在OpenERP中，工作流是管理一组“所做的事情”（与一些数据模型的记录关联）的人为现象。工作流提供了高级别的方式来组织记录要上做的事情。

具体地说，工作流是一个定向的路径，这里节点称为活动并且弧线称为流程进度。

活动定义了OpenERP应该处理的工作，比如改变某些记录的状态，或者发送邮件。

Transitions 控制了活动之间工作流的处理进度

在一个工作流定义中，一个达到了条件，就会触发进度的推进，这样工作流的行为取决于用户的actions（比如点击一个按钮），变更记录，或者任意的Python代码。

Basic（基本）

使用数据文件定义一个工作流是很直截了当的：针对activities和transitions的记录工作流是与记录一起给出。例如，这里是定义在XML中简单的一系列的两个activities。

[html] view plaincopy在CODE上查看代码片派生到我的代码片

<record id="test\_workflow" model="workflow">

<field name="name">test.workflow</field>

<field name="osv">test.workflow.model</field>

<field name="on\_create">True</field>

</record>

<record id="activity\_a" model="workflow.activity">

<field name="wkf\_id" ref="test\_workflow"/>

<field name="flow\_start">True</field>

<field name="name">a</field>

<field name="kind">function</field>

<field name="action">print\_a()</field>

</record>

<record id="activity\_b" model="workflow.activity">

<field name="wkf\_id" ref="test\_workflow"/>

<field name="flow\_stop">True</field>

<field name="name">b</field>

<field name="kind">function</field>

<field name="action">print\_b()</field>

</record>

<record id="trans\_a\_b" model="workflow.transition">

<field name="act\_from" ref="activity\_a"/>

<field name="act\_to" ref="activity\_b"/>

</record>

定义的工作流关联到一个特定的模型（这个模式由模型workflow的osv属性给出）。activities或者transations内指定的方法将在这个模型上调用。

在上路的例子代码中，创建了一个调用test\_workflow的工作流。其由两个activities，“a”和“b”，一个transation，从a到b，组成。

第一个activity有属性flow\_start，并设置为true，这样OpenERP知道在工作流实例化后从哪里开启工作流便利。因为工作流记录的on\_create设置为true，为每个新创建的记录实例化工作流（否则，要通过其他的方式实例化工作流，比如一些模块的Python代码）。

当实例化工作流时，从activity a开始。那个activity是一种函数，意味着模型test.workflow上的方法action pring\_a调用（通常传递cr，uid，ids，context参数给它）。

a和b之间的transation不指定任何条件。意味着处理a后工作流实例立即从a走向b，并稍后处理activity b。

Transation(迁移)

Transation提供了一个控制结构安排工作流。当一个activity完成时，工作流引擎就试着从一个完成的activity（活动）流向下一个activity（活动）。在它们最简单的表单中（如同上面的例子），它就继续连接activities（活动）：即之前的activities（活动）处理完成就处理后面的activities（活动）。

作为一下子运行所有的activities（活动）的替换，其也可能在transitions（迁移）上等待，仅当匹配了某些条件才能通过。标准是条件，信号或者触发。它们将在下面详细讨论。

Conditions（条件）

当完成一个activity时，检查它的输出Transation决定工作流实例处理它们并到达下一个activity。当只定义了一个条件时（没有信号，没有触发），OpenERP评估这个条件，如果评估为true，工作流实例处理通过transation。如果条件不匹配，在每次关联的记录修改时都会再评估一次，或者通过一个显示的方法调用做这件事。

默认地，属性condition（即评估表达式） 是true。注意条件可能有几行长，在那种情况下，最后的一个值决定了是否采取通过。

在条件评估环境中，定义了几个方便的符号（除了在OpenERP的safe\_eval环境中）

所有的模型列名，and

所有浏览器记录属性

Singals（信号）

除了条件外，一个transation可以指定一个信号名。当使用一个信号名时，transation不会直接通过，即使condition评估为true。阻塞transation，等待被唤醒。

为了唤醒一个由信号名定义的transation，信号必须发送给工作流实例。发送信号的通用方式是在用户界面使用一个按钮，使用元素<button/>并且信号名作为按钮的属性name。一旦点击了按钮，信号发送给当前记录的工作流实例。

注意

当信号发送给工作流实例时，仍要评估条件。

Triggers（触发）

条件评估为fasle，transation不会通过（并且它引导的activity不立即处理）。工作流实例仍然可以通过提供所谓的触发器处理transation。这是发生在条件不满足，触发器记录在数据库中的情况下。之后，可能唤醒工作流实例，其安装了那些触发器，提供它们重新评估transation条件。这种机制使得唤醒工作流实例很便宜，只是针对几个（安装了触发器的工作流）而不是全部

触发器以记录IDs记录在数据库中（与模型名字一起）并适用于等待那些记录的工作流实例。transation定义提供了一个模型名（属性trigger\_model）和一个Python表达式（属性trigger\_expression），在给定的模型中评估成一列记录IDs。那些记录的任何一个可以唤醒它们关联的工作流实例。

注意

无论什么时候transation重试，触发器不重装。

Splitting and joining transitions（拆分和联合过渡）

当多个transitions离开同一个activity，或者流向同一个activity，OpenERP提供了获取哪个transition的控制，或者如何处理到达的activity。activity的split\_mode和join\_mode 属性用于这样的控制。那些属性的可能值解释如下。

Activities

transition可以通过工作流的控制结构看见，activities是事件发生的地方，从改变记录状态到发送邮件。

存在不同的activities：Dummy，Function，Subflow和Stop all，当activity处理时每个做不同的事情。除了这些类型，activity还有其他属性，详情看下面。

Flow start and flow stop（流开始和流结束）

flow\_start属性是一个布尔值，指定了当工作流实例化时，activity是否在处理。多个activities能将其属性flow\_start设置为true。当为一个记录实例化一个工作流时，OpenERP简单地处理所有，并之后评估输出的transitions。

属性flow\_stop是一个布尔值，指定activity是否停止工作流实例。当所有的activity，其flow\_stop属性设置为true，都完成时，工作流实例认为是完成了。

对于OpenERP知道工作流实例完成了是很重要的。一个工作流可以有一个activity，其是另一个工作流（称为子流）。当子流完成了，这个活动也就完成了。

Subflow（子流）

一个活动可以嵌入一个完整的工作流，称为子流（嵌入的工作流叫做父工作流）。实例化的工作流通过属性subflow\_id指定。

注意

在GUI中，那个属性不能被设置除非activity的种类是Subflow

当其子流完成时这个activity被认为是完成了（它的输出transitions准备评估）

从子流中发送一个信号

当一个工作流嵌入了一个工作流的activity（作为子流），子流可以从其自己的activities发送一个信号到其父工作流，通过给的信号名（属性signal\_send）。OpenERP通过发送以subflow为前缀signal\_sen的值给父工作流处理那些activities.

换句话说，可能相互交互并且在父工作流中获取transition，作为子流中执行的activities。

Server actions（服务器action）

一个action可以运行一个Server Action 通过在属性action\_id中指定其ID。

Python action

一个acitivity可以执行一些Python代码，由属性action给出。评估环境与Condition部分解释的一样。

Split mode（分叉模式）

一个activity处理后，就评估其输出transitions。一般地，如果一个transition能被获取，OpenERP通过它并且继续通向transition导向的下一个activity。

实际上，当多个transition离开一个activity，OpenERP可能继续或者不，这取决于其他的transitions。即，transitions上的conditions能联合在一起，联合的结果通知OpenERP穿过0,1，或者所有的transitions。这种联合的方式由属性split\_mode控制。

有三种分叉模式：XOR，OR 和ＡＮＤ

XOR

当迁移与一个XOR分叉模式联合时，只要第一个满足迁移条件的迁移跳转，只有 一个跳转。

OR

使用OR模式，只要满足迁移条件即沿着该迁移跳转。有零个或者多个跳转。剩余的迁移条件将不再评估。

AND

使用AND模式，OpenERP只有所有迁移都满足迁移条件才跳转，而且是沿所有迁移跳转。有零个或全部跳转

Join mode（合并模式）

就像输出迁移条件可以合并起来决定它们能否通过一样，输入迁移能合并来决定一个activity（活动）是否将处理或者什么时候处理。属性join\_mode控制那个行为。

有两种可能的合并模式：XOR和AND

XOR

使用XOR模式，以本节点为终点的入迁移，只要有一个跳至本节点，即执行本节点的Action。

AND

使用AND模式﻿﻿，在处理activity前，OpenERP将等待直到所有的输入迁移已经通过。（以本节点为终点的入迁移，只有所有迁移都已经跳至本节点，才执行本节点的Action）

种类

Activities能是不同的种类：dummy，function，subflow，或者stopall。种类定义了一个activity能做的工作类型。

Dummy

activity的Dummy种类什么都不做，或者对应activites仅调用一个服务action。什么都不做的activities用于作为hubs来获取/分发translations

Funcation（函数）

对应activity（活动）的Funcation（函数）种类仅需要运行一些Python代码，并且可能是一个Server action。

Stop all

activity的stopall种类完全停止工作流实例并且标记其已经完成。另外其也可以运行一些Python代码。

Subflow

当activity的种类是Subflow时，activity插入到另一个工作流实例。当子流完成时，activity也认为完成。

默认地，the subflow is instanciated for the same record as the parent workflow。通过提供Python代码（返回一个记录ID(作为子流的同样的数据模型)）可能改变行为。嵌入的子消息流实例是一个所给的记录。

Transitions

Transitions provide the control structures to orchestrate a workflow. When an activity is completed, the workflow engine tries to get across transitions departing from the completed activity, towards the next activities. In their simplest form (as in the example above), they link activities sequentially: activities are processed as soon as the activities preceding them are completed.

Instead of running all activities in one fell swoop, it is also possible to wait on transitions, going through them only when some criteria are met. The criteria are the conditions, the signals, and the triggers. They are detailed in the following sections.

Conditions

When an activity has been completed, its outgoing transitions are inspected to determine whether it is possible for the workflow instance to proceed through them and reach the next activities. When only a condition is defined (i.e., no signal or trigger is defined), the condition is evaluated by Odoo, and if it evaluates to True, the worklfow instance progresses through the transition. If the condition is not met, it will be reevaluated every time the associated record is modified, or by an explicit method call to do it.

By default, the attribute condition (i.e., the expression to be evaluated) is just "True", which trivially evaluates to True. Note that the condition may be several lines long; in that case, the value of the last one determines whether the transition can be taken.

In the condition evaluation environment, several symbols are conveniently defined (in addition to the Odoo safe\_eval environment):

all the model column names, and

all the browse record's attributes.

Signals

In addition to a condition, a transition can specify a signal name. When such a signal name is present, the transition is not taken directly, even if the condition evaluates to True. Instead the transition blocks, waiting to be woken up.

In order to wake up a transition with a defined signal name, the signal must be sent to the workflow instance. A common way to send a signal is to use a button in the user interface, using the element <button/> with the signal name as the attribute name of the button. Once the button is clicked, the signal is sent to the workflow instance of the current record.

Note

The condition is still evaluated when the signal is sent to the workflow instance.

Triggers

With conditions that evaluate to False, transitions are not taken (and thus the activity it leads to is not processed immediately). Still, the workflow instance can get new chances to progress across that transition by providing so-called triggers. The idea is that when the condition is not satisfied, triggers are recorded in database. Later, it is possible to wake up specifically the workflow instances that installed those triggers, offering them to reevaluate their transition conditions. This mechanism makes it cheaper to wake up workflow instances by targetting just a few of them (those that have installed the triggers) instead of all of them.

Triggers are recorded in database as record IDs (together with the model name) and refer to the workflow instance waiting for those records. The transition definition provides a model name (attribute trigger\_model) and a Python expression (attribute trigger\_expression) that evaluates to a list of record IDs in the given model. Any of those records can wake up the workflow instance they are associated with.