害が発生したが、該当するアクティビティによって処理されていない、などの場合、ランタイムエンジンは「システムの障害」を生成します (この場合、2 つの障害 (タイムアウトとシステムの障害が 1 つずつ) が実際に生成されます)。システムの障害が発生すると、プロセスインスタンスにより、`_SystemFault` というメッセージパートを持つ *_SystemFault* というメッセージが呼び出されます。

実装

アクティビティのソフトウェアにおける具体的な使用。アクティビティには必ず実装が必要です。

終了条件

終了条件は、特定のアクティビティが正常に実行されたかどうかを示す (ユーザ定義の XPath 論理のランタイムでの評価により決定する) ブール値です。終了条件が true になるまで、および終了条件が true でない場合、送信コントロールリンクは実行されません。終了条件が false の場合、アクティビティは (タイムアウトと再試行の設定が許す限り) 再度実行されます。

信号

通常、関数またはファイルが使用できるかどうか (ファイルがロックされているかどうか) を示すために使用されるフラグ値。

スレッド

他のコンテキストで実行されている操作に対する時間的順序による従属関係のない、実行コンテキスト。

相関 ID

Process Manager では、相関 ID とは、特定のメッセージパートをトランザクションのコンテキストと関連付けるために使用する、任意のユーザ定義の文字列または数値を指します。相関 ID は、この種類のユーザ定義ラベルでは一般的な用語ですが、WSDL または WSFL の正式な概念ではありません。

操作

WSDL では、操作とは、(名前の付けられた入力メッセージと出力メッセージで記述された) 順序が指定されたメッセージの転送を意味します (後の「ポートタイプ」も参照)。

待機中のアクティビティ

アクティビティ (サブプロセスや Web Service Receive アクティビティなど) が、別のアクティビティによって非同期的に行われたリクエストに対する応答を受信することを待機しているため、「待機の状態」にある場合、このアクティビティは「待機中のアクティビティ」と呼ばれます。

長時間実行されるプロセス

プロセスには実行が完了するまでに数日または数週間かかるものがあります。このようなプロセスを「長時間実行されるプロセス」と言います。

通知

一方向操作は、サービスによりメッセージが積極的に送信され、それに対する応答は期待されない Web サービスの実行パターンです。これは「実行後削除」のパターンです。ただし、これはパートナーとの非同期的な往復の通信を遂行するために、一方向パターン (前の「一方向」を参照) でしばしば使用されます。このような場合、通知パターンを実装する Web サービスは通常、相関情報を送信メッセージに埋め込み、後で一方向操作により受信される情報が通知のトランザクションのコンテキストと「一致する」ようにします (「一方向」を参照)。

データリンク

データリンクは、データフローの原子単位であり、1 つまたは複数のデータソースと 1 つまたは複数のデータターゲットを指定します。ソースとターゲットは、実行中のプロセス内のアクティビティです。ほとんどの場合、データフローは制御フローを反映していますが、データがフロー内の特定のアクティビティを省略したり、制御フローに指定されているパスよりも、直接的なパスを使用してターゲットに到達したりできる可能性もあります。したがって、データリンクが常にコントロールリンクに従うとは限りません。

デッドパスの排除

デッドパスの排除とは、毎回プロセスサーバが false を返す条件式 (たとえば、リンク条件) を実行する、特別なルックアヘッド操作を指します。特定のパスによるフローが、false リンク条件のために実行不可能になると、すべてのダウンストリームリンクは、結合が操作の過程で評価されるために、false とマークされなければなりません (パスはデッドの状態から既知の false の状態に変わります)。これが行われない場合、ダウンストリームの結合はハングしたままになります。

トランジション条件 (リンク論理)

プロセスの実行では、実行エンジンは特定のアクティビティの終了を認識し、フローの次のアクティビティを特定し、ユーザ定義のトランジション論理に基づいて、次のアクティビティを呼び出すかどうかを決定することができなければなりません。「トランジション条件」では、フローが現在のパスに沿って進むかどうかが決定されます。トランジション条件は XPath で指定され、常に true または false を返します。

ビジネスプロセス管理 (BPM (Business Process Management))

「ビジネスプロセス管理」は、コンポーネントのアクティビティと参加者の役割におけるビジネス機能をモデル化する方法についての研究です。

非同期

作業を「単独および他の作業と平行して」行う操作のモードです (つまり、作業間には時間的順序による従属関係は存在しません)。ソフトウェア用語では、非同期のタスクとは、独自のスレッドで実行されるタスクを指します。「実行後削除」という用語は、非同期的に作成されたプロセスを指す場合にしばしば使用されます (「スレッド」および「*Spawn*」も参照)。

ファクタリング

プログラミングでは、ファクタリングとは、コードをより小さな、より一般的 (つまり再利用できる) 作業単位へ分割しようとすることを指します。

複合 Web サービス

Web サービスに基づくプロセスモデル。基本的には、すべての WSFL プロセス。

フローモデル

フローモデルは、ビジネスプロセスをモデル化する有向グラフの XML による表現です。つまり、特定のプロセスを構成する、アクティビティ、コントロールリンク、およびデータリンクをすべて含むセットです。フローモデルによってプロセスのコレオグラフィが明示的になるため、実行エンジンはランタイム時にプロセスをインスタンス化したり、プロセスのライフサイクルに対する制御のフローを管理したりすることができるようになります。

プロセス

特定のビジネスタスクの実行に使用するアクティビティ、制御フローパターン、およびデータフローの関係を記述したもの。WSFL (後の項目を参照) では、プロセスは「複合 Web サービス」として記述されます。WSFL では、プロセス (またはワークフロー) は自動化されていると見なされます。

ポートタイプ

WSDL では、「ポートタイプ」とは、名前の付けられた、操作のセットを意味します (一方、操作は特定のメッセージに対して特定の時間的順序を指定したものです)。WSDL では、一方向、リクエスト - 応答、依頼 - 応答、および通知という 4 種類のポートタイプがサポートされています (各項目の説明を参照)。

マップポリシー

マップポリシーでは、2 つまたは複数のデータリンクが同じメッセージパートをターゲットとするといった特別な場合に、どのようにデータをマップするかを指定します。Last Writer Wins (LWW) のポリシーでは、古いデータは新しい着信データで上書きされます。First Writer Wins (FWW) のポリシーでは、一度データが書き込まれると、その後の着信データは無視されます。マップ順序では、アクティビティのプロパティシートにある［Messages］タブに表示された順序で、XPath 対 XPath のマップが行われ、タイムスタンプは完全に無視されます。

メタデータ

データに関するデータ。Process Manager では、「メタデータ」によるプロセスの表現は、特定プロセスの実際の構造と属性を XML で非視覚的に表現したものを意味します。メタデータによるプロセスの概略は、Process Server がランタイム時にプロセスインスタンスを作成する際に使用されます。

メッセージ

WSFL と WSDL では、「メッセージ」は、あるまとまったデータセットの抽象的な定義です。メッセージ構造の一部としてまとめられた論理パートは、「メッセージパート」と呼ばれます (次を参照)。アクティビティはメッセージで実行されるため、アクティビティへのインタフェースは、入力および出力のメッセージで指定できます。

メッセージパート

WSFL と WSDL では、「メッセージパート」はメッセージの論理単位を意味します。Process Manager では、アクティビティの実装が検査、修正、および新しいメッセージの新しい部分へ変換される、XML ドキュメントに対応する部分を指します。

ライフサイクルインタフェース

ライフサイクルインタフェースは、WSDL で定義された Web サービスインタフェースです。これはすべての WSFL 処理でサポートされる必要のある、操作の基本的なセットを記述します。これらの操作には、*Spawn*、*Call*、*Suspend*、*Resume*、*Enquire*、および *Terminate* があります。これらは、グローバルなスコープで使用 (処理全体に適用)、管理できます。

リクエスト - 応答

リクエスト - 応答操作は、サービスがメッセージを受信した後、イニシエータに (相関的な) メッセージを返信する Web サービスの実行パターンです。リクエスト - 応答の Web サービスは、受動的な受信者です。出力メッセージで応答します。

リンク条件

リンク条件はブール値を解決する XPath 式です。この値により、ランタイム時に特定のリンクがプロセスエンジンによって作成されるかどうかが決まります。通常 XPath 式は、アップストリームアクティビティの出力よりデータを使用します。

ワークフロー

ビジネスプロセス管理のコンテキストでは、ワークフローはプロセスです。作成者は、最も自動化されたワークフローが最終的には Web サービスに依存することを意図しているため、WSFL では「プロセス」という言葉を使用します。(従来のワークフローシステムでは、アクティビティは人間が介在するアクティビティの周囲に存在する傾向があります)。

索引

M

O

P

Q

R

S

T

U

W

X

あ

い

う

え

お

へ

ほ

ま

め

や

よ

ら

り

る

れ

ろ

わ