

Overview

Business processes are workflows that describe how various operations and transactions are conducted within an e-commerce environment. These processes are crucial for the efficient operation of an e-commerce platform.

On Cluster aware processing, the Business Process Feature is built. Accordingly, the business process can be distributed throughout the nodes of a Hybris cluster and operate as needed. This suggests that the process is carried out in an asynchronous fashion. The user who wants to send an email or place an order can therefore see every business transaction in clear view.

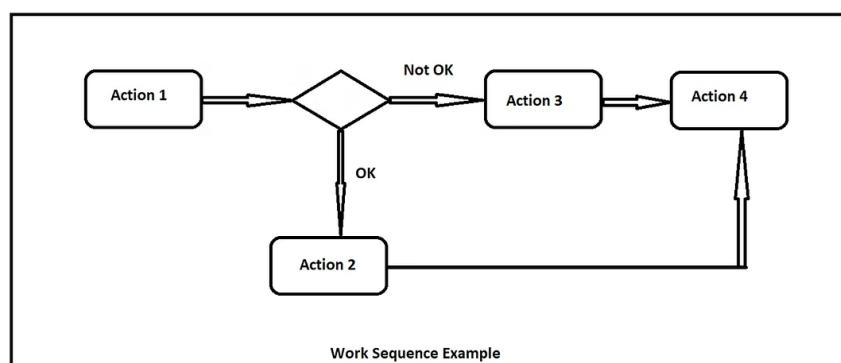
The process flow is technically described in an XML file, and each stage requires the implementation of a bean so that business logic may be executed inside каждого состояния и создать выход, который направляет процесс в определенное новое состояние. Необходимо знать, что Переход — это то, что перемещает процесс из одного состояния в другое.

Тот факт, что состояние постоянно сохраняется, является существенным преимуществом функции бизнес-процесса. В результате, в случае сбоя системы, processengine запомнит свое предыдущее местоположение и возобновит работу, как только контекст processengine будет восстановлен. Благодаря этому, есть существенная гарантия того, что, за исключением статистически небольшого числа случаев, процесс будет работать безопасно и не будет повторять никаких шагов.

Двигатель бизнес-процессов

Business Process Engine определяет бизнес-процесс через XML и запускает этот процесс асинхронным способом. XML-файл содержит каждый шаг и условия в упорядоченном виде. Мы видим, что эти процессы запускаются в порядке определений.

Мы можем определить события ожидания, решения потока, условия действия и уведомить группы пользователей о действиях. Эти файлы обычно определяются как часть расширения `acceleratorfulfilmentprocess`.



1) Определения последовательности рабочих процессов в виде XML:

```
<?xml version="1.0" encoding="utf-8"?>
<process xmlns="http://www.hybris.de/xsd/processdefinition" name="Example" start="Action1">
  <action id="Action1" bean="Action1">
    <transition name="OK" to="Действие2"/>
    <transition name="NOK" to="Действие3"/>
  </action>
</process>
```

```

</действие>
<action id="Действие2" bean="Действие2">
    <transition name="OK" to="Действие4"/>
</действие>
<action id="Действие3" bean="Действие3">
    <transition name="OK" to="Действие4"/>
</действие>
<action id="Действие4" bean="Действие4">
    <transition name="OK" to="successful"/>
</действие>
<end id="success" state="SUCCEEDED">Все было хорошо</end>
</process>

```

Определение процесса

Определение процесса определяет набор узлов, которые связаны друг с другом через идентификаторы. Мы можем создать процесс и выполнить его с помощью следующего кода.

- `businessProcessService.startProcess(id, processName);`
- `businessProcessService.createProcess(id, processName);`

1) Определение процесса: нам необходимо связать XML-файл процесса с файлом ProcessDefinitionResource

```

<bean id="checkorderProcessDefinitionResource" class="de.hybris.platform.processengine.definition.ProcessDefinitionResource">
    <имя_свойства>resource</имя_свойства>value="classpath:/processdemo/checkorder.xml"/>
</bean>

```

2) Действия: Мы можем определить часть действия процесса следующим образом.

```

<bean id="checkOrder" class="de.hybris.platform.fulfillment.actions.CheckOrder" parent="abstractAction">
    <имя_свойства>checkOrderService</имя_свойства>ref="checkOrderService"/>
</боб>

```

3) Корневой тег и процесс: определение процесса имеет корневой тег, который содержит имя, компонент запуска процесса и класс процесса .

```

<?xml version="1.0" encoding="utf-8"?> <process xmlns="http://www.hybris.de/xsd/processdefinition" name="consignmentFulfillmentCustomProcess" start="waitForTransaction" onError="onError" processClass="de.hybris.platform.fulfillment.model.ConsignmentProcessModel">
    ...
</процесс>

```

4) Типы узлов: Процесс имеет набор узлов, которые представляют шаг в данном процессе. Каждый узел должен быть определен в рабочем процессе в порядке выполнения процесса. Процесс имеет различные типы узлов, как определено ниже.

- **Узел действия:** Узлы действия являются важными узлами процесса, они содержат бизнес-логику и возвращаемый тип, на основе которого выполняется следующий шаг процесса.

```

<action id="customProcessCompleted" bean="custom ProcessCompleted">
    <transition name="OK" to="sendOrderCompletedNotification"/>
    <transition name="NOK" to="waitForWarehouseSubprocessEnd"/>
</действие>

```

- **Узел ожидания:** Этот узел используется для ожидания завершения внешнего процесса. Этот узел будет ждать завершения внешнего процесса и результатов внешнего процесса.

```

<wait id="waitForWarehouseSubprocessEnd" then="isProcessCompleted" prependProcessCode="false">
<event>${process.code}_ConsignmentSubprocessEnd</event>

```

</подождать>

- Уведомление об окончании: этот узел используется для уведомления группы пользователей или пользователя о текущем состоянии процесса.

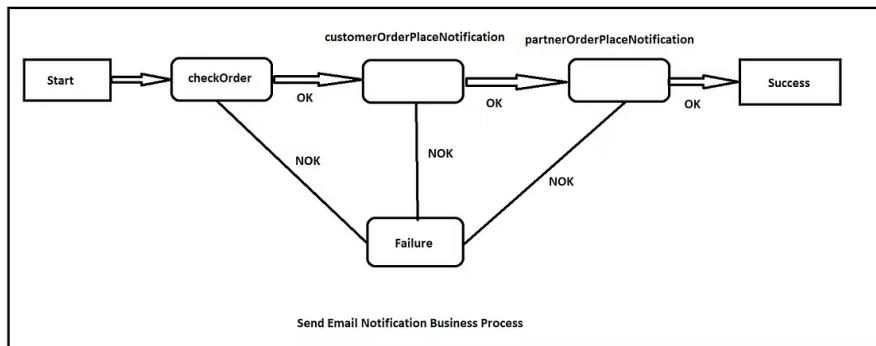
```
<notify id="notifyadmingroup"then="split">
    <userGroup name="admingroup"message="Выполнить действие"/>
    <userGroup name="othergroup"message="другое сообщение"/>
</уведомить>
```

- Конечный узел: этот узел является конечным узлом процесса и хранит конечный результат процесса.

```
<end id="error" state="ERROR">Все пошло не так.</end>
<end id="success" state="SUCCEEDED">Все было хорошо.</end>
```

Создание примера бизнес-процесса

У нас есть требование, что после размещения заказа мы выполним бизнес-процесс, который отправит уведомление по электронной почте Клиенту, а затем Партнерам B2BUnit. Мы не собираемся выполнять какие-либо платежи, мошенничество или процессы, связанные с отправкой. После отправки уведомления по электронной почте заказ будет извлечен сторонней системой из Hybris для выполнения.



Шаг 1: Создайте файл placeOrder-notification-process.xml .

```
<process xmlns="http://www.hybris.de/xsd/processdefinition" start=" checkOrder" name=" PlaceOrderNotification"
"processClass="de.hybris.platform.orderprocessing.model.OrderProcessModel">
<action id=" checkOrder" bean=" checkOrderAction">
    <transition name="OK" to=" customerOrderPlaceNotification" />
    <transition name="NOK" to="error"/>
</action>
<action id=" customerOrderPlaceNotification" bean=" customerOrderPlaceNotificationAction">
    <transition name="OK" to=" partnerOrderPlaceNotification" />
    <transition name="NOK" to="error"/>
</action>
<action id=" partnerOrderPlaceNotification" bean=" partnerOrderPlaceNotificationAction">
    <transition name="OK" to="success"/>
    <transition name="NOK" to="error"/>
</action>
<end id=" error" state=" ERROR">Все пошло не так.</end>
<end id=" failed" state=" FAILED">Тестовый бизнес-процесс не пройден.</end>
<end id=" success" state=" SUCCEEDED">Тестирование бизнес-процесса прошло успешно.</end>
</process>
```

шаг 2: зарегистрируйте компоненты процесса и действия в файле trainingfulfilmentprocess-spring.xml.

- Регистрация процесса:

```
<bean id="placeOrderNotificationProcess" class="de.hybris.platform.processengine.definition.ProcessDefinitionResource">
<имя_свойства="resource" значение="classpath:/trainingfulfilmentprocess/process/placeOrder-notification-process.xml" />
</боб>
```

- Регистрация компонентов действий:

```
<bean id="checkOrderAction" class="com.training.fulfilmentprocess.actions.CheckOrderAction" parent="abstractAction"/>
<bean id="customerOrderPlaceNotificationAction" class="com.training.fulfilmentprocess.actions.CustomerOrderPlaceNotificationAction"
parent="abstractAction"/>
<bean id="partnerOrderPlaceNotificationAction" class="com.training.fulfilmentprocess.actions.PartnerOrderPlaceNotificationAction"
parent="abstractAction"/>
```

Шаг 3: Реализуйте все классы действий, определенные на предыдущем шаге. Я привел примерный план ниже.

```
1.  пакет com.training.fulfilmentprocess.actions;
2.  import de.hybris.platform.orderprocessing.model.OrderProcessModel;
3.  импортировать de.hybris.platform.processengine.action.AbstractSimpleDecisionAction;
4.  импорт de.hybris.platform.task.RetryLaterException; импорт org.apache.log4j.Logger;
5.  открытый класс CheckOrderAction расширяет AbstractSimpleDecisionAction<OrderProcessModel>
6.  {
7.      частный статический финальный Logger LOG = Logger.getLogger(FirstAction.class);
8.      @Override public Transition executeAction(final OrderProcessModel var) выдает RetryLaterException, Exception
9.      {
10.          // XXX Автоматически сгенерированная заглушка метода
11.          LOG.info("CheckOrderAction!!!");
12.          возврат ПереходаOK;
13.      }
14.  }
```

Шаг 4: Запустите бизнес-процесс, следуя сервису согласно требованию. В приведенном выше сценарии мы можем разместить этот код в потоке заказов.

- Создайте экземпляра бизнес-процесса.

```
окончательная OrderProcessModel placeOrderNotificationProcess = ( OrderProcessModel ) getBusinessProcessService().createProcess("placeOrderNotificationProcess " + System.currentTimeMillis(), "placeOrderNotificationProcess");
```

- Запустите бизнес-процесс.

```
getBusinessProcessService().startProcess( placeOrderNotificationProcess );
```

Бизнес-процесс Действие Синхронный Путь

Мы можем запускать узлы действий в определении бизнес-процесса асинхронно или синхронно. В асинхронном режиме узел действия рассматривается как отдельная и отдельная задача при запуске обработчиком задач. В синхронном режиме узел действия запускается в той же задаче, что и предыдущий узел.

По умолчанию действия выполняются асинхронно, мы можем явно сделать их синхронными, добавив следующее свойство в файл local.property:

```
processengine.process.canjoinpreviousnode.default=false
```

Об авторе

Пиюш Сингх — опытный энтузиаст технологий и преподаватель, страстно желающий сделать сложные концепции доступными для всех. Имея более чем **10**-летний опыт работы в технологической отрасли в качестве консультанта, Пиюш специализируется на технологиях **Java** и **SAP Hybris** и обладает даром разбивать сложные темы на простые для понимания руководства.

Связаться с автором

[LinkedIn](#)

[Электронная почта](#)