JBoss Enterprise SOA Platform 5

ESB サービスガイド

JBoss 開発者およびビジネスルール作成者向け

エディッション 5.2.0

Last Updated: 2017-11-08

JBoss Enterprise SOA Platform 5 ESB サービスガイド

JBoss 開発者およびビジネスルール作成者向け
エディッション 5.2.0

JBoss ESB 開発チーム コミュニティーの協力

Red Hat Documentation Group

法律上の通知

概要

本書は、SOA Platform の中核をなすエンタープライズサービスバスに関するビギナーズガイドです。ESB のコアコンポーネントの多くを紹介していきます。

目次

第1章 基本的なコンセプトの紹介

サービス指向アーキテクチャー(SOA) は単一のプログラムやテクノロジーではなく、ソフトウェア設計のパラダイムと考えられるでしょう。

ご存知のように *ハードウェアバス* は、複数のシステムやサブシステムの間を結ぶ物理コネクターです。システム間を結ぶポイントツーポイントコネクターを大量に使うのではなくハードウェアバスを 1 つ使うと、各システムを単にセントラルバスへ接続するだけで済みます。*エンタープライズサービスバス* (ESB) も、ソフトウェアで全く同じ事を行います。

ESB は、メッセージングシステムのうえにあるアーキテクチャー層にあります。このメッセージングシステムにより、ESB 経由のサービスでの*非同期通信*をスムーズに行うことができます。

このコンテキストでは「サービス」は Web サービスとは限りません。ファイル転送プロトコル (FTP) や Java Message Service などのトランスポートを使った違うタイプのアプリケーションも「サービス」になりえます。

注記

この時点では、エンタープライズサービスバスはサービス指向アーキテクチャーと同じものかと考えている人もいるかもしれませんが、違います。ESB はサービス指向アーキテクチャー自体を形成するのではなく、構築に利用できるツールを多く提供しています。特に、SOA で必要な*疎結合*や*非同期式メッセージ受け渡し*を容易に行います。SOA は単なるソフトウェアではなく、一連の原則、パターン、ベストプラクティスと考えてください。

JBoss Enterprise Service Bus は Red Hat が提供するオープンソースの ESB で、以下に対応しています。

- 複数のトランスポートとプロトコル
- *listener-action モデル*(サービス間を疎結合できるようにするため)
- *コンテンツベースルーティング*(JBoss Rules エンジン、XPath、Regex、Smooks 経由)
- **JBoss Business Process Manager** との統合 (*サービスオーケストレーション*機能を提供)

第2章 開発者向けの例

本章には、開発者向けの JBoss Enterprise SOA Platform について紹介されています。

2.1. JBOSS ESB QUICK START (クイックスタート)

quick starts (クイックスタート) はサンプルのプロジェクトです。それぞれ、サービスを構築しやすくするため、各機能の利用方法を説明しています。***SOA_ROOT*/samples/quick starts/** ディレクトリには数十個の**quick starts** が含まれています。**Apache Ant** を使いすべてのクイックスタートを構築、デプロイしてください。

「"helloworld" クイックスタート」 は、**helloworld quick start** を詳しく説明しています。

quick start を詳しく学習する手順：

- クイックスタートの **readme.txt** ファイルを参照します。
- **quick-starts** ディレクトリで **ant help-quick starts** コマンドの実行を実行します。
- **quick-starts** ディレクトリで **ant help** コマンドを実行します。

クイックスタートの実行時は以下に注意してください。

1. 各クイックスタートは Apache Ant を使い構築、デプロイします。**スタートガイド**を参照してください。
2. 各クイックスタートは、**samples/quick starts/conf/quick starts.properties** ファイルを使い環境固有の設定オプションを保存します。サンプルのプロパティファイル– **quick starts.properties-example** – も含まれています。
3. クイックスタートごとに要件は違います。要件については個別の **readme.txt** に記載されています。
4. クイックスタートはすべてのサーバープロファイルで実行できるわけではありません。
5. jBPM クイックスタートでは、jBPM Console のユーザー名とパスワードが有効でなければなりません。以下のように、**samples/quick starts/conf/quick starts.properties** ファイルにプロパティとして追加し、ユーザー名とパスワードを提示します。

   ```
   # jBPM console security credentials
   jbpm.console.username=admin
   jbpm.console.password=adminpassword
   ```

   この要件を満たす必要のあるクイックスタートは、**bpm_orchestration1**、**bpm_orchestration2**, **bpm_orchestration3**、**bpm_orchestration4** です。
6. *headless* モードでサーバーが実行されていない場合、(**groovy_gateway** など) 実行できるクイックスタートには限りがあります (JBoss Enterprise SOA Platform はデフォルトでは headless モードで起動するよう設定されています)。

   Red Hat は headless モードでのみ実稼働サーバーを実行するよう推奨していますが、開発環境で利用するためにモードを変更することができます。詳細は、**スタートガイド**を参照してください。

これらの **quick starts** は、頻繁に利用される機能についてデモを提供しています。

helloworld	**business_service**	**aggregator**
helloworld_action	**business_rules_service**	**bpm_orchestration1**
custom_action	**scripting_groovy**	**bpm_orchestration2**
helloworld_file_action	**transform_CSV2XML**	**webservice_consumer1**
helloworld_ftp_action	**transform_XML2POJO**	**webservice_producer**
simple_cbr	**transform_XML2XML_simple**	
fun_cbr	**transform_XML2XML_date_manipulation**	

2.1.1. "helloworld" クイックスタート

helloworld quick start では、JBoss Enterprise SOA Platform の主要機能とコンセプトを簡単に説明しています。これは ***SOA_ROOT*/samples/quick starts/helloworld/** ディレクトリに置かれています。以下の手順に従い実行してください。

手順2.1 helloworld quick start の実行

1. **サーバーを起動します。**
   サーバーが実行中であることを確認します。詳細は **スタートガイド**を参照してください。

2. **quick start のプロパティを設定します。**
   ***SOA_ROOT*/samples/quick start/conf/quick starts.properties** ファイルが正しい設定およびホームディレクトリを使うように設定されているか確認します。

   ```
   # Location of your JBoss Application Server installation.
   org.jboss.esb.server.home=/opt/jboss-soa-p-5/jboss-as
   # JBossAS server name.
   org.jboss.esb.server.config=default
   ```

3. **クイックスタートを構築、デプロイします。**
   端末を起動し ***SOA_ROOT*/samples/quick starts/helloworld** ディレクトリに移動します。

   ant deploy コマンドを実行します。これで**helloworld.esb** アーカイブを構築しデプロイします。

4. **サービスを呼び出します。**
   ant runtest コマンドを実行します。これにより、JMS メッセージを先ほどデプロイした ESB サービスに送信します。

5. **結果に関してサーバーを確認します。**
   "hello world" メッセージが ***PROFILE*/log/server.log** ファイルに追加されています。

以下のようになっているはずです。

```
17:25:12,614 INFO [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
17:25:12,615 INFO [STDOUT] Body: Hello World
17:25:12,615 INFO [STDOUT] &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
17:25:12,615 INFO [STDOUT] Message structure:
17:25:12,615 INFO [STDOUT] [ message: [ JBOSS_XML ]
```

利用するサーバープロファイルにより、物理サーバーコンソールにも表示される場合もあります。**production** プロファイルの場合はログファイルにのみ出力されます。

図2.1「helloworld コンポーネントとイベントのシーケンス」は、**HelloWorld quick-start** のコンポーネントと実行時に発生するイベントのシーケンスを表します。

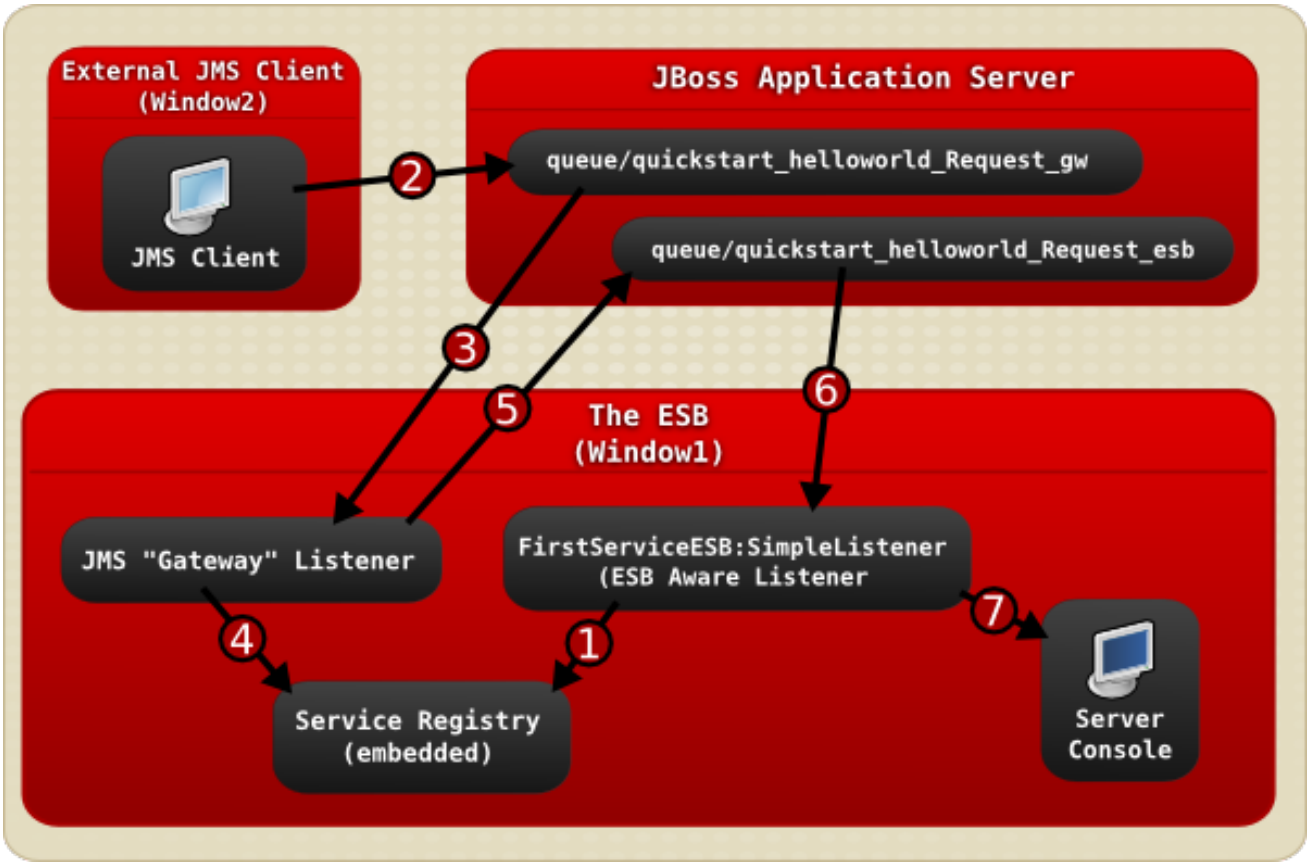


図2.1 helloworld コンポーネントとイベントのシーケンス

helloworld quick start は以下のコンポーネントを使います。

Gateway Listener

Gateway Listener は、**JBoss Enterprise SOA Platform** の主要アーキテクチャーコンポーネントの１つです。この種類の **listener** は、ESB と外部のサービスエンドポイントの間のインターフェースを提供します。今回の場合、Java Message Service Gateway を使います。

サービスレジストリ

これは **JAXR registry** 実装です。**registry** では、このクイックスタートに関し *Remote Method Invocation* ベースの通信を使います。**registry** の詳細については、**JUDDI Registry Guide** を参照してください。

ESB 対応のサービスリスナー

FirstService:ESB:SimpleListener は ESB 対応のメッセージがないかqueue/quick start_helloworld_Request_esb をリッスンします。*ESB 対応* および *非対応* メッセージについては、**本書**の次の章で説明します。

注記

アプリケーションクライアントとサービスは、*ESB 対応* とされます。これは、SOA Platform で利用するメッセージ形式やトランスポートプロトコルを理解できるという意味です (ゲートウェイを使い ESB 非対応のサービスをエンタープライズサービスバスで処理、ルーティング可能な形式にメッセージを変換します)。

図2.1「helloworld コンポーネントとイベントのシーケンス」にて、以下のイベントのシーケンスを説明します。

1. **JBoss Enterprise SOA Platform** サーバーは、**Window1** で起動し、helloworld quick start がデプロイされると**FirstServiceESB:SimpleListener** サービスはサービスレジストリサービスに追加されます。
2. JMS クライアントは ESB に対応しない "Hello World" メッセージ (プレーンな **String** オブジェクト) を JMS キューに (**queue/quick start_helloworld_Request_gw**) 送信します。
3. JMS Gateway Listener は ESB に対応しないメッセージを受け取り、ESB 対応のエンドポイントで利用できるように、そのメッセージから ESB 対応のメッセージを作成します。
4. **JMS Gateway Listener** は **registry** を使い **FirstServiceESB:SimpleListener** サービスの *エンドポイントリファレンス (EPR: end-point reference)*を検索します。この場合、EPR は **queue/quick start_helloworld_Request_esb** JMS キューです。
5. **JMS Gateway Listener** は、新規の ESB 対応メッセージを受け取り、**queue/quick start_helloworld_Request_esb** JMS キューに送信します。
6. **FirstServiceESB:SimpleListener** サービスはメッセージを受け取ります。
7. **FirstServiceESB:SimpleListener** サービスはメッセージからペイロードを抽出し、コンソールに出力します。

第3章 JBPM WEB アプリケーションを JBOSS ENTERPRISE SOA PLATFORM にデプロイ

JBoss Enterprise SOA Platform に Web アプリケーションをデプロイする際に従うべきベストプラクティスについては、本項を確認してください。

プロセス定義、長命のアイテムは Web アプリケーションとは別にデプロイしてください。Red Hat は、Web アプリケーションの前にプロセスをデプロイするよう推奨しています。これは、プロセスが常に利用できるとの前提で Web アプリケーションを操作できるためです。

デプロイメントのオプションは多数あり、 **GPD deployment** タブ、 **JSF console upload form**、**Ant DeployProcessTask** を使い選択します。

警告

jBPM ライブラリは含めないようにしてください。

jbpm.esb モジュールは jBPM アプリケーションで必要なライブラリと設定ファイルをすでに提供しています。クラスローディングのコンフリクトや設定の不一致といった問題を防ぐには、同梱のバージョンとデフォルト設定を常に利用するようにしてください。これはこれらのバージョンやデフォルト設定が Red Hat の幅広い Quality Engineering テストを行い改良が図られているためです。

マルチテナントのユースケースでは、各アプリケーションにそれぞれ違った設定が必要なアプリケーション多数を 1 つのサーバーでホストしています。Red Hat は、プラットフォームに同梱されているデフォルトの設定ファイルを変更しないように、各設定ファイルに一意名 (**jbpm.cfg.xml** 以外) を付けるようにお勧めしています。

第4章 ルールサービス

4.1. ルールサービスとは？

この章を参照し、ルールサービスのコンセプトと、ルールサービスの使用方法について学びます。**JBoss Business Rules Management System** (BRMS) のバックグラウンドを理解していると、この章を読み進めていく際に役立ちます。

4.1.1. はじめに

JBoss Enterprise SOA Platform の*ルールサービス*があると、ESB 上で *サービス*として貴社のビジネスアナリストが記述した*ビジネスルール*をデプロイできます。これには大きく２つの利点があります。

1. ルールをあるアプリケーション環境に統合するのに必要なクライアントコードの量が大幅に削減される
2. *アクションチェーン*から、あるいは*オーケストレーションされたビジネスプロセス*内からルールにアクセスできる

注記

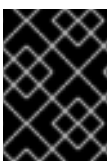
JBoss Business Rules Management System は対応オプションですが、ご希望であれば **rule engine** を利用できます。

ルールサービス機能は、**BusinessRulesProcessor** と **DroolsRuleService** のアクションクラスで提供され、後者は **RuleService** インターフェースも実装します。

BusinessRulesProcessor クラスにより、クラスパスからルールをロードすることができます。これらのルールは **.drl** および **.dsl** ファイルや*決定テーブル*(**.xls** 形式) で定義されます。ただし、１つの **BusinessRulesProcessor** アクションを複数のルールファイルで指定する方法はないため注意してください。これらのファイルベースのルールは、主にプロトタイプのテストができるように存在します。より複雑なルールサービスは、**JBoss Rules KnowledgeAgent** を使う必要があります。

RuleService は、**KnowledgeAgent** を使い、**Business Rules Management System** からか、ローカルファイルシステムから*ルールパッケージ*にアクセスします。ルールパッケージには、様々なソースを持つ何千ものルールが含まれています。例えば以下が挙げられます。

1. **JBoss Business Rules Management System**
2. インポートされた **DRL** ファイル
3. *ドメイン固有言語*(DSL) ファイル
4. *決定テーブル*



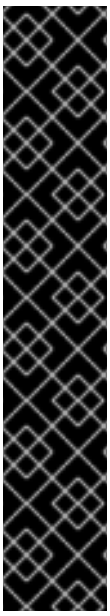
重要

Red Hat は、実稼働システムでは **KnowledgeAgent** アプローチを使うよう推奨しています。

BusinessRulesProcessor アクションは、 *ステートレス*と*ステートフル*の JBoss Rules の実行モデルに対応します。

ルールサービスの多くがストートレスモデルに準拠します。このモデルでは、メッセージがルールサービスに送信されます。このメッセージのボディに含まれているものは、ルールエンジンにより処理されるファクトすべてです。このルールが実行され、メッセージやファクトを更新します。

反対に、ステートフル実行はより長い時間をかけて複数のメッセージをルールサービスに送信します。メッセージが送信されるたびに、ルールが再度実行され、メッセージやファクトが更新されます。プロセスの最後に、最終メッセージがルールサービスがステートフルセッションの*作業メモリ*を消去するか通知し、その後ルールエンジンから完全に消去されます。



重要

RuleAgent は SOA Platform の ESB では利用されなくなっており、設定方法が違う KnowledgeAgent が代わりに利用されます。DRL 変更のポーリング設定は **ruleAgentProperties** ファイルの poll プロパティから設定されなくなっており、今は**esb.deployer/jbossesb-properties.xml** 内にある org.jboss.soa.esb.services.rules.resource.scanner.interval プロパティからグローバルに設定可能です (デフォルト値は 60)。つまり、KnowledgeAgents すべてでリソース変更があるかシステムが６０秒おきにチェックします。

また、これらのプロパティを提供し、基本認証でセキュリティが確保された URL にアクセスする必要があります。

```
username=admin
password=admin
enableBasicAuthentication=true
```

注記

ステートフルモデルには制限があります。それは、メッセージフロー内にルールサービスを１つしか置くことができませんが、今後この制限はなくなる可能性があります。

4.2. JBOSS RULES でのルールサービスの利用

4.2.1. はじめに

JBoss Rules は、**SOA Platform** に*ルールサービス*機能を提供するエンジン名です。詳細については本章を参照してください。

JBoss Rules は以下のコンポーネントを使い SOA Platform とやり取りを行います。

- **BusinessRulesProcessor** アクションクラス
- **JBoss Rules**、DRL、DSL、決定テーブル、**Business Rule Editor** のいずれかで記述されたルール
- Enterprise Service Bus メッセージ (これは JBoss Rules Engine の作業メモリに挿入されます)
- Enterprise Service Bus メッセージのコンテンツ (JBoss Rules エンジンに送信されるメッセージにある "fact" オブジェクトで構成されている)

メッセージを **BusinessRulesProcessor** に送信すると、*ルールセット* がそのメッセージに含まれているファクトオブジェクトを処理します。それぞれ、このルールセットはファクトオブジェクトの１つを更新するか、メッセージ自体を更新します。

4.2.2. ルールセットの作成

JBoss Developer Studio を使いルールセットを作成します。メッセージが*グローバル*として作業メモリに送信されるため、**jbossesb-rosetta.jar** ファイルを **JBoss Rules** プロジェクトに追加する必要があります。

注記

ルール作成と JBoss Rules 言語自体の詳しい説明は、同梱の **JBoss Rules リファレンスガイド** を参照してください。

JBoss Enterprise SOA Platform でサービスとしてデプロイメントのルールを記述する場合、これらの要件に必ず準拠します。

名前とアクションクラスの両方が必要です (名前はユーザー定義)。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderDiscountRuleService">
```

以下のうち１つが必要となります:

- **DRL** ファイル

  ```
  <property name="ruleSet" value="drl/OrderDiscount.drl" />
  ```

- **DSL** あるいは **DSLR** (*ドメイン固有言語*) ファイル

  ```
  <property name="ruleSet" value="dsl/approval.dslr" />
  <property name="ruleLanguage" value="dsl/acme.dsl" />
  ```

- クラスパスの **decisionTable**

  ```
  <property name="decisionTable" value="PolicyPricing.xls" />
  ```

- クラスパス上のプロパティファイル (**rule agent** にルールパッケージの検索方法を伝えることを目的ととする)。その方法ですが、URL あるいはローカルファイルへのパスのいずれかを指定します。

  ```
  <property name="ruleAgentProperties"
            value="brmsdeployedrules.properties" />
  ```

以下に設定例を示します。

1. ここでは、ルールが DRL ファイルに置かれており、ステートレスモデルに従い実行されます。

   ```
   <action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
     name="OrderDiscountRuleService">
       <property name="ruleSet" value="drl/OrderDiscount.drl" />
       <property name="ruleReload" value="true" />
       <property name="object-paths">
   ```

```
        <object-path esb="body.Order" />
    </property>
</action>
```

2. ここでは、ルールが DRL ファイルに置かれており、ステートフルモデルに従い実行されます。

   このシナリオでは、一定期間にクライアントが複数のメッセージをルールサービスに送信することができます。例えば、最初のメッセージには顧客オブジェクトが含まれており、続きのメッセージはその顧客に対する注文がそれぞれ含まれています。メッセージが受信されるたびにルールが実行されます (クライアントは、作業メモリのコンテンツを消去するようルールサービスに伝える最終メッセージにプロパティを追加することができます)。

注記

1. 同期セッションインスタンス１つをステートフルセッションデプロイメントの同期実行すべてにて共有します。これにより、ステートフルモデルのユースケースの数が大幅に制限されます。複数のクライアント指向セッションサービスデプロイメントが必要な場合、代わりに jBPM あるいは BPEL ソリューションの利用を検討してください。
2. ステートフルセッションは *永続的*ではありません。
3. ステートフルセッションは *クラスター化されていません*。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderDiscountMultipleRuleServiceStateful">
    <property name="ruleSet"
              value="drl/OrderDiscountOnMultipleOrders.drl" />
    <property name="ruleReload" value="false" />
    <property name="stateful" value="true" >
    <property name="object-paths">
        <object-path esb="body.Customer" />
        <object-path esb="body.Order" />
    </property>
</action>
```

3. この例では、ルールは DRL ファイルに置かれており、ステートフルモデルを使い、監査ロギングが有効になっています。JBoss Rules clockType、eventProcessingType、*チャネル* を設定し*複合イベント処理 (CEP: complex event processing)*を容易にします。

   ruleMultithreadEvaluation が **false** に設定されているため、*ナレッジベースパーティショニング* は有効ではありません(また、ruleMaxThreads が **1** に設定されている点も注意してください)。

注記

複合イベント処理のシナリオでは、ステートフルモデルを使っているため、前述した例と同じように、これらのシナリオはステートフルセッションデプロイメントの同期実行すべてにおいて共有されます。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="OrderEventsRuleServiceStateful">
    <property name="ruleSet" value="drl/OrderEvents.drl" />
```

```
    <property name="ruleReload" value="false" />
    <property name="stateful" value="true" >
    <property name="ruleFireMethod" value="FIRE_UNTIL_HALT" />
    <property name="ruleAuditType" value="THREADED_FILE" />
    <property name="ruleAuditFile" value="myaudit" />
    <property name="ruleAuditInterval" value="1000" />
    <property name="ruleClockType" value="REALTIME" />
    <property name="ruleEventProcessingType" value="STREAM" />
    <property name="ruleMultithreadEvaluation" value="false" />
    <property name="ruleMaxThreads" value="1" />
    <property name="object-paths">
        <object-path esb="body.OrderStatus"
            entry-point="OrderStatusStream" />
        <object-path esb="body.OrderInfo"
            entry-point="OrderInfoStream" />
    </property>
    <property name="channels">
        <!-- chan1 and chan2 are equivalent
            (but timeout only applies if async is false) -->
        <send-to channel-name="chan1"
            service-category="cat1" service-name="svc1" />
        <send-to channel-name="chan2"
            service-category="cat1" service-name="svc1"
            channel-
class="org.jboss.soa.esb.services.rules.ServiceChannel"
            async="true" timeout="30000"
            set-payload-
location="org.jboss.soa.esb.message.defaultEntry" />
        <!-- chan3 is a custom channel -->
        <send-to channel-name="chan3"
            channel-class="com.example.MyChannel" />
    </property>
</action>
```

4. この例では、ルールはドメイン固有言語形式で、ステートレス実行モデルが採用されています。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="PolicyApprovalRuleService">
    <property name="ruleSet" value="dsl/approval.dslr" />
    <property name="ruleLanguage" value="dsl/acme.dsl" />
    <property name="ruleReload" value="true" />
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
    </property>
</action>
```

5. この例ではルールは決定テーブルにあり、ステートレス実行モデルが採用されています。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="PolicyPricingRuleService">
    <property name="decisionTable"
              value="decisionTable/PolicyPricing.xls" />
    <property name="ruleReload" value="true" />
```

```
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
    </property>
</action>
```

6. この例では、ルールは BRMS 形式で、ステートレス実行モデルが採用されています。

```
<action class="org.jboss.soa.esb.actions.BusinessRulesProcessor"
  name="RuleAgentPolicyService">
    <property name="ruleAgentProperties"
              value="ruleAgent/brmsdeployedrules.properties" />
    <property name="object-paths">
        <object-path esb="body.Driver" />
        <object-path esb="body.Policy" />
    </property>
</action>
```

これはアクションタグの属性です。これらの属性を使い実行すべきアクションと、このアクションを渡す名前を指定します。

BusinessRulesProcessor アクション設定属性

属性	説明
クラス	アクションクラス
名前	カスタムアクションの名前

アクションプロパティが表 2 に示されています。これらのプロパティは、このアクションで使うルールセット (ruleSet) を指定します。

BusinessRulesProcessor アクション設定プロパティ

プロパティ	説明
ruleSet	これはコンテンツの評価に利用するルールセットである **ruleSet** を含むファイルへの参照（任意）です。各 **rule service** インスタンスに対して **ruleSet** 1 つのみを渡すことができます。
ruleLanguage	これは、ルールセットの評価に利用するドメイン固有言語の定義を含むファイルへの参照（任意）です。この定義はルールセットの評価に利用できます。利用されている場合、**ruleSet** のファイルは **dslr** であるようにします。

プロパティ	説明
ruleReload	**rule sets** *のホットデプロイメント*を有効にするには、このプロパティ (任意) を **true** に設定します (ただし、この機能を有効にするとルール処理のオーバーヘッドが増加します)。ルールが存在する **.esb** アーカイブが再デプロイされた場合、ルールもリロードされる点に注意してください。
decisionTable	これは、ルール仕様のスプレッドシートの定義を含むファイルへの参照 (任意) です。
stateful	この任意のプロパティを**true** に設定し **rule service** が一定期間中に複数のメッセージを受信するよう指定します (新しいファクトが **rule engine** の**working memory** に追加されルールが毎回再実行されます)。
object-paths	メッセージオブジェクトを JBoss Rules の Working Memory に渡す任意のプロパティ
ruleFireMethod	ステートフルルール実行メソッドを定義する任意のプロパティ。有効な値は FIRE_ALL_RULES (デフォルト) と FIRE_UNTIL_HALT (自身のスレッドをキックオフ。複合イベント処理 (CEP) に便利)
ruleAuditType	Drools に監査ロギングを実行させる任意のプロパティ。ログを Drools Eclipse プラグインに読み込ませ確認することができます。有効な値は CONSOLE、FILE、THREADED_FILE で、デフォルトでは監査ロギングが実行されます。
ruleAuditFile	監査ロギングのファイルパスを定義する任意のプロパティ。FILE あるいは THREADED_FILE ruleAuditType のみに適用されます。デフォルトは "event" です。JBoss Drools は ".log" を自動で追加します。このファイルのデフォルトの場所は "." (現在の作業ディレクトリ、つまり JBoss では bin/ ディレクトリ) となっています。
ruleAuditInterval	監査イベントを監査ログにフラッシュする頻度を定義する任意のプロパティ。これは THREADED_FILE ruleAuditType のみに適用されます。デフォルトは 1000 (ミリ秒) です。
ruleClockType	JBoss Drools が利用するクロックを定義する任意のプロパティ。有効な値は REALTIME および PSEUDO です。デフォルトは REALTIME です。

プロパティ	説明
ruleEventProcessingType	Drools で使うイベント処理オプションを定義する任意のプロパティ。有効な値は CLOUD あるいは STREAM (CEP に便利) で、デフォルトは CLOUD です。
ruleMultithreadEvaluation	ナレッジベースパーティショニングを有効にするかを定義する任意のプロパティ。デフォルトは null で、これは false となっている Drools のデフォルトに委譲します。
ruleMaxThreads	ナレッジベースパーティショニングを利用するスレッドの数を定義する任意のプロパティ。これは、ruleMultithreadEvaluation が true の場合のみ考慮されます。デフォルトは null で、これは JBoss Rules のデフォルトに委譲します。
channels	ルールがオブジェクトを送信できるチャネルを定義する任意のプロパティ。チャネルのコンセプトは従来 ExitPoint として知られるものです。「チャネル」の章を参照してください。

4.2.3. オブジェクトパス

オブジェクトパスについてはこのセクションを参照してください。

JBoss Rules エンジンはオブジェクトを*シャロー*とみなします。このソフトウェアはパフォーマンスを最適化するためこのように設計されました。

`object tree` 階層の下の方にあるオブジェクトを評価したい場合は、任意の**`object tree`** を使います。**`ESB Message Object Path`** を使い、このプロパティを取り出し、このプロパティを設定します。

JBoss Rules エンジンは *MVFLEX Expression Language* (MVEL) を使いオブジェクトを抽出します。お使いのパスは、この構文に従う必要があります。

```
location.objectname.[beanname].[beanname]...
```

この場合：

- **`location`** は {body, header, properties, attachment} のいずれかでなければなりません。
- **`objectname`** はオブジェクトの名前です (attachment は名前をつけるか、数字をつけることができます。そのため、attachment が複数ある場合、数字にすることも可能です)。
- **`beannames`** は任意です。このタグを使い、*bean graph* を探索します。

他の MVEL 式言語例についても以下に示しています。

MVEL 式言語の例

式	結果
properties.Order	これを使い、**Order**という名前のプロパティオブジェクトを取得します。
attachment.1	最初の attachment オブジェクトを取得
attachment.AttachmentOne	**AttachmentOne** という名前の attachment を取得
attachment.1.Order	添付オブジェクト上で**getOrder()**の*リターンオブジェクト*を取得
body.Order1.lineitem	メッセージのオブジェクトから**Order1** という名前のオブジェクトを取得。次に、このオブジェクトで **getLineitem()** が呼び出されます。bean グラフをトラバースするために、他の要素をクエリに追加することもできます。

重要

ルールセットにインポートしたオブジェクトに、**java import** ステートメントを追加するのを忘れないでください。

注記

*オブジェクトマッパー*はコレクション全体を「フラット化」します。 フラット化の必要がある場合、まずメッセージで変換プロセスを行うことでコレクションを展開します。

4.2.4. チャネル

オブジェクトを JBoss Rules エンジンチャネルに送信するには、この DRL 構文を使います (*ルール定義*の右側に行きます)。

```
channels["mychannel"].send(myobject);
```

このコードスニペットを機能させるには、**mychannel** を自身のチャネルに追加します。方法は、**jboss-esb.xml** 設定ファイルに channels プロパティセクションを追加します。

同じあるいは違う *チャネル名*を使うことで、チャネルはいくつでも定義できます (各チャネル名ですが、チャネルは設定ファイルに表示されている順番で実行されます)。

以下のタイプのチャネルに対応しています。

- デフォルトで提供されている **ServiceChannel**。機能させるには send-to 要素の属性として、channel-name、service-category、service-name を指定する必要があります。

  他にasync、timeoutset-payload-location という任意の属性があります (チャネルに送信されるオブジェクトは新しい ESB メッセージに置かれます)。

  このコードは属性の設定方法を表しています。

```
<property name="channels">
    <send-to channel-name="mychannel"
        service-category="cat1" service-name="svc1" />
</property>
```

重要

ターゲットサービスにて **invmScope="GLOBAL"** が定義されているよう確認します。

- 独自の **org.drools.runtime.Channel** 実装クラスを使い指定可能なカスタムのチャネル。この**send-to** 属性は**channel-class**です。お使いの実装には *public な引数なしのコンストラクター*が必要です。

  実装を設定可能にするには、**org.jboss.soa.esb.Configurable** インターフェースを実装し**setConfiguration(ConfigTree()** メソッドが呼び出されるようにします。こうすることで、属性とサブプロパティ要素をカスタムチャネルに渡すことができます。

  このコードサンプルは、カスタムのチャネル設定の方法を示しています。

```
<property name="channels">
    <send-to channel-name="mychannel"
        channel-class="com.example.MyChannel" />
</property>
```

4.2.5. ルールのパッケージングとデプロイ

Red Hat は、ルールコードを機能単位にパッケージするよう推奨しています。これは**.esb** パッケージに入れることで行います**(rule sets**を使う **rule services** の横にルーティングルールをパッケージ化するのが目的です**)**。

jbrules.esb アーカイブを作成すると、**deployment.xml** ファイルを使いデプロイ、参照します。このコードは以下を行う方法を示しています。

```
<jbossesb-deployment>
 <depends>jboss.esb:deployment=jbrules.esb</depends>
</jbossesb-deployment>
```

第5章 コンテンツベースルーティング

5.1. コンテンツベースルーティングとは

5.1.1. はじめに

通常、エンタープライズサービスバスのデータは、 **メッセージ**という形式でパッケージ化、転送、格納されます。メッセージはエンドポイント参照 (これはサービスかクライアントを参照します) があて先となります。エンドポイント参照のロールは、メッセージのコンテンツを参集的に処理するマシン、プロセス、オブジェクトを特定します。しかし、指定のアドレスが無効の場合はどうなるのでしょうか。このシナリオに陥ってしまう状況には、サービスに問題がある、あるいは削除されている場合などが挙げられます。

また、サービスが特定のタイプのメッセージを処理しなくなっている可能性もあります。このような場合、おそらく他のサービスが元の機能を行いますが、それでもメッセージ自体がどのように処理されるのかが疑問に残ります。

元の受け取り側の隣にある他のサービスがこのメッセージの内容を必要としている場合はどうなるのでしょうか。あて先が指定されていない場合はどうでしょう。

5.1.2. コンテンツベースルーティングの紹介

このようなシナリオに対応する方法の１つに、*コンテンツベースルーティング*を使うことが挙げられます。この技術は、指定のエンドポイント参照を使うのではなく、実際のメッセージの内容からあて先を探しメッセージをルーティングしようとします。

コンテンツベースルーティングの機能の仕方ですが、メッセージが開かれそのコンテンツにルールセットを適用し、メッセージのルーティングを行います。これらのルールを使い、どのパーティがメッセージを必要としているか確認することで、エンタープライズサービスバスが送信先を決定することができます。これにより、送信アプリケーションがメッセージの送信先を把握する必要がなくなります。

コンテンツベースルーティングシステムは、*ルーター* (１つのみ) および *サービス* (通常複数個)の２つのコンポーネントをベースに構築されています。

サービスは、最終的にメッセージを「消費する」コンポーネントです。各サービスがルーターに対して特定のメッセージタイプを必要するかを指定する方法は実装により左右されますが、ルーターが正しく指示を出せるようにするには、メッセージタイプ (メッセージコンテンツの別のアスペクト) やサービスの間になんらかのマッピングが存在する必要があります。

ルーターという名前が示しているように、これはメッセージをルーティングします。メッセージを受領するとコンテンツを検証し、コンテンツにルールを適用後ルールの指示どおりメッセージを転送します。

ルーターとサービス以外に、*ハーベスター*を含むシステムもあります。これらのツールのロールは、興味深い情報を探し出し、フォーマットされたメッセージのようなかたちでパッケージされルーターに送信されます。ハーベスターは、 *メール転送エージェント*メッセージストア、ニュースサーバー、データベース、他のレガシーシステムなど、多くの情報源を「マイニング」します。

5.2. XPATH を使ったコンテンツベースルーティング

5.2.1. はじめに

コンテンツベースのルーティングを簡単に実行するにはは、**ContentBasedRouter** アクションで *XPath ルールプロバイダー*を実行します。このプロバイダーの利用は非常に簡単で、*インライン*および*外部ルールの定義*の両方に対応しています。

手順5.1 インラインルールの定義

1. cbrAlias プロパティを **XPath** に設定します。
2. (container destinations プロパティ にある) **route-to** 設定のルーティングルールを定義します。

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="XPath"/>
   <property name="destinations">
       <route-to service-category="BlueTeam"  service-name="GoBlue"
expression="/Order[@statusCode='0']" />
       <route-to service-category="RedTeam"   service-name="GoRed"
expression="/Order[@statusCode='1']" />
       <route-to service-category="GreenTeam" service-
name="GoGreen" expression="/Order[@statusCode='2']" />
   </property>
</action>
```

手順5.2 外部ルールの定義

1. cbrAlias プロパティを **XPath** に設定します。
2. **.properties** ファイルのルーティング式を定義します。ここでは*プロパティキー*があて先の名前で、該当のあて先へのルーティングに関するプロパティ値が XPath 式となっています。
3. container destinations プロパティの **route-to** 設定にてルーティングルールを定義します。外部 **properties** ファイルに定義されているように destination-name 属性は、*XPath ルールキー*を参照します。

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="XPath"/>
   <property name="ruleSet" value="/rules/xpath-rules.properties"/>
   <property name="ruleReload" value="true"/>
   <property name="destinations">
       <route-to destination-name="blue"  service-
category="BlueTeam"  service-name="GoBlue" />
       <route-to destination-name="red"   service-
category="RedTeam"   service-name="GoRed" />
       <route-to destination-name="green" service-
category="GreenTeam" service-name="GoGreen" />
   </property>
</action>
```

XPath ルールは **.properties** ファイルにあります。以下の構文で表現されます。

```
blue=/Order[@statusCode='0']
red=/Order[@statusCode='1']
green=/Order[@statusCode='2']
```

5.2.2. 名前空間

*XML 名前空間プリフィックスと URI マッピング*は、name-space 要素で定義します (これらは namespaces コンテナープロパティ内に置かれています)。

手順5.3 XML 名前空間プリフィックスと URI のマッピングの定義

- XML 名前空間プリフィックスと URI マッピングを外部もインラインルール定義も同じ方法で定義します。

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
    <property name="cbrAlias" value="XPath"/>
    <property name="namespaces">
        <route-to prefix="ord" uri="http://www.acne.com/order" />
    </property>
    <property name="destinations">
        <route-to service-category="BlueTeam"
        service-name="GoBlue"
expression="/ord:Order[@statusCode='0']" />
        <route-to service-category="RedTeam"
        service-name="GoRed"
expression="/ord:Order[@statusCode='1']" />
        <route-to service-category="GreenTeam"
        service-name="GoGreen"
expression="/ord:Order[@statusCode='2']" />
    </property>
</action>
```

5.3. REGEX を使ったコンテンツベースルーティング

5.3.1. はじめに

ContentBasedRouter アクションの *Regex ルールプロバイダー*を使いコンテンツベースのルーティングを実行することも可能です。このプロバイダーは、インラインおよび外部のルール定義の両方に対応しています。

手順5.4 インライン Regex ルーティングルールの定義

1. cbrAlias プロパティを **Regex** に設定します。
2. (container destinations プロパティ にある) **route-to** 設定のルーティングルールを定義します。

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="Regex"/>
   <property name="destinations">
       <route-to service-category="BlueTeam"  service-name="GoBlue"
```

```
expression="#*111#*" />
       <route-to service-category="RedTeam"   service-name="GoRed"
expression="#*222#*" />
       <route-to service-category="GreenTeam" service-
name="GoGreen" expression="#*333#*" />
   </property>
</action>
```

手順5.5 外部のルール定義の設定

1. cbrAlias プロパティを **Regex** に設定します。
2. **.properties** ファイルのルーティング式を定義します。ここではプロパティキーがあて先の名前で、該当のあて先へのルーティングに関するプロパティ値が Regex 式となっています。
3. container destinations プロパティの **route-to** 設定にてルーティングルールを定義します。外部 **.properties** ファイルに定義されているように destination-name 属性を設定し、Regex ルールキーを参照します。

```
<action class="org.jboss.soa.esb.actions.ContentBasedRouter"
name="ContentBasedRouter">
   <property name="cbrAlias" value="XPath"/>
   <property name="ruleSet" value="/rules/regex-rules.properties"/>
   <property name="ruleReload" value="true"/>
   <property name="destinations">
       <route-to destination-name="blue"  service-
category="BlueTeam"  service-name="GoBlue" />
       <route-to destination-name="red"   service-
category="RedTeam"   service-name="GoRed" />
       <route-to destination-name="green" service-
category="GreenTeam" service-name="GoGreen" />
   </property>
</action>
```

XPath ルールは **.properties** ファイルにあります。以下のシンプルな形式で表現されます。

```
blue=#*111#*
red=#*222#*
green=#*333#*
```

5.4. JBOSS RULES エンジンを使ったコンテンツベースルーティング

5.4.1. はじめに

JBoss Rules は、コンテンツベースルーターのルールプロバイダーエンジンです。Enterprise Service Bus は、３つのルーティング **action classes** を使い、このエンジンを統合します。この３つの action classes は以下のとおりです。

- **JBoss Rules** エンジンの DRL 言語で記述されたルーティングルールセット (あるいは、希望であれば DSL 言語を利用可能)
- メッセージコンテンツ。これは、JBoss Rules エンジンへ入るデータです (XML 形式か、メッセージに埋め込まれたオブジェクト)。

- デスティネーション (エンジンから出された結果情報)

メッセージは **content-based router** に送信されると、ルールセットはコンテンツを評価し、*サービスのあて先*を返します。

本章の残りで、ルールセットの作成方法、メッセージコンテンツの評価方法、この評価プロセスから出されたあて先で何ができるかについて説明します。

5.4.2. 3 種類のルーティングアクションクラス

JBoss Enterprise SOA Platform には、ルーティング *アクションクラス*が同梱されており、それぞれ少し違います。これらは、*エンタープライズ統合パターン*を実装しています。 以下にその３つのアクションクラスを挙げます。

1. **org.jboss.soa.esb.actions.ContentBasedRouter**

   このアクションクラスは、コンテンツベースのルーティングパターンを実装します。メッセージンコンテンツとコンテンツの評価するルールセットに従い、メッセージを１つ以上のあて先サービスにルーティングします。あて先が指定のルールセットあるいはメッセージの組み合わせと一致しない場合、コンテンツベースのルーターは例外をスローします。このアクションは、それ以降のパイプライン処理を終了するためパイプラインの最後に置かれます。

2. **org.jboss.soa.esb.actions.ContentBasedWiretap**

   これは、*WireTap* パターンを実装します。**WireTap** は、メッセージのコピーを制御チャネルへ送信するエンタープライズ統合パターンです。**WireTap** は、標準のコンテンツベースルーターの機能と同じですが、パイプラインは終了しません。この機能が **wire-tap** として利用するのに適しているため、そのような名前がついています。

3. **org.jboss.soa.esb.actions.MessageFilter**

   これは *メッセージフィルターパターン*を実装します。あるコンテンツ要件を満たされなかったためメッセージが単にフィルタリングされる場合にメッセージフィルターパターンを使います。つまり、コンテンツベースルーターと同じように機能しますが、ルールセットがあて先と一致しない場合に例外をスローせず単にメッセージをフィルタリングするだけという点が違います。

5.4.3. ルールセットの作成

ルールセットを作成するには、**JBoss Developer Studio.** を使います (このアプリケーションには、特別なプラグインが含まれており、このタスクを容易に実行できるようになります)。

通常のルールセットをコンテンツベースルーティングに利用できるようにするには、**ESB** メッセージを評価し、ルール一致により *サービスあて先名*のリストを出力するようにする必要があります。これを行う前に以下を確認します。

- ルールセットが **ESB** メッセージをインポートするようにします。また、テストを行うにはこのコードを使います。

  ```
  import org.jboss.soa.esb.message.Message
  ```

- ルールセットがあて先一覧を作成する以下のグローバル変数を定義しているようにします。

  ```
  global java.util.List destinations;
  ```

このメッセージは JBoss Rules エンジンの作業メモリに送信されました。

5.4.4. XPath ドメイン固有言語

XML ベースメッセージの XPath ベース評価は便利であると感じるかもしれません。Red Hat は *ドメイン固有言語* 実装を同梱することで、これに対応しています。この実装を使い XPath 式をルールファイルに追加します。

追加の方法は、**XPathLanguage.dsl** ファイルで式を定義し、以下のコードとあわせてルールセットのファイルを参照します。

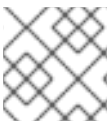
```
expander XPathLanguage.dsl
```

XPath 言語により、メッセージが **JBOSS_XML** のタイプで、以下のアイテムが定義されていることを確認します。

1. **xpathMatch** <*element*>: この名前による要素がマッチする場合は、**true** を出します。
2. **xpathEquals** <*element*>、 <*value*>: 要素が見つかりその値が指定値と同じ場合に **true** を出します。
3. **xpathGreaterThan** <*element*>、<*value*>: 要素が見つかりその値が指定値よりも大きい場合に **true** を出します。
4. **xpathLessThan** <*element*>、 <*value*>: 要素が見つかりその値が指定値よりも小さい場合に **true** を出します。

注記

fun_cbr クイックスタートは XPath の使用方法について説明します。



注記

全く違うドメイン固有言語を定義することができます。

5.4.4.1. XPath および名前空間

XPath と名前空間を使うには、名前空間プリフィックスを XPath 式で使うかを指定しmさう。名前空間プリフィックスは、この形式 (**prefix=uri,prefix=uri**) でコンマ区切り一覧として指定します (上述した各種 XPath 式でも実行可能)。

1. **xpathMatch expr "<expression>" use namespaces "<namepaces>"**
2. **xpathEquals expr "<expression>", "<value>" use namespaces "<namspaces>"**
3. **xpathGreaterThan expr "<expression>", "<value>" use namespaces "<namspaces>"**
4. **xpathLowerThan expr "<expression>", "<value>" use namespaces "<namespaces>"**

名前空間対応のステートメントはすべて、XPath 式の最初に **expr** というキーワードをつける必要があります (これを追加することで、DSL ファイルで XPath 対応でないステートメントと衝突を起こさないようにするためです)。このプリフィックスは、評価する際に XML で利用したものとマッチする必要

はありません。統一資源識別子 (URI: Uniform Resource Identifier) が同じにすることが重要です。

5.4.4.2. 設定

これらはそれぞれ、構成設定を介してすべて接続されており、**jboss-esb.xml** ファイルに保存されます。以下の **service configuration** は、サービス設定のフラグメント例です。このフラグメントでは、サービスは Java Message Service キューをリッスンしています。

```
<service
         category="MessageRouting"
         name="YourServiceName"
         description="CBR Service">
         <listeners>
                  <jms-listener name="CBR-Listener"
                           busidref="QueueA" maxThreads="1">
            </jms-listener>
               </listeners>
  <actions>
       <action class="org.jboss.soa.esb.actions.ContentBasedRouter"
                name="YourActionName">
                     <property name="ruleSet" value="JBossESBRules.drl"/>
                     <property name="ruleReload" value="true"/>
                     <property name="destinations">
                           <route-to destination-name="xml-destination"
        service-category="category01"
              service-name="jbossesbtest1" />
                           <route-to destination-name="serialized-
destination"
                                                 service-category="category02"
        service-name="jbossesbtest2" />
                     </property>
                     <property name="object-paths">
    <object-path esb="body.test1" />
    <object-path esb="body.test2" />
        </property>
                   </action>
  </actions>
            </service>
```

各メッセージは **ContentBasedRouter** アクションクラスに渡され、特定のルールセットにロードされます。次に、JBoss Rules エンジンの作業メモリへメッセージを送信、ルールを実行、あて先一覧を取得、メッセージのコピーをサービスに送信します。

この場合、**JBossESBRules.drl** ルールセットを使い **xml-destination** と **serialized-destination** の 2 つのあて先にマッチします。これらの名前は、**route-to** セクションの真のサービスにマッピングされます。

このテーブルは action タグの属性を示しています。これらの属性は使用する**action** と渡す名前を指定します。

表5.1 CBR アクション設定の属性

属性	説明
Class	アクションクラスで、 以下のいずれかになります :**org.jboss.soa.esb.actions.ContentBasedRouter**、**org.jboss.soa.esb.actions.ContentBasedWiretap** あるいは **org.jboss.soa.esb.actions.MessageFilter**
Name	カスタムアクションの名前

この表はアクションプロパティを示しています。このプロパティは、どのルールセット (**ruleSet**) をアクション内で利用するか指定します。

表5.2 アクション設定プロパティ

プロパティ	説明
ruleSet	これは、**JBoss Rules engine****ruleSet** を含むファイル名で、メッセージコンテンツを評価するのに利用するルールセットです(各コンテンツベースルーティングインスタンスに対し、**ruleSet** １つのみを渡すことができます)。
ruleLanguage	これは、ルールセットの評価に利用するドメイン固有言語の定義を含むファイルへの参照（オプション）です。
ruleReload	これはオプションのプロパティで、ルールセットのホット再デプロイメントを有効にするには **true** に設定します。この機能はルール処理のオーバーヘッドを引き起こします。また、ルールが置かれている **.esb** アーカイブが再デプロイされると、ルールがリロードされる点にも注意してください。
ステートフル	これはオプションのプロパティですが、RulesService に (呼び出しと呼び出しの間にファクトを記憶させる) ステートフルセッションを使うよう伝えます。詳細については「ステートフルルール」の章を参照してください。
destinations	これは route-to プロパティセットで、レジストリ内に参照されたサービスカテゴリとサービス名にあわせ、サービスカテゴリと名前あて先の論理名を含みます。論理名はルールセットで利用する必要のある名前です。
object-paths	これはオプションのプロパティで **message** オブジェクトを**working memory**に渡します。
ruleAuditType	これはオプションのプロパティで、JBoss RUles エンジンが監査ロギングを実行できるようにします。ログは JBoss Developer Studio プラグインに読み込まれ検証されます。有効な値は、CONSOLE、FILE、THREADED_FILE です。デフォルトでは、監査ロギングは実行されません。
ruleAuditFile	これはオプションのプロパティで、監査ロギングのファイルパスが定義できるようになります。FILE か THREADED_FILE の ruleAuditType のみに適用される点に注意してください。デフォルトは "event" です。JBoss Rules エンジンは".log" を追加します。このファイルのデフォルトの場所は、"."と現在の作業ディレクトリ (JBoss では bin/ ディレクトリ内) です。

プロパティ	説明
ruleAuditInterval	これはオプションのプロパティで、監査イベントを監査ログにフラッシュする頻度を定義することができます。これは THREADED_FILE ruleAuditType のみに適用されます。デフォルトは 1000 (ミリ秒) です。

5.4.4.3. ステートフルルールセッション

ステートフルセッションを使うと、ファクトオブジェクトは各呼び出しの間で記憶されます (言い換えると、ステートフルが **true**に設定されると、作業メモリは消去されません)。

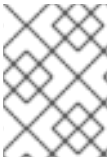
メッセージプロパティを使い現在のセッションを継続するか、消去するかステートフルルールサービスに伝えます。

既存のステートフルセッションを継続するよう通知するには、以下の２つのメッセージプロパティを設定します。

```
message.getProperties().setProperty(“dispose”, false);
message.getProperties().setProperty(“continue”, true);
```

最後にルールを実行する場合、常に dispose を **true** に設定し JBoss Rules エンジンの作業メモリを消去します。

```
message.getProperties().setProperty(“dispose”, true);
message.getProperties().setProperty(“continue”, true);
```



注記

ステートフルルールセッションの利用に関する詳細は、**business_ruleservice_stateful** クイックスタートを参照してください。

5.4.4.4. 事前コンパイル済みのルールパッケージの利用

コンパイル済みのルールパッケージを使うには **KnowledgeAgent** を使います。これらのパッケージはローカルファイルシステム、あるいはリモートロケーション (URL 経由でアクセス) に置くことが可能です。

5.4.4.5. ビジネスルールの実行

ビジネスプロセスに従いメッセージのコンテンツを変更するというルールを実行するのは、ルーティングのルールを実行する場合と非常によく似ています。**business_rule_service** というクイックスタートのサンプルでこれについて例示しています (このクイックスタートは **org.jboss.soa.esb.actions.BusinessRulesProcessor** アクションクラスを利用します)。

このプロセスは*Business Rule Processor*と呼ばれるコンポーネントを使います。これはルーターにほぼ同じで、違いは以降の処理のために**action pipeline** に変更メッセージを返す点です。

希望であればビジネスとルーティングルールを１つのルールセットで組み合わせることもできます (ただし、前述した３つのルーティング**action** クラスの１つを使う場合のみ機能します)。

5.4.4.6. ルールサービス実装の変更

このルールサービスは **org.jboss.soa.esb.services.rules.RuleService** インターフェースを実装します。デフォルトルールサービスを利用したくない場合、アクション設定で使いたいクラスを指定します。

```
<property name="ruleServiceImplClass" value="org.com.YourRuleService" />
```

第6章 メッセージの変換

メッセージのフォーマットを変換する方法としてメカニズムが３つ存在します。

Smooks

Smooks は　フラグメントベースのデータ変換および分析ツールです。Smooks は、XML、EID、CSV、JAVA など幅広いソースやターゲットデータの形式を理解します。また、データ処理や操作機能も含まれています。

XSLT

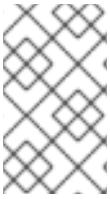
XSLT は Smooks の代わりとなるものです。

ActionProcessor データ変換

Smooks が特定の種類の変換を処理できない場合、このインターフェース (**`org.jboss.soa.esb.actions.ActionProcessor`**) を使いカスタムソリューションを実装します (この機能は廃止予定であるため、 ActionPipelineProcessor を利用してください)。

手順6.1 Smooks の使用

- **`SmooksAction`** コンポーネントを使い Smooks を ESB アクションパイプラインへプラグインします。



注記

`samples/quick starts` ディレクトリに変換のデモを行うクイックスタートがいくつかあります(これらのクイックスタートの変換名は **`transform_`** とのプリフィックスがついています)。

注記

ESB プログラマーガイドには変換に関する詳細情報が含まれています。

第7章 JBPM 統合

7.1. はじめに

JBoss Business Process Manager(JBPM) 統合について学習するには、この章を参照してください (JBPM の基礎を把握しているとの前提で進めています。基礎を把握してない場合は、**JBPM Reference Guide** をまず参照してください)。

JBoss Business Process Manager はワークフローおよび *business process management* エンジンです。JBPM により、ビジネスプロセスの設計を行うように、人、アプリケーション、サービスを連携することができます。

ビジネスのプロシージャにて手順を図で作成したい場合は、JBPM の *Process Designer* 機能をご利用ください (このツールのもう 1 つ利点は、ビジネスアナリストと技術開発者との良好な関係を構築しやすくなります)。

JBoss Enterprise Service Bus が **JBPM** コンポーネントと統合される理由は 2 つあります。

1. サービスオーケストレーション

   *プロセス定義*を作成することで、**Business Process Manager** を使いサービスの*オーケストレーション*が可能になります。

2. ヒューマンタスク管理

   Business Process Manager により、マシンベースのサービスと人が行うタスク管理を統合することができます。

7.2. 統合の設定

完全な JBPM ランタイムエンジンとコンソールは JBoss Enterprise SOA Platform に含まれており、**`jbpm.esb`** ディレクトリに置かれています。

エンジンとコンソールは共通のデータベースを使っています。データベースを作成するには **`DatabaseInitializer`** *m-bean* を起動します (**`jbpm.esb/jbpm-service.xml`** ファイルの最初の設定要素に MBean 関連の設定があります)。

```
<classpath codebase="deploy" archives="jbpm.esb"/>
<classpath codebase="deploy/jbossesb.sar/lib"
   archives="jbossesb-rosetta.jar"/>

<mbean code="org.jboss.internal.soa.esb.dependencies.DatabaseInitializer"
   name="jboss.esb:service=JBPMDatabaseInitializer">
   <attribute name="Datasource">java:/JbpmDS</attribute>
   <attribute name="ExistsSql">select * from JBPM_ID_USER</attribute>
   <attribute name="SqlFiles">
       jbpm-sql/jbpm.jpdl.hsqldb.sql,jbpm-sql/import.sql
   </attribute>
   <depends>jboss.jca:service=DataSourceBinding,name=JbpmDS</depends>
</mbean>

<mbean code="org.jboss.soa.esb.services.jbpm.configuration.JbpmService"
   name="jboss.esb:service=JbpmService">
</mbean>
```

表7.1 DatabaseInitializer MBean のデフォルト値

プロパティ	説明	デフォルト
Data Source	JBPM データベースのデータソース	java:/JbpmDS
ExistsSql	この SQL コマンドを使いデータベースの存在を確認します。	JBPM_ID_USER から * を選択
SqlFiles	これらのファイルには、SQL コマンドが含まれており、データベースがない場合は JBPM データベースを作成します。	jbpm-sql/jbpm.jpdl.hsqldb.sql, jbpm-sql/import.sql

デフォルトでは、**DatabaseInitializer** MBean が設定され、**JbpmDS** がデプロイされるまで待機し、その後 MBean が自身でデプロイを行います。

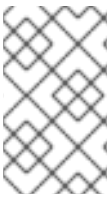
２つ目の MBean **JbpmService** は、**JBoss Business Process Manager** の*ジョブエグゼキューター*のライフサイクルと **jbpm.esb**のライフサイクルを連携します。**job executor** インスタンスを起動時に開始し、シャットダウン時に終了します。

警告

JbpmDS データソースを **jbpm-ds.xml** ファイルで定義します。デフォルトでは、**Hypersonic** データベースを使います。実稼働環境では、常にこれを実稼働品質のデータベースに変更してください。

警告

JBoss Enterprise SOA Platform にはインメモリ参照データベースの **Hypesonic** も同梱されています。これに関してもテスト環境でのみ利用してください。



注記

jbpm.esb デプロイメントはすべて同じデータベースインスタンスを共有するようにしてください (様々なエンタープライスサービスバス (ESB) ノードが同じプロセス定義にアクセスできるようにするためです)。

サーバーが起動すると、**JBPM Console** にアクセスできます。これは web ベースのアプリケーションで、http://localhost:8080/gpd-deployer のアドレスからアクセスしてください。

このアプリケーションのセキュリティ設定を変更する方法については、**JBPM Reference Guide** を参照してください。方法は、**conf/login-config.xml** ファイルの設定を変更します。

JBPM Console を使い JBPM プロセスとヒューマンタスク管理のプロシージャの両方をデプロイ、監視することができるようになりました (カスタマイズされたタスク一覧がユーザーごとに表示されるため、自分のタスクに関する作業を行うことができます)。**bpm_orchestration4** クイックスタートはこの機能のデモを提供しています。

jbpm.esb/META-INF ディレクトリには、**deployment.xml** と **jboss-esb.xml** ファイルが含まれています。

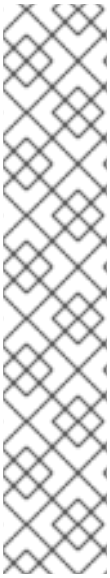
deployment.xml ファイルは、ESB アーカイブが従属する ２つのリソースファイルを指定します (**jbossesb.esb** と **JbpmDS** のデータソースファイルがあります)。 これらのファイルの情報を使いデプロイメントの順番を判断します。

```
<jbossesb-deployment>
  <depends>jboss.esb:deployment=jbossesb.esb</depends>
  <depends>jboss.jca:service=DataSourceBinding,name=JbpmDS</depends>
</jbossesb-deployment>
```

jboss-esb.xml ファイルは **JBpmCallbackService** と呼ばれる、内部サービスを１つデプロイします。

```
<services>
  <service category="JBossESB-Internal" name="JBpmCallbackService"
    description="Service which makes Callbacks into jBPM">
    <listeners>
      <jms-listener name="JMS-DCQListener"
        busidref="jBPMCallbackBus" maxThreads="1" />
    </listeners>
    <actions mep="OneWay">
      <action name="action"
        class="org.jboss.soa.esb.services.jbpm.actions.JBpmCallback"/>
    </actions>
  </service>
</services>
```

この内部サービスは **jBPMCallbackBus** をリッスンします。このjBPMCallbackBus はデフォルトでは **JBossMQ** (**jbmq-queue-service.xml** ファイル経由) あるいは **JBossMessaging** (**jbm-queue-service.xml** ファイル経由) になるように設定されます。後者は *Java Message Service Queue* のメッセージプロバイダーです。このファイルのいずれかのみが **jbpm.esb** アーカイブにデプロイされるようにしてください。



注記

ご自身のメッセージングプロバイダーを利用になりたい場合は、**jboss-esb.xml** ファイルの該当セクションにて、そのメッセージングプロバイダーを参照するように変更してください。以下に例を示しています。

```
<providers>
  <jms-provider name="CallbackQueue-JMS-Provider"
    connection-factory="ConnectionFactory">
    <jms-bus busid="jBPMCallbackBus">
      <jms-message-filter dest-type="QUEUE"
        dest-name="queue/CallbackQueue" />
    </jms-bus>
  </jms-provider>
</providers>
```

7.3. JBPM の設定

JBoss Business Process Manager の設定は、**jbpm.cfg.xml**、**hibernate.cfg.xml**、**jbpm.mail.templates.xml** の３つのファイルに保存されます。

jbpm.cfg.xml ファイルを JBPM が *JTA Transaction Manager* を使うように設定します。

```
<service name="persistence">
  <factory>
    <bean class="org.jbpm.persistence.jta.JtaDbPersistenceServiceFactory">
      <field name="isTransactionEnabled"><false/></field>
      <field name="isCurrentSessionEnabled"><true/></field>
      <!--field name="sessionFactoryJndiName">
      <string value="java:/myHibSessFactJndiName" />
      </field-->
    </bean>
  </factory>
</service>
```

また、**hibernate.cfg.xml** ファイルは、JBPM が **JTA Transaction Manager** を使うように指示をだします。

```
<!-- JTA transaction properties (begin) ===
    ==== JTA transaction properties (end) -->
<property name="hibernate.transaction.factory_class">
  org.hibernate.transaction.JTATransactionFactory</property>

<property name="hibernate.transaction.manager_lookup_class">
  org.hibernate.transaction.JBossTransactionManagerLookup</property>
```

注記

データベーススキーマの作成には、**Hibernate** を使わないでください。代わりに、**DatabaseInitializer** MBean を使うようにしてください。

デフォルトでは、**jbpm.mail.templates.xml** ファイルは空となています。

注記

これらの設定ファイルの詳細は、**JBPM Reference Guide** を参照してください。

7.4. プロセス定義の作成とデプロイ

Red Hat は **JBoss Developer Studio** の *JBPM Process Designer Plug-in* (KA-JBPM-GPD) を使いプロセス定義を作成するよう推奨しています。手動でダウンロードしインストールするか、**JBoss Developer Studio** を使い実行してください。

JBPM Graphical Editor を使いプロセス定義を視覚的に作成します。ノードとノード間の移行を追加、削除、変更できます。このソフトが XML 形式でプロセス定義を保存します。

このファイルが保存されると、JBPM インスタンス (つまりデータベース) にデプロイできます。プロセスインスタンスをデプロイする度に JBPM はそのインスタンスをバージョニングし古いコピーも保存します。これにより、実行中のプロセスは既存のインスタンスを使って完了します。反対に、新規作成されたインスタンスはプロセス定義の最新のバージョンを使います。

プロセス定義をデプロイするには、サーバーが実行中か確認します。

次に、Graphical Editor の **Deployment** タブに移動し、*プロセスアーカイブ*を有効にします。

注記

processdefinition.xml ファイルしかデプロイする必要がない場合もありますが、通常は*タスクフォーム*といった、他の**artifacts** もデプロイします。

警告

process archive に Java クラスをデプロイすることも可能です。つまり、最終的にデータベースに送られ、そこで保存、バージョニングされます。Red Hat では、Process archive への Java クラスのデプロイは推奨していません。理由としては、クラスローディングの問題が発生する可能性があるためです。代わりに、これらのクラスをサーバーの **lib** ディレクトリにデプロイしてください。

次に、以下の方法を使いプロセス定義をデプロイします。

1. **JBoss Developer Studio** の **Deploy Process Archive** ボタンをクリックする方法 (まず、デプロイヤーを使いアップロードサーブレットを設定してください)。**Deployment** ビューで確認できます。

2. **Deployment** ビューからローカル **.par** ファイルにデプロイメントを保存し、JBPM Console を使いアーカイブを有効にする方法 (これには管理者権限でコンソールにログインしている必要があります)。

3. **DeployProcessToServer** JBPM **ant** タスクを使う方法

7.5. ENTERPRISE SERVICE BUS から JBPM へ

JBoss Enterprise Service Bus は、**BpmProcessor** アクションを使い **JBoss Business Process Manager** へ呼び出すことができます。このアクションは、**JBPM Command API** を使います。利用可能な JBPM コマンドを以下に挙げています。

表7.2 JBPM コマンド

コマンド	説明
NewProcessInstanceCommand	このコマンドは、新規の **ProcessInstance** を開始し、そのうち、関連のプロセス定義が JBPM にデプロイされています。**NewProcessInstanceCommand** は **start** のステータスににプロセスインスタンスをのコアします。例えば、*actor* のタスクリストに１つある場合などです。
StartProcessInstanceCommand	これは、新規プロセスインスタンスが **start** の位置から最初のノードに自動的に移動される以外は、**NewProcessInstanceCommand** と同じです。
GetProcessInstanceVariablesCommand	このコマンドは、プロセスインスタンス識別子を使い、プロセスインスタンスにルートノードを取ります。
CancelProcessInstanceCommand	このコマンドは、**ProcessInstance** を取り消します。これは、イベントを受け取り **ProcessInstance** 全体をキャンセルする必要がある場合などに利用します (このアクションは JBPM コンテキスト変数をメッセージに設定する際に必要になります。特に **ProcessInstance** 識別子などです)。

このアクションに対する **jboss-esb.xml** の設定は以下のようになります。

```
<action name="create_new_process_instance"
  class="org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">

  <property name="command" value="StartProcessInstanceCommand" />
  <property name="process-definition-name" value="processDefinition2"/>
  <property name="actor" value="FrankSinatra"/>

  <property name="esbToBpmVars">
  <!-- esb-name maps to getBody().get("eVar1") -->
    <mapping esb="eVar1" bpm="counter" default="45" />
    <mapping esb="BODY_CONTENT" bpm="theBody" />
  </property>

</action>
```

以下の２つの属性を利用する必要があります。

1. 名前

action pipelineで一意であれば、この name 属性の値のいずれかを使います。

2. クラス

   常に、この属性を **org.jboss.soa.esb.services.jbpm.actions.BpmProcessor** に設定します。

これらの任意のプロパティも設定することができます。

表7.3 設定プロパティ

プロパティ	説明	必須
command	これは、**NewProcessInstanceCommand**、**StartProcessInstanceCommand**、**GetProcessInstanceVariablesCommand**、**CancelProcessInstanceCommand** のいずれかでなければなりません。	はい
process-definition-name	このプロパティは、process-definition-id プロパティを使わない場合、**NewProcessInstanceCommands** と **StartProcessInstanceCommands** に必要です。このプロパティの値は、新規インスタンスを作成したいと考えているデプロイ済みのプロセス定義を参照する必要があります (このプロパティは**CancelProcessInstanceCommand** に適用されません)。	場合による
process-definition-id	これは、process-definition-name プロパティが使用されていない場合、**NewProcessInstanceCommands** と **StartProcessInstanceCommands** には必須プロパティです。このプロパティの値は、新規インスタンスを作成したいと考えているデプロイ済みのプロセス定義を参照する必要があります (このプロパティは **CancelProcessInstanceCommand** に適用されません)。	場合による
actor	このプロパティを使い **JBPM actor** 識別子を指定します (**NewProcessInstanceCommand** と **StartProcessInstanceCommand** にのみ適用します)。	いいえ
key	このプロパティを使い JBPM キーを指定します。このキーは、プロセスインスタンス上にある文字列ベースのビジネスキープロパティです。ビジネスキーが提供されている場合、ビジネスキーとプロセス定義の組み合わせは一意でなければなりません。キーの値は、MVEL 式を保持することができ、EsbMessage から希望の値を抽出します。例えば、メッセージボディに**businessKey** と呼ばれるパラメーターを置きたい場合、body.businessKey を使います (このプロパティは **NewProcessInstanceCommand** と **StartProcessInstanceCommands** にのみ適用します)。	いいえ

プロパティ	説明	必須
transition-name	これは **StartProcessInstanceCommand** のみに適用します。現在のノードからの遷移が複数ある場合にのみ使います。このプロパティが指定されていない場合は、このノードからのデフォルトの遷移が取られます。 デフォルトの遷移は JBPM **processdefinition.xml** のノードに定義された遷移一覧にある一番最初のものです。	いいえ
esbToBpmVars	これは **New-** と **StartProcessInstanceCommand** の任意のプロパティです。これは、ESB メッセージから抽出し特定のプロセスインスタンスに対し JBPM コンテキストに設定する必要のある変数一覧を定義します。この一覧はマッピング要素で構成されており、それぞれ以下の属性を含みます。 • esb これは必須属性です。ここに MVEL 式を置き、ESB メッセージの中から値を抽出するのに利用します。 • bpm これは任意属性で、JBPM 側で利用する名前が含まれています (省略されている場合、エンタープライズサービスバスの名前を利用します)。 • default これは任意の属性で、ESB の MVEL 式により ESB メッセージ内に設定されている値を発見できない場合に利用するデフォルト値を保持することができます。 • bpmToEsbVars これは上記の esbToBpmVars と構造的には同じです。**GetProcessInstanceVariablesCommand** とあわせて利用し JBPM**process instance variables** (**root token** 変数) を ESB メッセージにマッピングします。 • reply-to-originator これは **New-** と **StartProcessInstanceCommands** の任意のプロパティです。 Specify a value of **true** を指定すると、プロセスインスタンスがそのインスタンス内に呼び出し中のメッセージエンドポイント参照の **ReplyTo/FaultTo** 値を保存します。これらの値を後続の **EsbNotifier/EsbActionHandler** 呼び出しで利用しメッセージを **ReplyTo/FaultTo** アドレスに配信することができます。	いいえ

EsbMessage のボディに変数が設定されました。

表7.4 ボディの設定プロパティ

プロパティ	説明	必須
jbpmProcessInstId	GetProcessInstanceVariablesCommand と CancelProcessInstanceCommand コマンドに適用する必須の ESB メッセージボディパラメーター。この値を EsbMessage ボディに名前付きのパラメーターとして設定するかはユーザー次第です。	はい

7.5.1. ESB-to-JBPM 例外処理

ESB が呼び出された場合、**JbpmException** が JBPM Command API からスローされる可能性があります。この例外は **action pipeline** に渡されます。このパイプラインでエラーをロギングし、メッセージを **DeadLetterService** に転送後、エラーを **faultTo** エンドポイント参照に送信します (faultTo が設定されていると仮定)。

7.6. JBPM-TO-ESB 統合

JBPM-to-JBossESB の通信により、サービスオーケストレーションに JBPM を利用できるようになります。

JBPM アクションハンドラークラス 2 つ (**EsbActionHandler** と **EsbNotifier**) を利用しこの統合を実現します。**EsbActionHandler** は request-reply タイプアクションでメッセージをサービスに送信しレスポンスを待ちます。反対に、**EsbNotifier** はレスポンスを待ちません。

注記

エンタープライズサービスバスとのやり取りは*非同期的*であるため、サービス実行中にプロセスインスタンスをブロックしません。

警告

JBPM と JBoss ESB サービスを渡す変数を表現するクラスが JBPM プロセス、対象の ESB サービス、ESB JBPM コールバックサービスで見える状態であることが重要です。このような理由から、Red Hat はこれらのクラスをサーバーの**lib** ディレクトリにデプロイするように推奨しています。

EsbNotifier アクションは **EsbActionHandler**のサブセットを実装するため、まずこのアクションについて学習します。

7.6.1. ESB Notifier Action

EsbNotifier アクションは、出力遷移に紐付けする必要があります。これは、バックグラウンドで ESB サービスへのリクエストを実行している間にJBPM が継続して処理ができるようにするためです。**EsbNotifier** を JBPM **processdefinition.xml** ファイルの *出力遷移　(outgoing transition)* に紐付けする必要があります。

-

```
<node name="ShipIt">
  <transition name="ProcessingComplete" to="end">
    <action name="ShipItAction"
      class="org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
      <esbCategoryName>BPM_Orchestration4</esbCategoryName>
      <esbServiceName>ShippingService</esbServiceName>
      <bpmToEsbVars>
        <mapping bpm="entireCustomerAsObject" esb="customer" />
        <mapping bpm="entireOrderAsObject" esb="orderHeader" />
        <mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
      </bpmToEsbVars>
    </action>
  </transition>
</node>
```

以下の属性を指定することができます。

- 名前

  これは必須です。アクションのユーザー固有名。

- クラス

  必須。**org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier** に設定する必要があります。

以下のサブ要素を指定することも可能です。

- **esbCategoryName**

  これは ESB サービスのカテゴリ名で、reply-to-originator 機能を利用しない場合は必須です。

- **esbServiceName**

  これは ESB サービス名で、reply-to-originator 機能を利用しない場合は必須です。

- **replyToOriginator**

  これを使い、作成時にプロセスインスタンスに保存されていた reply あるいは fault 送信元アドレスを指定します。

- **globalProcessScope**

  この要素は任意の Boolean 値パラメーターです。これを使い、**bpmToEsbVars** 変数の検索を行うデフォルト範囲を設定します。**globalProcessScope** が**true**に設定されている場合、*トークン階層* 内の変数を検索します (**process-instance** の範囲) 。反対に、**false**に設定されている場合、トークンの範囲内の変数をリトリーブします。トークン自体が指定名の変数を持たない場合、トークン階層を使いその変数を検索します。この要素が省略されている場合は**globalProcessScope** がデフォルトで **false**に設定されます。

- **bpmToEsbVars**

  この要素は任意です。サブ要素のリストで、これらを使い JBPM コンテキスト変数を ESB メッセージの場所にマッピングします。これらのマッピングサブ要素に以下の属性を持たせることができます。

- **bpm** - これは必須属性です。JBPM コンテキストの変数名です。この名前は MVEL 型式にすることもできるため、大きめのオブジェクトから指定のフィールドを抽出することができます。MVEL root を JBPM “ContextInstance”に設定する場合、以下のようなマッピングを利用できます。

```
<mapping bpm="token.name" esb="TokenName" />
<mapping bpm="node.name" esb="NodeName" />
<mapping bpm="node.id" esb="esbNodeId" />
<mapping bpm="node.leavingTransitions[0].name" esb="transName" />
<mapping bpm="processInstance.id" esb="piId" />
<mapping bpm="processInstance.version" esb="piVersion" />
```

  JBPM コンテキスト変数は直接参照することも可能です。

- **esb**

  この属性は任意です。エンタープライズバスメッセージの変数名です。MVEL 型式も可能です (つまり、上記の例の属性値 TokenName は **body.TokenName** と同等です。**BODY_CONTENT** と呼ばれる特別値はボディを直接指定します)。デフォルトでは、変数がESB メッセージのボディに名前付きのパラメーターとして設定されます。**esb** 属性を省略するには、**bpm** 属性の値と置き換えてください。

- **process-scope**

  この属性は任意です。これパラメーターでこの中にはマッピング用に **globalProcessScope** 設定をオーバーライドする際に利用する Boolean 値を含めることが可能です。

重要

変数マッピング設定で作業する場合は、**debug** レベルのロギングを常に有効にしてください。

7.6.2. ESB のアクションハンドラー

ノードが入った場合にアクションを呼び出したい場合は、**EsbActionHandler** をノードに紐付けします。**EsbActionHandler** が実行すると、遷移の合図を待つノードを退出します (他の実行中スレッドからも入れますが、通常は **JBossESB callback** サービスが送信します)。以下に設定方法を示しています。

```
<action name="create_new_process_instance"
  class="org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">

  <property name="command" value="StartProcessInstanceCommand" />
  <property name="process-definition-name" value="processDefinition2"/>
  <property name="actor" value="FrankSinatra"/>

  <property name="esbToBpmVars">
  <!-- esb-name maps to getBody().get("eVar1") -->
    <mapping esb="eVar1" bpm="counter" default="45" />
    <mapping esb="BODY_CONTENT" bpm="theBody" />
  </property>

</action>
```

EsbActionHandler は **EsbNotifier** の設定にも依存します。拡張には以下のサブ要素が含まれています。

表7.5 サブ要素

プロパティ	説明	必須
esbToBpmVars	これは**BpmProcessor** 設定の **esbToBpmVars** プロパティと同じです。サブ要素は、ESB メッセージから抽出し、特定のプロセスインスタンスに対しビジネスプロセスマネージャーに設定する必要のある変数一覧を定義します。これを指定せず、変数が設定されている場合は **globalProcessScope** 値はデフォルトで**true** に設定されます。 このリストには、マッピング要素が含まれており、これらの要素には以下の属性を含めることが可能です。 • **esb** これは必須属性で、MVEL 式を含むことができます。これを使い、値を抽出し別の場所から ESB メッセージに置くことができます。 • **bpm** これは任意の属性で、JBPM で利用する名前を含みます。これが提示されていない場合は **esb** の名前を代わりに利用します。 • default これは任意の属性で、**esb** の MVEL 式により Enterprise Service Bus メッセージ内に設定されている値を発見できない場合に利用するデフォルト値を保持することができます。 • **process-scope** これは Boolean 値で構成される任意のパラメーターです。これを使いこのマッピングの **globalProcessScope** の設定をオーバーライドします。	いいえ
exceptionTransition	これは、サーバー処理中に例外が発生した場合に使用する遷移の名前です。この要素では、現在のノードに複数の出力遷移があり、そのうちの１つが*例外処理*を行う必要があります。	いいえ

希望であればタイムアウト値 (任意) を指定することができます。指定方法は、JBPM ネーティブの*タイマー*をノードに追加してください。この例では、このノードに入って１０秒以内にシグナルを受け取らない場合、**time-out** と呼ばれる遷移がトリガーされるような設定になっています。

```
<timer name='timeout' duedate='10 seconds' transition='time-out'/>
```

7.6.3. JBPM-to-ESB 例外処理

例外は以下の２つの状況で発生します。

1. メッセージを **Enterprise Service Bus**に送信できない場合、**ServiceInvoker** は **MessageDeliveryException** をスローします。これはユーザーがサービス名のスペルを間違えた場合などに発生します。この種類の例外は、**EsbNotifier** と **EsbActionHandler** の両方からスローされます。**ExceptionHandler** (**TB-JBPM-USER**) を JBPM ノードに追加しこのような状況に対応することもできます (詳細は http://docs.jboss.com/jbpm/v3/userguide/processmodelling.html を参照してください)。

2. ２つ目の例外は、リクエストを正常に受け取りはするものの、次の処理中に問題が発生した場合に発生します。呼び出しが**EsbActionHandler** から出された場合のみ、例外は**JBoss Business Process Manager**に報告されます。これは、呼び出しが**EsbNotifier** から行われた場合、JBPM 処理はすでに進められており、問題をプロセスインスタンスに報告してもあまりに意味がないためです。

標準の JBPM 機能で設定できるエラー処理を例示するシナリオをいくつか以下に示しています。

7.6.4. シナリオ１：タイムアウト

EsbActionHandler アクションを利用し、ノードがコールバックを待っている場合、待機時間に制限を設けると便利です。制限時間を設定するにはノードにタイマーを追加します (以下のプロセス定義スニペットで **Service1** が設定されているように行います)。このタイマーは一定期間設定でき、今回の場合は１０秒に設定されています。

```
<node name="Service1">

  <action class=
    "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>MockCategory</esbCategoryName>
      <esbServiceName>MockService</esbServiceName>
  </action>

  <timer name='timeout' duedate='10 seconds'
    transition='time-out-transition'/>
  <transition name="ok" to="Service2"></transition>
  <transition name="time-out-transition" to="ExceptionHandling"/>

</node>
```

Service1 には２つの出力遷移があります。１つ目は**ok** で、２つ目は **time-out-transition** と呼ばれるものです。

通常の処理条件では、コールバックはデフォルトの遷移 (ここでは **ok**、一番に設定されているため) にシグナルを送ります。しかし、サービス処理に１０秒以上かかった場合、タイマーが実行されます。このタイマーの transition 属性は **time-out-transition** にされており、タイムアウトするとこの遷移が取られることになります。

図を参照していただくと、この処理が **ExceptionHandling** ノードで終わっていることがわかります。ここから、代わりの作業を行うことができます。

7.7. シナリオ２：例外遷移

exceptionTransition を定義し、サービス処理中に発生する例外を処理することができます。こうすることで、**faultTo** エンドポイント参照をメッセージに設定し、エンタープライズサービスバスがこのノードにコールバックします。このコールバックにより **exceptionTransition** にシグナルを送ります。

Service2 には２つの出力遷移があります。正常に行われている場合は**ok** 遷移をとり、名前が示すように処理中にサービスが例外を送出した場合に**exception** 遷移を取ります。

```
<node name="Service2">
  <action class=
  "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
     <esbCategoryName>MockCategory</esbCategoryName>
     <esbServiceName>MockService</esbServiceName>
     <exceptionTransition>exception</exceptionTransition>
   </action>
   <transition name="ok" to="Service3"></transition>
   <transition name="exception" to="ExceptionHandling"/>
</node>
```

Service2 定義の前に、アクションの exceptionTransition は **exception** と設定されています。このシナリオでは、この処理自体が **ExceptionHandling** ノードに到達するように設定されています。

7.8. シナリオ３：例外の決定

最後のシナリオを理解するには、**Service3** とその次の **exceptionDecision** ノードの設定を学習してください。**Service3** は通常どおりに処理し、ノードからの遷移は予想どおりに行われます。

しかし、サービス実行中に **errorCode** が設定され、**exceptionDecision** ノードが同じ名前の変数が呼び出されていないか確認します。

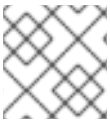
```
<node name="Service3">
  <action class=
  "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
     <esbCategoryName>MockCategory</esbCategoryName>
     <esbServiceName>MockService</esbServiceName>
     <esbToBpmVars>
        <mapping esb="SomeExceptionCode" bpm="errorCode"/>
     </esbToBpmVars>
  </action>
  <transition name="ok" to="exceptionDecision"></transition>
</node>

<decision name="exceptionDecision">
   <transition name="ok" to="end"></transition>
   <transition name="exceptionCondition" to="ExceptionHandling">
      <condition>#{ errorCode!=void }</condition>
   </transition>
</decision>
```

esbToBpmVars マッピング要素は、メッセージのボディから **SomeExceptionCode** と呼ばれる **errorCode** を抽出し、 *JBPM コンテキスト*に設定します (これは **SomeExceptionCode** が設定されているとの前提で行われます)。

次のノードで、名前付きの **exceptionDecision** ですが、通常処理の場合は**ok** の遷移が取られますが、**errorCode** と呼ばれる変数が JBPM コンテキストにない場合は**exceptionCondition** 遷移が取られます。

システムをこの方法で設定するには、JBPM の*決定ノード*機能を使う必要があります。これにより、条件内に複数の遷移をネストすることができます。

```
<condition>#{ errorCode!=void }</condition>
```



注記

条件遷移の詳細については、**JBPM リファレンスガイド**を参照してください。

第8章 サービスオーケストレーション

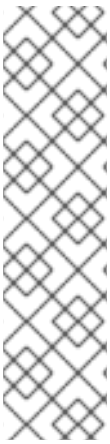
8.1. はじめに

本項を参照し、サービスオーケストレーションと **JBoss Business Process Manager** 機能を統合する方法を学習します。

サービスオーケストレーション

サービスオーケストレーションという用語は、ビジネスプロセスの整理という意味です。従来、Business Process Execution Language (BPEL) を使い SOAP ベースの Web サービスを実行していました。Red Hat は JBPM を使いプロセスのオーケストレーションを行うよう推奨しています。

8.2. ウェブサービスの統制



注記

Red Hat および Enterprise Service Bus チームは、ActiveEndpoints (受賞歴のある **ActiveBPEL** WS-BPEL Engine を構築) と特別なサポート契約を交わしています。

ActiveBPEL を使い効果的に連携することで、Web サービスインターフェースを公開しないサービス群のうえに WS-BPEL-ベースのオーケストレーション層を提供することできます。JBoss Enterprise Service Bus は、Web サービス統合を提供しており、**ActiveBPEL** は *プロセスオーケストレーション*を提供しています。**Flash-** ベースのクイックスタートがオンラインで (http://labs.jboss.com/jbossesb/resources/tutorials/bpel-demos/bpel-demos.html) 多数提供されています。

8.3. オーケストレーションダイアグラム

*オーケストレーションダイアグラム*はフローチャートです。これを使い、ビジネスのサービスオーケストレーションプロセスをデプロイする方法を計画します。**jBPM Integrated Development Environment** を使いこれらのダイアグラムを作成します。

このオーケストレーションダイアグラムの例は、単純な受注プロセスを示しています(この例は **bpm_orchestration4** クイックスタートを元にしています)。

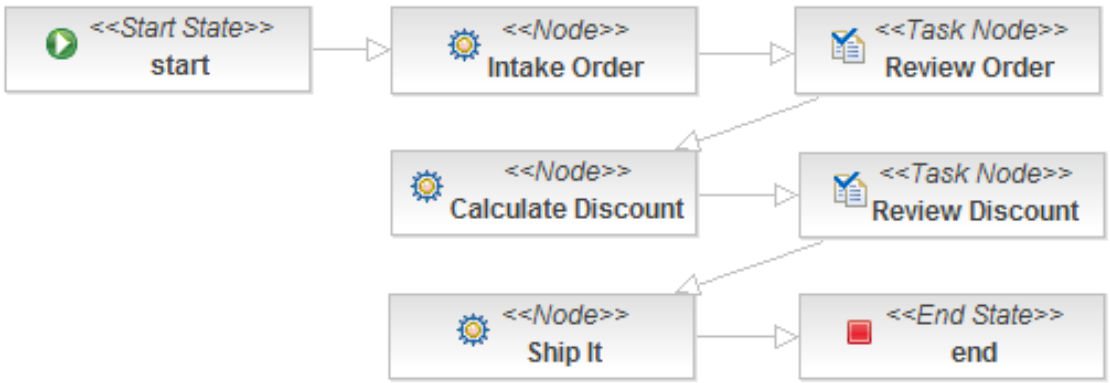


図8.1 bpm_orchestration4 クイックスタートのオーケストレーションダイアグラム

各ノードのクラス名は JBoss ESB サービスで、これらは**Intake Order**、**Calculate Discount**、**Ship It** となっています。通常の**nodes** であるため、**<<Node>>** との名前がついていま

す。これらのノードはそれぞれ **EsbActionHandler** に付いています。つまり、JBoss Business Process Manager ノードは、リクエストをサービスに送り、Enterprise Service Bus がサービスからのレスポンスによりコールバックするまで「待機」のステータスでとどまります。このレスポンスは、**JBoss Business Process Manager** コンテキスト内で利用していくことができます。

例えば、**Intake Order** サービスが応答すると、このレスポンスを使い**Review Order** フォームを生成します。**Review Order** ノードは*タスクノード*となります。このタスクノードは、人とのやりとりを目的として作成されています (この場合、注文プロセスを発生させるには誰かがこの注文をレビューしなければなりません)。

手順8.1 オーケストレーションダイアグラムの作成手順

1. **File > New > Other** を選択します。
2. **Selection** ウィザードから**JBoss jBPM Process Definition**を選択します。
3. プロセス定義を保存します (Red Hat は、整理の観点から、プロセス定義毎に別の辞書を使うよう推奨しています。これはプロセス設計ごとに最終的に複数のファイルが形成されるためです)。
4. まず、 **jBPM Integrated Development Environment** のメニューパレットから**Process Design** ビューにアイテムをドラッグアンドドロップします。追加した XML 要素を確認するために、デザインとソースモードの間で切り替えることも可能です。

   統合に必要な XML フラグメントはここに追加します (最近、*ESB サービス* と呼ばれる新しい種類のノードが追加されました)。
5. 注文プロセスの図を構築する前に、３つのサービスを作成、テストします。通常の ESB サービスですので、**jboss-esb.xml** ファイルにて定義します。

   Red Hat は **bpm_orchestration4** クイックスタート内にある **jboss-esb.xml** ファイルを参照し学習を進めていくよう推奨していますが、サービスオーケストレーション関連で把握しておくべきことはサービス名とカテゴリーです。**jboss-esb.xml** ファイルからの抜粋を以下に示しています。

```
    <services>
  <service category="BPM_orchestration4_Starter_Service"
  name="Starter_Service"
  description="BPM Orchestration Sample 4: Use this service to start
a
process instance">
   <!-- .... -->
  </service>
  <service category="BPM_Orchestration4" name="IntakeService"
  description="IntakeService: transforms, massages, calculates
priority">
   <!-- .... -->
  </service>
  <service category="BPM_Orchestration4" name="DiscountService"
    description="DiscountService">
  </service>
  <service category="BPM_Orchestration4" name="ShippingService"
    description="ShippingService">
   <!-- .... -->
  </service>
</services>
```

EsbActionHandler あるいは **EsbNotifier** のいずれかを使い、これらのサービスを参照します **(JBoss Business Process Manager** がレスポンスが必要な場合は**EsbActionHandler** を、必要でない場合は **EsbNotifier** を選択します**)**。

6. ESB サービスについて学習しましたので、**Start** ステートノードを設計ビューにドラッグします。新規プロセスインスタンスはこのノードで開始します。

7. ノードをドラッグします **(**あるいは利用可能であれば ESB をドラッグします**)**。

8. このノードに **Intake Order** という名前を付けます。

9. メニューから**Transition** を選択し、**Start** と**Intake Order** ノードのうえでクリックすることで、それぞれを接続できます。**(**これらの接続する矢印が表示されます。最初の **Intake Order** 方向を向いています**)**。

10. Intake ノードにサービスとカテゴリー名を追加します。**Source** ビューを選択します。**Intake Order** ノードのソースコードを参照できます。ソースコードは以下のようになっているはずです。

```
<node name="Intake Order">
    <transition name="" to="Review Order"></transition>
</node>
```

11. **EsbHandlerAction** クラス参照の後に、サービスカテゴリーと名前のサブ要素構成設定、**BPM_Orchestration4**、**IntakeService** を追加します。以下に、この方法についてのサンプルコードを示します。

```
<node name="Intake Order">
  <action name="esbAction" class=

"org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
    <esbCategoryName>BPM_Orchestration4</esbCategoryName>
    <esbServiceName>IntakeService</esbServiceName>
    <!-- async call of IntakeService -->
  </action>
  <transition name="" to="Review Order"></transition>
</node>
```

12. 次に、サービス呼び出しと共に**JBoss Business Process Manager** コンテキスト変数を送信します **(**この例では、**entireOrderAsXML** という名前に変数があり、メッセージボディのデフォルトの位置に設定されます**)**。これを行うには以下のコードを追加します。

```
<bpmToEsbVars>
  <mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
```

これにより、**entireOrderAsXML** 変数の **XML** ベースコンテンツがメッセージのボディに入るようになります。これにより、パイプライン内で各アクションを通り流れるため、**IntakeService** がメッセージにアクセスし処理可能になります。最後のアクションに到

達すると replyTo プロパティをチェックしメッセージが **JBpmCallBack** サービスに送信されます。

JBpmCallBack は**JBoss Business Process Manager**にコールバックし、**Intake Order** ノードから次のノードで遷移するよう合図を送ります (今回の場合、**Review Order**です)。

13. メッセージから変数を送信したいとします。コンテキストがオブジェクトのクラスをロードできれば、オブジェクト全体を送信することができます。

    JBoss Business Process Manager へマップバックする機能を保持するには**esbToEsbVars** 要素を追加します。

```
<node name="Intake Order">
<action name="esbAction" class=
 "org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
<esbServiceName>IntakeService</esbServiceName>
<bpmToEsbVars>
<mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
<esbToBpmVars>
<mapping esb="body.entireOrderAsXML" bpm="entireOrderAsXML"/>
<mapping esb="body.orderHeader" bpm="entireOrderAsObject" />
<mapping esb="body.customer" bpm="entireCustomerAsObject" />
<mapping esb="body.order_orderId" bpm="order_orderid" />
<mapping esb="body.order_totalAmount" bpm="order_totalamount" />
<mapping esb="body.order_orderPriority" bpm="order_priority" />
<mapping esb="body.customer_firstName" bpm="customer_firstName" />
<mapping esb="body.customer_lastName" bpm="customer_lastName" />
<mapping esb="body.customer_status" bpm="customer_status" />
</esbToBpmVars>
</action>
<transition name="" to="Review Order"></transition>
</node>
```

    このサービスが返されると、以下の変数は**JBoss Business Process Manager**のコンテキストに保存されます。

    - **entireOrderAsXML**,
    - **entireOrderAsObject** および
    - **entireCustomerAsObject**

    さらに、デモ目的でフラット化された変数も存在します。

    - **order_orderid**
    - **order_totalAmount**
    - **order_priority**
    - **customer_firstName**
    - **customer_lastName** および

- **customer_status**

14. ここで人による注文プロセスの確認を行う必要があります。そのため、**task node**と**Order Review**というタイトルのタスクを追加します。これらのジョブは、actor_id **user**を持つ誰かが実行する必要があります。

    XML フラグメントは以下のようになっているよう確認します。

```
<task-node name="Review Order">
<task name="Order Review">
<assignment actor-id="user"></assignment>
 <controller>
<variable name="customer_firstName"
access="read,write,required"></variable>
<variable name="customer_lastName" access="read,write,required">
<variable name="customer_status" access="read"></variable>
<variable name="order_totalamount" access="read"></variable>
<variable name="order_priority" access="read"></variable>
<variable name="order_orderid" access="read"></variable>
<variable name="order_discount" access="read"></variable>
<variable name="entireOrderAsXML" access="read"></variable>
</controller>
</task>
<transition name="" to="Calculate Discount"></transition>
</task-node>
```

15. これらの変数が **jbpm-console** の形式で表示できるように、XHTML データ形式を作成します。

注記

詳細は **bpm_orchestration4** クイックスタートの **Review_Order.xhtml** ファイルを参照してください。

16. これらの設定を **forms.xml** ファイルに追加し、このデータフォームを task node とリンクします。

```
<forms>
<form task="Order Review" form="Review_Order.xhtml"/>
<form task="Discount Review" form="Review_Order.xhtml"/>
</forms>
```

17. この場合、２つのタスクノードに同じフォームが適用される点に注意してください。以下のサンプルコードで示されているように、**Review Order** フォームでは変数への参照があります(こうすることで、 **JBoss Business Process Manager**のコンテキストに設定される変数を参照します)。

```
<jbpm:datacell>
<f:facet name="header">
<h:outputText value="customer_firstName"/>
</f:facet>
<h:inputText value="#{var['customer_firstName']}" />
</jbpm:datacell>
```

18. このプロセスが **Review Node** に到達すると、jBPM Console にログインし **Tasks** をクリックすることでアイテム一覧を参照することができます。

19. タスクをクリックし詳細を見ていきます。

20. **Save and Close** をクリックして終了します。この時点でプロセスは次のノードに移動します。

21. これは **Calculate Discount** ノードです。こちらも ESB サービスノードで、この設定ファイルは以下のようになります。

```
<node name="Calculate Discount">
<action name="esbAction" class="
org.jboss.soa.esb.services.jbpm.actionhandlers.EsbActionHandler">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
<esbServiceName>DiscountService</esbServiceName>
<bpmToEsbVars>
<mapping bpm="entireCustomerAsObject" esb="customer" />
<mapping bpm="entireOrderAsObject" esb="orderHeader" />
<mapping bpm="entireOrderAsXML" esb="BODY_CONTENT" />
</bpmToEsbVars>
<esbToBpmVars>
<mapping esb="order"
bpm="entireOrderAsObject" />
<mapping esb="body.order_orderDiscount" bpm="order_discount" />
</esbToBpmVars>
</action>
<transition name="" to="Review Discount"></transition>
</node>
```

このサービスは **customer** と **orderHeader** の両オブジェクトだけでなく **entireOrderAsXML** データも受け取ります。次に割引を参集tします。そのレスポンスにより **body.order_orderDiscount** 値を order_-discountと呼ばれる **JBoss Business Process Manager** コンテキスト変数にマッピングします。そうすると、このプロセスは、**Review Discount** ノードに移動するように印付けされます。

22. 8.5 に設定されている割引をレビューします。**Save and Close** をクリックすると、プロセスは ESB サービスである **Ship It** に移動します。

Ship It サービスの完了前の注文プロセスを回避するため、**EsbNotifier** アクションハンドラーを利用し出力遷移に添付します。以下のコードで、方法を説明しています。

```
<node name="ShipIt">
<transition name="ProcessingComplete" to="end">
<action name="ShipItAction" class=
"org.jboss.soa.esb.services.jbpm.actionhandlers.EsbNotifier">
<esbCategoryName>BPM_Orchestration4</esbCategoryName>
<esbServiceName>ShippingService</esbServiceName>
 <bpmToEsbVars>
<mapping bpm="entireCustomerAsObject" esb="customer" />
 <mapping bpm="entireOrderAsObject" esb="orderHeader" />
 <mapping bpm="entireOrderAsXML" esb="entireOrderAsXML" />
 </bpmToEsbVars>
```

```
 </action>
</transition>
</node>
```

ShippingService に通知後、注文プロセスは **end** ステータスに移動し終了されます (**ShippingService** 自体は終了途中の場合もあります)。

bpm_orchestration4 クイックスタートでは、**JBoss Rules** エンジンを使い、この注文が通常の手段あるいは、エクスプレス手段いずれを使って出荷されるか決定します。

8.4. プロセス定義のデプロイ

前のセクションで、プロセス定義のインスタンスが実行されていると仮定されていました。この仮定は、プロセスフローを説明するために立てられました。しかし、 **processdefinition.xml** ファイルが作成されたので、以下のいずれかを使いこのファイルを **JBoss Business Process Manager** にデプロイできます。

- 統合開発環境 (以下の例で利用)
- **ant**
- jBPM コンソール

以下のファイルがデプロイされます。

- **Review_Order.xhtml**
- **forms.xml**
- **gpd.xml**
- **processdefinition.xml**
- **processimage.jpg**

統合デプロイメント環境により **.PAR** アーカイブが作成され、**jBPM** データベースにデプロイされます。

警告

Red Hat は、クラスローディングの問題が発生する可能性があるため、**.PAR** アーカイブで Java コードをデプロイしないようアドバイスをしています。代わりに、クラスのデプロイには **.JAR** あるいは **.ESB** アーカイブを使います。

手順8.2 デプロイメントのインスタンス化

- プロセス定義がデプロイされると新規プロセスインスタンスを作成します (**StartProcessInstanceCommand**コマンドを利用できます。このコマンドを使うことで、事前設定された初期値を使いプロセスインスタンスを作成できます)。

  このサンプルコードをご覧ください。

```
<service category="BPM_orchestration4_Starter_Service"
name="Starter_Service"
description="BPM Orchestration Sample 4: Use this service to start a
 process instance">
<listeners>
</listeners>
<actions>
<action name="setup_key" class=
"org.jboss.soa.esb.actions.scripting.GroovyActionProcessor">
<property name="script"
value="/scripts/setup_key.groovy" />
</action>
<action name="start_a_new_order_process" class=
"org.jboss.soa.esb.services.jbpm.actions.BpmProcessor">
<property name="command"
value="StartProcessInstanceCommand" />
<property name="process-definition-name"
value="bpm4_ESBOrderProcess" />
<property name="key" value="body.businessKey" />
<property name="esbToBpmVars">
 <mapping esb="BODY_CONTENT" bpm="entireOrderAsXML" />
</property>
</action>
</actions>
</service>
```

ここでスクリプトを使い新規プロセスインスタンスが呼び出されます。入ってくる注文 CML ファイルにより、jBPM キーが OrderId の値に設定されます。

同様の XML は結果、**esbToBpmVars** マッピングを使い、**jBPM** コンテキストに入れられます。

bpm_orchestration4 クイックスタートでは、XML は**Seam DVD Store** から来ており、**SampleOrder.xml** は以下のようになります。

```
<Order orderId="2" orderDate="Wed Nov 15 13:45:28 EST 2006"
statusCode="0"
 netAmount="59.97" totalAmount="64.92" tax="4.95">
<Customer userName="user1" firstName="Rex" lastName="Myers"
state="SD"/>
<OrderLines>
<OrderLine position="1" quantity="1">
<Product productId="364" title="Gandhi"
price="29.98"/>
</OrderLine>
<OrderLine position="2" quantity="1">
<Product productId="299" title="Lost Horizon" price="29.99"/>
</OrderLine>
</OrderLines>
</Order>
```

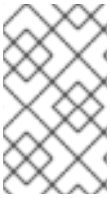
注記

Enterprise Service Bus と JBoss Business Process Manager のデプロイメントは両方、「ホット」デプロイメントと呼ばれます。

jBPM は、プロセスデプロイメントをバージョン化する特別機能です。新しく作成されたプロセスインスタンスが最新版となり、既存のものは起動したプロセスデプロイメントを使い終了されます。

8.5. まとめ

本項のサンプルを学習することで、**JBoss Business Process Manager** を使いサービスオーケストレーションや「ヒューマンタスク管理」を行うことができます。



注記

ご希望の jBPM 機能を自由にお使いいただけます。たとえば、**fork** と **join** 機能の使用方法を学習したい場合は、**bpm_orchestration2**という名前のクイックスタートを確認してください。

付録A GNU GENERAL PUBLIC LICENSE 2.0

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but

```
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange;
or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.
```

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that

system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED

TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
    <one line to give the program's name and a brief idea of what it
does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
along
    with this program; if not, write to the Free Software Foundation,
Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show
w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

```
You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program
into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with
the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

付録B 改訂履歴

改訂 5.2.0-0.2.400	**2013-10-31**	**Rüdiger Landmann**
Rebuild with publican 4.0.0		
改訂 5.2.0-0.2	**Thu Sep 13 2012**	**Credit Translator's**
日本語へ翻訳		
改訂 5.2.0-0.1	**Thu Sep 13 2012**	**Credit Translator's**
翻訳終了と XML ソース 5.2.0-0 を同期		
改訂 5.2.0-0	**Tue June 1 2011**	**David Le Sage**
SOA-P 5.2 向けに更新		
改訂 5.1.0-0	**Mon Mar 7 2011**	**David Le Sage**
SOA-P 5.1 向けに更新		
改訂 5.0.2-0	**Wed May 26 2010**	**David Le Sage**
SOA-P 5.0.2 向けに更新		
改訂 5.0.1-0	**Tue Apr 20 2010**	**David Le Sage**
SOA-P 5.0.1 向けに更新		
改訂 5.0.0-0	**Fri Jan 22 2010**	**David Le Sage**
初版作成		