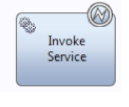
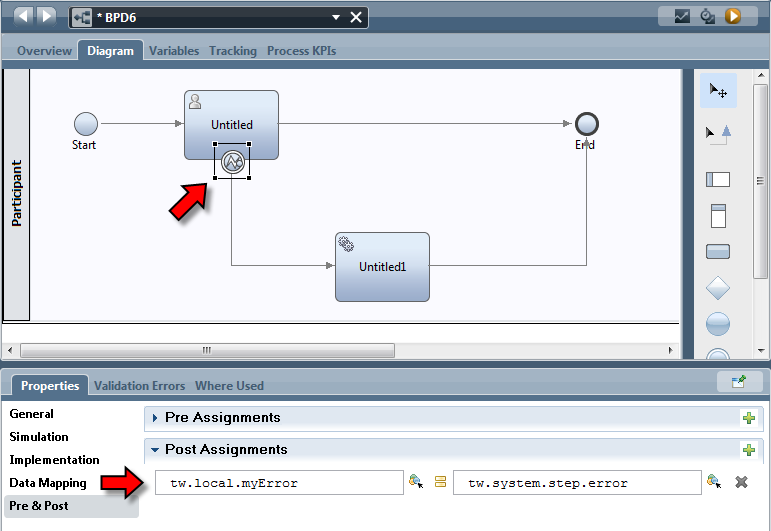
198



The exception occurrence details are available within the variable called tw.system.step.error. Note that this variable only has a value within the activity that populates it. This means that downstream from the Exception handler, it will have no value. The details of the exception should be copied out of the Intermediate Exception Event to a local variable of type XMLElement in a Post-Assignment expression if they are to be subsequently used.

发生异常中可用细节的变量称为tw.system.step.error。注意,这个变量只有在发生异常的活动里才会有一个值。这意味着下游异常处理程序,它就没有值。如果他们将随后使用，异常的细节应该复制到中间异常事件的局部变量类型为XMLElement 在*Post-Assignment*表达式,



The exception details can then be retrieved from the XML element variable. In general, the format of the XML returned is as follows:

异常的细节能够从XML element的变量里获取，通常，返回的XML 格式如下：

<error type="..." description="...">

<cause type="..." description="..." />

<localizedMessage type="..." description="...">text</localizedMessage>

<message type="..." description="...">text</message>

<messageKey type="..." description="..." />

<stackTrace type="..." description="...">

<element type="..." description="...">text</element>

...

</stackTrace>

<toString>text</toString>

<stackTrace>text</stackTrace>

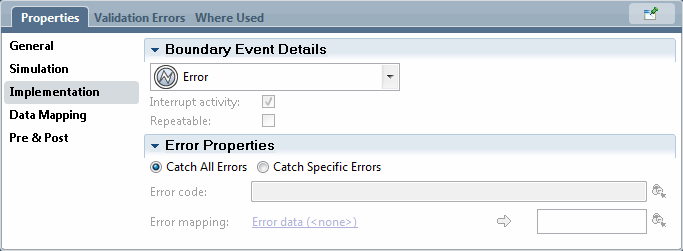
</error>

The Implementation section of this activity has additional properties.

这一活动的实现部分有额外的属性。

198 end

199



See also

参考：

DeveloperWorks - WebSphere Lombardi exception handling and logging - 2011-05-25

# Error End Event

错误结束事件

If a process detects that an error condition has occurred, it can terminate the remaining and current steps of the process and throw an Exception. This exception can be caught by higher level exception handlers or passed on to the user.

To add this item, drag an End Event activity from the palette and change the implementation type. The resulting icon in the canvas looks as follows:

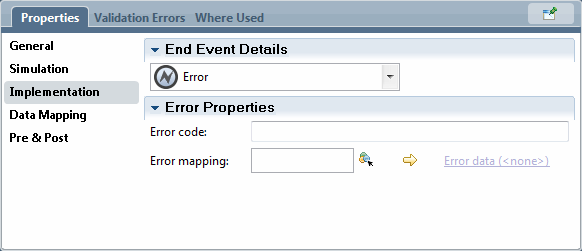
如果一个流程检测到一个错误条件发生了,它可以终止剩余和当前流程的步骤并且抛出异常。这个异常可以被更高级别的异常处理程序或传递给用户。

添加这个一项,从面板中拖动结束事件活动,改变实现类型。由此产生的图标在画布看起来如下:



It has properties that look like:

它的属性看起来像：



Here an error code and error data can be supplied that are passed onwards with the exception details.

这里可以在异常的细节里提供一个错误代码和错误数据传递在。

# Gateways, Conditionals and Joins

网关、条件和连接

Gateways are decision points in the BPD. When reached, an expression is evaluated and based on the outcome of that expression, control flows in different paths. In IBPM, the JavaScript language is used to evaluate the decision's outcome.

网关是BPD的决策点。到达时,一个表达式求值并且基于表达式的结果,控制流在不同的路径。IBPM的JavaScript语言是用来评估决策的结果。

The simplest gateway is called the *Exclusive Gateway*. The outcome of the expression is a boolean (true or false) outcome. The output of the decision is always that a single path is taken.

最简单的网关被称为独占网关。表达式的输出结果是一个Boolean类型的。决策的输出总是在一个路径里获得。

199 end

200

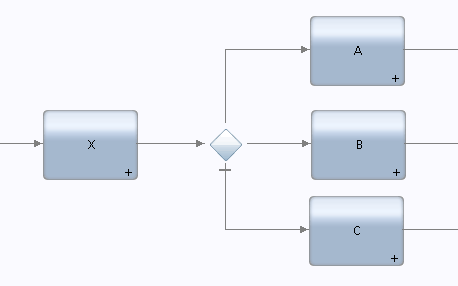
On the diagram, it looks as follows

在图中，它看起来像下面那样



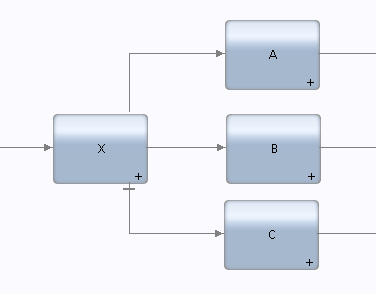
In an actual BPD usage situation, it may look like:

在实际的BPD的使用情况,它可能看起来像:



The inclusion of the decision gateway in this fashion is called an explicit decision gateway because the decision icon is explicitly included in the diagram. An alternative construct called the implicit decision gateway looks as follows:

上图中这种方式的决策网关被称为一个显式的的决策网关,因为决策网关的图标显式地包含在图。另一种构造称为隐式决策网关看起来如下:



Here, the decision links come straight from the preceding activity. Although semantically equivalent and hence the visual style is all that separates them, the explicit decision gateway is much more commonly seen than the implicit style.

在这里,决策链接直接来自前面的活动。虽然语义上等价,因此视觉风格是分开,显式决策网关比隐式更常见。

The second type of gateway is the Parallel Gateway. It does not have an expression associated with it. When reached, all the paths associated are followed in parallel. Each followed path is termed a branch.

第二种类型的网关是并行网关。它没有与之关联的一个表达式。当达到时,所有相连接的路径并行执行。每个路径称为一个分支。????

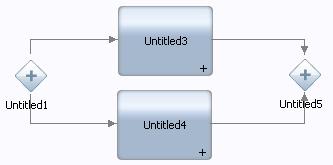


To join the paths again, the same primitive can be used as a join

要再次合并这些路径,可以使用相同的加入？？？

200 end

201



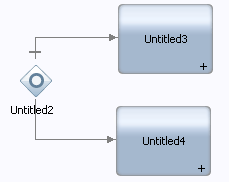
The third type of gateway is called the Inclusive Gateway. Like the simple decision, there are expressions involved but in this case, control can flow from this primitive in one or more directions assuming multiple conditions evaluate to true.

第三种类型的网关被称为包容网关。和简单决策一样,但在这种情况下有多个表达式计算, 假设多个条件评估为true时控制可以从一个或多个方向流动。



For an Inclusive Gateway and Simple Decision, one path is defined as the default and is taken if none of the expressions associated with the other paths evaluates to true. The default path is visually marked as the default by having a “bar” through it.

对于包容性网关和简单决策,一条路径被定义为默认,如果没有与其他路径相关联的表达式计算为true时会执行默认路径。默认路径用一个穿过它的“门闩”来视觉标记。

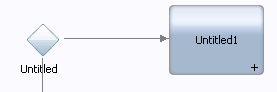


When a link is made from a decision type gateway, it is a good practice to give the link a name and make the name associated with the link visible.

当一个链接是来自决策类型网关,它是一个良好的实践是给链接名字并且使连接相关联的名称可见。

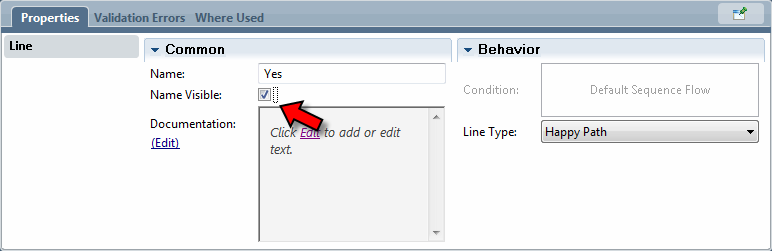
Here we see a decision with no visible label on the link connection:

在这里,我们看到一个决策,没有明显的标签在链接连接的地方:



By selecting the link between the decision and the next component, in the properties view, a tab called Line is shown. Now we can give it a name (“Yes” in this case) and check the box that the name should be visible.

选择决策网关和下一个组件之间的连接,在properties属性视图中,会看见一个称为线的属性选项卡。现在,我们可以给它一个名字(在这个案例中名字是“Yes”)和选择能够让名字可视的复选框。

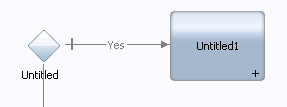


201 end

202

The result is that the name is now shown on the line:

其结果是,名字在线上显示:

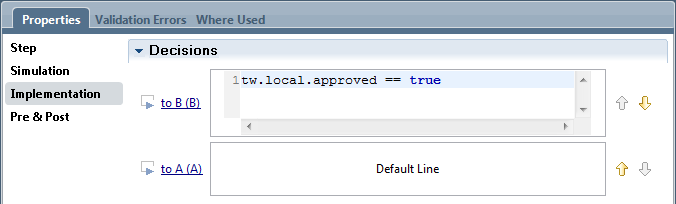


This makes the overall diagram much more readable as now one can tell just by looking at a decision box in a diagram the logic associated with a decision outcome.

这使得整个图可读性更抢,仅仅通过观察在图中的决策框，与决策相关的逻辑结果就出来了。

For gateways that have expressions associated with them, the expression can be entered in the Implementation section in the Properties view:

网关具有与其相关联的表达式,表达式可以在Properties视图实施部分中输入:



The expressions are supplied as JavaScript expressions. When expressions are defined, there is always one item that is defined as the default line. This link will be followed if none of the expressions evaluate to true.

表达式提供的JavaScript表达式。定义表达式时,总有一个项被定义为默认。如果没有这个链接将表达式计算为true。

The order of the expressions is important. For an Exclusive Gateway, only one path is followed. If multiple expressions could evaluate to true, the first expression that is true is the one followed. This means that the order of evaluation has a bearing on the outcome of the logic flow. The expressions can be re-ordered up or down using the arrows to the right of the expressions. The default line is always the last in the list.

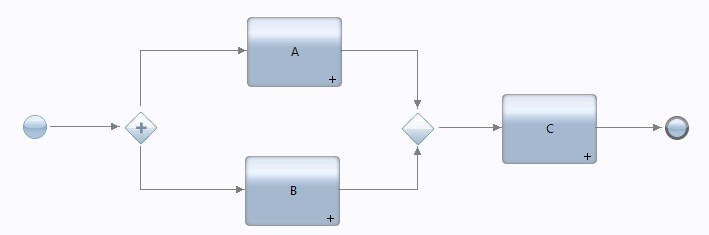
表达式的顺序是很重要的。对于独占网关,只有一个路径跟踪。如果多个表达式可以评估为true,第一个是真的表达式会跟踪。这意味着计算的顺序对逻辑流程的结果有影响。表达式可以使用右边的箭头重新调上或者调下。默认的线总是在列表的最后面。

The Joins in a process diagram also need some explanation. If we look at the BPD palette we see three possible different types of joins so let us look at how they differ and how they might be used.

流程图的合并也需要一些解释。如果我们会看到在BPD 画板里三种可能的不同的合并，让我们看看他们如何不同,他们如何被使用。

The first one we will look at is the exclusive join.

第一个,我们将看看独占合并。



￼In this case, if either A or B complete, then C will start. There is no synchronization. If one thinks about this, there is a new question to be asked. Obviously, one of A or B will complete first and C will be started ... but what happens when the other of A or B then completes? The answer is that a second instance of C will start. From a token perspective, the token from A will pass to C and also the token from B will also to pass to C.

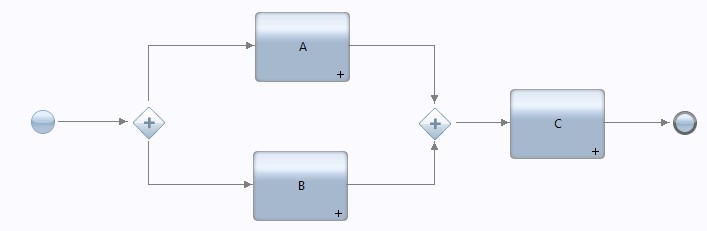
￼在这个案例中,如果A或B完成了,C就会开始。没有同步。如果一个人认为,有一个新的问题要问。显然,A或B中某一个将完成并且C将开始……但剩下A或B中某一个也要完成吗?答案是C的第二个实例会开始。从令牌的角度来看,来自A的令牌传给C，B也会一样将令牌传递给C。

The next join we look at looks like this. This is a synchronizing join. In this instance both A and B must complete before C is allowed to start.

接下来加入我们看这样子。这是一个同步加入。在这个实例中A和B 在C允许开始之前必须完成。

202 end

203

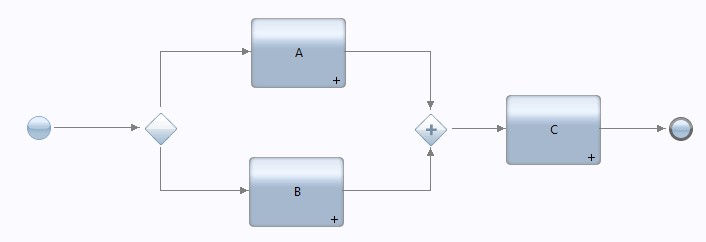


From a token perspective, the tokens from A and B are merged at the join to form a single token which is propagated onwards. With this join, all inputs must be satisfied before C can progress.

从令牌的角度来看,a和B的令牌是合并在一起形成一个单一传播令牌。这个连接,在C可以前进之前必须满足所有输入。

Still looking at the same join, give some thought to this pattern:

仍然在同样的结合点上,给出这种模式的一些想法:

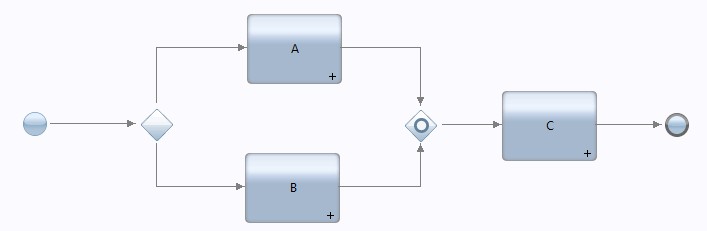


Notice that only one of A or B will execute as the condition gateway is an either/or. The join expects all inputs to be satisfied before continuing onwards towards C. Since only one of A or B will execute, this is an invalid diagram as the join will never be satisfied.

请注意, A或B中只有一个将执行因为条件网关是一个非此即彼的。合并网关期望加所有输入都能够满足才能继续去往c .既然只有A或B中某一个将执行,那么这是一个无效的图因为合并网关的条件将永远不会满足。

This brings us to the last type of join.

这给我们带来了最后的连接类型。



This join has somewhat "magical" properties. Again, it is a synchronizing join and will wait for all possible inputs to arrive before it continues but the magic here is my use of the word "possible" inputs.

这个合并网关有一些“神奇”的特性。再一次,这是一个同步的汇聚合并并且将等待所有可能输入到达之后才能继续运行但是魔术在这是我对“可能的”输入的称呼用法。

In this diagram, it will know that a token will arrive only from one of A or B and when one arrives, it will pass on to "C". However, if there were other diagram structures where other tokens could arrive, it will wait for those.

在这个图中,它将知道令牌只会从一个a或B,当其中一个到达时,它会传递给“C”。然而,如果有其他的令牌可以到达的其他图结构的话,它将等待这些。

Think about this join as being able to do work to see the bigger picture and know what possible tokens *could* arrive and for the ones that will come in the future, will wait but if there are no further tokens that will arrive, it will allow control to pass onwards.

想想这个合并网关能够看到更大的图景和工作知道令牌可能到达和那些将会在未来到来的令牌，合并网关会等待,但如果没有进一步的令牌将到来,它将使控制开始继续向后执行。

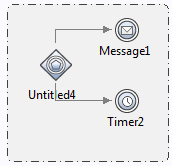
There is one final gateway type that is called the Event Gateway. Strictly speaking, this should really be considered an intermediate message event but it is grouped in the gateway section in the

PD tooling.

最后还有一个网关类型称为事件网关，严格地说，这应该被视为一个中间消息事件但是在PD的工具里被分组在网关这一部分里。

203 end

204

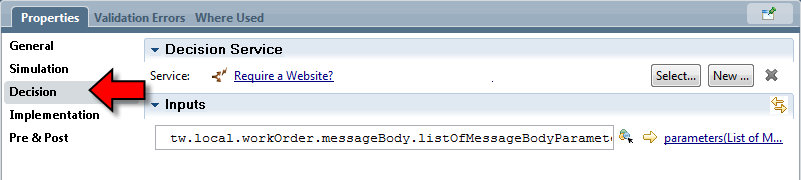


The event gateway waits for a message to arrive but, if a message has not been delivered within a time period, the gateway times out and follows the path associated with the timer. This allows us to wait for a message and, if none has arrived, we can timeout.

这个事件网关等待一个消息到达，但是如果一个消息没有在一个时间段里发出的话，网关超时并遵循计时器的那条路径，这让我们等待一个消息到来，如果没有到来，我们就执行超时事件。

In addition to using the expression mechanism described above, one can also use the Decision selection:

除了使用上面的表达机制，我们还可以使用决策。



In the Decision section, a Decision Service can be mapped to be called. It can be passed any mandatory parameters needed for its execution. This decision service is invoked when the gateway is reached. The resulting output parameters from this decision are available in the variable name- space that starts "tw.decision.\*". These values can then be used in the expressions to determine which path the process should follow.

在决策这一部门里，可以映射到决策服务，它可以传递任何强制执行的所需要的参赛，当到达网关的时候会调用这个决策服务，”tw.decision.\*”开始的命名空间变量可以获取决策服务生成的输出参数。表达式中使用这些值可以确定流程应该执行那条路径。