**Modeling sub-processes**

**子流程建模**

Within a BPD, we can create a sub-process. A sub-process is simply a grouping together of a set of activities into an aggregate area. For example, consider the following process diagram:

在一个流程定义，我们能够创建一个子流程。一个子流程是简单地一分组聚集的一系列在集合领域的活动。举个例子， 参考下面的流程图表：



Here we see us take an order, package the order (getting more stock if we are out) and finally shipping the order. What we need to pay attention to is the center part of our diagram. The details of ordering stock (if needed) is perhaps not necessary to understanding the overall flow of our process. Instead, we might want the diagram to look as follows:

这里我们看见我们订了个订单， 打包订单（如果我们是空的，获得更多的进货）并且最终运送订单。我们需要注意的是我们图表的中心部分。订购进货的详情（如果必须）可能是不必要去理解整个我们流程。相反，我们可能想要图表看起来是下面这样：



Notice the marker on the Build Order step. This indicates that this is a sub-process. If we drill down into this step, we find:

注意在Build Order这一步的标记。这个表明这是一个子流程。如果我们进入这里面，我们发现：



What we have done is nested the steps for the Build Order activity as a grouped/hidden sequence of steps. This sub-process shares the same variables as the parent. In addition, the names of the steps defined in the sub-process must be distinct from those of the parent. It is as though the

steps were placed "in-line" in the parent process and have simply been hidden for readability. An important aspect to note is that there is **no** re-usability of these sub-process steps. Their hiding in a sub-process is for readability only.

我们已经做了的是对Build Order活动做嵌入的步骤，作为一聚合/隐藏顺序的步骤。 这个字流程和父流程共享相同的变量。另外，定义在子流程的名字必须是与父流程那些不同。

它好像是被‘内嵌的’放置在父流程并且简单地对可读性隐藏。一个重要方面要注意的是没有那些子流程步骤的可以重用性。它们隐藏在一个子流程是仅仅是为了可读性。

Experimentation has shown that if a Terminate Event is met within a sub-process then the sub process is terminated and control is returned to the step following the sub-process definition.

实验已经显示如果一个中止事件相遇在一个子里面，然后子流程被中止并且控制被返回到紧接着的子流程定义的步骤

See also:

• Terminate Event

参考：

。中止事件

**Modeling Linked Processes**

**链式流程建模**

An alternative style of creating sub-processes is the notion of the Linked Process. In this story, a separate BPD is created that contains re-usable BPD activities. Input and Output variables are defined which describe the expected inputs and outputs from the linked process. In the calling process, a Linked Process activity is defined and a reference added to the target BPD that should be invoked when the parent reaches it.

一个创建子流程替代的类型是链式流程。在这个故事， 一个单独的流程定义被创建，它包括可重用的流程定义活动。输入和输出变量被定义，它们描述了从链式流程里期待的输入和输出。在一个正在请求的流程，一个链式流程活动被定义，

Again we can see the marker that indicates that it contains additional steps. Notice the heavy border around the activity which marks it as distinct from a sub-process.

我们可以看到标记,表明它包含额外的步骤。注意周围的加重的边框活动标记为独特的子流程。

A step in the process can dynamically choose which linked-process to invoke without explicitly having to define the name of the BPD to be called. To achieve this, create a variable of type String and populate its value with the name of the linked-process to invoke. In the Advanced section of the Implementation area, select that variable as the source of the name of the process. At run-time, the variable will be consulted and a dynamic call to the process with that name will be made.

过程的一个步骤可以动态地选择要调用哪个链式流程无需显式定义的被调用BPD名称。为了达到这个目标,创建一个字符串类型的变量和填充它的价值为链式流程调用的名称。在进一步实现区域的部分,选择该变量作为流程的名称的来源。在运行时,该变量将被顾及并且动态调用过程的名 称。

If parameters are to be passed to a dynamically called process then each process that may be potentially selected to be called must have the same set of parameters. Think of this as the dynamically invoked process having a template.

如果参数被传递给一个动态调用过程每一个过程,可能是潜在的选择被称为必须有相同的一组参数。认为这是动态调用过程模板

The BPD that is named as the process to be started **must** have a Start Event contained within it and this will be the starting point for the new sub-process.

命名为启动的过程的BPD必须有一个事件包含在它开始,这将是新的子流程的起点。

When a parent BPD invokes a child BPD any variables passed in as parameters are passed by reference. What this means is that if a child process changes the values of these passed in variables then the changes will also occur in the parent process. Care must be taken here to watch out for unexpected side effects.

当调用一个父BPD调用子BPD任何变量作为参数通过引用传递。这意味着,如果一个子过程变化这些传入的值变量变化将会发生在父过程。这里必须小心提防意想不到的副作用。 The BPD Process ID of the parent is the same Process ID used for the child.

父流程的流程ID和子流程的ID一样

**Modeling Event Sub-processes**

**事件子流程建模**

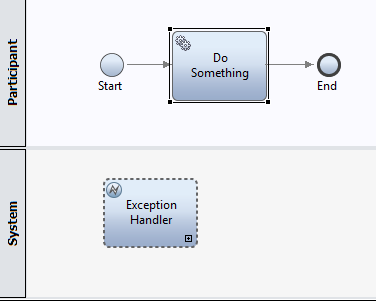
For certain types of events, we can create a "sub-process" which will be invoked if such an event is detected but is not otherwise handled elsewhere. For example, if we have a set of steps that we wish to be executed whenever an exception is thrown we don't want to have to wire this code to every activity in our process that may throw an exception. Instead, we create an Exception Event Sub-process that will be invoked when **any** uncaught exception is thrown.

对于某些类型的事件,我们可以将调用创建一个“子流程”,如果检测到此类事件,但在其他地方无法处理。例如,如果我们有一组要执行的步骤,我们希望每当抛出异常我们不想要这个代码连接到每一个可能抛出异常的活动在我们的流程。相反,我们创建一个异常事件子流程时,它将被调用当有任何未捕获异常。



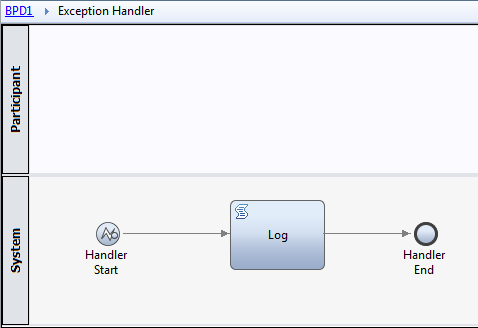
For example, looking at the following BPD fragment, we see an activity called "Do Something" that presumably does something. Under normal circumstances, "Do Something" will end and that will indicate the end of the process as a whole. But, what if "Do Something" throws an exception? That is where the event sub-process that we called "Exception Handler" comes into play. It contains a set of activities that will be executed whenever an exception event is thrown.

例如,看着下面的BPD片段,我们看到一个活动叫“Do Something”,大概做了一些。在正常情况下,“Do Something”将结束,这将显示整个过程的结束。但是,如果“Do Something”抛出一个异常呢?这就是我们称为“异常处理程序”的事件子流程。它包含了一组活动,它将执行事件当抛出一个异常。



If we double click to expand the Exception Handler, we will see that it itself contains steps:

如果我们双击扩开异常处理程序,我们将会看到,它本身包含步骤:



In this case, it is simply a Script fragment that logs to the console. So, if the parent activity called "Do Something" throws an exception, control will be given to the steps contained within the"Exception Handler" which will log data.

在本例中,它是一个简单的脚本片段记录到控制台。所以,如果父活动称为“Do Something”抛出一个异常,控制会将给予步骤包含在“异常处理程序”,将记录日志数据。

Note that although Process Designer allows us to create multiple Sub-process handlers for the same type of event, it is an error to do so. It isn't clear which of the two will be executed.

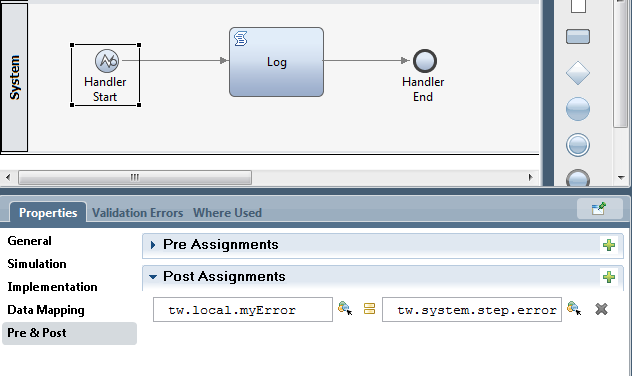
注意,尽管过程设计工具允许我们创建多个子流程为相同类型的事件处理程序,它是一个错误。目前还不清楚这两个那个会被执行。

Because the Sub-process event handler doesn't have any inputs or outputs, there is no follow-on work from this step.

To gain access to the exception details in an Exception Sub-Process handler, a variable of type XMLElement must be created and assigned from the tw.system.step.error variable in the Post Assignment of the Start node in the diagram:

因为子流程事件处理程序没有任何输入或输出,从这一步没有后续的工作。

访问异常细节子流程异常处理程序,一个XMLElement的类型变量必须创建并分配从tw.system.step.error这个变量在变量赋值后的开始节点图:



The Event Sub-process provides handling for event types other than exceptions. Message and Timer events can also be modeled in a similar fashion.

For Event Sub-processes which have starts defined for Message, we have the opportunity to supply a correlation id value to ensure that the correct instance of the process is woken by a corresponding event.

Take extra special care when defining an Event Sub-process which is triggered by the arrival of a message. The parameters called "Interrupt Parent Process?" and "Repeatable" come into play. If "Interrupt Parent Process" is checked, the the arrival of the message causes the container of the Sub-process to be terminated. If the "Repeatable" option is not checked, then messages after the first one will be ignored.

事件子流程提供了处理例外情况以外的事件类型。消息和计时器事件也可以以类似的方式建模。

事件子流程已开始为消息定义,我们有机会提供相关id的值,以确保正确的实例过程被对应的事件唤醒。

特别注意在定义一个事件子流程触发消息的到来的参数称为“打断父过程?”和“重复”。如果选中“中断父过程”,消息的到来使容器的子流程终止。如果“重复”选项没有选中,然后消息后第一个将被忽略。

See also

• Message Start Event

• DeveloperWorks - Designing event-driven business processes in IBM Business Process Manager - 2012-06-27

另请参阅

•消息开始事件

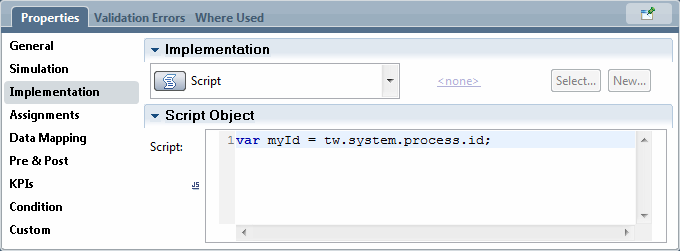
• DeveloperWorks - Designing event-driven business processes in IBM Business Process Manager - 2012-06-27

**Script Activities**

**脚本活动**

An activity can have an implementation type of Script.

一个活动可以以脚本类型的方式实现。



Script allows the programmer to include JavaScript code in-line within the BPD process. The JavaScript can utilize the IBPM supplied JavaScript classes. Although this option is exposed to be used it is unlikely to be a good long term strategy. Code entered here is not re-usable by other activities or services. A better idea would be to build a General Service which includes a Server Script element and invoke the General Service.

脚本允许程序员包括JavaScript代码内联在BPD的过程。JavaScript可以利用IBPM JavaScript类。虽然这个选项是暴露在使用它不太可能是一个很好的长期的策略。代码输入在这里不是被其他活动或服务可重用的。更好的想法是建立一个通用服务包括一个服务器脚本元素和调用通用服务。

**External Implementation**

**外部实现**

When an activity in a BPD is reached, it is associated with and implemented by one of the implementation Service types associated with the IBPM product. These include the common General Service, Integration Service and Human Service types. IBPM provides another type of implementation that is called an "External Implementation". This can be subtle to understand so we will take it slowly. Also note that in previous releases, the External Implementation used to be called an External Activity.

当一个活动在一个BPD,它是被相关联的和被一个已经实现IBPM产品相关联的服务类型。这些包括常见的通用服务、集成服务和人服务类型。IBPM提供了另一种类型的实现,被称为一个“外部实现”。这可以微妙的理解,所以我们将把它缓慢。还需要注意的是,在以前的版本中,外部实现使用被称为外部活动。

The overall goal of the concept of the External Implementation is that some application or code **outside** of the IBPM environment is going to perform some work on behalf of the overall execution of the process. This is not an uncommon situation and IBPM provides a variety of ways in which external applications can be **called** to perform work and return their results. This includes Web Services, REST and other Integration mechanisms. The External Implementation concept though is something different. It is much closer to the concept of a Human Service than it is to an Integration Service.

When an External Implementation is reached in the BPD, a new IBPM Task is created and the BPD process suspends itself until the task completes. Unlike a Human Service task, there are no sets of Coaches or other UI components provided by IBPM associated with this Task. However, the Task still exists and can be queried by the REST API or Web API. These APIs can be used by an arbitrary external application to:

外部的概念实现的总体目标是,一些应用程序或代码IBPM之外的环境将执行一些工作代表整体的执行过程。这是很普遍的情况,IBPM提供了各种方式可以调用外部应用程序执行工作并返回结果。这包括Web Services, REST和其他集成机制。外部实现的概念虽然是不同的东西。这是更接近人服务和集成服务相比。

当外部实现在BPD碰到,创建一个新的IBPM任务和BPD过程中止,直到任务完成。与人服务的任务不同,没有Coaches或者其他UI组件由IBPM提供与此相关的任务。然而,任务仍然存在,可以通过REST API查询或Web API。这些api可以被任意使用外部应用程序:

• Query for the existence of External Implementation tasks

• Obtain the parameters passed as input to an External Implementation

• Set the parameters to be returned from an External Implementation

• Utilize custom properties set on the External Implementation implementation

• Complete the External Implementation associated task

•查询存在的外部实现的任务

•获得的参数作为输入传递给外部实现

•设置参数返回从外部实现

•使用自定义属性设置在外部实现的实现

•完成相关的外部实现任务

If we think about this for a while, we see that a BPD activity calling an external application through an Integration Service is an explicit invocation of that application while an External Implementation is much more focused on the External Implementation associated application polling and working

with the data.

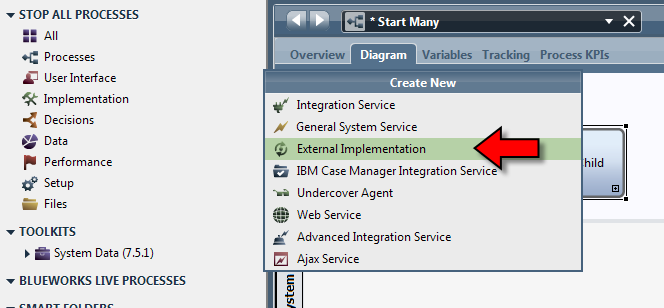
To use external implementations, this capability must first be enabled in IBPM PD.

An External Implementation can be created from the Implementation Category:

如果我们考虑这个一会,我们看到,一个BPD活动调用外部应用程序通过一个集成的服务是一个显式的调用的应用程序，然而外部实现更专注于应用程序相关联的外部实现工作与数据。

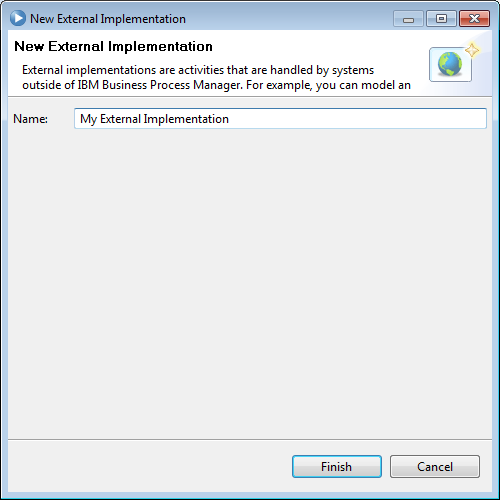
使用外部实现,此功能必须先启用IBPM PD。

可以创建一个外部实现的实现类:



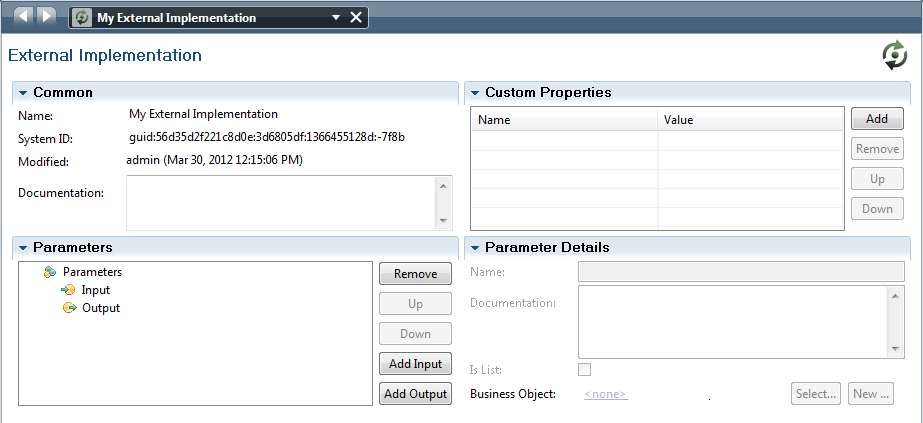
A dialog is presented that allows us to enter the name of the activity to be created:

提出了一个对话框,允许我们进入创建活动的名称:



Once done, the properties of the External Implementation can be entered:

一旦完成,可以输入外部实现的属性:



To use an External Implementation in a BPD, create an activity and change its implementation type to be User Task. Next, the identity of the external implementation previously defined can be used.

使用外部实现在一个BPD里,创建一个活动并改变其实现类型对于用户的任务。其次,外部的身份实现可以使用之前定义。

**Completing an External Implementation – REST API**

**完成一个外部实现-REST API**

When we obtain the data for a TaskInstance object of a task, we can get a property from that object called the ExternalActivityID. If that value is null, then the task does **not** represent an external activity. However, if it is not null, then its value represents an External Activity identifier. A REST request called

当我们获得的数据TaskInstance对象的任务,我们可以得到一个属性对象称为ExternalActivityID。如果该值为null,那么这个任务并不代表外部活动。然而,如果不为空,那么它的值代表一个外部活动标识符。一个REST请求叫

GET /rest/bpm/wle/v1/externalActivity/{externalActivityID}/model[?

parts={string}]

can be used to return the model of the external activity. This model describes the possible input and output data types for this activity. Here is an example of what is returned:

可用于返回外部活动的模型。这个模型描述了可能的输入和输出数据类型为这个活动。下面是一个示例返回的是什么:

{

"status":"200",

"data":

{

"name":"EA1",

"customProperties":{},

"inputs":

{

"in1":

{

"isList":false,

Page 211

"type":"BO1"

}

},

"outputs":

{

"out1":

{

"isList":false,

"type":"BO1"

}

},

"validations":

{

"BO1":

{

"properties":

{

"f1":

{

"isList":false,

"type":"String"

},

"f2":

{

"isList":false,

"type":"String"

},

"f3":

{

"isList":false,

"type":"String"

}

},

"type":"object"

}

},

"ID":"60.0b902a52-8df2-44ed-85e2-04bbd980b01d"

}

}

Let us look at the inputs and outputs structures to begin with. These correspond to a list of variables where each variable has a "type" attribute. The type attribute matches an entry in the "validations" section.

When it comes time to obtain the input data for the External Activity, that data can be found in the Task Instance details object. To return data as output to an External Activity, use the REST Task Finish method (see: Finishing a Task).

When working with an External Activity, it is a good idea to build a notepad document that looks as follows:

让我们看看输入和输出结构。这些对应的列表变量,每个变量都有一个“type”属性。type属性匹配“验证”一节中的一个条目。

当获得的外部活动的输入数据,这些数据可以在任务实例对象的细节。返回数据作为输出到外部活动,使用REST任务完成方法(见:完成一个任务)。

在处理外部活动,它是一个好主意来构建一个记事本文档看起来如下:

EA1

Inputs:

order

customerName String

orderType String

amount Decimal

Outputs:

Page 212

approval

approved Boolean

comments String

For a task input:

{

data:

{

data:

{

variables:

{

order:

{

customerName:

orderType:

amount:

}

}

}

}

}

For task completion:

{

approval:

{

approved:

comments:

}

This document starts with the name of the External Activity, notes its inputs and outputs and then shows the JSON for a GET request of a task. Finally it shows the response JSON to be passed when the task is completed. Building this notepad document and keeping it handy while designing and coding can save much time and effort.

本文从外部活动的名称开始,指出它的输入和输出,然后显示了JSON的GET请求的任务。最后它显示了JSON响应通过在任务完成。构建这个记事本文档和保持它，当设计和编码可以节省很多时间和精力。