火龙果•整理
uml.org.cn

jBPM User Guide

jBPM 用户指南

目录

目录

第一章 概述.....1
1.1 什么是 jBPM？.....1
1.2 概览.....2
1.3 核心引擎.....3
1.4 Eclipse 编辑器.....4
1.5 基于 Web 的设计器.....5
1.6 jBPM 控制台.....5
1.7 文档结构.....6
第二章 获得开始.....7
2.1 下载.....7
2.2 获得开始.....7
2.3 社区.....8
2.4 资源.....8
2.41 许可证.....8
2.42 源代码.....9
2.4.3 从源文件构建.....9
第三章 安装.....9

3.1 必备的.....9
3.2 下载 installer.....9
3.3 演示安装.....10
3.4 10 分钟指南： 使用 Eclipse 工具.....10
3.5 10 分钟向导：使用 jBPM 控制台.....12
3.6 10 分钟指南：使用 Guvnor 知识库和设计器.....13
3.7 如果遇到问题或有问题应该怎么去做.....14
3.7 经常被问的问题.....14
第四章：核心引擎的 API.....15
4.1 jBPM API.....16
4.11 Knowledge Base.....17
4.12 会话.....17
4.13 事件.....19
4.2. Knowledge-based API.....21
第五章 核心引擎：基础.....21
5.1 创建一个流程.....21
5.1.1 Eclipse BPMN2.0 图形编辑器插件.....22
5.1.2 使用 XML 定义流程.....22
5.1.3 使用流程 API 定义流程.....24
5.2 不同流程构造的细节概述.....25
5.3 流程属性的细节.....26
5.4 事件细节.....26
5.4.1 开始事件.....26
5.4.2 结束事件.....27
5.4.3 中间事件.....28

5.5 动作细节.....29
5.5.1 脚本任务.....29
5.5.3 服务任务.....30
5.5.3 用户任务.....31
5.5.4 重用的子流程.....32
5.5.5 业务规则任务.....32
5.5.6 嵌入子流程.....33
5.5.7 多个实例子流程.....33
5.6 Gateways 细节.....34
5.6.2 合并网关.....35
5.7 在你的应用中使用流程.....35
5.8 其他特性.....36
5.8.1 数据.....36
5.8.2 约束.....37
5.8.3 动作脚本.....38
5.8.4 事件.....39
5.8.5 计时器.....40
5.8.6 更新流程.....41
第六章 BPMN 2.0 核心引擎.....43
6.1 Business Process Model and Notation（BPMN）2.0 Specification 业务流程建模和符号 2.0 规范.....43
6.2 例子.....47
6.3 支持的元素/属性.....48
第七章 核心引擎：持久化和事务.....54
7.1 运行时状态.....54

7.1.1 二进制持久化.....54
7.1.2 Safe Pionts.....54
7.1.3 配置持久化.....55
7.1.4 事务.....58
7.2 流程定义.....58
7.3 历史日志.....59
7.3.1 将流程事件存储到数据库.....59
第八章 核心引擎：示例.....60
8.1 jBPM 示例.....60
8.2 示例.....61
8.3 单元测试.....61
第九章 Eclipse BPMN 2.0 Plugin.....62
9.1 安装.....62
9.2 创建你的 BPMN 2.0 流程.....62
9.3 元素和属性的顾虑.....65
第十章 设计器.....66
10.1 安装.....67
第十一章 控制台.....68
11.1 安装.....68
11.2 运行流程管理控制台.....68
11.2.1 管理流程实例.....69
11.2.2 人工任务列表.....71
11.2.3 报告.....72
11.3 添加新的流程/任务表.....73
11.4 REST interface.....74

第十二章 Human Tasks..... 75
12.1 流程内的 Human tasks..... 75
12.1.1 Swimlanes..... 78
12.2 Human task 管理组件..... 78
12.2.1 任务生命周期..... 78
12.2.2 将任务组件链接到 jBPM 引擎..... 79
12.2.3 启动任务管理组件..... 81
12.2.4 和任务管理组件的交互..... 83
12.3 人工任务管理接口..... 84
12.3.1 Eclipse integration..... 84
12.3.2 基于 Web 的任务视图..... 84
第十三章 特定领域的流程..... 84
13.1 介绍..... 84
13.2 示例：通告..... 85
13.2.1 创建工作定义..... 86
13.2.2 注册你的工作定义..... 86
13.2.3 在你的流程中使用你最新的工作条目..... 87
13.2.4 执行服务节点..... 89
第十四章 测试和调试..... 91
14.1 单元测试..... 91
14.1.1 使用 Helper 方法创建你的 session..... 92
14.1.2 断言..... 92
14.1.3 测试与外部服务的结合..... 93
14.1.4 配置持久化..... 94
14.2 调试..... 94

14.2.1 流程实例视图.....95

14.2.2 人工任务视图.....96

14.2.3 核查视图.....96

第十五章 流程知识库.....97

第十六章 业务活动监视器.....99

16.1 报告.....99

16.2 直接接入.....101

第十七章 复杂流程.....102

第十八章 和 Maven，OSGi，Spring 的整合.....105

18.1 Maven.....105

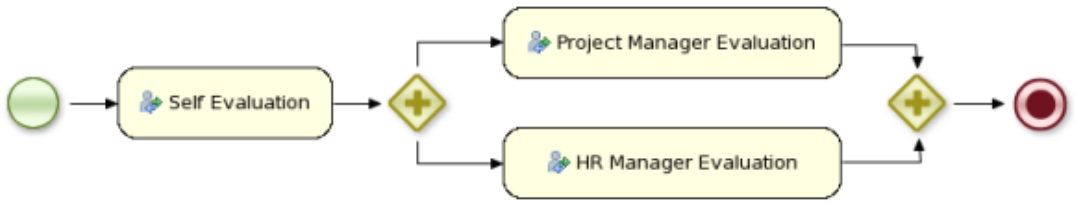
18.2 OGSi.....107

第一章 概述

1.1 什么是 jBPM？

jBPM 是一个灵活的业务流程管理组件。它是轻量级，完全开放并且是使用 java 语言编写。它允许你建模、执行和监控业务流程，遍及整个的生命周期。

业务流程允许你去建模一个通过描述要达到的目标并按一定次序执行的每一步来作为一个流程表。这样能够明显的提高你的业务流程的可视化和灵活性。jBPM 关注于可执行的业务流程，这些业务流程包含足够的细节而使他们能够真正的在 BPM 引擎上执行。可执行的业务流程解决了业务用户和开发人员的隔阂，这些可执行的业务流程是更高级的并且应用了能够被业务用户所理解并且能够直接执行的特有的域概念。



jBPM 的核心是一个纯 java 语言编写的轻量级、可扩展的工作流引擎，它允许你去执行使用了最新的 BPMN 2.0 规范的业务流程。它能够在任何的 java 环境中运行或者嵌入到你的应用中或者作为一个服务。

在核心引擎的顶端，有很多的特性和工具被提供用来支持遍及整个业务流程的生命周期。

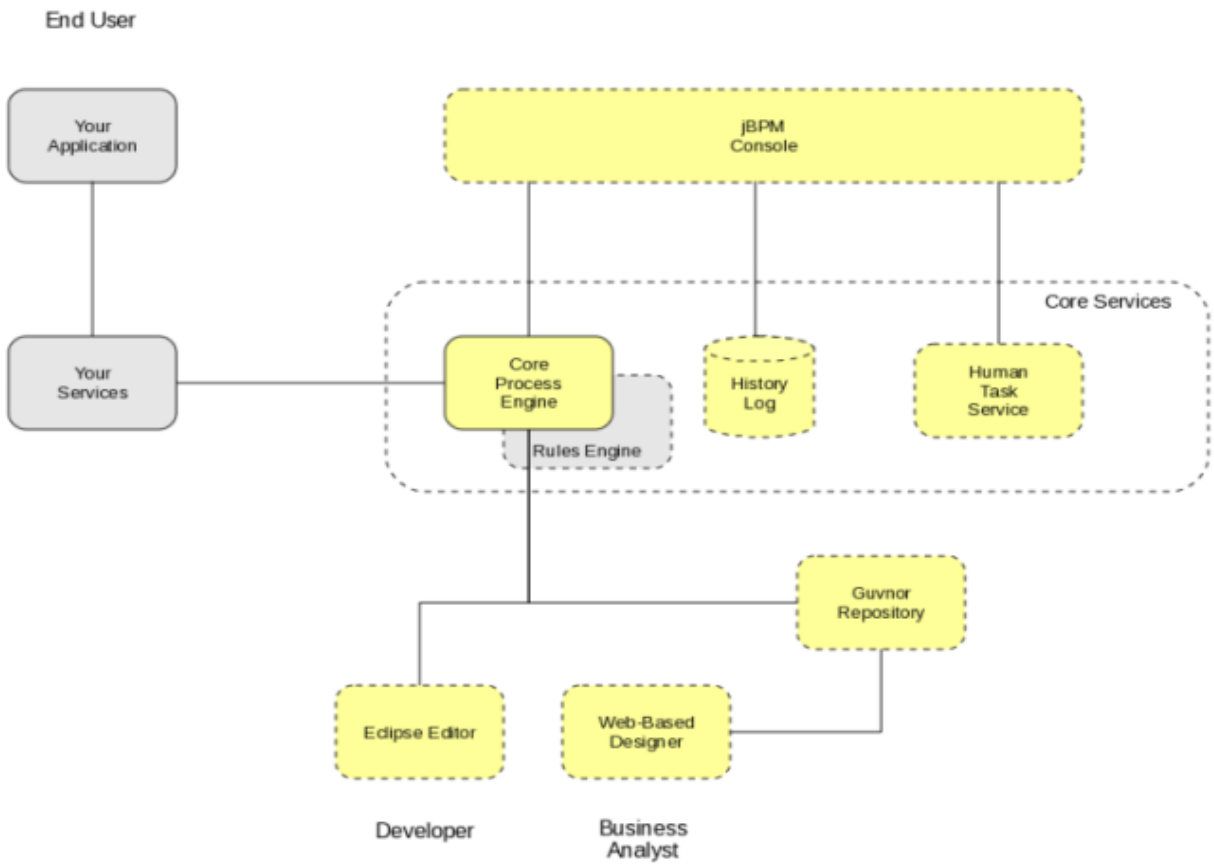
- 基于 Eclipse 和 Web 的编辑器支持支持你的业务流程的图形化创建。
- 持久化插件和基于 JPA/JTA 的事务处理。
- 基于 WS 人工任务的人工任务服务插件针对于包含了各种需要被人工角色执行的任务。
- 控制台管理支持流程实例的管理，任务列表和任务表的管理，和报告。
- 可选择的流程知识库能够配置你的流程
- 历史日志（用于查询）
- 可以和 Seam, Spring, OSGi 进行整合

BPM 架起了业务分析，开发人员和最终用户之间的桥梁，并且提供了业务用户和开发人员都喜欢的流程管理特性和工具的方式。特有的域节点能够被插入到模板中，使得流程能够使业务用户更加容易理解。

JBPM 提供了可适应性和动态性流程，流程需要灵活的去建模复杂的、真正生命的那些不能够容易的使用死板的流程所描述的情形。我们提供了最终用户的控制权限通过允许他们去控制应该被执行的流程的每一个方面和动态的变更流程等等。

当然 jBPM 也不仅仅是一种独立的业务引擎，复杂的业务逻辑能够作为一种拥有业务规则的业务流程和复杂事件流程的结合而被建模。jBPM 能够和 Drools 项目相结合来支持统一的环境，结合这些范例你可以建模你的业务逻辑作为流程、规则和事件的结合。除了核心引擎自身之外，还有一些可添加和可选的组件你能够使用，例如基于 Eclipse 和 Web 的设计器和控制台管理。

1.2 概览



图片 1.1

这张图片提供了 jBPM 项目不同组件的概览。jBPM 能够整合很多其他的服务，但是这我们过关注的组件式 jBPM 项目自身的一部分。

- 流程引擎是项目的核心并且是必须的，如果你想去执行业务流程，你的应用服务在任何必须的时候都可以调用核心引擎。
- 有一个可选择的核心服务时历史日志，它将会记录你所有的流程实例的当前和先前状态的有关信息。
- 另外一个可选的核心服务时人工任务服务，如果人工角色参与了流程，它将会管理人工任务的生命周期。
- 提供了两种类型的图形编辑器来定义你的业务流程。
- Eclipse 插件是 Eclipse 集成开发环境的一种扩展，目标针对于开发人员，同时允许你通过拖拽的方式创建你的业务流程，提供高级的测试。
- 基于 Web 的设计器允许业务用户在基于 Web 的环境下管理业务流程。
- Guvnor 知识库是一个可选的组件，它能够被用来存储你所有的业务流程。它支持协作和编译等等。Eclipse 插件和基于 Web 的设计器的结合，支持迂回流动的工具。
- jBPM 控制台是基于 Web 的控制台，它允许业务用户去管理他们的业务流程、任务列表和查看报告。

每个组件的更多细节将会在下面被描述。

1.3 核心引擎

jBPM 核心引擎是项目的核心。它是一种轻量级的工作流引擎用来执行你的业务流程。它可以作为你的应用的一部分被嵌入，或者配置作为一项服务。这里还有很多的特性：

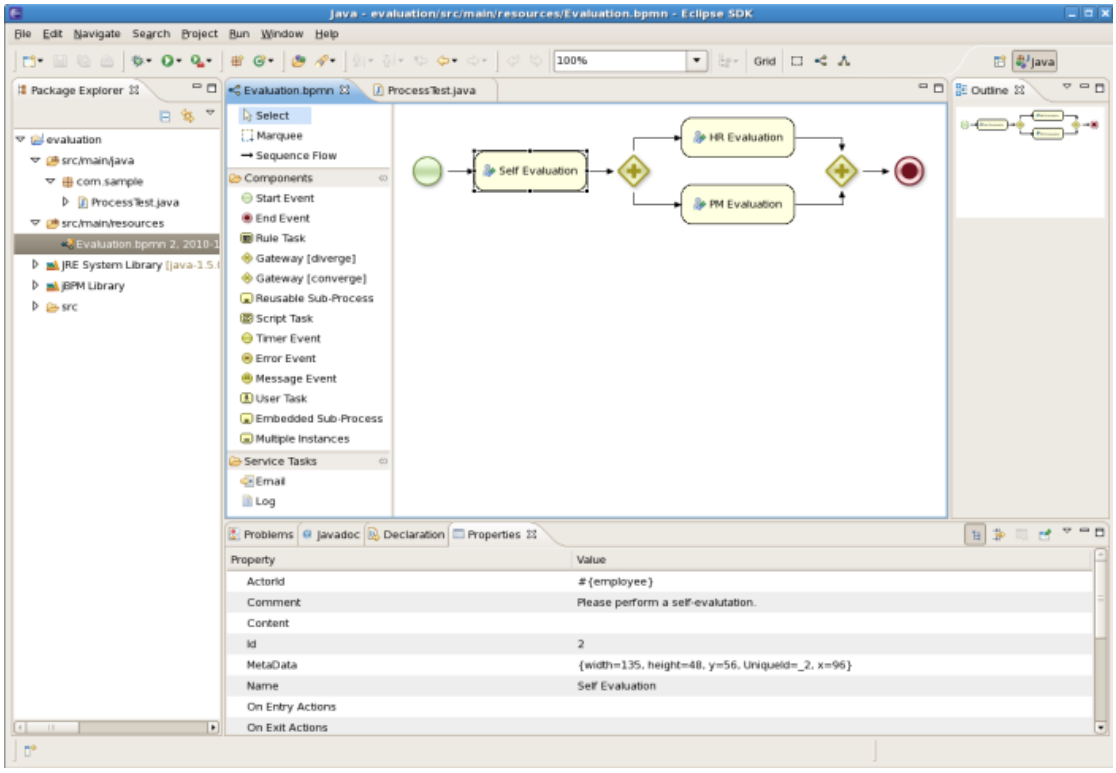
- 可靠地、稳定地核心引擎用来执行你的流程实例
- 对 BPMN 2.0 规范的本地支持，执行业务流程
- 强壮的执行和监控能力
- 轻量级的（能够被配置在几乎任何的支持简单的 java 运行环境的设备上，而不需要任何的 Web 容器）
- （可选择的）带有一个默认的 JPA 执行的持久化插件
- 带有一个默认的 JTA 执行的事务支持插件
- 作为一个普通的流程引擎工具，它能够被扩展来支持新的节点类型或者其他的流程语言
- 监听器可以通报各种事件
- 具有移植运行中的流程实例到他们流程定义的一个新版本中的能力

当然，核心引擎能够和其他的核心服务相整合：

- 当人工角色需要参与流程的时候，人工任务服务能够被用来管理人工任务。它是一个完整的插件并且是基于 WS 人工任务规范的默认工具，管理任务的生命周期、任务列表、任务表和一些更高级的特性，例如自动调整、委托、基于规则的分配等等。
- 历史日志可以存储所有在引擎上的流程的执行的有关信息。如果你需要访问历史信息作为运行时的持久化仅仅存储所有活动流程实例的当前状态这就是必需的了。历史日志能够存储活动的和完整的流程实例的所有的当前和历史状态。它能够被用来查询任何和流程实例执行相关的信息，用来做监测分析等等。

1.4 Eclipse 编辑器

Eclipse 编辑器是针对于 Eclipse 集成开发环境的插件，它允许你整合你的业务流程到你的开发环境中。它的目标是针对于开发者，并且有一些向导去开始，有一个图形编辑器用来创建你的业务流程和许多的高级测试和调试能力。



图片 1.2 Eclipse 编辑器创建 BPMN2 流程

它包含的特性有：

- 创建一个新的 jBPM 项目的向导
- 针对 BPMN 2.0 流程的图形编辑器
- 插入你拥有的特有域节点
- 验证
- 运行时支持（你能够选择你喜欢用的 jBPM 版本）
- 图形化调试，可以查看一个被选择的会话的所有运行时流程实例，可以显示一个特有的流程实例的当前状态等等。
- 审核视图得到一个在运行时所发生的概述
- 和知识库集成

1.5 基于 Web 的设计器

基于 Web 的设计器允许你在基于 Web 的环境中建模你的业务流程。它所针对的目标更多的是业务用户和提供一个图形编辑器来显示和编辑你的业务流程（使用拖拽），类似于 Eclipse 插件。它支持 Eclipse 编辑器和基于 Web 设计器的迂回流动。

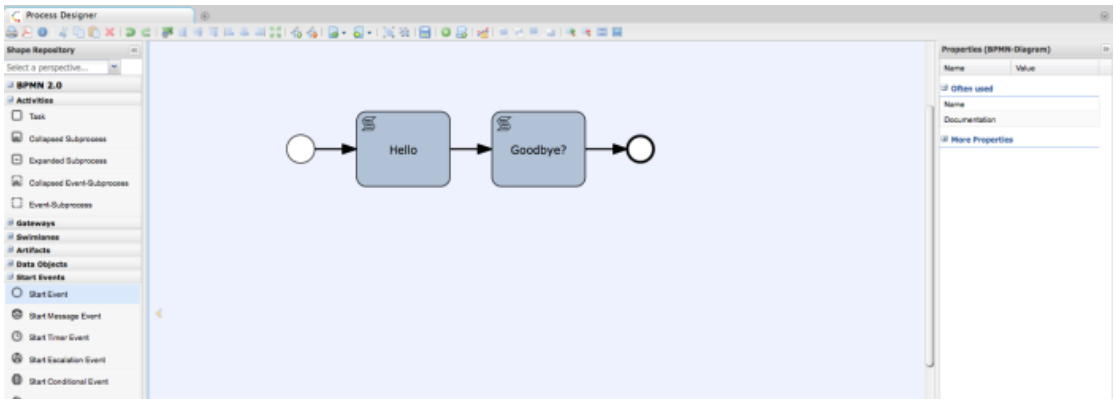


图 1.3 基于 Web 的设计器创建 BPMN2 流程

你能够随意的使用一个或者多个知识库来存储你的业务流程。基于 Web 的设计器被结合到 Guvnor 知识库中，它所针对的目标是业务用户和允许你从你的应用中分离处理管理你的流程。它支持：

- 一个知识库服务用来存储你的业务流程和相关的人工品，使用一个 JCR 知识库，它可以作为一个文件系统或者使用 REST 服务来支持译本、远程接入。
- 基于 Web 的用户接口用来管理你的业务流程，目标针对于业务用户，支持你流程的可视化，当然还有分类，方案测试。部署配置等等。
- 协作特性对于有很多角色的共同工作可以在同一个流程中定义。
- 知识库代理可以在知识库中很容易的创建一个基于流程定义的新会话。如果一个新的流程已经被配置，它支持动态的升级。

1.6 jBPM 控制台

业务流程能够通过一个 Web 控制台管理。它是主要针对于业务用户，它的主要特性有：

- 流程实例管理：启动一个新的流程实例的能力，得到运行时流程实例的列表，可视化查看一个明确的流程实例等等。
- 人工任务管理：能够得到所有当前的任务列表，完善任务列表上的任务。
- 报告：得到你的应用状态的概览，系统动态产生报告，给你一个关键执行指示的概述

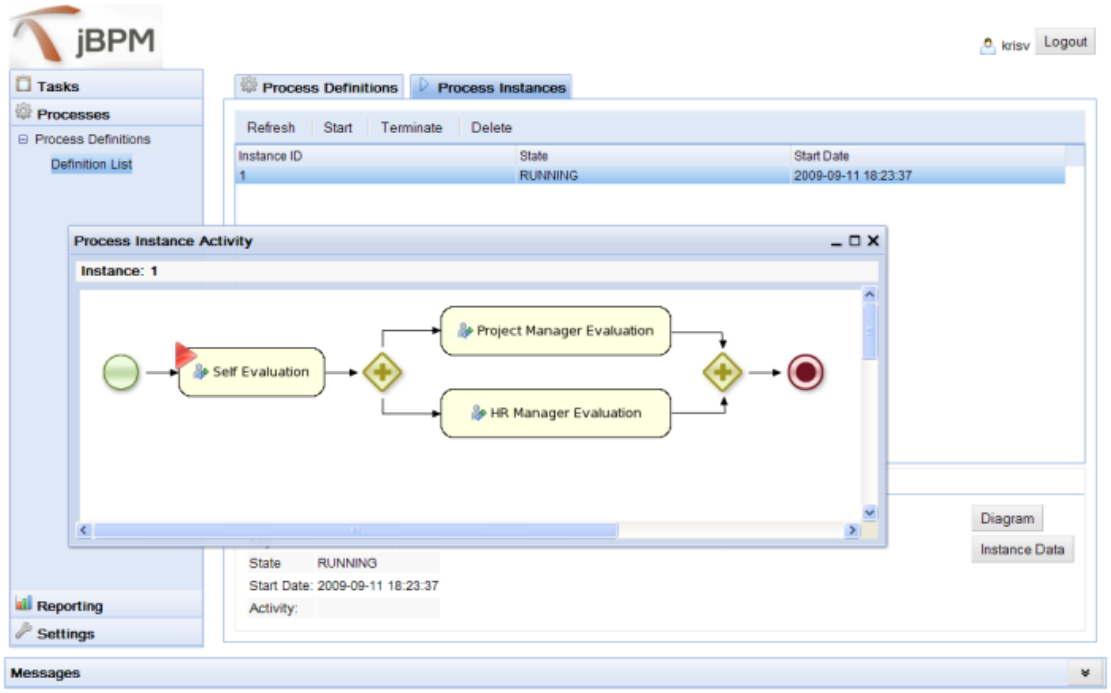


图 1.4 管理你的流程实例

1.7 文档结构

文档的结构如下：

- 概述：概述章节给你一个不同组件的概述
- 获得开始：获得开始章节教你从哪下载源文件和包含很多有用的链接
- 安装：安装章节帮助你得到一个能够运行的演示程序，包括很多的 jBPM 组件并且通过他们来应用一个简单的例子，还有包括录屏的 10 分钟指南。
- 核心引擎：下面的四章描述了针对于开发人员的 Eclipse 插件，有一个旧版本的和一个新的 BPMN 2.0 工具正在开发当中。
- 设计器：描述了基于 Web 设计器允许业务用户去编辑在一个基于 Web 的上下文中的业务流程。
- 控制台：jBPM 控制台能够被用来管理流程实例，人工任务列表和报告
- 重要的特性

    1. 人工任务：当使用人工角色的时候，你需要一个人工任务服务区管理任务，任务列表的生命周期等等。
    2. 特有的域流程：可以在你的流程中插入你自己的高级的、特有域的节点

3. 测试和调试：如何去测试和调试你的流程
4. 流程知识库：一个流程知识库能够用来管理你的业务流程

- 复杂的概念
  1. 业务活动监视：事件处理用来监视你的系统状态
  2. 灵活的处理：建模很多适用性、灵活性的流程用来升级流程构造并使业务规则和事件处理相结合
  3. 整合：怎样和其他的技术，例如 maven，OSGi 等等整合

第二章　获得开始

2.1 下载

所有的版本都可以从 SourceForge[https://sourceforge.net/projects/jbpm/files/]上下载。选择你想下载的版本然后选择你想要的产品：

- bin:所有的 jBPM 的二进制 JAR 包和他们所依赖的
- src:核心组件的源文件
- gwt-console: jBPM 控制台，一个 ZIP 文件包含了服务器端和客户端的 WAR 包
- docs:文档
- examples：一些 jBPM 例子，能够被引入到 Eclipse 中
- installer：包含 jbpm-installer，下载和安装一个 jBPM 的演示程序
- installer-full：包含 bpm-installer，下载和安装一个 jBPM 的演示程序，已经包含了一系列所依赖的包

2.2 获得开始

如果你喜欢采用快速指南，它将通过使用一个简单的例子中众多组件来指导你，看一下安装章节吧！这将教你如何去下载并使用 installer 去创建一个演示程序，包含许多的组件。通过一个简单的例子来指导你涉及很多重要的特性。可利用的录屏将很好的帮助你解决问题。

如果你喜欢阅读更多的第一手信息，下面的章节将首要关注核心引擎（API，BPMN 2.0 等等）。在下一章将会描述其他的组件和其他更复杂的主题像 特有域流程，灵活流程等等。当阅读和核心章节之后，你应该能够跳到其他你可能感兴趣的章节。

当然你也可以通过演示一些例子来开始，这些例子已经被提供单独下载。查看一下例子章节，来看看如何通过这些来开始。

在那之后，你应该准备开始创建你自己的流程并整合引擎到你的应用，例如通过从 installer 开始或者另外的例子或者通过自己的理解来开始尝试一下也是不错哦~

2.3 社区

这有很多有用的链接如果你想成为 jBPM 社区的一部分：

- 与 jBPM 相关的博客入口[http://planet.jboss.org/view/feed.seam?name=jbossjbpm]

Jbossjbpm 的相关说明：[http://twitter.com/jbossjbpm].

一个用户论坛：[http://www.jboss.com/index.html?module=bb&op=viewforum&f=217]，可以提问和给出回答

一个 JIRA 的 bug 跟踪系统：[https://jira.jboss.org/jira/browse/JBPM]，对于 bugs 和一些特性的要求和指示。

一个连续的构建服务：[https://hudson.jboss.org/hudson/job/jBPM/]，得到最新的快照：[https://hudson.jboss.org/hudson/job/jBPM/lastSuccessfulBuild/artifact/jbpmdistribution/target]

请免费加入我们的在线聊天频道：irc.codehaus.org #jbpm。这有很多的实时在线关于项目的生产的讨论，能够在你需要的时候可以发现很多的在线开发人员。没有在线聊天的客户端安装程序吗？简单的得到：*http://irc.codehaus.org*,输入你渴望的昵称，然后登陆获得乐趣吧！

2.4 资源

2.41 许可证

jBPM 的代码本身使用了 Apache 的 2.0 版本许可证。

我们所整合的其他组件也有他们自己的许可证：

- 新的 BPMN2 插件是 Eclipse Public License (EPL) v1.0.
- The web-based designer is based on Oryx/Wapama and is MIT License
- The BPM console is GNU Lesser General Public License (LGPL) v2.1
- The Drools project is Apache License v2.0.

2.42 源代码

现在 jBPM 使用 git 作为它的源代码的版本控制系统。jBPM 项目的资源可以在这被找到（包括从 jBPM 5.0-CR1 的所有版本）：*https://github.com/droolsjbpm/jbpm*

我们所整合的其他组件的资源可以在这找到：

- 与jBPM和Drools项目相关的其他组件：[https://github.com/droolsjbpm].
- jBPM的Eclipse插件：[http://anonsvn.jboss.org/repos/jbosstools/trunk/bpmn/plugins/org.jboss.tools.jbpm/]
- 基于Web的设计器：[https://github.com/tsurdilo/process-designer]
- BPM控制台：[https://github.com/bpmc/bpm-console]

2.4.3 从源文件构建

如果你对于构建源代码感兴趣，贡献版本等等。确保阅读这个：[https://github.com/droolsjbpm/droolsjbpm-build-bootstrap/blob/master/README.md].

第三章　安装

这个向导将会在安装和运行一个包含各种组件的 jBPM 项目演示程序帮助时帮助你。如果你有对于如何改进向导的任何反馈，如果你遇到了问题，或者你想解决问题，不要犹豫，联系 jBPM 社区作为所描述的那样：如果遇到问题或者有问题如何去做的片段。

3.1 必备的

这里假设你已经安装了 java JDK 1.5+（设置了 JAVA_HOME）,并安装了 Ant 1.7+.如果你没有，用下面的链接去下载并安装它们：

Java: *http://java.sun.com/javase/downloads/index.jsp*
Ant: *http://ant.apache.org/bindownload.cgi*

3.2 下载 installer

首先，你需要下载 installer [https://sourceforge.net/projects/jbpm/files/jBPM%205/]，这里有两个版本，一个 full installer 和一个最小版本的 installer。总的来说，下载一个 full installer 可能会是最好的：jBPM-{version}-installer-full.zip

你页可以在这找到最新发布的版本，不过只有最小版本的。
https://hudson.jboss.org/jenkins/job/jBPM/lastSuccessfulBuild/artifact/jbpm-distribution/target/

[https://hudson.jboss.org/jenkins/job/jBPM/lastSuccessfulBuild/artifact/jbpm-distribution/target]

3.3 演示安装

最容易的方法就是运行安装脚本来开始，很容易的进入安装文件夹并在运行（注意是在 cmd 中）：

ant install.demo

这将会：

- 下载 JBoss AS
- 下载 Eclipse
- 安装 Drools Guvnor into JBoss AS
- 安装 Oryx Designer into JBoss AS
- 安装 the jBPM gwt-console into JBoss AS
- 安装 the jBPM Eclipse plugin
- 安装 the Drools Eclipse plugin

这样将要花费一会（网速慢的话将会花很长时间，因为下载有应用服务，Eclipse 安装程序，即使你已经下载了 full installer）.脚本将显示正在下载的文件。如果你想避免下载特定的组件，在下面检查运行仅仅特定的演示的部分或者直接安装一个已经安装的组件。

为了限制需要下载的数据量，我们不用下载 Eclipse BIRT 插件通过默认的报告，如果你想在 jBPM 控制台输出报告，确定要下载 BPM.birt 在属性文件中设置一下，在运行安装之前。

一旦演示安装完成，你能够开始通过启动 demo.setup 来演示各种组件。

ant start.demo

这个将会：
• Start the H2 database
• Start the JBoss AS
• Start Eclipse
• Start the Human Task Service

一旦所有的组件都启动了，你就可以启动 Eclipse 工具、Guvnor 知识库和 jBPM 控制台，在下面的片段中将会解释。

3.4 10 分钟指南： 使用 Eclipse 工具

下面的录屏将给你一个如何去运行一个简单的演示流程的概述：
[http://people.redhat.com/kverlaen/install-eclipse-jbpm.swf]，它将会展示给你：

- 怎么将已经存在的项目引入到你的工作空间中去
- 一个简单的 BPMN2 流程用来请求一个执行评估
- 一个简单的 java 类来启动流程
- 怎样开始流程

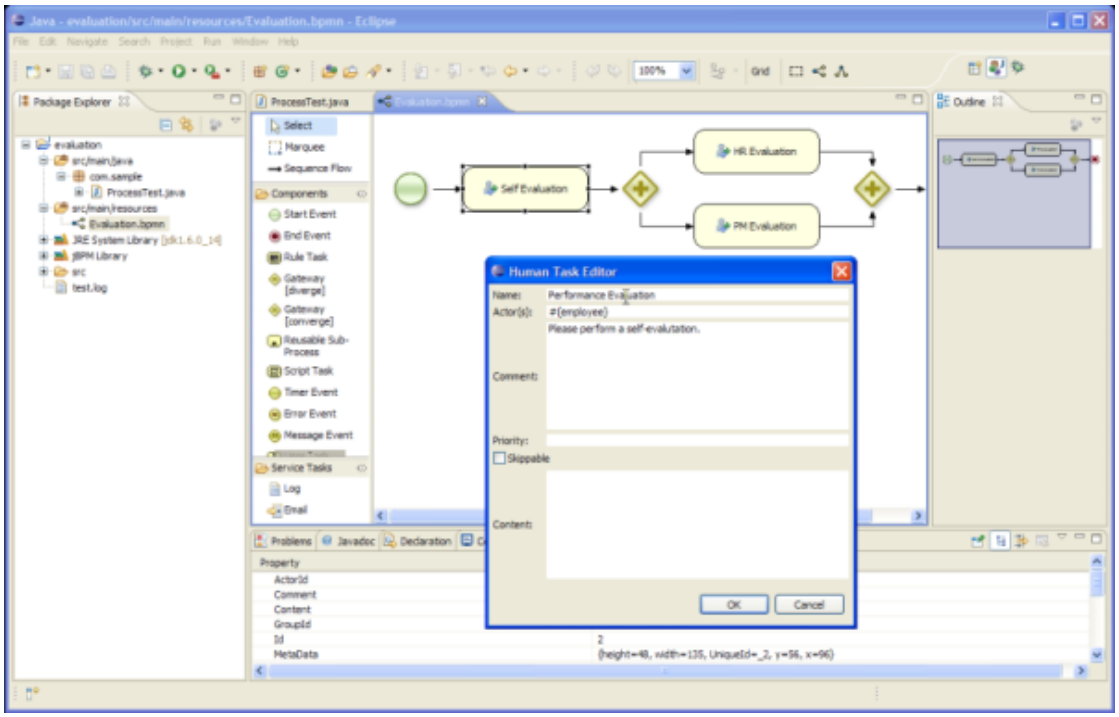


图 3.1 [http://people.redhat.com/kverlaen/install-eclipse-jbpm.swf]

接下来要做的：

- 一旦启动了 Eclipse，就是简单的引入了，如果不会，那么你就回家卖红薯吧！！这个项目包含一个简单的 BPMN2 流程和一个 java 文件用来启动流程。
- 你可以通过双击打开这些文件
- 现在我们就可以 debug 流程了，我们可以通过用 debug tooling 来可视化查看它的运行时状态，首先要在流程测试类的文件中定义断点，然后就是•••
- 查看各种 debug 视图："Window - Show View -> Other ...",，选择流程实例视图和流程实例视图（在 Drools 目录下）和人工任务视图并单击 OK。
- 程序在开始流程之前将会遇到断点，单击“Step Over”（F6）来启动流程。在这种情况下，它会很容易的启动流程，这将导致创建一个新的用户任务对于用户”krisv”在人工任务服务中。之后流程就会等待它的执行。进入人工任务视图，在用户 ID 下面，填写”krisv”并单击刷新。一个新的执行评估任务应该被展现。

- 展示你已经启动了的流程实例的状态，单击流程实例视图并在变量视图中选择 ksession 变量。这将展现在所选会话中的所有活动的流程实例。这种情况下，仅仅有一个流程实例。双击去查看在流程表中有注解的流程实例的状态。

- 现在让我们回到任务视图，选择执行的任务并首先启动然后完成选择的任务。现在我们回到流程实例视图并再次双击流程实例来查看它新的状态。

你应该能够用 jBPM 项目向导创建一个新的项目。这个简单的项目包含一个简单的 HelloWorld BPMN2 流程和一个相关联的 java 文件用来启动流程。

3.5 10 分钟指南：使用 jBPM 控制台

打开流程管理控制台：*http://localhost:8080/jbpm-console*

登陆：用krisv / krisv作为用户名和密码。下面的录屏 [http://people.redhat.com/kverlaen/install-gwt-console-jbpm.swf]将给你一个怎么去管理你的流程实例的概述。它展现给你：

- 怎样启动一个新的流程
- 怎样查看一个正在运行的流程实例的当前状态
- 怎样去查看你的任务
- 怎样完成一个任务
- 怎样去产生一个报告区监视你流程的执行

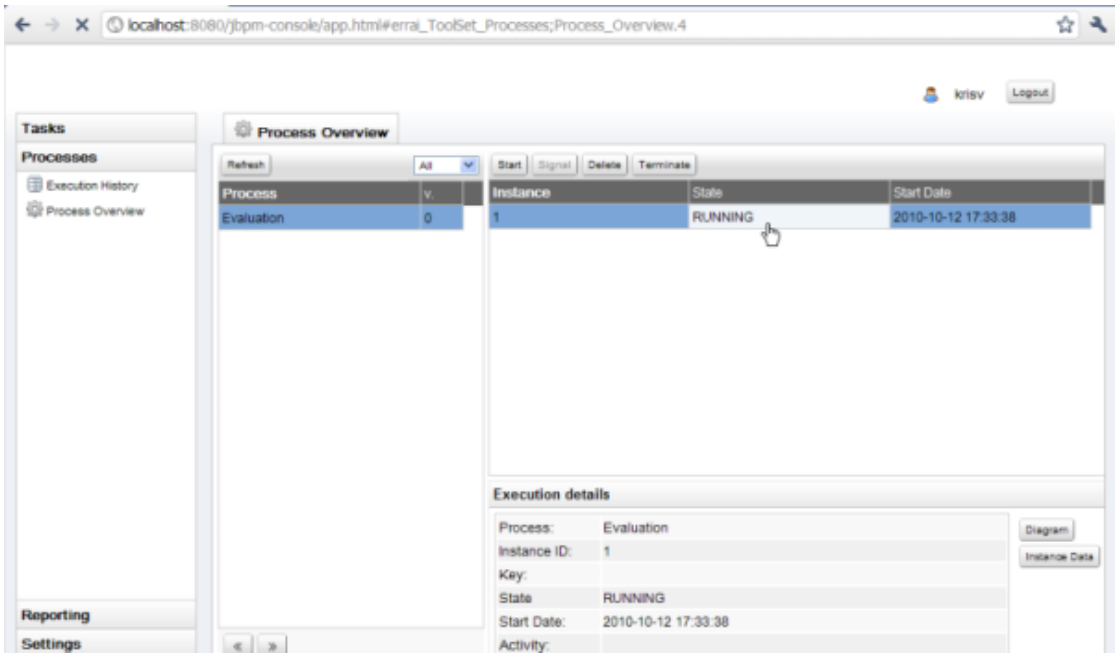


图 3.2 [http://people.redhat.com/kverlaen/install-gwt-console-jbpm.swf]

- 管理你的流程实例，单击”Processes”标签，在选择”Process Overview”的左边。在一个轻微额延迟之后（如果你正在使用你的应用在第一时间，由于会话的初始化），“process“列表应该展现所有已知的流程。在演示程序的jbpm控制台的当前链接的所有流程都存储在"jbpm-installer/sample/evaluation"路径中的“evaluation”项目中的"src/main/resources"路径中。你点击流程，它将会显示出所有当前的运行实例。当然，如果在这一点没有运行的实例，那么实例表中就是空的。

- 你能够通过单击“start”按钮来启动一个新实例。当你确认有这个新的流程已经执行启动，那么你就可以通过填写必要的信息来启动一个流程。这种情况你需要填写用户名“krisv”，在你完成之后就可以关闭窗口了。一个新的实例应该被展现在“instance”列表中。乳沟你单击流程实例，你能够在下面查看它的细节、图形和实例数据通过分别点击“Diagram“和”Instance Data”按钮。你刚刚启动的流程实例将会在第一时间要求用户的处理并一直等待到用户完成这个任务。
- 通过选择左边“Tasks”标签并 选择“Personal Tasks”你就可以查看分配给你的任务。个人任务表将会展现出你要完成的处理。你能够通过选择并单击“View”按钮来完成这个任务。为了完成处理将会打开一个任务表。你要填写必要的数据并完成整个表，然后关闭表。在完成任务之后，你可以查看“Process Overview”并进一步查看你的流程实例的进展。你应该能够看到有一个流程正在等待你的ＨＲ经理和项目经理去执行任务。你可以通过"john" / "john"和"mary" / "mary"来记录已经完成的任务。
- 在你启动并正在完成一些流程实例和人工任务之后，你能够产生一个到目前为止所有事件的报告。在“Reporting“下选择"Report Templates"。控制台默认的有两个报告模板，一个是产生所有流程的一般的概述，一个是为了监控曾经特有流程的定义。如果你选择了后者，你要确认在用流程定义的ID去查看活动的处理流程时要输入"com.sample.evaluation"。点击"Create Report”按钮来产生一个当前状态的实时报告。注意报告的初始化可能会花一点时间，特别是在你第一次使用你的应用时。

3.6 10 分钟指南：使用 Guvnor 知识库和设计器

Guvnor 知识库可以被用来作为一个流程知识库来存储业务流程。当然它也提供一个基于 Web 的接口来管理你的流程。这包括基于 Web 的编辑器来查看和编辑流程。

打开 Drools Guvnor: *http://localhost:8080/drools-guvnor*

你可以使用任何非空的用户名密码来登录。下面的录屏教你如何去管理你的知识库: [http://people.redhat.com/kverlaen/install-guvnor-jbpm.swf]，它会告诉你：

- 如何从 Eclipse 中导入一个已经存在的流程到 Guvnor 中
- 怎样在一个 Web 编辑器中打开你的处理流程
- 怎样创建一个用来创建会话的包

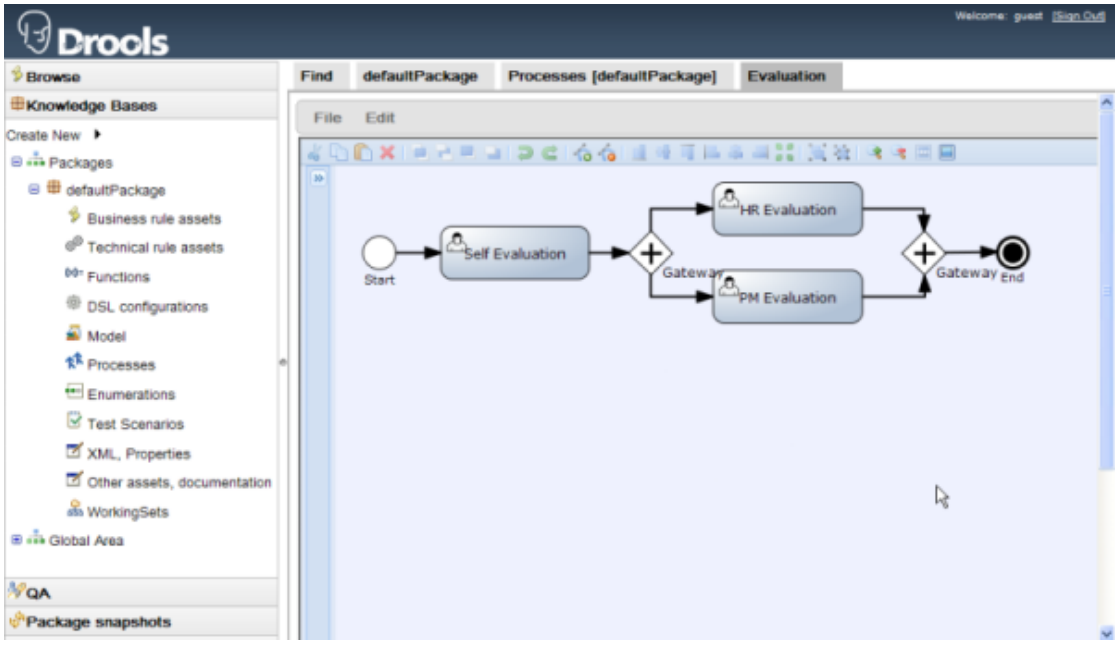


图 3.3 [http://people.redhat.com/kverlaen/install-guvnor-jbpm.swf]

如果你想了解更多，我们建议你去看一下 Drools Guvnor 文档。

你曾经已经做过的： ant stop.demo 并很容易关闭所有的 rest（不知怎么翻译好）相关线程吧！！！

3.7 如果遇到问题或有问题应该怎么去做

若有安装问题，可这样子：

Email: jbpm-dev@lists.jboss.org

IRC: #jbpm at irc.codehaus.org

jBPM 用户论坛

http://community.jboss.org/en/jbpm?view=discussions

FAQ：

1. 脚本不能下载组件？

检查网络，或直接手动下载后放到 jbpm-installer/lib 文件夹。

2. 下载的组件不能解压？

文件有问题，重新下载。

3.7 经常被问的问题

一下常见问题的解释：

问题：如果 installer 不能安装某个组件？

回答：你已经连接到因特网了吗？你的防火墙开着吗？你是否需要一个代理？这些都有可能当我们正在下载组件时暂时离线。试着手动下载组件，如何把他们放到 jbpm-installer/lib 文件夹下面。

问题：installer 不能够解压缩包含的压缩文件？

回答：当下载组件时下载失败，可能是 installer 正在尝试使用一个未完成的文件。尝试在 jbpm-installer/lib 文件下删除问题组件并重新安装，它将会重新下载。

问题：如果我已经更改了我的安装配置并向重新再一次通过清理原来的配置来启动，该怎么做？

回答：你可以用 ant clean.demo 来移除所有安装的组件，结束后你再次刷新一下安装配置就可以了。

问题：当在尝试停止或重启确定服务时有时候会看到异常，我应该怎么做？

回答：如果你在关闭的时候看到错误，你要确定是否还有服务在运行。如果在重启时看到异常，你要确定先前启动的服务是否被成功的关闭。如果有必要的话你可以尝试强制关闭所有的服务。

问题： 在运行 Eclipse 的时候似乎有些异常但不知怎么去做？

回答：首先查看控制台的输出，例如错误信息或者栈回溯。你也可以检查 Eclipse 的错误日志。尝试添加一个审查日志来记录你的会话信息来描绘在运行时或者在尝试 debug 你的应用时所发生的事件。

问题：在运行基于 Web 的应用时似乎有些错误，例如 jbpmconsole, Guvnor and the Designer

回答：你可以在目录 jbpm-installer/jboss-4.2.3.GA/server/default/log 查看一下服务日志来看一下可能的异常。

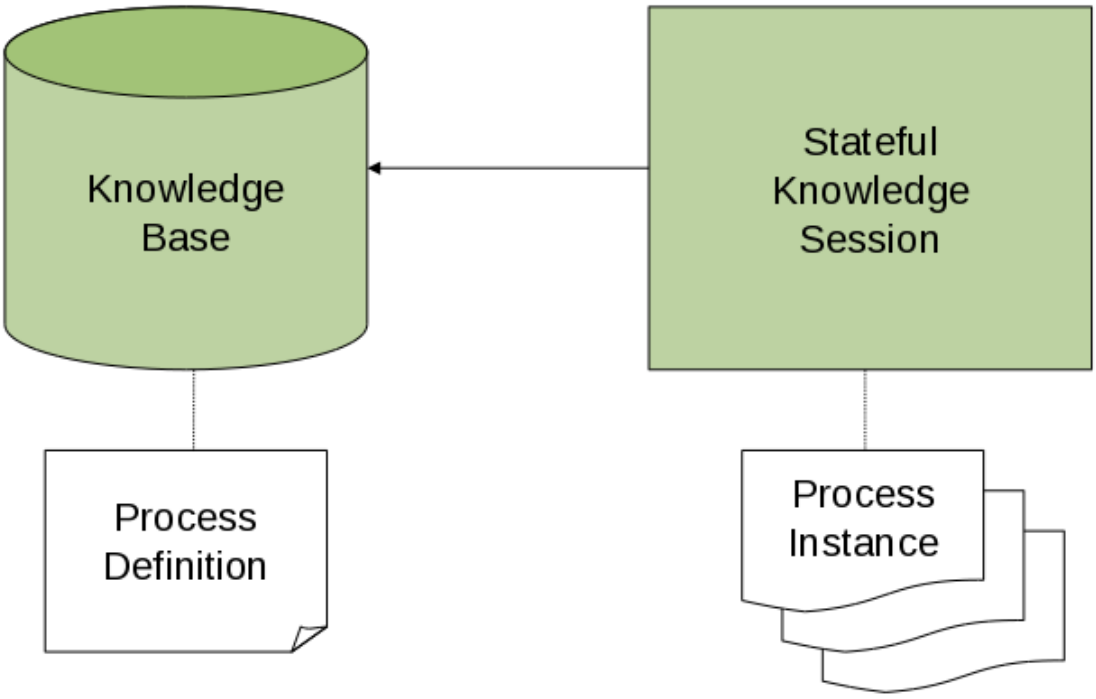
对于其他问题，尝试练习 jBPM 社区。

第四章：核心引擎的 API

这章将会介绍在你需要连接和执行你的流程是所要用到的 API。怎样去定义流程本身的更多细节可以去查看 BPMN 2.0 章节。

和流程引擎的链接，你需要去建立一个会话。这个会话用来和流程引擎联系。会话需要有一个从 knowledge base 引用，包含了所有相关流程定义的引用。这个 knowledge base 是用来在任何有必要的时候来查看流程的定义。要去创建一个会话，你首先需要去创建一个 knowledge base，链接到所有的必需已定义的流程，然后初始化一个会话。

一旦你建立了一个会话，你就可以用它来执行流程了。一个流程无论什么时候被启动，一个新的流程实例都将被创建，并维持特有的流程实例的状态。



例如，想象一下你正在写一个销售订单的流程应用。你可以定义一个或多个流程，定义这些订单应该怎样去执行。在启动你对应用时，你首先需要创建一个包含了所有已经定义了的流程的 knowledge base。然后你就可以在基于 knowledge base 的基础上建立一个会话，当你哥新的销售订单进入的时候，一个与销售订单对应的新的流程实例也将启动。流程实例包含了特定销售需求的流程状态。

一个 knowledge base 能够通过会话来共享并且通常只需要在应用启动时创建一次。knowledge base 能够动态的更改。

基于 knowledge base 被创建的会话被用来执行流程和与引擎互动。你可以按你想的创建很多独立的会话，创建一个会话是被认为是轻量级的。创建多少会话全都依你而定。总的来说，许多简单的情况只需要建立一个会话，然后可以在你应用的其他地方调用。如果你想有多个独立的流程单元，你可以决定创建多个会话，或者由于你出于可监控的原因可以建立多个会话。如果你不知道怎么去做，可以通过建立一个包含所有流程定义的 knowledge base 来开始，然后创建一个会话，然后你就可以用来执行你的流程了。

4.1 jBPM API

jBPM 项目和 API 直接由很清晰的区别，应该和真正的实现类来相结合。对于可用的公共 API 的众多特性，我们相信普通的用户能够安全的使用并且应该保持使用相对稳定的版本。专家用户仍然能够访问内部类，但是要清楚他们正在做什么，并且内部 API 在将来仍然可能改变。

以上所做的解释，jBPM 应该被用来创建一个 knowledge base，其包含你的流程定义，再就是创建一个用来启动新的流程实例、单独存在的，或者注册监听器等等的会话。

4.11 Knowledge Base

jBPM 允许你先创建一个 knowledge base。这个 knowledge base 应该包含你所有的流程定义，就是需要被会话执行的流程。创建一个 knowledge base，先用 knowledge builder 来连接不同资源的流程，然后从 builder 创建一个新的 knowledge base。下面的代码片段展示了怎样去创建一个 knowledge base，这只包含一个流程定义。

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add(ResourceFactory.newClassPathResource("MyProcess.bpmn"),
ResourceType.BPMN2);
KnowledgeBase kbase = kbuilder.newKnowledgeBase();
```

对于从文件系统连接文件、URL、输入流，阅读器等链接和 ResourceFactory 的方法相似。

4.12 会话

一旦你装载了 knowledge base，你应该创建一个会话来和引擎结合。然后会话就能够被来启动一个流程或单独的事件。下面的代码片段展现了基于先前创建的 knowledge base 来创建一个会话是多么的容易，并可以通过 id 来启动流程。

```
StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
ProcessInstance processInstance = ksession.startProcess("com.sample.MyProcess");
```

流程运行时接口定义了会话的所有方法来处理流程，就像下面展示的一样。

```
/**
 * 启动一个新的流程实例。 流程通过流程的id引用来获得流程
 * * @param processId 被启动的流程id
 * @return the ProcessInstance 被启动的流程实例对象
*/
ProcessInstance startProcess(String processId);
/**
 * Start a new process instance. The process (definition) that should
 * be used is referenced by the given process id. Parameters can be passed
 * to the process instance (as name-value pairs), and these will be set
 * as variables of the process instance.
```

```
* @return the ProcessInstance that represents the instance of the process that wasstarted
*/

ProcessInstance startProcess(String processId, Map<String, Object> parameters);

/**
* Signals the engine that an event has occurred. The type parameter defines
* which type of event and the event parameter can contain additional information
* related to the event. All process instances that are listening to this type
* of (external) event will be notified. For performance reasons, this type of event
* signaling should only be used if one process instance should be able to notify
* other process instances. For internal event within one process instance, use the
* signalEvent method that also include the processInstanceId of the process instance
* in question.
*
* @param type the type of event
* @param event the data associated with this event
*/
void signalEvent(String type,Object event);
/**
* Signals the process instance that an event has occurred. The type parameter defines
* which type of event and the event parameter can contain additional information
* related to the event. All node instances inside the given process instance that
* are listening to this type of (internal) event will be notified. Note that the event
* will only be processed inside the given process instance. All other process instances
* waiting for this type of event will not be notified.
*
* @param type the type of event
* @param event the data associated with this event
* @param processInstanceId the id of the process instance that should be signaled
*/
void signalEvent(String type,Object event,long processInstanceId);
/**
* Returns a collection of currently active process instances. Note that only process
* instances that are currently loaded and active inside the engine will be returned.
* When using persistence, it is likely not all running process instances will be loaded
* as their state will be stored persistently. It is recommended not to use this
* method to collect information about the state of your process instances but to use
* a history log for that purpose.
*
* @return a collection of process instances currently active in the session
*/
Collection<ProcessInstance> getProcessInstances();
```

```
/**
* Returns the process instance with the given id. Note that only active process instances
* will be returned. If a process instance has been completed already, this method will return
* null.
*
* @param id the id of the process instance
* @return the process instance with the given id or null if it cannot be found
*/
ProcessInstance getProcessInstance(long processInstanceId);
/**
* Aborts the process instance with the given id. If the process instance has been completed
* (or aborted), or the process instance cannot be found, this method will throw an
* IllegalArgumentException.
*
* @param id the id of the process instance
*/
void abortProcessInstance(long processInstanceId);

/**
* Returns the WorkItemManager related to this session. This can be used to
* register new WorkItemHandlers or to complete (or abort) WorkItems.
*
* @return the WorkItemManager related to this session
*/
WorkItemManager getWorkItemManager();
```

4.13 事件

会话提供了监听器的注册和删除的方法，一个流程事件监听器可以被用来监听和流程相关的事件，比如开始或完成一个流程，进入或离开一个节点等等。下面是流程事件监听器类的各种方法的展示。一个事件对象提供相关信息的访问，比如流程实例和与事件链接的节点实例。你能够使用这个 API 来注册你自己的事件监听器。

```
/**
* Returns the WorkItemManager related to this session. This can be used to
* register new WorkItemHandlers or to complete (or abort) WorkItems.
*
* @return the WorkItemManager related to this session
*/
WorkItemManager getWorkItemManager();

void afterProcessCompleted( ProcessCompletedEvent event );
void beforeNodeTriggered( ProcessNodeTriggeredEvent event );
```

```
void afterNodeTriggered( ProcessNodeTriggeredEvent event );
void beforeNodeLeft( ProcessNodeLeftEvent event );
void afterNodeLeft( ProcessNodeLeftEvent event );
void beforeVariableChanged(ProcessVariableChangedEvent event);
void afterVariableChanged(ProcessVariableChangedEvent event);
```

jBPM out-of-the-box 提供了一个用来创建审查日志的监听器。这个审查日志包含所有在运行时所发生的不同事件。它是很容描绘发生了什么。注意：这些日志仅仅被用来作为调试的目的。下面的日志执行已经被默认支持：

1. 控制台记录器：记录器将多有事件都输出到控制台上
2. 文件记录器：将多有的事件写入到文件或者 XML 格式的文件，这个日志文件可能在集成开发环境中被用到，在执行过程中产生一个基于树的可视化的事件发生记录。
3. 线程文件记录器：当关闭记录器或者在记录器重一系列事件到达一个提起定义的等级的时候，文件记录器才会将事件写到硬盘上。在运行时不能够被用来调试流程。线程文件记录器将会在每隔一个特定的时间就会输出事件，它使得可以实时的用日志来查看流程成为可能。

KnowledgeRuntimeLoggerFactory 可以让你添加一个日志记录器到你的会话中，就像下面展现的一样。当创建一个控制台记录器的时候，由于记录器的需要要创建一个 knowledge session，并且必须要作为一个变量来传递。文件记录器需要日志的文件名来创建，而线程文件记录器是需要每隔一段时间就把事件保存起来。你一直应该在结束你的应用的时候关闭记录器。

```
KnowledgeRuntimeLogger logger =
KnowledgeRuntimeLoggerFactory.newFileLogger( ksession, "test" );
// add invocations to the process engine here,
// e.g. ksession.startProcess(processId);
...
logger.close();
```

日志文件时被基于文件的包含基于 XML 预览记录器创建来记录在运行时所发生的所有事件。它可以在 Eclipse 中打开，在 Drools Eclipse Plugin 中使用审查视图。事件都是以树形来展示。事件的发生前后，事件是作为事件的孩子被展示。下面的截屏是一个简单的例子，这是一个流程的启动，导致了开始节点的激活，一个动作节点和一个结束节点，当流程完成之后。

```
RuleFlow started: ruleflow[com.sample.ruleflow]
  RuleFlow node triggered: Start in process ruleflow[com.sample.ruleflow]
    RuleFlow node triggered: Hello in process ruleflow[com.sample.ruleflow]
      RuleFlow node triggered: End in process ruleflow[com.sample.ruleflow]
        RuleFlow completed: ruleflow[com.sample.ruleflow]
```

4.2. Knowledge-based API

你可能已经注意到了，jBPM 项目所暴露的 API 就是 knowledge API。那就意味着不仅仅要关注流程，潜在的其他 knowledge 类型也要允许被加载。这个对于用户的影响几乎是非常小的。这就意味着代替了 ProcessBase 或 ProcessSession，你正在使用的是 KnowledgeBase 和一个 KnowledgeSession。

无论如何，如果你曾经打算使用业务规则或者复杂的事件流程来作为你应用的一部分，，那么knowledge-based API允许用户可以添加不同类型的资源，例如流程和规则，几乎是相同的方式引入到同一个knowledge base。这就要求一个用户指定如何去使用jBPM来启动，Drools Expert (for business rules) or Drools Fusion (for event processing) 几乎是瞬时的在，作为API和工具对于不同类型的knowledge是统一的。

第五章 核心引擎：基础

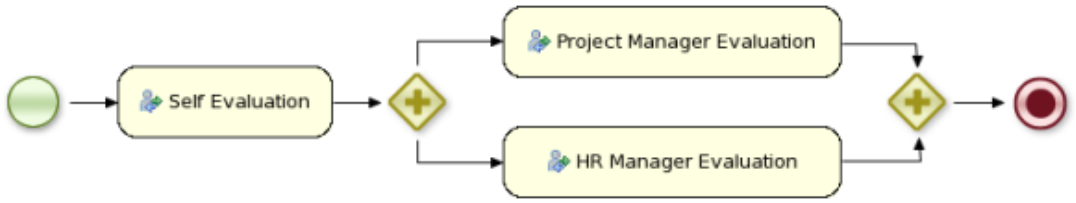


图 5.1

业务流程是一张流程图表，描述一系列执行步骤的顺序。它由多个结点组成，结点之间互相连接。一个流程是由节点的集合组成，节点通过连续链接。箭头代表结点之间的连接，它指定如何从一个结点流向另外一个结点。提起定义的节点类型的大部分选择都已经被定义。这章主要描述了怎样去定义这样的流程和如何在你的应用中使用它们。

5.1 创建一个流程

流程的创建可以通过下面三种方式：

1. Eclipse 图形编辑器插件

2. 使用 XML 文件，根据已经定义了的符合 BPMN2.0 规范 XML 格式定义的 XML 流程格式。

3. 通过流程 API 直接创建

5.1.1 Eclipse BPMN2.0 图形编辑器插件

Eclipse BPMN2.0 图形编辑器允许你通过拖拽不同的节点到面板上并编辑这些节点的属性的方式来创建一个流程。Eclipse BPMN2.0 图形编辑器是 jBPM / Drools Eclipse 插件的一部分。如果你已经建立了一个 jBPM 项目，你就可以开始添加流程了。通过新建文件来选择新建一个流程文件，然后就可以用流程编辑器打开。

首先，你要确保你能够在 Eclipse 窗口的属性视图中看到流程的属性，在你流程中的元素不同的属性需要填写时必须的。如果你不能看到属性视图，点击“Window”，然后选择"Show View" and "Other...",，然后再“General”文件夹下选择属性视图（Properties View）。

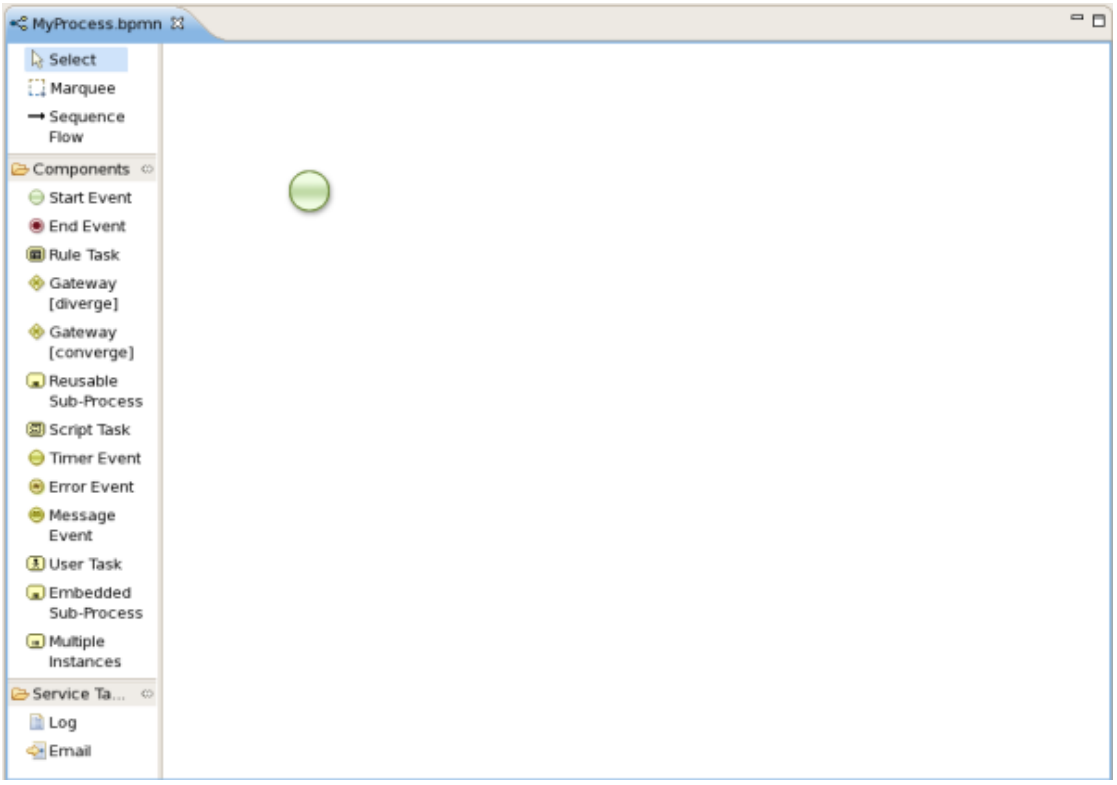


图 5.2 新流程

流程编辑器的组成部分就不翻译了，插件已经安装好的，直接新建一个 流程就 OK 了。

5.1.2 使用 XML 定义流程

使用符合 BPMN 2.0 规范的 XML 来直接指定流程页是可能的。XML 流程的语法已经在 BPMN 2.0 XML Schema 中定义了。例如，下面的 XML 片段展示了一个简单的流程，这个

流程包含一个开始事件、一个脚本任务在控制台打印“Hello World”，和一个结束事件的次序。

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions id="Definition"
targetNamespace="http://www.jboss.org/drools"
typeLanguage="http://www.java.com/javaTypes"
expressionLanguage=http://www.mvel.org/2.0

xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"Rule Task
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL BPMN20.xsd"
xmlns:g="http://www.jboss.org/drools/flow/gpd"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:tns="http://www.jboss.org/drools">
<process processType="Private" isExecutable="true" id="com.sample.hello"
name="Hello
Process" >
<!-- nodes -->
<startEvent id="_1" name="Start" />
<scriptTask id="_2" name="Hello" >
<script>System.out.println("Hello World");</script>
</scriptTask>
<endEvent id="_3" name="End" >
<terminateEventDefinition/>
</endEvent>

<!-- connections -->
<sequenceFlow id="_1-_2" sourceRef="_1" targetRef="_2" />
<sequenceFlow id="_2-_3" sourceRef="_2" targetRef="_3" />
</process>
<bpmndi:BPMNDiagram>
<bpmndi:BPMNPlane bpmnElement="com.sample.hello" >
<bpmndi:BPMNShape bpmnElement="_1" >
<dc:Bounds x="16" y="16" width="48" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="_2" >
<dc:Bounds x="96" y="16" width="80" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="_3" >
<dc:Bounds x="208" y="16" width="48" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNEdge bpmnElement="_1-_2" >
<di:waypoint x="40" y="40" />
```

```
<di:waypoint x="136" y="40" />
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdge bpmnElement="_2-_3" >
<di:waypoint x="136" y="40" />

<di:waypoint x="232" y="40" />
</bpmndi:BPMNEdge>
</bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
</definitions>
```

XML 流程文件的结构不做翻译，具体的请看上面的例子。

5.1.3 使用流程 API 定义流程

推荐使用图像编辑器或者在 XML 下定义流程，当然也可以使用 API 直接定义。很多重要的流程模型元素已经定义在包：org.jbpm.workflow.core and org.jbpm.workflow.core.node.中。灵活的 API 允许你使用工厂的易读方式很容易的构建一个流程。最后，你需要确认你手动构建的流程。

5.1.3.1 例子

这是一个带有脚本任务的基本流程的简单示例：

```
RuleFlowProcessFactory factory =
RuleFlowProcessFactory.createProcess("org.jbpm.HelloWorld");
factory
// Header
.name("HelloWorldProcess")
.version("1.0")
.packageName("org.jbpm")
// Nodes
.startNode(1).name("Start").done()
.actionNode(2).name("Action")
.action("java", "System.out.println(\"Hello World\");").done()
.endNode(3).name("End").done()
// Connections

.connection(1, 2)
.connection(2, 3);
RuleFlowProcess process = factory.validate().getProcess();
```

你可以看到我们可以通过从 RuleFlowProcessFactory 类调用 creatProcess()静态方法来开始。这个方法根据给出的 id 创建一个新的流程，并返回一个用来创建流程的 RuleFlowProcessFactory。一个典型的流程由三部分组成。头部由像流程的名称，引入

的变量等全局元素组成。包含了所有不同节点的节点片段组成了流程。连线片段最终链接每一个节点来创建一个流程图。

在这个例子中，头部包含了流程的名称、版本和包的名称。之后，你就可以在当前的流程中添加节点了。如果你已经自动完成了，你能够看到你有不同的方法来创建每一个你已经配置的支持的节点类型。

在例子中你通过调用 startNode(),actionNode()和 endNode()方法在流程中添加节点，这些方法将返回一个特定的 NodeFactory，允许你设定节点属性。一旦你配置完节点，done()方法返回当前的一个 RuleProcessFactory，如果有必要，你可以添加更多的节点。

当你在添加节点之后，你必须通过创建连线来连接它们。这样就可以被 connection 方法调用先前创建的节点。

最后，你能够通过调用 validate()方法来验证产生流程，并纠正创建的 RuleFlowProcess 对象。

5.2 不同流程构造的细节概述

接下来的章节将会描述在你建模你的流程细节的时候会用到的不同构造。在 BPMN 中执行的流程由用有次序的流程连接的不同类型的节点组成。BPMN2.0 规范定义了三种主要的节点类型：

- Events：是用来建模特定事件的发生。事件有开始事件，结束事件，中间事件。
- Activities：这些动作定义了在流程执行过程当中需要被执行的不同动作。不同类型的任务是要依赖于你正在建模的动作的类型，并且动作是可以嵌套的。
- Gateways：你能够在流程中定义多个路径。依赖 gateway 的类型，这可以是指示并行的执行、选择等待。

下面的片段将会描述本身的属性和每一个不同节点类型的细节，作为 Eclipse 插件所支持的并在流程图中显示出来。Eclipse 属性编辑器可以显示更多的所支持的节点类型的属性，但是，只有那些在使用 BPMN2.0 XML 格式被支持的定义在片段当中的属性才可以被显示。

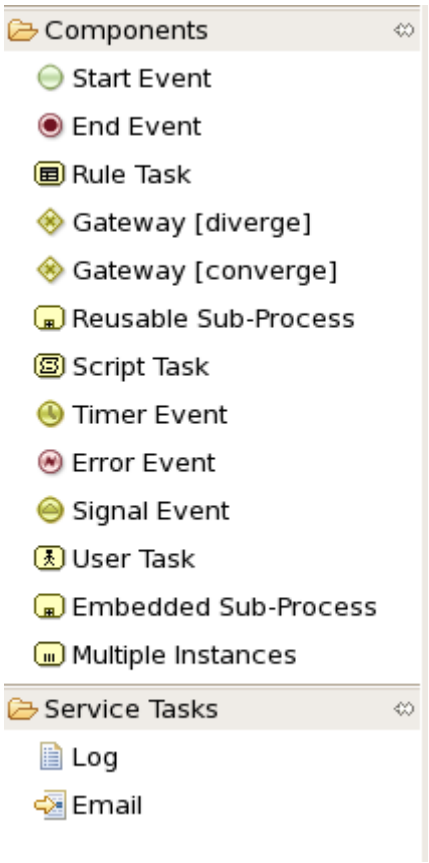


图 5.3 不同类型的 BPMN2 节点

5.3 流程属性的细节

BPMN2 流程图式就是使用连线将不同类型的节点链接起来。流程本书由以下属性：

•id：流程的唯一标识

•Name：流程的显示名称

•Version：流程的版本

•Package：流程被定义所在的包

•Variables：在流程执行过程中变量可以被定义用来存储数据，请看数据细节片段。

•Swimlanes：暂翻译为泳道。在流程中指定的泳道用来分配人工任务。查看具体细节“Human Tasks”章节

5.4 事件细节

5.4.1 开始事件

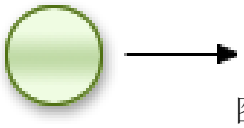


图 5.4 开始事件

开始事件是流程的开始，一个流程应该有一个正确的开始节点，这个节点不能够有进入链接，并且应该有一个输出链接。一个流程无论在什么时候被启动、执行，都将在开始节点开始并自动从第一个节点链接开始事件等等。它包含以下的属性：

•Id：节点的 Id

•Name：节点的展示名字

5.4.2 结束事件

5.4.2.1 结束事件

图 5.5 结束事件

代表流程的结束。一个流程应该有一个或者多个结束事件。一个结束事件应该有一个进入链接二没有输出链接。它包含以下属性：

•Id：节点 ID

•Name：节点的名称

•Terminate：结束事件可以终止整个流程也可以是一个路径。当一个流程实例被终止的时候，就意味着流程的状态被设为完成，而流程实例中所有活动的节点都将被删除。没有终结结束事件只是简单的结束一个路径，其他的并行路径仍然能够继续。一个流程实例在路径中如果没有动作，那么这个流程会自动完成。终止结束事件可以通过事件节点的内部循环看成它的作用，而非终止结束事件则是没有作用。注意：如果你在一个子流程中使用了终止结束事件，你将终止高一级的流程实例，而不仅仅是子流程。

5.5.2.2 抛出错误事件

图 5.6 抛出错误事件

一个错误事件可以在流程中发出一个异常信号。它应该有一个输入连接而没有输出链接。在流程中当一个错误事件发生的时候，它将抛出一个错误并给出错误名称。而流程将会搜索能够使得解决这种错误的处理方法。如果没有错误处理被发现，流程实例将会被终止。一个错误事件包括了以下属性：

•Id：节点 Id

•Name：节点的名称

•FaultName：错误名称。这个名称用来搜索合适的能够处理这种错误能力的处理方法。

错误处理能够被指定用来处理边界事件。现在只能通过在 XML 当中直接来实现。我们将在新的 BPMN2 编辑器中提供图形编辑的支持。

5.4.3 中间事件

5.4.3.1 获取定时器事件

图 5.7 获取定时器事件

代表着在给出一段时间后定时器会引发一次或多次。一个计时器事件应该有一个输入连接和一个输出链接。计时器延迟指定了应该等待多长时间之后第一时间引发。在流程中，当一个计时器事件发生的时候，流程将会和计时器相关联。如果计时器节点被取消，那么计时器也将被取消。参考“Timers”可以查看更多的信息。计时器事件包含以下属性：

•Id：节点 Id

•Name：节点的名称

•Timer delay：在节点被第一次触发之前节点要等待的时间。其格式应该为[#d][#h][#m][#s][#[ms]]。这就意味着你可以指定天，小时，分钟，秒，毫秒的大小。例如，“1h”代表在触发计时器之前等待一个小时。也可以通过使用#{}动态设定基于一些流程变量的延迟。这可以是一个流程变量，或者是一个基于流程变量的复杂表达，例如(e.g. myVariable.getValue())。

•Timer period：计时器间隔，或者译为时期。就是在两个子序列引发之间的那段时期。如果间隔为 0，那么计时器就会被立刻触发。其格式也应该是[#d][#h][#m][#s][#[ms]]。

5.4.3.2 获取信号事件

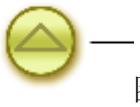


图 5.8 获取信号事件

在流程的执行当中，一个信号事件可以用来作为对内部或外部事件的回应。它指定了预期事件的类型。无论这个事件类型什么时候被探测到，与它相连接的事件节点都将被触发。它包含以下属性：

•Id：节点的 Id

•Name：节点的名称

•EventType：所期望的节点的类型

•variableName：当这个事件发生的时候，变量名称就会和这个事件的数据相关联。

流程实例可以定义一个特定事件的信号事件通过使用：ksession.signalEvent(eventType, data, processInstanceId)。

在等待的事件类型的流程实例中，所有的事件节点都将被触发。如果事件节点指定了一个变量名称，当事件发生的时候这个数据将会被复制到变量中。

当然，也可以用在子流程的内部事件节点中。这些事件节点在子流程被激活的时候也会被激活。

你能够在流程实例的内部产生一个信号。一个脚本能够使用：
kcontext.getKnowledgeRuntime().signalEvent(
eventType, data, kcontext.getProcessInstance().getId());

抛出一个信号事件也可以用来模拟一个事件的信号。不过这个现在也只能直接在 XML 中实现。我们在新的 BPMN2 编辑器重将会添加图形编辑的支持。

5.5 动作细节

5.5.1 脚本任务

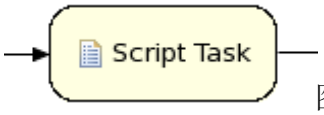


图 5.9 脚本任务

代表一个脚本在流程中应该被执行。一个脚本任务应该有一个输入链接和一个输出链接。相关联的动作应该指定什么要被执行，还有编码动作的方言，和正在的动作代码。这段代码可以访问任何变量和全局参数。这还有一个提前定义的变量 kcontext 从 ProcessContext 对象引入。在流程中，当一个脚本任务被访问时，它将会执行动作，然后继续到下一个节点。它包含以下节点：

•Id：节点的 Id

•Name：节点的名称

•Action：与这个动作节点相关联的动作脚本

注意：你可以在脚本代码内部写任何有效地 java 代码。这基本上允许你在脚本代码内做任何事。然而，这里也有一些警告：

•当你试着去创建一个高级的业务流程时，你应该确定能够被业务用户所理解，在流程内部避免低级的执行细节将会是明智的。包括在脚本任务的内部。脚本任务仍然可以用

来快速处理变量等。但是像服务任务等其他的任务能够被用来在高级方式中建模更加复杂的行为。

•脚本应该是 immediate（这里不知道翻译成什么好），他们使用引擎线程来执行脚本。脚本的执行可能会花一些时间，它的执行可能会被以异步的服务任务方式来进行。

•你应该尽量避免通过脚本节点和外部服务相关联。这不仅仅违法了前面两个的警告，它将会影响除了 knowledge 引擎之外的其他外部服务，这将会产生问题，特别是在持久化和转换时。总的来说，使用服务任务来和外部的服务相关联是相当明智的。

•脚本不应该抛出异常。运行时异常应该被捕获，例如在脚本内部进行管理，或者转换成信号，或者产生的错误能够在流程内部处理。

5.5.3 服务任务

图 5.1. 服务任务

代表一个在流程中应该被执行的工作单元。在流程引擎外部执行的所有的工作都应该使用服务任务来描绘或执行。BPMN 2.0 已经提前定义了不同的服务类型，例如发送邮件服务，消息日志等等。用户可以定义特有域的服务或工作条目，使用唯一的名称并定义所用到的参数，最终的输出和工作类型相关。查看 domain-specific processes 章节你可以获得更多的细节解释和在你的流程中怎样去定义和使用工作条目的说明。一个服务任务应该有一个输入连接和一个输出链接。

•Id：节点的 Id

•Name：节点的名称

•Parameter mapping：参数映射，允许复制流程变量的值到工作条目的参数。在工作条目创建的时候，值就会被复制。

•Result mapping：允许复制工作条目的结果参数的值到流程变量。每一个工作类型都能够定义结果参数，在工作条目完成之后返回参数的值。结果映射就是用来复制结果参数的值到流程的变量。例如，“FileFinder”工作条目返回符合给出的搜索参数的文件列表。这个文件列表然后与流程变量绑定。在完成了工作条目之后，这些值将会被复制。

•On-entry and on-exit：进入、退出动作，当进入虎退出节点的时候分别执行的动作。

•Additional parameter：额外的参数，每一个工作天明能够定义额外的参数，这些参数和工作的类型相关。例如，“Email”工作条目定义的额外参数有“from, To, subject, body”。用户可以直接填写这些参数的值，或者定义一个参数映射，这个映射用来复制流程中变量的值到给出的参数，如果两者都被指定，那么这个映射具有优先权。String 类型的参

数可以用#{expression}来侵入到一个 String 值中。这个值在创建工作条目的时候还可以被找回，并且表达式的值将会被参数调用 toString()返回的结果所代替。表达式也可以是简单的变量名，但是，使用 MVEL 表达可能会更好（`#{person.name.firstname}`）

5.5.3 用户任务

图：5.11 用户服务

流程包含的任务由人工角色来执行。用户任务代表一个有人工角色来执行的最初任务。它应该有一个输入连接和一个输出链接。用户任务可以被用来和 Swimlance 泳道相结合来给同一个角色分配多个人工任务。在 human tasks 章节可以查到更多的细节。一个用户服务就是一个服务节点特有类型。一个用户任务包含以下属性：

•id

•Name

•TaskName：任务名

•Priority：优先权，一个整形数代表人工任务的优先权

•Comment：和人工任务相关的注释

•ActorId：代表执行这个人工任务的角色 Id，一个角色列表可以用逗号分开来指定

•GroupId：代表执行这个人工任务的组，一个组列表可以用逗号分开来指定

•Skipable：指定人工任务十分要跳过去，角色可以决定是否去执行。

•content：和这个任务相关联的数据

•Swimlane:Swimlane（泳道）是人工任务节点的一部分，泳道使分配多个人工任务给相同的角色变得非常容易。查看跳过任务章节可以得到更多关于泳道的使用细节。

•On entry and on exit actions：在进入或退出节点时动作脚本将会分别执行

•Parameter mapping：允许复制流程变量的值到人工任务的参数，人工任务的创建，值就会被复制。

•Rusult mapping:允许复制人工任务参数的结果值到流程变量。在人工任务完成之时，值就会被复制。人工任务有一个结果变量“Result”，它包含人工角色的返回数据。变量 ActorId 包含了真正执行任务的角色 Id。

用户任务应该定义需要执行的任务类型和需要完成它的人。注意：如果有和特定流程实例相关的数据，最终用户应该在执行任务的时候，数据应该作为任务的内容内传递。这个任务不需要访问流程变量。查看人工任务章节活动如何在人工任务和流程实例之间传递数据。

5.5.4 重用的子流程

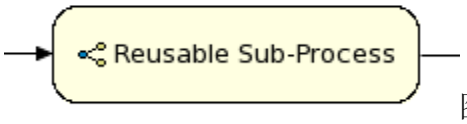


图 5.12 重用的子流程

代表当前流程中的另一个流程的调用。子流程节点应该有一个进入链接和一个退出链接。当流程中一个重用子流程的节点，引擎将会通过 Id 来启动这个流程。它包含以下的属性：

•Id：

•Name

•ProcessId

•Wait for completion：如果这个属性设为 True，那么这个子流程的节点仅仅在子流程启动并完成它的任务时才会继续。不然的话，即设为 False 的时候，节点会在启动流程之后直接执行。

•Independent：如果这个属性设为 True，那么子流程将作为一个独立的流程启动，这意味着即使父流程完成子流程也不会终止。不然当父流程在终止的时候子流程的活动将被取消。

•On-entry and on-exit actions：在进入或退出节点的时候会分别被执行。

•Parameter in/out mapping：一个子流程能够针对变量定义 in-和 out-映射，当流程启动时，在“in”映射中给出的变量将作为参数。当子流程完成的时候，被定义在“out”子流程中的变量将会被复制到这个流程的变量。注意：你只有在“Wait for completion”设为 true 的时候“out”映射才能使用。

5.5.5 业务规则任务

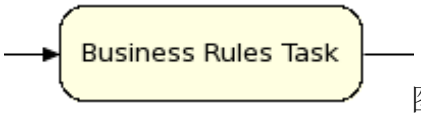


图 5.13 业务规则任务

代表一组规则需要被“evaluated”（这里不知道怎么翻译好）。当节点被链接到时，规则会被“evaluated”。一个规则任务应该有一个进入链接和一个退出链接。规则使用 DroolS rule formate 被定义在单独的文件中。在规则文件的头部加入 ruleflow-group 特性时规则就成为了指定规则流程组的一部分。在流程中，当一个规则任务到达时，引擎将

会执行相应的规则流程组的部分规则。如果在规则流程组中没有其他的动作规则，将会自动继续执行下一个节点。这就意味着在一个规则流程组的执行过程中，由于日程的更改，一个属于当前活动规则流程组的一个新的动作有可能会通过其他的规则被添加到这个日程中成为事实。注意：如果一个流程有一个规则流程组，而在流程组内没有动作规则，那么流程将会直接进入下一个节点。如果规则流程组已经被激活，规则流程组将会保持激活状态并执行，只有当在规则流程组中所有激活的规则被执行完才会继续向下执行。它包含以下属性：

•id

•Name

•RuleFlowGroup：规则流程组的名字，代表这个规则流程组节点的一组规则。

5.5.6 嵌入子流程

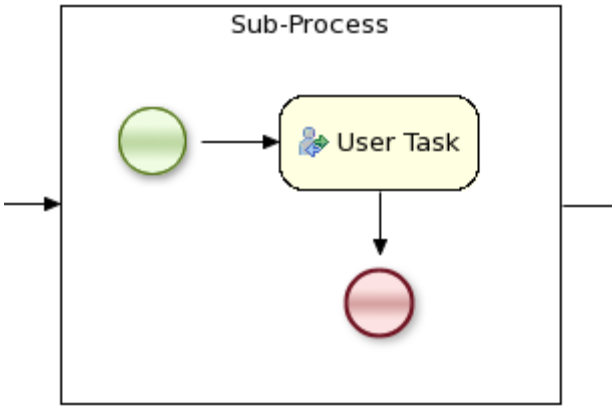


图 5.14 嵌入子流程

一个子流程就是一个能够包含其他节点的节点，事实上讲就是一个节点容器。这不仅仅允许在这样一个子流程节点嵌入流程的一部分，而且可以添加容器内所有节点都可以访问的变量。一个子流程应该有一个进入链接和一个退出链接。它也应该包含一个开始节点，这个开始节点定义了在进入子流程时应该从哪里启动。它当然也应该包含一个或多个结束事件。注意：如果你在子流程内部使用了终止节点，你将终止高一级的流程实例，而不仅仅是子流程，所有，你应该在子流程内使用非终止节点。在子流程中如果没有活动节点存在，那么这个子流程将会结束。它包含以下属性：

•id

•Name

•Variables:在这个节点执行中，可以添加变量来存储数据。详细内容参见“Data”。

5.5.7 多个实例子流程

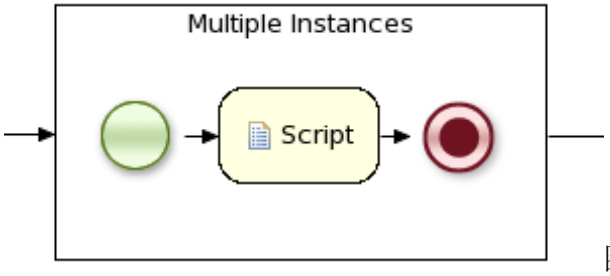


图 5.15 多实例子流程

一个多实例子流程是一个指定类型的子流程，它允许你多次执行所包含的流程片段，每一个元素都会在集合当中。一个多个实例子流程应该有一个进入链接和一个退出链接。在继续执行之前，要等嵌入的流程片段中给出的集合中每一个元素都被完成才能继续。它包含以下属性：

•id

•Name

•CollectionExpression：一个变量名称，代表着应该被迭代结束的元素集合。这个集合变量应该是 java.util.Collection 类型的数组。如果集合表达值为空或为一个空集合，那么多个实例子流程将会直接完成并沿着它的流出链接流出。

•VariableName：变量的名称，包含当前集合的元素。复合节点内（应该就是说容器内）的节点能够访问所愿的元素。

5.6 Gateways 细节

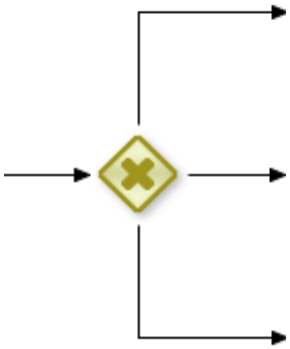


图 5.16 分支网关

允许你在你的流程中创建一个分支。一个分支网关应该有一个进入链接和两个或多个流出链接。当前所支持的有三种类型的网关节点：

•AND or parallel：与或并行，意味着控制流程对流出链接同时进行处理

•XOR or exclusive: 异或，排斥，意味着在流出链接中选择一个正确的。决定是由链接每一个的流程连接的约束作出的。有最低优先权的数字（evaluates to true）被选择。约束

能够用不同的方言来指定。注意：你应该确保在运行时至少有一个流程连接被评估为正确的。

•OR or inclusive：或，包含，意味着在所有的流出链接中，无论谁被评估为true，都将被选择。条件和排斥的网关相似，没有优先权的除外。注意：你应该确保至少有一个流程连接在运行时被评估为true，因为在运行时不能决定一个流出链接流出将会抛出一个异常。

它包含以下属性：

•id

•Name

•Type：分离节点的类型，AND，XOR or OR。

•Constraint：链接每一个流出链接的约束

5.6.2 合并网关

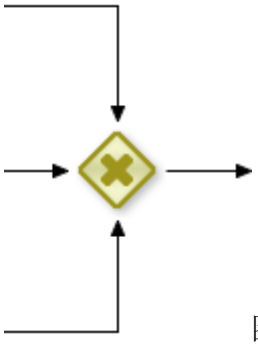


图 5.17 合并网关

允许你同步多个分支。一个合并网关应该有两个或多个进入链接和一个流出链接。当前支持两种类型的分离：

•AND or parallel：在继续执行之前要等所有的进入分支完整

•XOR or exclusive：异或，排斥，意味着只要有一个进入分支完成就可以继续向下进行。如果有多个进入链接被触发，那么它将针对这些引发每一个来引发下一个节点。

它包含以下属性：+

•id

•Name

•Type：联合节点的类型，AND or XOR

5.7 在你的应用中使用流程

在 API 章节已经做了很多的解释，为了能够执行你应用中的流程，你需要做两件事：(1)你需要创建一个 Knowledge Base，要包含一个流程定义；(2)你需要通过创建一个 session 来启动流程，并通过 session 来和流程引擎链接。

1. 创建一个 Knowledge Base：你需要有一个可用的流程，然后添加流程到 Knowledge Base：

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newClassPathResource("MyProcess.bpmn2"),
ResourceType.BPMN2 );
```

在添加所有的流程到 builder 之后，你可以像下面一样创建一个新的 Knowledge base：

```
KnowledgeBase kbase = kbuilder.newKnowledgeBase();
```

注意：如果 knowledge base 包含错误，这将会抛出一个异常。

2. 启动一个流程：启动一个指定的流程，你需要用你的的 session 去调用 startProcess 方法并传递你想要启动的流程的 id。例如：

```
StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();
ksession.startProcess("com.sample.hello");
```

startProcess 方法的参数是需要启动的流程的 id。当定义一个流程时，流程的 id 需要作为流程的一个属性来指定。

当你启动流程时，你可以使用 startProcess(String processId, Map parameters)方法来指定额外的参数来传递流程输入的额外数据。额外的一组参数就是一组名值对。这些参数能够作为流程的高级变量被复制到被新创建的流程实例，所以他们能够直接访问你流程的 remainder（不知怎么翻译）。

5.8 其他特性

5.8.1 数据

虽然流程表关注于指定的流程控制流向，但是从数据的角度来查看流程一般也是很有必要的。一个流程的执行，数据能够获得、存储、传递和使用。

为了存储运行时的数据，在流程的执行过程中，可以使用流程变量。一个变量通过变量名称和数据类型来定义。这可以是一个基本数据类型，例如 boolean、int 或者 String，或者任何的对象子类型。变量能够在变量域内定义。高级的域就是流程的变量域。子域在一个子流程中定义。在子流程中定义的变量只能访问这个域中的节点。

当一个变量被访问时，流程将会在相应的变量域中查找定义的变量。变量域的嵌套是允许的。一个节点在它的父容器中可以一直查找到。如果变量不能够查找到，那么它将会在它的父容器中查找，直到查找到流程实例自身为止。如果一个变量不能被找到，那么读访问会被设置为 null，并且一个写访问会产生一个错误信息，但流程会继续执行。

变量可以以多种方式使用：

•Process-level variables：流程级变量，能够在调用 startProcess 方法通过提供的参数映射来启动流程时被设置。这些参数将会被作为流程域的变量设置。

•Script actions：脚本动作能够直接访问变量，在脚本中作为一个本地参数就可以通过使用变量的名来访问。例如：如果在流程中定义了一个"org.jbpm.Person"类型的变量，在流程的脚本中就可以直接的访问：

```
// call method on the process variable "person"
person.setAge(10);
```

通过 knowledge context 可以直接在脚本中改变变量的值：

```
kcontext.setVariable(variableName, value);
```

•Service tasks：服务任务能够通过映射变量到流出参数来传递流程变量的值到外部（或者另一流程实例）。例如，一个服务任务的参数映射在服务正在被调用时能够定义流程变量 X 的值应该映射到一个任务参数 Y。当然你也可以经流程变量的值注入到一个硬编码的参数 String 类型，使用#{expression}。例如，一个人工任务的描述能够被定义为：#{person.getName()}，当服务需要去呗调用时，可以通过使用实际的 Person 类名来代替这个表达。使用结果映射，一个服务的结果可以相似的被复制回一个变量。

•各种其他的节点也能够访问数据。例如，事件节点可以存储数据到相关联的事件变量中。查看不同类型节点的属性获取更多的信息。

最后，流程和规则都可以访问全局变量，就是在 Knowledge Session 中全局定义变量和数据。在动作中全局变量可以直接访问就像变量一样。在全局变量能够被使用之前，全局变量需要作为流程的一部分被定义。例如，当在 Eclipse 动作属性编辑器中指定一个动作脚本时，你能够通过点击全局按钮来定义全局变量。你也可以在外部使用 ksession.setGlobal(name, value)或从流程内部的脚本使用 kcontext.getKnowledgeRuntime().setGlobal(name,value)来设置全局变量的值。

5.8.2 约束

流程可以用在你流程的各种地方。例如在一个分离网关中。jBPM 支持两种类型的约束：

•Code constraint：代码约束就是布尔表达，直接判断他们什么时候到达。我们现在对于表达这种代码约束支持两种方言：Java 和 MEVL。Java 和 MVEL 代码约束都可以直接的

访问在流程中定义的全局变量和变量。这有一个 java 约束的例子，person 是流程中的一个变量：

```
return person.getAge() > 20;
```

相似的可用 MVEL 代码约束：

```
return person.age > 20;
```

•规则约束等同于正常的 Drools rule 条件。用 Drools Rule 语言的语法来表达可能复杂的约束。就像其他的规则一样，这些规则能够在工作内存中引用数据。当然也可以直接引用全局变量。下面有一个可用的规则约束：

```
Person( age > 20 )
```

这测试了在工作内存中年龄大于 20 的人。

规则约束不能直接访问在流程中定义的变量。但是可以在一个规则约束的内部引用当前流程的实例，这要通过添加流程实例到工作内存，并在规则约束中匹配流程实例。我们已经添加了指定的逻辑来确保 WorkflowProcessInstance 类型的变量 processInstance 在工作内存中仅仅只匹配当前的流程实例而不是其他的流程实例。注意：无论怎么样，在你的流程中使用 Java 代码或者一个 on-entry、on-exit、explicit 动作，插入流程实例到 session，或者是更新都是你自己的责任。下面的规则约束例子将会搜索相同名字的人并最为值存储到流程的变量“name”中：

```
processInstance : WorkflowProcessInstance()

Person( name == ( processInstance.getVariable("name") ) )

# add more constraints here ...
```

5.8.3 动作脚本

动作脚本可以以不同的方式使用：

•在脚本任务中使用

•作为一个进入或退出动作

动作可以访问在流程中被定义的全局变量和变量，和预定义的变量 kcontext.这个变量是 org.drools.runtime.process.ProcessContext 类型的并能够在服务任务中使用：

•得到当前节点的实例。节点实例会查询相关数据，例如它的名称和类型。你也能够取消当前的节点实例。

```
NodeInstance node = kcontext.getNodeInstance();

String name = node.getNodeName();
```

•得到当前的流程实例。一个流程实例能够被查询数据（名称、id、processid 等等），终止或引发一个内部事件。

```
ProcessInstance proc = kcontext.getProcessInstance();

proc.signalEvent( type, eventObject );
```

•得到或设置变量的值

•运行时访问 Knowledge，允许你启动一个流程，发出事件信号，插入数据等等。

jBPM 当前支持两种方言 Java 和 MVEL。Java 动作应该是可用的 Java 代码。MVEL 动作使用业务脚本语言 MVEL 来表达动作。MVEl 可以接受任何的可用 Java 代码，但是要额外提供对参数访问的潜入支持和其他脚本的改进。从而，MVEL 表达对于业务用户来说更为方便。例如，一个打印人的名字的动作：

```
// Java dialect

System.out.println( person.getName() );

// MVEL dialect

System.out.println( person.name );
```

5.8.4 事件

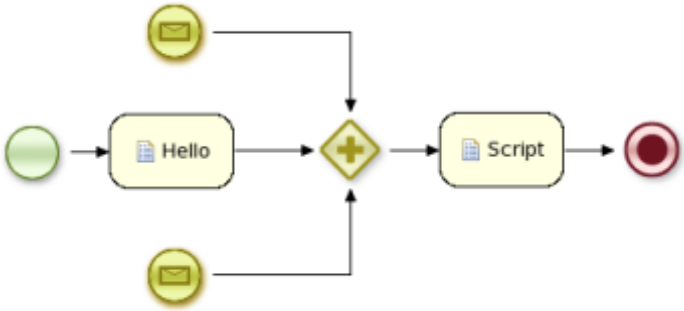


图 5.18 使用事件的简单流程

在一个流程的执行过程中，流程引擎要根据流程计划，通过请求工作条目的的执行并等待结果来确保所有的相关任务被执行。然而，流程应该响应那些不是被流程引擎直接请求的事件。明确的描绘在流程中的事件，允许流程作者去指定对于这种事件流程应该做出如何反应。

事件有一个类型并可能有和他们相关的数据。用户可以自由的定义他们自己的事件类型和他们相关的数据。

一个流程能够通过使用 Message Event 来指定对事件应该如何响应。一个事件节点需要去指定事件的类型。它也能够定义变量的名称，用来接收和事件相关的数据。这就允许在流程中的并发节点去访问事件数据并基于这些数据采取响应的动作。

一个事件可以有一系列的方式向运行中的流程实例发出信号：

•Internal event：在流程中的任何动作都可以发内部事件的发生信号给周围的相关实例，使用以下代码：

```
kcontext.getProcessInstance().signalEvent(type, eventData);
```

•External event：一个流程实例可以被一个外部事件通告，使用代码如下：

```
processInstance.signalEvent(type, eventData);
```

•External event using event correlation:代替了直接通告流程实例，使用基于事件类型的 event correlation 来让引擎自动决定哪一个流程实例相对应哪一个事件。一个包含监听外部某种类型的节点的事件节点的实例，无论事件什么时候发生，都会通告给流程实例，发送事件信号给流程引擎，代码如下：

```
ksession.signalEvent(type, eventData);
```

事件也可以用来启动一个流程。无论什么时候，Message Start Event 定义了一个指定类型的事件触发器，一个新的流程实例将会在事件类型每次发送信号给流程引擎后都会启动。

5.8.5 计时器

计时器是在触发之前等待提起定义的时间量，它允许执行一次或重复执行。计时器能够被用来在一个特定周期之后触发特定的逻辑，或者在特定的间隔内重复某些动作。

计时器提供了一个延迟和一个周期。在触发计时器之前，节点激活之后，延迟指定的等待时间。周期定义的时间在并发触发动作之间。一个周期是在 0 到一小段时间的时间间隔。

period 和 delay 的表达格式应该是[#d][#h][#m][#s][#[ms]]，具体的结束在前面已经翻译。

计时器服务要确保计时器在适当的时间触发。计时器也可以被取消，意味着计时器将不会再被触发。

在流程中计时器有两种使用方式：

•可以在流程当中添加一个计时器事件。它的动作就是在它触发时，一次或者重复的启动计时器，它激活计时器节点的继承者，应该是流程的下一步。这就意味着带有确定时期的流出链接可以被多次触发。取消一个计时器节点也就取消了相关联的计时器，之后就不会再有触发发生了。

•计时器可以作为一个边界事件来和一个子流程相关联。当前只能通过 XML 来直接的设置。在新的 BPMN2 编辑器中将会添加支持。

5.8.6 更新流程

在经过一段之后，流程可以要优化，例如由于流程自身需要改进，又或者因为需求的改变。实际上，你不能够真正的升级一个流程，你仅仅能够配置一个新版本的流程，旧版本的流程将仍然存在。因为已经存在的流程实例可能仍然需要定义好的流程。所以新的流程应该有一个不同的 id，但是名称可以相同。你可以使用版本参数来显示一个流程被升级了。

流程无论什么时候被更新，确定在运行中的流程实例已经发生了什么是非常重要的。有各种策略用来考虑每一个运行中的实例：

- Proceed：运行中的流出实例会按照平常一样继续下去，following the process (definition) as it was defined when the process instance was started（不知怎么翻译，应该是新启动的流程实例是更新之后的流程实例）。这样的结果就是已经运行的流程实例的过程将不会给更新。而新的实例将会使用更新后的流程来启动。

- Abort（and restart）：已经运行的实例将被终止，如果有必要流程实例可以使用新定义的流程来重新启动。

- Transfer：流程实例将会被移植到新定义的流程，这意味着一旦移植成功，流程实例将会基于更新的流程逻辑来继续执行。

默认的，jBPM 使用流程步骤、处理（approach 不知道怎么翻译好），意味着相同流程的多个版本能够被延迟，但是存在的流程实例将会基于旧的定义好的流程来继续执行，在一个新的流程启动的时候。当然，使用流程管理 API 也能够中止运行中的流程实例。流程实例的移植是非常困难的，下面的章节将会做出解释。

5.8.6.1 流程实例的移植

一个流程实例包含所有的运行时信息，流程实例需要及时的在之后的一些点继续执行。这包括所有连接这个流程实例的数据，但是当然也有流程表中的当前状态。对于当

前活动的每一个节点，都是通过节点实例来代表的。节点实例也包含连接到仅仅指定节点的执行的额外状态。对于不同的节点类型有不同的节点实例类型。

一个流程实例仅仅包含运行时状态，并被链接到代表流程逻辑的指定流程，当执行一个流程实例的时候，流程逻辑需要被跟踪。这样，更新一个流程实例到新版本就是使用新的流程逻辑代替旧的流程逻辑，这只需简单的改变从旧的流程 id 到新的流程 id 的索引。

无论如何，这不用考虑需要被移植的流程实例的状态。这种情况下流程仅仅被扩展，所有存在的等待状态被保存。这样就非常的直白了，流程实例的运行状态不需要去改变。当一个存在的等待状态被移除或分成多个等待状态，一个存在的流程实例的等待状态不能够被简单的更新。或者当一个新的流程变量被引进的时候，这个变量需要正确的实例化，以至于它能够在更新的流程的其他部分使用。

WorkflowProcessInstanceUpgrader 能够被用来升级一个工作流程实例到一个新版本的流程实例。当然你需要提供流程实例和新版本流程的 id。默认的，jBPM 会自动的映射旧的节点 id 到新的节点 id。具体还是看例子吧！！

```
// create the session and start the process "com.sample.process"
KnowledgeBuilder kbuilder = ...
StatefulKnowledgeSession ksession = ...
ProcessInstance processInstance = ksession.startProcess("com.sample.process");
// add a new version of the process "com.sample.process2"
kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add(..., ResourceType.BPMN2);
kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
// migrate process instance to new version
Map<String, Long> mapping = new HashMap<String, Long>();
// top level node 2 is mapped to a new node with id 3
mapping.put("2", 3L);
// node 2, which is part of composite node 5, is mapped to a new node with id 4
mapping.put("5.2", 4L);
WorkflowProcessInstanceUpgrader.upgradeProcessInstance(
ksession, processInstance.getId(),
"com.sample.process2", mapping);
```

如果这种类型的映射仍然不奏效，你可以描述你自己的传统映射管理来应当特定的情况。确保首先断开与流程实例的连接，因此改变状态并重新连接流程实例，和 WorkflowProcessinstanceUpgrader 的做法是相似的。

第六章 BPMN 2.0 核心引擎

6.1 Business Process Model and Notation（BPMN）2.0 Specification 业务流程建模和符号 2.0 规范

BPMN 的最初目标就是提供一个能够被所有业务用户容易理解的符号，它来自于流程最初草稿的创建的业务分析，到技术开发人员负责实现能够执行这些流程的技术，最终，对于业务人员能够管理并监视这些流程。

Business Process Model and Notation（BPMN）2.0 Specification 不仅仅定义了怎样去绘画一个流程图的标准的一个 OMG（The Object Management Group 对象管理组织；Open Media Framework Interchange 开发式媒体框架交换，不知道应该是哪一个！！！）规范，而且现在还包括对于元素定义语法的扩展，和一个 XML 格式怎样去存储一个流程定义。

jBPM5 允许你运行用 BPMN 2.0 XML 格式定义的流程。这就意味着你能够使用所有各种不同的 jBPM5 的工具区建模，执行，管理和监视你的业务流程，而这个要执行的业务流程是使用 BPMN 2.0 格式说明的。实际上，完整的 BPMN 2.0 规范也包括怎样去描绘一件事情的细节，例如编排和协助。jBPM 项目仍然是用来说明执行流程规范的一部分。

在 BPMN 中能够执行的流程是由不同类型的节点组成，并且每一个节点之间是使用有向连线连接。BPMN2.0 中定义了三种主要的节点类型：

- Events：事件是用来建模特定事件的发生。有 start event, end event, intermediate event.
- Activities：这定义了在流程的执行过程中需要执行的不同动作。存在的不同类型任务，依赖于你想要去建模的动作类型，这些动作时可以嵌套的。
- Gateways：用来定义在流程中多种路径。这依赖于 gateway 的类型，这可能是表示是并行执行或选择等等。

jBPM5 没有实现在 BPMN 2.0 规则中所定义的所有元素和属性。我们只是支持规范的子集，包括许多的能够在可执行的流程内部使用的公共节点类型。这里包括在 BPMN 2.0 规范的“Common Executable”子集中定义的所有元素和属性，并扩展了一些我们相信在上下文中有价值的额外的元素和属性。下面是所有支持的元素和属性：

- Flow Objects

- Events
  - Start Event(None, Conditional, Signal, Message, Timer)
  - End Event(None, Terminate, Error, Escalation, Signal, Message, Compensation)
  - Intermediate Catch Event(Signal, Timer, Conditional, Message)
  - Intermediate Throw Event (None, Signal, Escalation, Message, Compensation)
  - Non-interrupting Boundary Event (Escalation, Timer)
  - Interrupting Boundary Event (Escalation, Error, Timer, Compensation)
- Activities
  - Script Task
  - Task
  - Service Task
  - User Task
  - Business Rule Task
  - Manual Task
  - Send Task
  - Receive Task
  - Reusable Sub-Process (Call Activity)
  - Embedded Sub-Process
  - Ad-Hoc Sub-Process
  - Data-Object
- Gateways
  - Diverging
    - Exclusive
    - Inclusive
    - Parallel

        - Event-Based
    - Converging
        - Exclusive
        - Parallel
    - Lanes
- *Data*
    - Java type language
    - Process properties
    - Embedded Sub-Process properties
    - Activity properties
- *Connecting objects*
    - Sequence flow

例如：查看下面的 BPMN 2.0 流程“Hello World”，当流程被启动时，仅仅输出”Hello World”.

这个流程的可执行版本是使用 BPMN 2.0 XML 来表示，具体如下：

```
[键入文档的引述或关注点的摘要。您可将文本框放置在文档中的任何位置。可使用“文本框工具”选项卡更改重要引述文本框的格式。]<?xml version="1.0" encoding="UTF-8"?>
<definitions id="Definition"
targetNamespace="http://www.example.org/MinimalExample"
typeLanguage="http://www.java.com/javaTypes"
expressionLanguage="http://www.mvel.org/2.0"
xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL BPMN20.xsd"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:dc=http://www.omg.org/spec/DD/20100524/DC

xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xmlns:tns="http://www.jboss.org/drools">
<process processType="Private" isExecutable="true" id="com.sample.HelloWorld"
name="Hello
World" >
```

```
<!-- nodes -->
<startEvent id="_1" name="StartProcess" />
<scriptTask id="_2" name="Hello" >
<script>System.out.println("Hello World");</script>
</scriptTask>
<endEvent id="_3" name="EndProcess" >
<terminateEventDefinition/>
</endEvent>
<!-- connections -->
<sequenceFlow id="_1-_2" sourceRef="_1" targetRef="_2" />
<sequenceFlow id="_2-_3" sourceRef="_2" targetRef="_3" />
</process>
<bpmndi:BPMNDiagram>
<bpmndi:BPMNPlane bpmnElement="Minimal" >
<bpmndi:BPMNShape bpmnElement="_1" >
<dc:Bounds x="15" y="91" width="48" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="_2" >
<dc:Bounds x="95" y="88" width="83" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="_3" >
<dc:Bounds x="258" y="86" width="48" height="48" />
</bpmndi:BPMNShape>
<bpmndi:BPMNEdge bpmnElement="_1-_2" >
<di:waypoint x="39" y="115" />
<di:waypoint x="75" y="46" />
<di:waypoint x="136" y="112" />
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdge bpmnElement="_2-_3" >
<di:waypoint x="136" y="112" />
<di:waypoint x="240" y="240" />
<di:waypoint x="282" y="110" />
</bpmndi:BPMNEdge>
</bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>

</definitions>
```

使用 BPMN 2.0 格式创建你自己的流程：

•使用 Drools Eclipse 插件向导创建一个新的流程文件，在向导的最后一页，确保选择 Drools 5.1 编码兼容性。这将使用 BPMN 2.0 XML 格式创建一个新的流程。注意：现在还没有正确的 BPMN 2.0 编辑器，就像仍然使用不同的属性名一样等等。无论怎样，可以

使用可用的 BPMN 2.0 语法来保存一个流程。也要注意：编辑器并不支持已经在执行引擎中所支持的所有的节点类型和属性。

•设计器是基于 web 的并开源的，并支持 BPMN 2.0 格式。我们已经嵌入进入 Guvnor 中对于 BPMN 2.0 流程的可视化编辑。你能够用设计器去创建或编辑 BPMN 2.0 流程，并然后以 BPMN2.0 格式输出，或者保存到 Guvnor 中并输出，以至于他们能够执行。

•正在开发的新 BPMN2 Eclipse 插件将全面支持 BPMN2 规范。现在仍然在开发当中，并仅仅支持有限的构造和属性，但能够已经用来创建一个简单的 BPMN2 流程。在这个编辑器中创建一个新 BPMN2 文件，使用向导创建一个新 BPMN2 文件，这将产生一个.bpmn2 文件和一个包含图形信息的.prd 文件。双击.prd 文件使用图形编辑器编辑。更多的细节，请查看新的 BPMN2 插件。

•你通过直接写 XML 来手动创建你的 BPMN 2.0 流程文件。你可以使用 BPMN 2.0 XSD 来验证你流程的语法，或者使用在 Eclipse 插件中的验证器来检查你建模的语法和完整性。

下面的代码片段展示了你应该怎样加载一个 BPMN2 流程到你的 knowledge base…

```
private static KnowledgeBase createKnowledgeBase() throws Exception {
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add(ResourceFactory.newClassPathResource("sample.bpmn2"),
ResourceType.BPMN2);
return kbuilder.newKnowledgeBase();
}
```

…和怎样去执行这个流程…

```
KnowledgeBase kbase = createKnowledgeBase();
StatefulKnowledgeSession ksession = kbase.newStatefulKnowledgeSession();

ksession.startProcess("com.sample.HelloWorld");
```

更多的细节请查看 API 和基础章节。

6.2 例子

BPMN 2.0 规范定义了每一个节点类型的属性和语义。

jbpm-bpmn2 模块包含每一个不同节点类型的单元测试。这些测试流程也可以作为监督的例子：他们不能真正的代表一个业务流程的整个生命，但是能够明确的用来展现特定的特性是怎么用的。例如，下面的图表就展现流这些例子的流程图。整个表可以在 jbpm-bpmn2 [http://github.com/krisv/jbpm/tree/master/jbpm-bpmn2/src/test/resources/] 的 the src/main/resources 文件夹下找到。

6.3 支持的元素/属性

表 **6.1** 关键字

Table 6.1. Keywords

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
definitions		rootElement BPMNDiagram		
process	processType isExecutable name id	property laneSet flowElement	packageName adHoc version	import global
sequenceFlow	sourceRef targetRef isImmediate name id	conditionExpression	priority bendpoints	
interface	name id	operation		
operation	name id	inMessageRef		
laneSet		lane		
lane	name id	flowNodeRef		
import*		name		
global*		identifier type		
Events				
startEvent	name id	dataOutput dataOutputAssociation outputSet eventDefinition	x y width height	

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
endEvent	name id	dataInput dataInputAssociation inputSet eventDefinition	x y width height	
intermediateCatchEvent	name id	dataOutput dataOutputAssociation outputSet eventDefinition	x y width height	
intermediateThrowEvent	name id	dataInput dataInputAssociation inputSet eventDefinition	x y width height	
boundaryEvent	cancelActivity attachedToRef name id	eventDefinition	x y width height	
terminateEventDefinition				
cancelEventDefinition				
compensateEventDefinition	activityRef	documentation extensionElements		
conditionalEventDefinition		condition		
errorEventDefinition	errorRef			
error	errorCode id			
escalationEventDefi	escalationRef			

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
nition				
escalation	escalationCode id			
messageEventDefinition	messageRef			
message	itemRef id			
signalEventDefinition	signalRef			
timerEventDefinition		timeCycle timeDuration		
Activities				
task	name id	ioSpecification dataInputAssociation dataOutputAssociation	taskName x y width height	
scriptTask	scriptFormat name id	script	x y width height	
script		text[mixed content]		
userTask	name id	ioSpecification dataInputAssociation dataOutputAssociation resourceRole	x y width height	onEntry-script onExit-script
potentialOwner		resourceAssignmentExpression		

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
resourceAssignmentExpression		expression		
businessRuleTask	name id		x y width height ruleFlowGroup	onEntry-script onExit-script
manualTask	name id		x y width height	onEntry-script onExit-script
sendTask	messageRef name id	ioSpecification dataInputAssociation	x y width height	onEntry-script onExit-script
receiveTask	messageRef name id	ioSpecification dataOutputAssociation	x y width height	onEntry-script onExit-script
serviceTask	operationRef name id	ioSpecification dataInputAssociation dataOutputAssociation	x y width height	onEntry-script onExit-script
subProcess	name id	flowElement property loopCharacteristics	x y width height	
adHocSubProcess	cancelRemainingInstances name id	completionCondition flowElement property	x y width height	

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
callActivity	calledElement name id	ioSpecification dataInputAssociation dataOutputAssociation	x y width height waitForCompletion independent	onEntry-script onExit-script
multiInstanceLoopCharacteristics		loopDataInputRef inputDataItem		
onEntry-script*	scriptFormat		script	
onExit-script*	scriptFormat		script	
Gateways				
parallelGateway	gatewayDirection name id		x y width height	
eventBasedGateway	gatewayDirection name id		x y width height	
exclusiveGateway	default gatewayDirection name id		x y width height	
inclusiveGateway	default gatewayDirection name id		x y width height	
Data				
property	itemSubjectRef id			

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
dataObject	itemSubjectRef id			
itemDefinition	structureRef id			
ioSpecification		dataInput dataOutput inputSet outputSet		
dataInput	name id			
dataInputAssociation		sourceRef targetRef assignment		
dataOutput	name id			
dataOutputAssociation		sourceRef targetRef assignment		
inputSet		dataInputRefs		
outputSet		dataOutputRefs		
assignment		from to		
formalExpression	language	text[mixed content]		
BPMNDI				
BPMNDiagram		BPMNPlane		
BPMNPlane	bpmnElement	BPMNEdge BPMNShape		
BPMNShape	bpmnElement	Bounds		

Element（元素）	Supported attributes（支持的属性）	Supported elements 支持的元素	Extension attributes 扩展的属性	Extension elements 扩展的元素
BPMNEdge	bpmnElement	waypoint		
Bounds	x y width height			
waypoint	x y			

第七章 核心引擎：持久化和事务

jBPM 支持确定信息，流程运行时状态，历史信息的持久化存储。

7.1 运行时状态

当一个流程启动时，一个流程实例就被创建，这代表着流程在特定的上下文中执行。例如，当执行一个指定如何去处理订单的流程时，针对每一个销售请求都将创建一个流程实例。流程实例代表着当前在特定的上下文中执行的状态，包含了所有和流程实例相关的信息。注意：它仅仅包含最小限度的运行时状态，这些状态是流程实例在之后执行所需要的状态，但是他不包括流程实例历史的有关信息，如果这些信息在流程实例中不再需要。执行流程的运行时状态能够被持久化在数据库中。这就允许在不希望错误发生的情况下，可以恢复所有运行流程的执行状态，或者暂时将运行实例从内存移除并在之后恢复他们。jBPM 允许你插入不同的持久化策略。默认的，如果你不配置流程引擎的话，流程实例就不会持久化。

7.1.1 二进制持久化

jBPM 提供了二进制持久化机制，它允许你以二进制数据集的方式来保存流程实例的状态。这样，运行流程实例的状态就能够一直被存储在持久固定的位置。注意：这些二进制数据集相对来说一般较小，因为他们只包含最小限度的流程实例的执行状态。对于一个简单的流程实例，这一般包含一个或几个节点实例，任何当前正在执行的节点和一些变量的值。

7.1.2 Safe Pionts

在流程引擎在执行过程中，流程实例的状态被存储在一个被称为“safe points”的地方。无论一个流程实例什么时候允许，在它从一个等待状态开始或继续，引擎会一直继续执行，直到没有动作能够被执行。在那一点，引擎已经到达了下一个安全状态，所有有可能受影响的流程实例状态和其他所有的流程实例都将被持久化存储。

7.1.3 配置持久化

默认的，引擎不会持久的保存运行时的数据。但是，要配置引擎去存储这些数据也是非常简单的，只需添加一个配置文件和必要的依赖。持久化自身是基于 Java 的持久化 API，这样就能够在几个持久化机制下工作了。我们可以使用默认的 Hibernate，但是也可以自由选择其他的。在下面将使用一个 H2 数据库来存储数据，但是也可以选择你自己的。

首先，你需要添加必需的依赖到你类路径。如果你使用 Eclipse IDE，你可以添加 jar 文件到你的 jBPM 运行目录，或者手动添加这些依赖到你的项目。首先你需要添加 jar 文件：jbpm-persistence-jpa.jar，因为它包含了保存运行状态的代码。下一步，你需要其他各种依赖，同时也需要你需要的持久化环境和数据库。Hibernate 作为 JPA 持久化提供的默认结合，H2 数据库和 Bitronix 是基于 JTA 的事务管理，下面的列表是必须要添加的依赖：

1. jbpm-persistence-jpa (org.jbpm)

2. drools-persistence-jpa (org.drools)

3. persistence-api (javax.persistence)

4. hibernate-entitymanager (org.hibernate)

5. hibernate-annotations (org.hibernate)

6. hibernate-commons-annotations (org.hibernate)

7. hibernate-core (org.hibernate)

8. dom4j (dom4j)

9. jta (javax.transaction)

10.btm (org.codehaus.btm)

11.javassist (javassist)

12.slf4j-api (org.slf4j)

13.slf4j-jdk14 (org.slf4j)

14.h2 (com.h2database)

15.commons-collections (commons-collections)

下面，你需要配置 jBPM 引擎在任何必要的时候来保存引擎的状态。最简单的方式就是使用 JPAKnowledgeService 来创建你的 knowledge session，基于 knowledge base，一

个 knowledge session 和一个环境。环境需要包含从你的 Entity Manager Factory 来索引。例如：

```
// create the entity manager factory and register it in the environment
EntityManagerFactory emf =
Persistence.createEntityManagerFactory( "org.jbpm.persistence.jpa" );
Environment env = KnowledgeBaseFactory.newEnvironment();
env.set( EnvironmentName.ENTITY_MANAGER_FACTORY, emf );
// create a new knowledge session that uses JPA to store the runtime state
StatefulKnowledgeSession ksession =
JPAKnowledgeService.newStatefulKnowledgeSession( kbase, null, env );
int sessionId = ksession.getId();
// invoke methods on your method here
ksession.startProcess( "MyProcess" );
ksession.dispose();
```

你也可以使用 JPAKnowledgeService 来基于一个特定的 session id 来重新创建一个 session：

```
// recreate the session from database using the sessionId
ksession = JPAKnowledgeService.loadStatefulKnowledgeSession( sessionId, kbase, null,
env );
```

注意：我们只保存流程实例在之后的点需要继续执行的最小限度的状态。这就意味着它不会保存已经执行的节点的信息，如果这些节点再也不会用的，又或者已经完成的实例，或者中止并从数据库删除的情况都不会保存。如果你想搜索历史相关的信息，你应该使用历史日志，之后会解释为什么。

你需要使用 Hibernate 和 H2 数据库，并添加一个持久化配置到你的类路径去配置 JPA，在 META-INF 目录下调用 persistence.xml。关于怎样更改你自己的配置的更多信息，我们已经查阅了 JPA 和 Hibernate 文档获取更多的信息。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence
version="1.0"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/persistence">

<persistence-unit name="org.jbpm.persistence.jpa">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
```

这个配置文件引用的数据源为："jdbc/processInstanceDS"。下面的 Java 代码片段用来建立数据源，这里我们使用的是 H2 数据库。

```
PoolingDataSource ds = new PoolingDataSource();
ds.setUniqueName("jdbc/testDS1");
ds.setClassName("org.h2.jdbcx.JdbcDataSource");
ds.setMaxPoolSize(3);
ds.setAllowLocalTransactions(true);
ds.getDriverProperties().put("user", "sa");
ds.getDriverProperties().put("password", "sasa");
ds.getDriverProperties().put("URL", "jdbc:h2:file:/NotBackedUp/data/process-instance-db");
```

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
<local-tx-datasource>
<jndi-name>jdbc/testDS1</jndi-name>
<connection-url>jdbc:h2:file:/NotBackedUp/data/process-instance-db</connection-url>
<driver-class>org.h2.jdbcx.JdbcDataSource</driver-class>
<user-name>sa</user-name>
<password>sasa</password>
</local-tx-datasource>
</datasources>
```

中从引擎调用的方法。如果这种方式能够被接受，你就不需要做其他的了。当然，你也可以自己指定事务边界。这就允许你可以将多个明天添加到事务当中。

在使用用户定义的事务之前，你需要在环境中注册一个事务管理器。下面是使用 Bitronix 事务管理的实例代码。下面我们就使用 Java Transaction API 来指定事务边界：

```
// create the entity manager factory and register it in the environment
EntityManagerFactory emf =
Persistence.createEntityManagerFactory( "org.jbpm.persistence.jpa" );
Environment env = KnowledgeBaseFactory.newEnvironment();
env.set( EnvironmentName.ENTITY_MANAGER_FACTORY, emf );
env.set( EnvironmentName.TRANSACTION_MANAGER,
TransactionManagerServices.getTransactionManager() );
// create a new knowledge session that uses JPA to store the runtime state
StatefulKnowledgeSession ksession =
JPAKnowledgeService.newStatefulKnowledgeSession( kbase, null, env );
// start the transaction
UserTransaction ut =
(UserTransaction) new InitialContext().lookup( "java:comp/UserTransaction" );
ut.begin();
// perform multiple commands inside one transaction
ksession.insert( new Person( "John Doe" ) );
ksession.startProcess( "MyProcess" );
// commit the transaction
ut.commit();
```

7.2 流程定义

流程定义文件通常是使用 XML 格式编写的。在开发中，这些文件能够很容易的存储在文件系统中。无论你想什么时候使你的 knowledge 访问产品中的一个或多个引擎，我们建议使用 knowledge repository（知识库），知识库就是集中你的 knowledge 到一个或多个知识库中。

Guvnor 是 Drools 子项目所提供的工具。它由一个存储各种不同类型的 knowledge、流程定义、rules、对象模型的 repository 知识库组成的。使用 WebDAV 或一个 knowledge 代理都很容易检索 knowledge，当专家一个 knowledge base 的时候，knowledge 代理将会从 Guvnor 下载信息，并提供了一个 web 应用来允许业务用例来查看并有可能在 knowledge repository 中更新信息。查看 Drools Guvnor 文档获取关于怎么做的更多信息。

7.3 历史日志

很多情况下，历史日志用来存储流程实例执行相关的信息，以至于这些信息在之后可以用来核实一个指定的流程实例的动作执行的内容，或者用来监视和分析一个特定流程的效率。存储历史信息到运行时的数据库不是一个很好的注意，这有可能造成运行时数据的不断增加，并且监视和分析查询会影响运行时引擎的执行。这就是为什么关于流程实例执行的历史信息要分开存储了。

执行信息的历史日志基于事件的产生通过流程引擎的执行被创建的。jBPM 运行时引擎提供了一个监听不同类型事件的通用机制。必要的信息将很容易从事件中获取并持久化到一个数据库当中。过滤器被用来仅仅存储你找到的相关信息。

7.3.1 将流程事件存储到数据库

jbpm-bam 模块包含了一个事件监听器，这个监听器使用 JPA 或者 Hibernate 来直接存储流程相关信息。数据库包含两个表，一个用来存储流程实例信息，一个用来存储节点实例信息。

1. ProcessInstanceLog：这个表包括所有流程实例的 id、流程 id、开始时间和结束时间。
2. NodeInstanceLog：这个表包含节点在每个流程实例中实际执行的节点的更多详细信息。无论什么时候，节点实例都是从它的进入链接中进入，从它的推出链接中推出，这些信息都存储在这个表中。它存储流程实例的 id，正在执行的流程实例的流程 id，节点实例的 id，和相应的节点 id，并且事件的类型和事件的状态都也会被存储。

ID	PROCESSINSTANCEID	PROCESSID	START_DATE

ID	TYPE	NODEINSTANCEID	NODEID	PROCESSINSTANCEID	PROCESSID	LOG_DATE

为了记录历史信息到数据库当中，你需要在你的 session 中注册一个记录器，像下面所示：

```
StatefulKnowledgeSession ksession = ...;
JPAWorkingMemoryDbLogger logger = new JPAWorkingMemoryDbLogger(ksession);
// invoke methods one your session here
logger.dispose();
```

注意：这个记录器和其他的核查记录器一样，这就意味着你能够添加一个或多个过滤器，通过调用 addFilter 方法来确保只有相关的信息被存储在数据库中。信息只能被所有的过滤器访问。你应该在不再需要记录的时候去除记录器。

编辑 persistence.xml 文件来指定存储信息的数据库，这个文件也包括审核日志的类型。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence
version="1.0"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/persistence">
```

火龙果•整理
uml.org.cn

所有的这些信息都可以被容易的查询，并在很多不同的使用情况下使用，创建一个指定流程实例的历史日志用来分析指定流程的所有实例的执行。

审核日志是被认为是一个默认的执行。我们不知道你需要存储哪些信息在之后用来分析，或由于执行的原因，它仅仅建议存储相关的数据。这样依赖你的情况，你可以定义你自己的数据模型来存储你需要的信息，并使用流程事件监听器来获取信息。

第八章 核心引擎：示例

8.1 jBPM 示例

这有一个分离的 jBPM 示例模块，它包含了一组流程的示例，并展示了如何使用 jBPM 引擎和行为，或如何通过 BPMN 2.0 规范来定义不同的流程构造。

为了能够去使用这个示例，先解压文件到某处，并打开安装了所有必须的插件的 Eclipse。如果你不知道怎么做，你可以查看一下 installer 章节，你会学到如何创建一个

demo 环境，包括一个完整配置的 Eclipse IDE，通过使用 jBPM installer。如果你想手动安装和配置这些你能够通过查看 Eclipse plugin 章节。

你只需将下载的示例项目导进 Eclipse 中就可以查看示例了，浏览你解压 jBPM 项目的文件夹选择项目并单击完成。这应该将你的示例项目导入了你的工作空间了，所以你能够开始查看流程并指那些类。

8.2 示例

这个示例包含从基础到高级的很多例子：

• Looping：一个示例展现了你能够怎么使用高级的 gateway 网关来循环你流程的一部分，一直到循环条件不再可用。流程设置一个‘count’（需要循环次数的值）作为输入并打印每一个循环的状态直到流程完成。

• MultiInstance：这个示例展示了怎样去执行子流程对于在一个集合中每一个元素。如果在列表中的每一个人都执行了任务，那么流程就完成了。

• Evaluation：一个执行评估流程展现了如何去把人工角色集成到流程中。基本的例子简单的展现了任务分配给预定义的用户，高级版本展现了数据在流程、任务和返回，组的分配、任务委托直接的传递。

• HumanTask：这是一个使用人工任务的高级示例。它展示了怎么在 tasks , task forms, swimlaces 之间传递数据。这个示例也配置了 Guvnor 知识库并执行了 jBPM 控制台。

• Request：这个高级示例展示了流程和规则一起作用的多种方式，像规则任务调用验证规则，规则作为在流程内部的表达语言约束，对于异常处理的规则，用来监视的事件流程，对更加复杂的流程添加规则。

8.3 单元测试

这个示例项目包含了大量的简单 BPMN2 流程，流程包含 jBPM5 所支持的每一个不同节点。在 junit 文件夹 src/main/resources 下，你能够在流程示例中找到例如开始事件，分离网关等元素。你想查看一个简单的展现指定元素行为的工作示例，这里你应该能够找到。文件夹中包含了 50 个简单流程。双击就可以在图形编辑器中打开。

每一个流程都有一个小单元测试，测试构造的执行。
org.jbpm.examples.junit.BPMN2JUnitTests 在测试源文件夹中包含了每一流程的测试。你可以执行这些测试来测试你想要执行的选择的方法。

如果你想学习更多关于如何测试示例流程的知识，请查看测试和调试章节。

第九章 Eclipse BPMN 2.0 Plugin

我们正在使用 BPMN 2.0 XML 图形语法来开发 BPMN 2.0 Eclipse editor，允许你来编辑流程。编辑器本身是基于 Eclipse Graphiti 框架和 Eclipse BPMN 2.0 模型。

特性：

•它几乎支持所有的 BPMN 2.0 流程的构建和属性。

•支持 jBPM5 介绍的一些自定义属性

•当建模流程时，允许你配置你想使用的元素和属性

非常感谢 Codehoop 在开发新版本编辑器所作出的努力。

9.1 安装

需要：

• Eclipse 3.6 or newer

• Graphiti framework, using update site
http://download.eclipse.org/graphiti/updates/0.7.1/

安装之前，首先启动 Eclipse 并安装 Graphiti，从上面的更新地址来安装。通过 help 菜单选择 install new software，然后输入网址并选择，然后安装，然后通过 *http://codehoop.com/bpmn2 来在 Eclipse 中安装最新版本的 BPMN 2.0 编辑器。*

录屏地址为：[http://vimeo.com/22022128]
资源地址：*https://github.com/droolsjbpm/bpmn2-eclipse-editor*

9.2 创建你的 BPMN 2.0 流程

你可以使用简单的向导来创建一个新的 BPMN 2.0 流程：File -> New - Other ...

选择 BPMN - BPMN2 Diagram

关于一些简单的流程的视频：

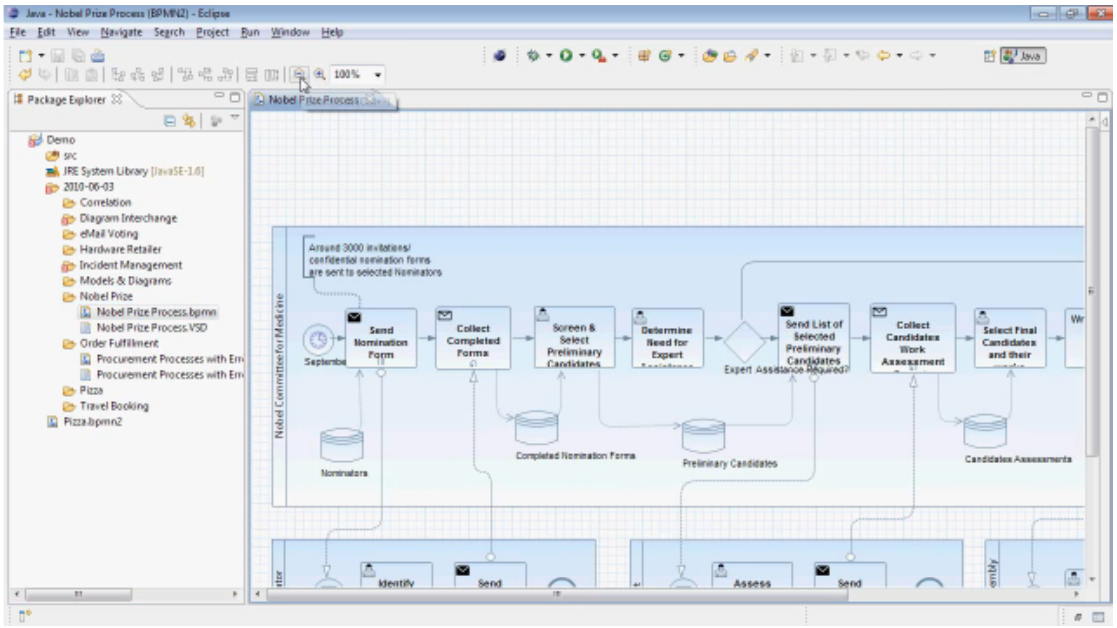


图 9.1 [http://vimeo.com/22021856]

这里有一些编辑器动作的截图：

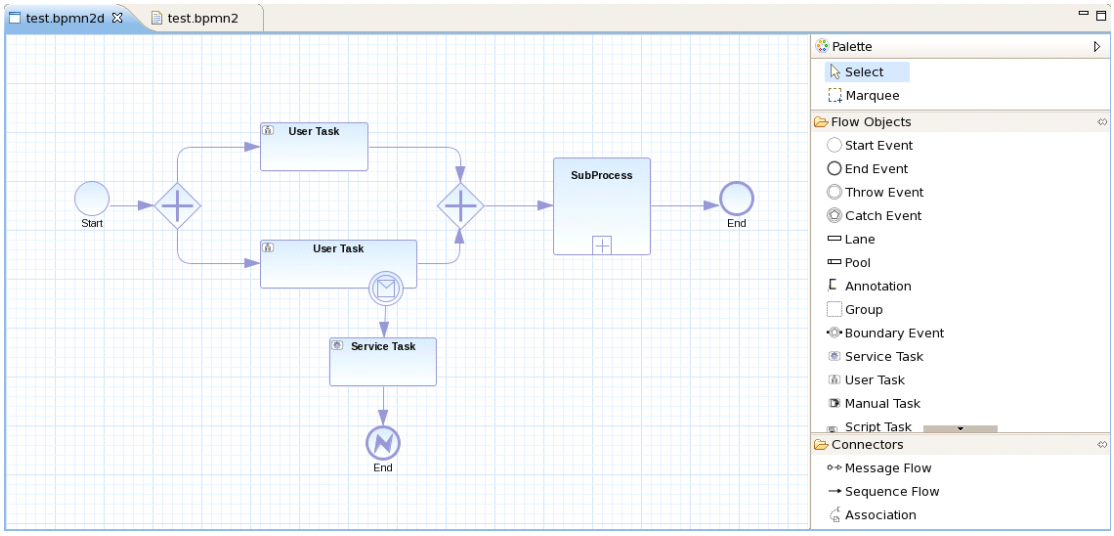


图 9.2

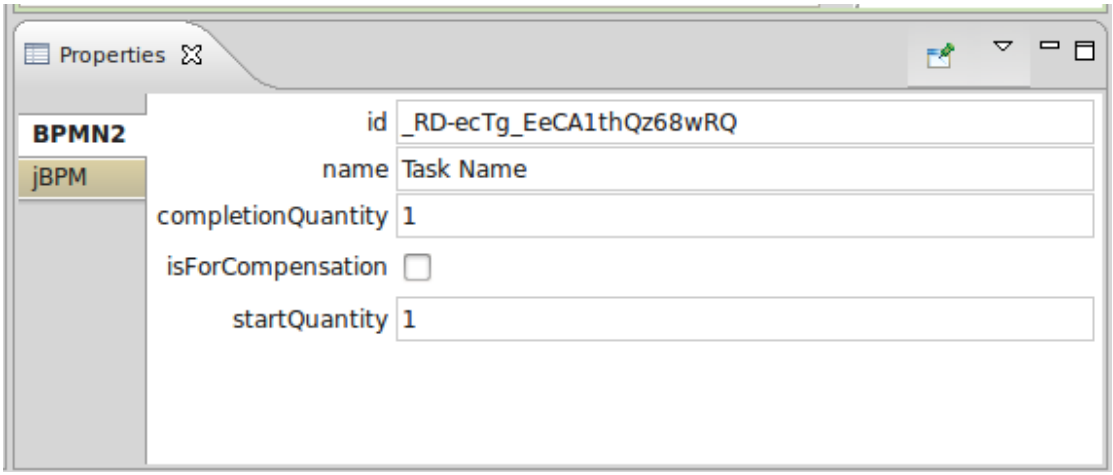

Properties
BPMN2
jBPM
id
_RD-ecTg_EeCA1thQz68wRQ
name
Task Name
completionQuantity
1
isForCompensation
startQuantity
1

图 9.3

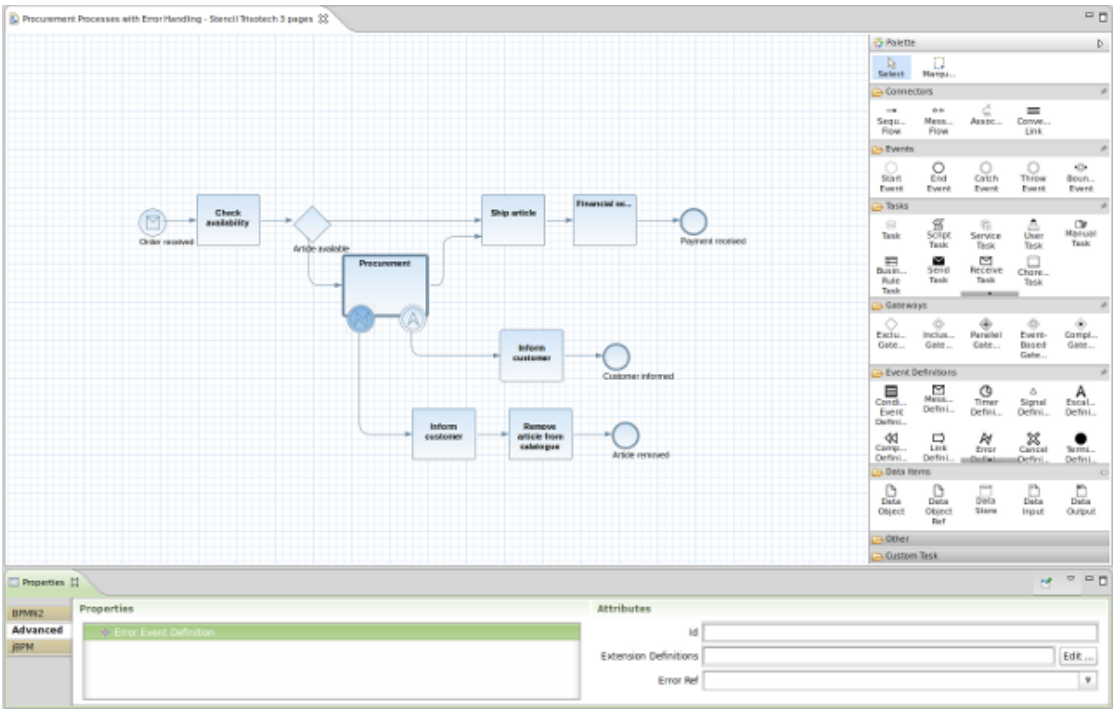

Check availability
Ship article
Procurement
Inform customer
Inform customer
Remove article from catalogue
Palette
Properties
Attributes

图 9.4

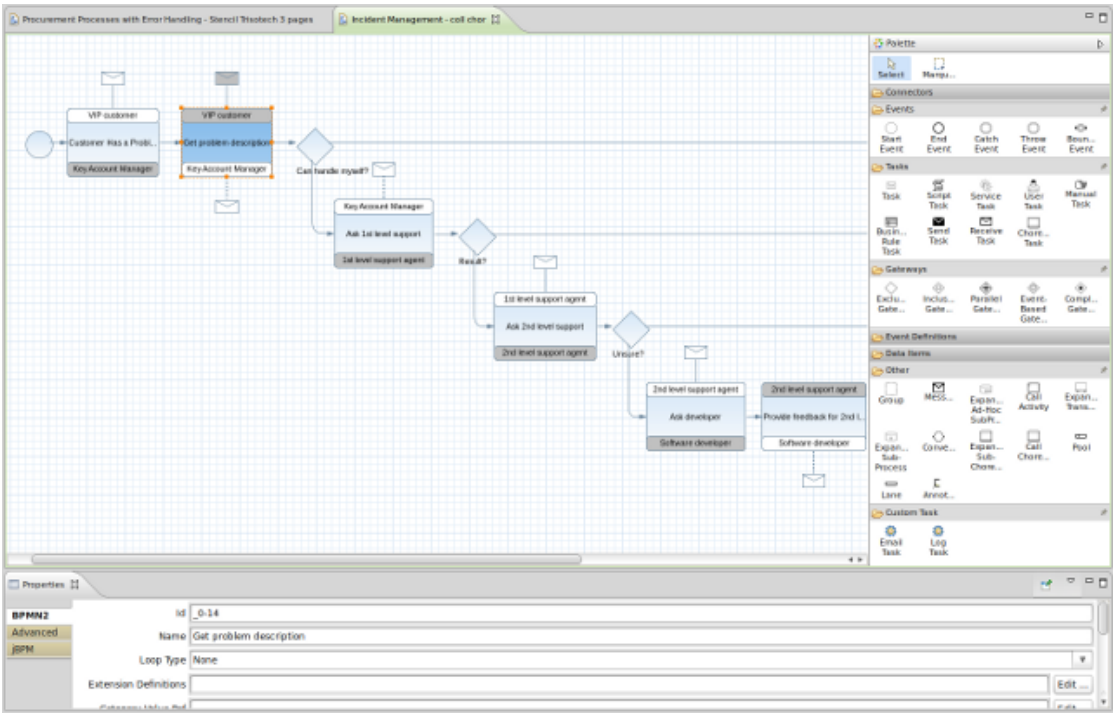
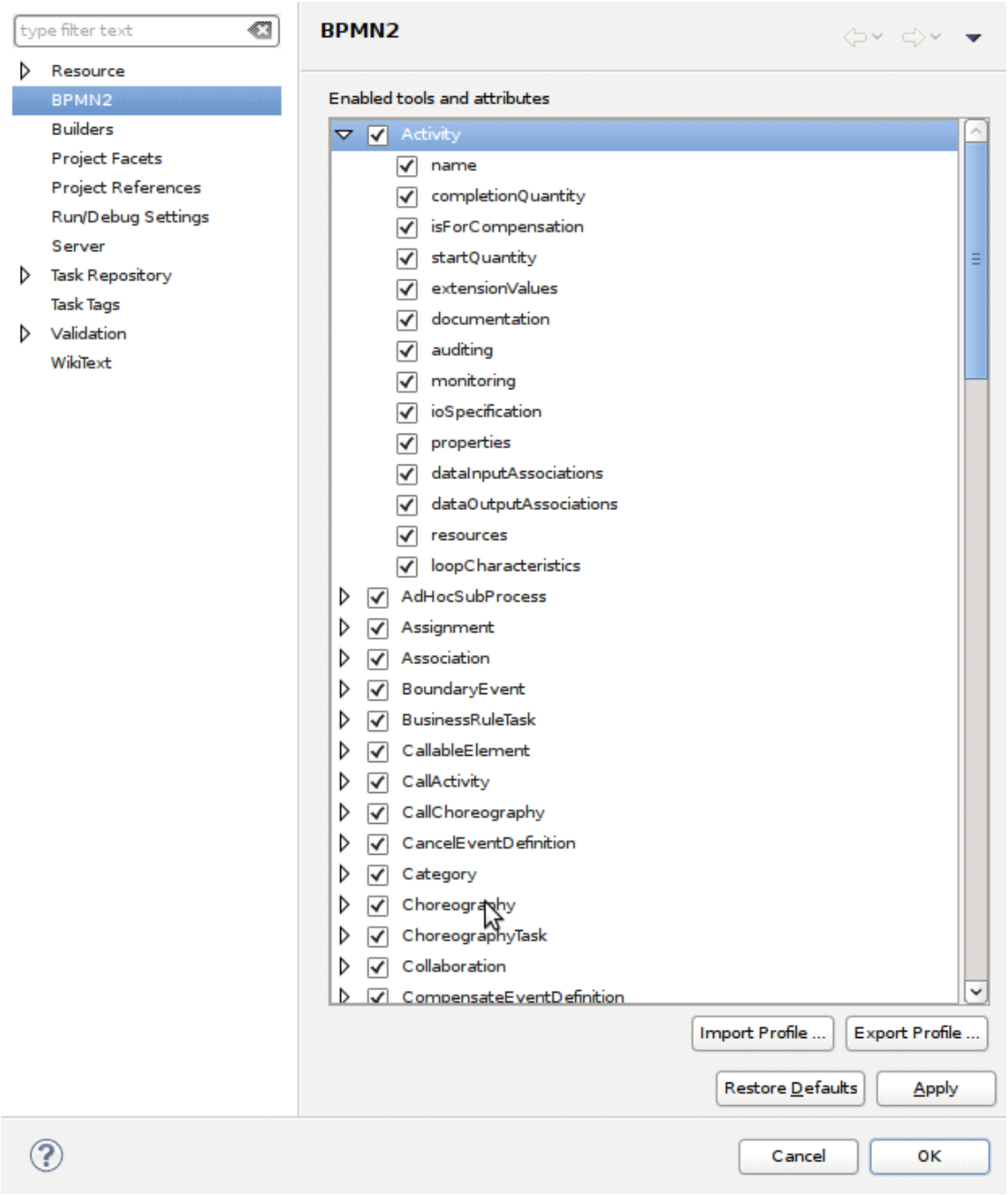


图 9.5

9.3 元素和属性的顾虑

当描述你的 BPMN 2.0 图表时，你能够定义你想要使用的 BPMN 2.0 元素和属性。

因为 BPMN 2.0 规范是相当的复杂，兵器包含大量不同的节点类型和属性对于每一个节点，这些节点都是你想在你项目中使用的元素和属性。使用 BPMN 2.0 的参数选择策略能够使元素和属性在项目层有效或失效。BPMN2 参数选择包含一个所有支持的元素和属性入口，你能够使选择框中的元素和属性有效或失效。



第十章 设计器

设计器是基于 web 的编辑器，能够用来查看，创建和编辑你的业务流程。相对于基于 Eclipse 的设计器是非常小的，允许在 web 上下文中创建 BPMN2 流程。这里在左边有一个调色板，右面有一个属性面板。

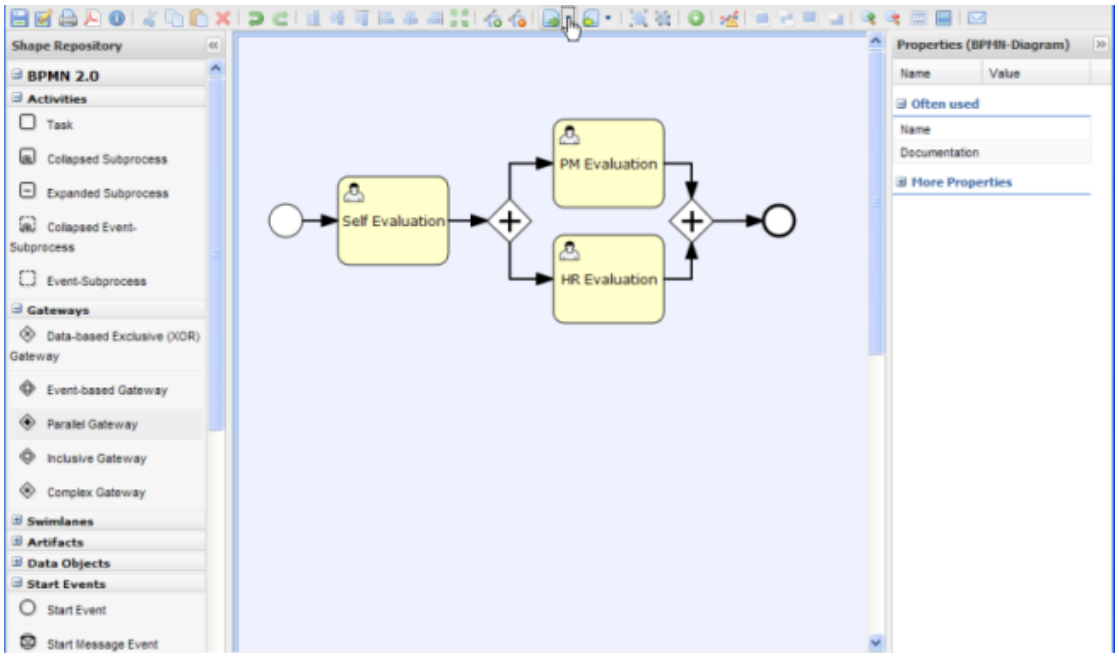


图 10.1

设计器的目标有以下几点：

• View existing BPMN2 processes：设计器允许你在一个 web 上下文中打开一个已经存在的 BPMN2 流程。

• Prototyping new BPMN2 processes：用户可以在设计器中创建一个新的 BPMN2 流程，使用编辑的能力来初始化一些细节的东西。这就允许业务用户创建他们想创建的业务流程的第一个原型版本。这里流程能够被导入 Eclipse 中，并添加所有的细节就可以完全的执行。

在这一点上，设计器并不能够支持 BPMN2 流程的所有操作（由于在解析和编辑上的限制）。我们建议使用 Eclipse 插件添加所有的执行细节。最终结果可以在 Guvnor 中被执行，以至于业务用户能够看到最终流程。我们正在努力在下一版本中添加所有的功能。

10.1 安装

如果你使用 jBPM installer，就会自动下载并为你安装最新版本的设计器。手动安装设计器，只需简单的讲 designer War 拖入你的应用服务配置文件夹。除了 JBoss AS 5.1.0 需要更改外，其他都不需要配置。注意：如果你想配置到其他的服务上去，你必须在 War 内添加合适的依赖，war 是基于你的应用服务所提供的默认库。最新版本的设计器: [http://people.redhat.com/tsurdilo/oryx/1.0.0.051/]。

打开设计器 *http://localhost:8080/drools-guvnor*，打开 Guvnor ，http://localhost:8080/drools-guvnor

设计器也能够通过下面的链接 *http://localhost:8080/designer/editor?profile=jbpm&uuid=123456*打开单独的流程。注意：以这种方式运行的设计器仅仅能够查看存在的流程，而不能够保存编辑和创建任何的流程。对于集成设计器到你自己的应用的信息请查看：http://blog.athico.com/2011/04/using-oryx-designerand-guvnor-in-your.html。

第十一章 控制台

业务流程能够通过一个 web 控制台来管理。这包括的属性有流程实例的管理，查看你的任务列表，执行这些任务，产生报告。

由两个 war 包组成的 jBPM 控制台必须配置在你的应用服务中，并包含必须的库，实际的应用。一个 jar 包包含服务应用，另一个是客户端。

11.1 安装

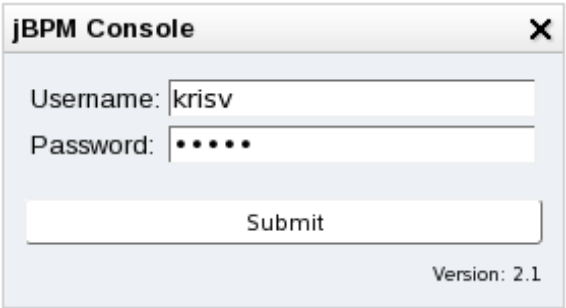
最简单的方式就是使用 installer。所有的下载安装配置等操作都自动完成。具体查看 installer 章节。

控制台是一个分离的子项目，通过不同的项目被共享，想 jBPM 和 RiftSaw。jBPM5 当前的版本源代码可以在 SVN 上找到：[http://anonsvn.jboss.org/repos/soag/bpm-console/tags/bpm-console-2.1/]，最新版本的控制台可以在[https://github.com/bpmc]找到。

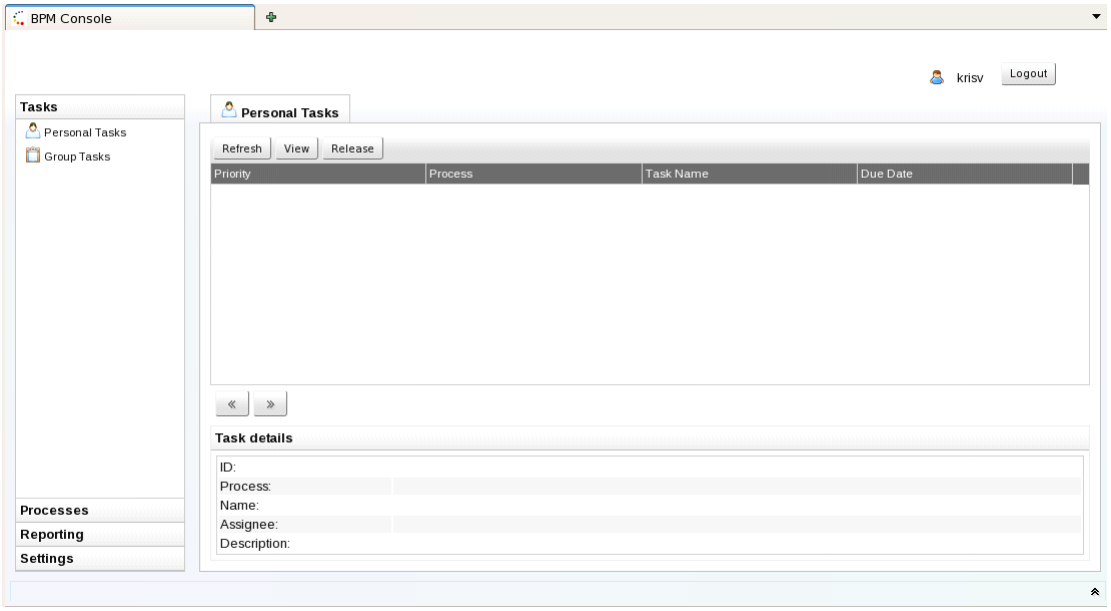
11.2 运行流程管理控制台

在导航栏中输入 *http://localhost:8080/jbpm-console* 访问。

跳出登陆界面，要求填写用户名和密码。默认的用户名密码已经配置好了：krisv/krisv, admin/admin, john/john and mary/mary。



登陆之后，流程管理平台应该被打开，展现的截屏如下。

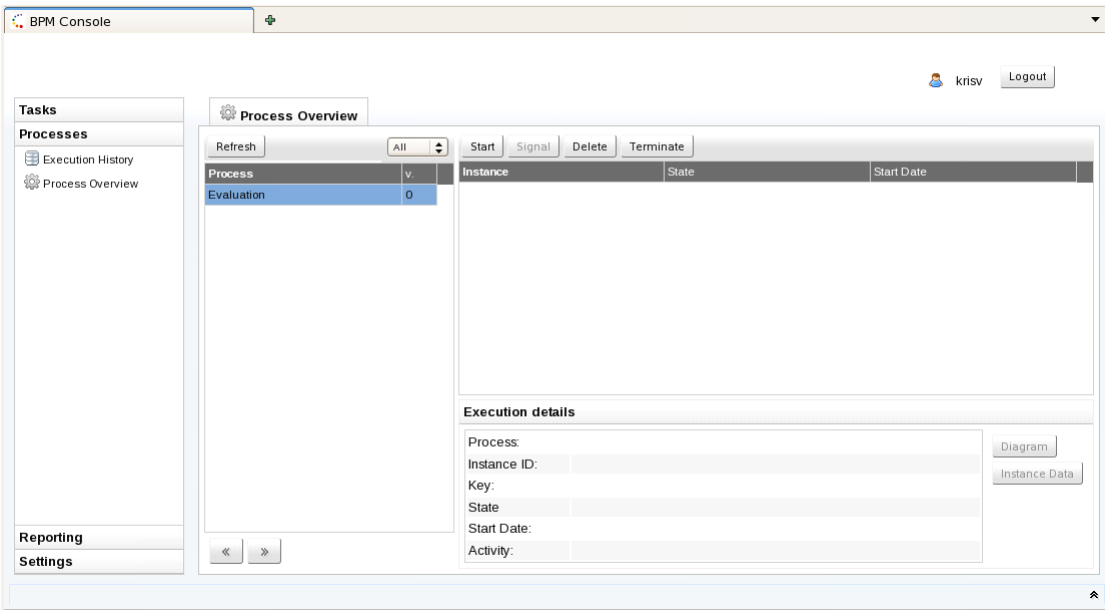


11.2.1 管理流程实例

“Pricesses”片段允许你查看流程的定义，即当前已经安装的 knowledge base 的一部分，启动一个流程流程。管理运行中的流出实例。

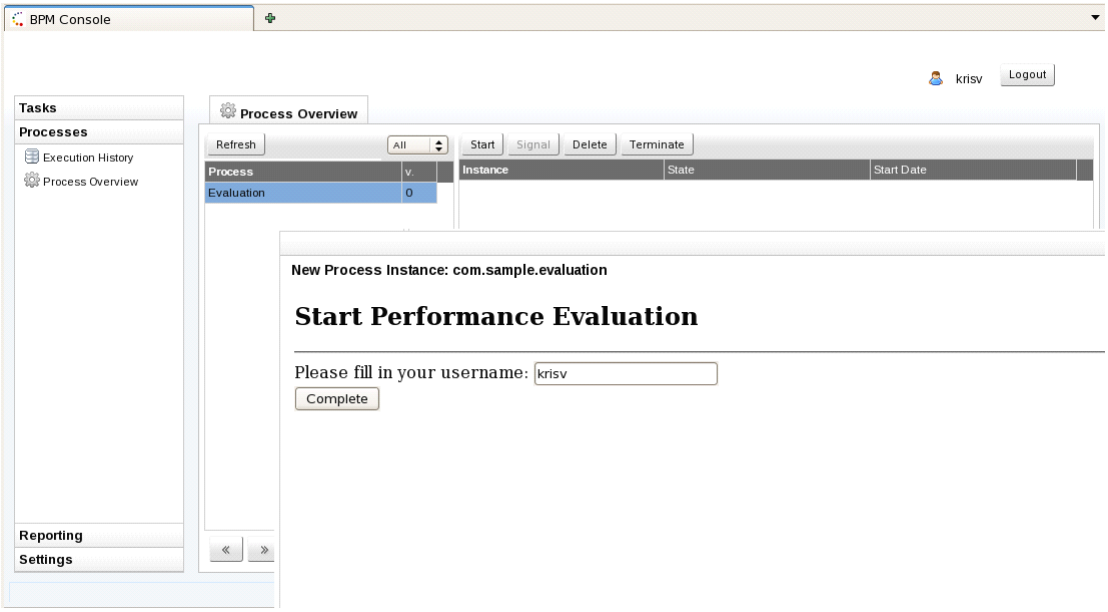
11.2.1.1 查看流程的定义

当你打开流程定义列表的时候，所有能识别出的流程定义都会被展示。你也可以查看指定流程的流程实例或者启动一个新的流程实例。



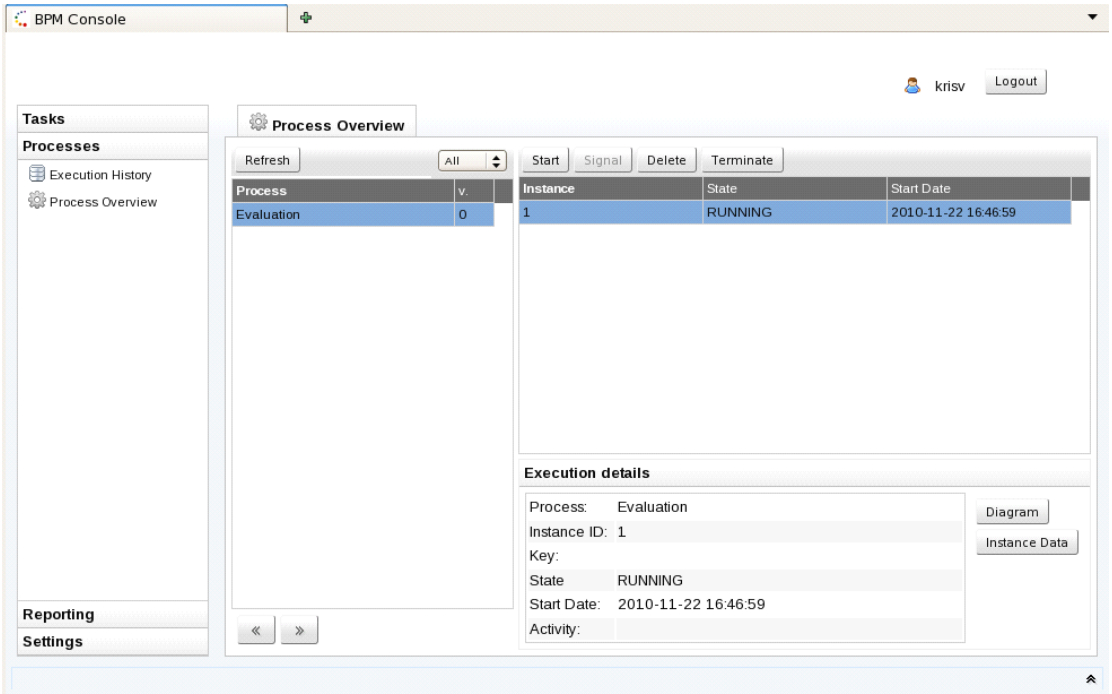
11.2.1.2 启动新的流程实例

启动一个特定的流程定义的流程实例，选择流程定义列表中的流出定义。单击“Start”按钮，启动一个新的特定流程实例。这时候有一个和特定流程相关联的表，这个表将会被展示。完成这个表之后，流程将会根据提供的信息启动。



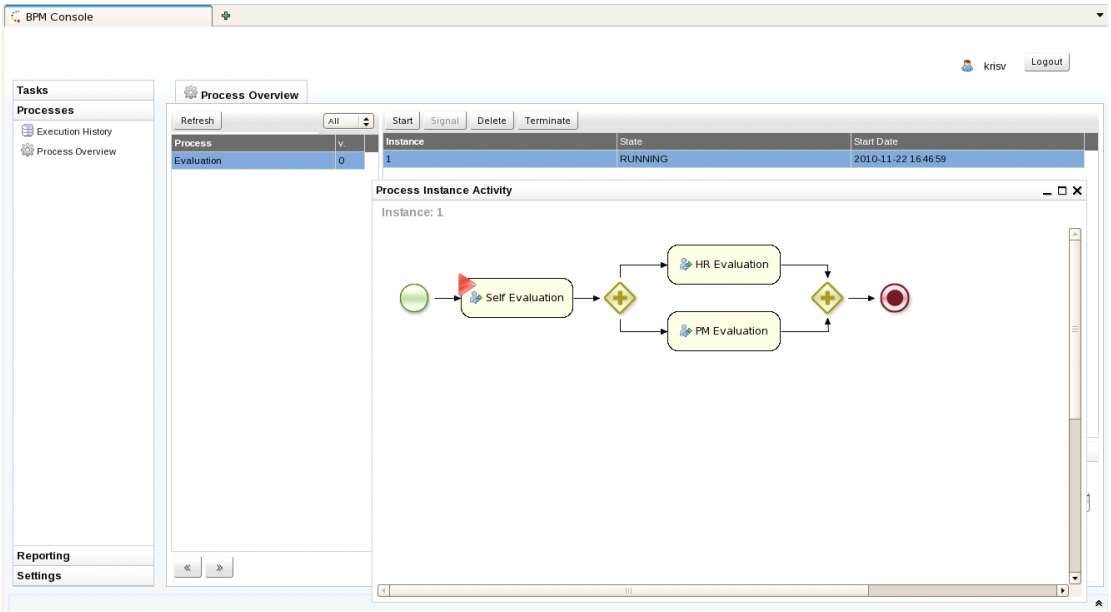
11.2.1.3 管理流程实例

流程实例表展现了所有的特定流程定义的运行中的实例。选择一个流程实例会展现指定流程实例的细节信息。



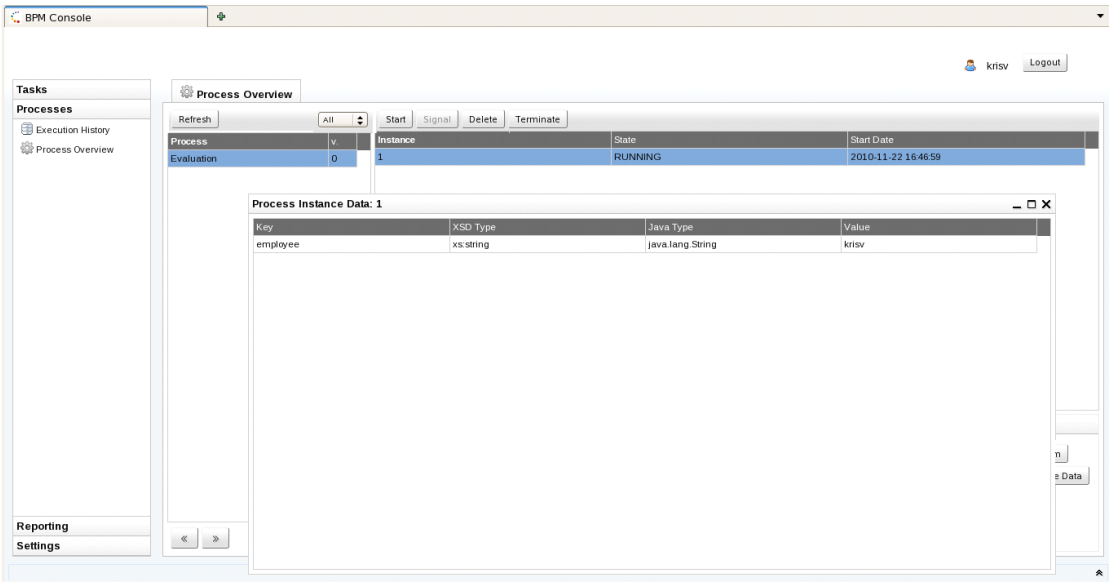
11.2.1.4 查看流程实例状态

你可以通过点击“Diagram ”按钮来查看特定流程实例的状态。这将会展现流程图表，有一个红色的三角展示当前活动的节点。



11.2.1.5 查看流程实例的变量

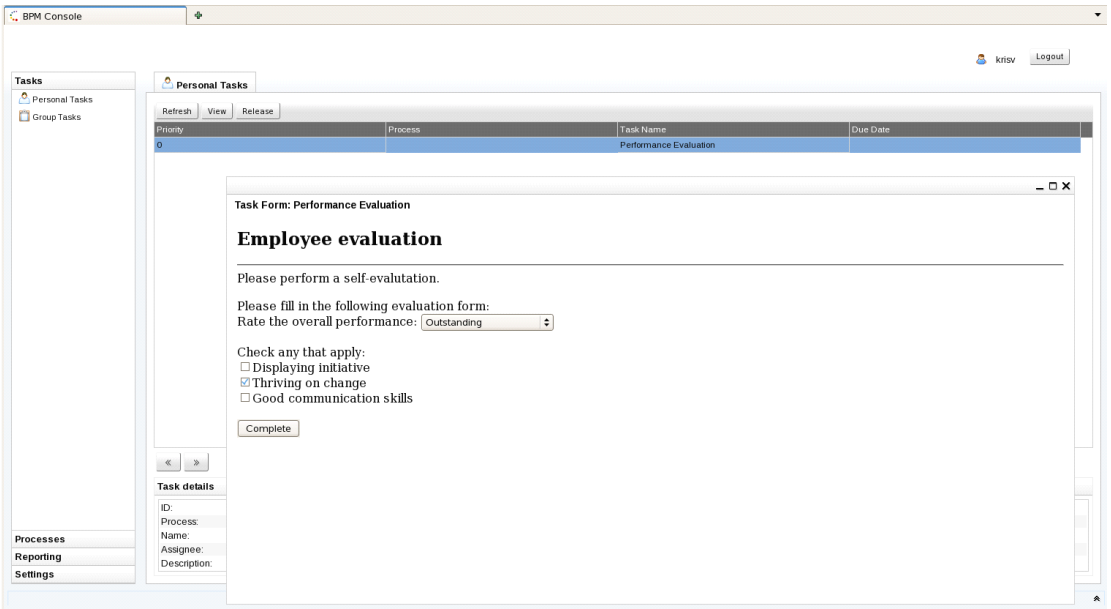
你能够通过点击“Instance Data”按钮来查看特定流程实例的变量。这将会展现在流程中定义的和特定流程实例相关联的变量值。



11.2.2 人工任务列表

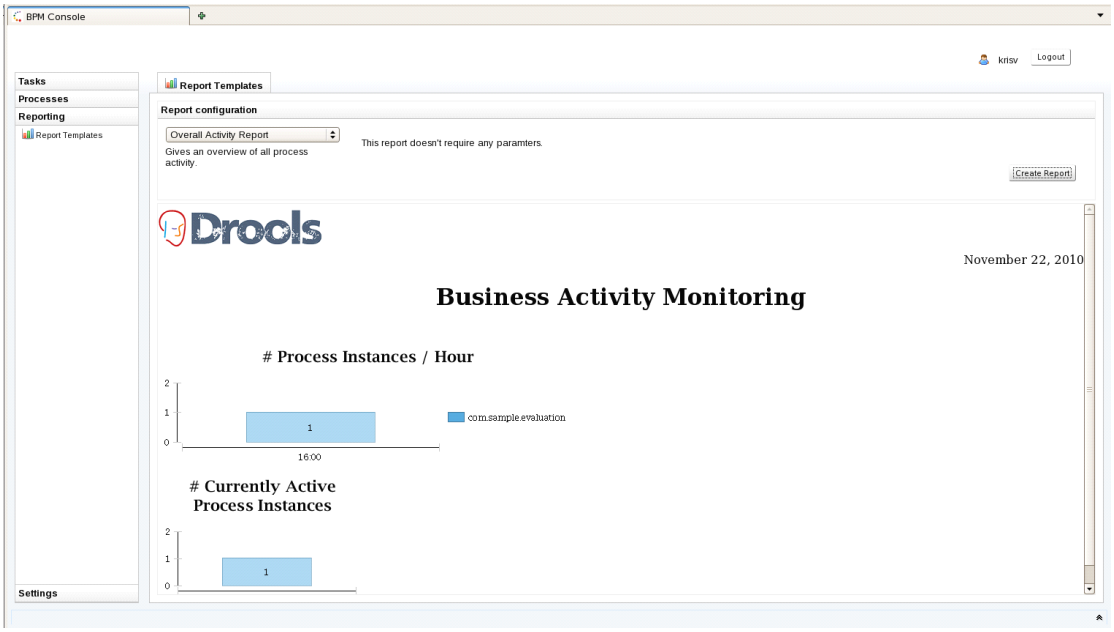
任务管理片段允许用户查看用户自己当前的任务列表。这一组任务列表展现了未分配给一个特定用户，但是当前登陆的用户可以获得任务。个人任务列表展现了所有已经分配给当前登陆的用的任务。执行一个任务，在你的个人任务列表中选择“View”，如果有一个和所选的任务相关联，那么这个表就会出现。完成表之后，任务将会被完成。

火龙果•整理
uml.org.cn

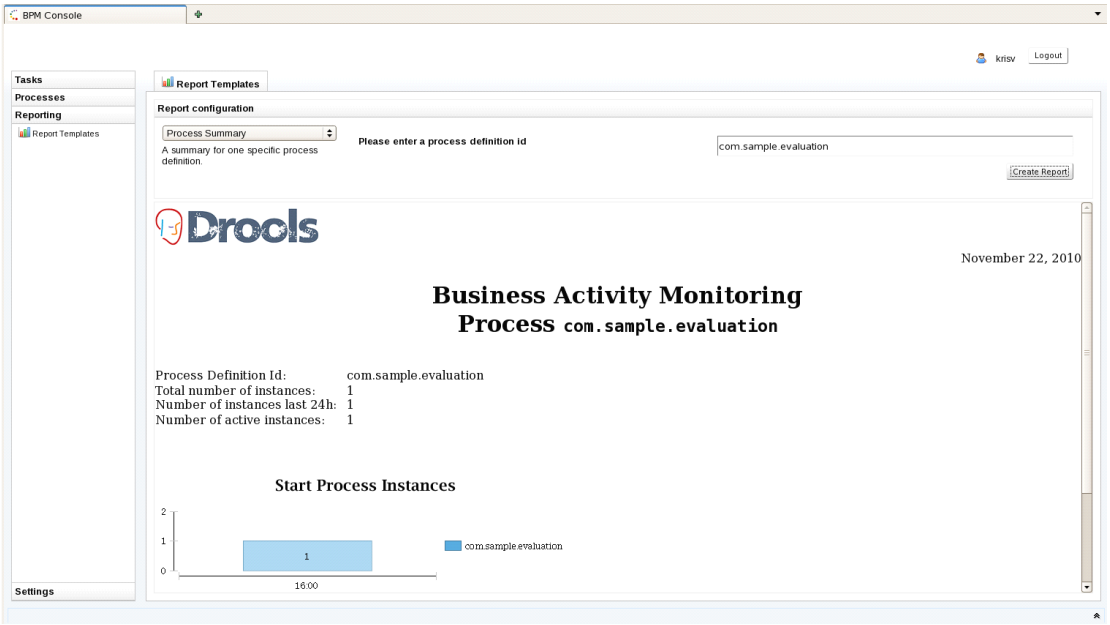


11.2.3 报告

报告页允许你查看流程执行的相关报告。



产生一个特定流程实例的报告。



jBPM 提供了一些简单的报告，这报告能够用来显示普遍的执行特点，像获得每个流程的活动流程实例的数量。但是，通过代替报告文件夹下的报告模板，来产生显示你公司任务是重要的信息。

11.3 添加新的流程/任务表

表能够被用来：（1）开始一个新流程或者（2）完成一个人工任务。我们使用自动产生模板来动态创建一个表。创建一个特定流程定义的表，通过{processId}的名字创建一个自动产生模板。这个模板自身是使用 HTML 代码来建模的表。

```
<html>
<body>
<h2>Start Performance Evaluation</h2>
<hr>
<form action="complete" method="POST" enctype="multipart/form-data">
Please fill in your username: <input type="text" name="employee" /></BR>
<input type="submit" value="Complete">
</form>
</body>
</html>
```

相似的，指定类型的人工任务的任务表能够通过创建一个命名为{taskName}的模板来链接人工任务。这个表访问 task 参数，参数代表当前的人工任务。所以他允许你根据 task 的输入来动态的调整任务表。任务参数是作为在 jbpm-human-task 模块中定义的任务建模对象。这就允许你基于描述或者和那个任务相关的输入数据；来定制任务表。

```
<html>
<body>
<h2>Employee evaluation</h2>
<hr>
${task.descriptions[0].text}<br/>
<br/>
Please fill in the following evaluation form:
<form action="complete" method="POST" enctype="multipart/form-data">
Chapter 11. Console
112
Rate the overall performance: <select name="performance">
<option value="outstanding">Outstanding</option>
<option value="exceeding">Exceeding expectations</option>
<option value="acceptable">Acceptable</option>
<option value="below">Below average</option>
</select><br/>
<br/>
Check any that apply:<br/>
<input type="checkbox" name="initiative" value="initiative">Displaying initiative<br/>
<input type="checkbox" name="change" value="change">Thriving on change<br/>
<input type="checkbox" name="communication" value="communication">Good
communication
skills<br/>
<br/>
<input type="submit" value="Complete">
</form>
</body>
</html>
```

在完成任务时，用户填写任务表时提供的数据将会被作为一个参数被添加。例如，完成上面的任务时，结果参数的映射将会包括结果变量，通过调用"performance", "initiative", "change" and "communication"。相关流程能够通过参数到流程变量的映射来访问结果参数。

表被包括在 jbpm-gwt-form.jar 的 server war 中。

11.4 REST interface

控制台提供了休眠接口，这样控制台就可以很容易的和流程引擎的 starting process instances, retrieving task lists 的特性很好的结合。

你在导航栏中输入 *http://localhost:8080/gwt-console-server/rs/server/resources*，*你就可以查看 REST interface*。

这允许你使用/gwt-console-server/rs/task/{taskId}/close 来关闭一个任务，使用 /gwt-console-server/rs/process/definition/{id}/new_instance 来开始一个新流程。

第十二章 Human Tasks

工作流和 BPM 的一个重要方面就是人工任务管理。虽然流程中的一部工作能够被自动执行，但是有些任务需要和人工角色的互动才能执行。jBPM 支持人工任务在流程内部的使用，在流程中会使用特定的代表这种互动的用户任务节点。这个节点允许流程设计者定义和任务相关的任务类型、角色、数据。我们也可以执行一个任务服务来管理这些用户任务。

为了能够在你的流程内部使用人工任务，你首先需要：（1）在你的流程内有用户任务节点（2）结合你选择的任务管理组件和（3）最终用户通过用户接口类型和人工任务管理组件互动。在下面，这些元素的更多细节将被谈论。

12.1 流程内的 Human tasks

jBPM 通过使用特定的用户任务节点来支持在流程内部使用人工任务。一个用户任务节点代表一个需要被人工角色执行的原子任务。虽然 jBPM 在流程内部的人工任务节点是特定的用户任务，人工任务被简单的认为任何其他的外部服务类型，这些外部服务需要被调用并作为工作条目的一个特定类型实现。唯一的事情就是指定添加 swimlanes 的支持的用户任务节点，使人工任务能够很容易的分配任务给用户。一个用户任务节点包含以下属性：

• *Id*: The id of the node (which is unique within one node container).

• *Name*: The display name of the node.

• *TaskName*: The name of the human task.

• *Priority*（*优先权*）: An integer indicating the priority of the human task.

• *Comment*（*注释*）: A comment associated with the human task.

• *ActorId*（*参与者 id*）: The actor id that is responsible for executing the human task. A list of actor id's can be specified using a comma (',') as separator.

• *Skippable*（*可略过的*）: Specifies whether the human task can be skipped (i.e. the actor decides not to execute the human task).

• *Content*（内容）: The data associated with this task.

• *Swimlane*(泳道): The swimlane this human task node is part of. Swimlanes make it easy to assign multiple human tasks to the same actor. See below for more detail on how to use swimlanes.

• *Wait for completion*（等待完成）: If this property is true, the human task node will only continue if the human task has been terminated (i.e. completed or any other terminal state); otherwise it will continue immediately after creating the human task.

• *On-entry and on-exit actions*: Actions that are executed upon entry and exit of this node.

• *Parameter mapping*: Allows copying the value of process variables to parameters of the human

task. Upon creation of the human tasks, the values will be copied.

• *Result mapping*: Allows copying the value of result parameters of the human task to a process variable. Upon completion of the human task, the values will be copied. Note that can only use result mappings when "Wait for completion" is set to true. A human task has a result variable "Result" that contains the data returned by the human actor. The variable "ActorId" contains the id of the actor that actually executed the task.

• *Timers*: Timers that are linked to this node (see the 'timers' section for more details).

• *ParentId*: Allows to specify the parent task id, in the case that this task is a sub task of another.(see the 'sub task' section for more details)（以上的翻译请参考上面第五章的5.5.3.

当你选择用户任务节点时候，你能够编辑在属性视图中的变量，或者你可以通过双击用户任务节点来编辑许多的重要属性，之后就有一个传统的用户任务编辑器打开：

Properties

Property	Value
ActorId	Sales Representative
Comment	You should call #{customer.name} to confirm the order.
Content	
Id	4
Name	Human Task
On Entry Actions	
On Exit Actions	
Parameter Mapping	{}
Priority	3
Result Mapping	{}
Skippable	true
Swimlane	
TaskName	Call customer
Timers	
Wait for completion	true

Human Task Editor

Name: Call customer

Actor(s): Sales Representative

Comment: You should call #{customer.name} to confirm the order.

Priority: 3

☑ Skippable

Content:

OK Cancel

注意：你能够直接指定不同参数的值，或者基于流程实例内部的数据来指定这些参数。例如，String 类型的参数能够使用#{expression}来嵌入一个 String 值。这个值将会在创建工作条目时被获取，#{...}将会代替 String 的 toString 的值。表达也可以简单的使用变量的名，或者更久高级的 MVEL 表达，例如：{person.name.firstname}。例如，当发送一个邮件的时候，邮件的主体部分能够包含一些像："Dear #{customer.name}的表达。对于其他类型的变量，通过使用参数映射来映射变量的值到参数。

12.1.1 Swimlanes

用户任务节点能够和 Swimlanes 结合来分配多个人工任务来相同的角色。在相同 swimlanes 中的任务将会被分配给相同的参与者。无论在 swimlanes 中的任务什么时候被创建，这个任务已经指定了 actorid，actorid 也会被分配给 swimlanes.之后，会根据 actorid 来创建所有其他的任务，即使 actorid 指定了任务。

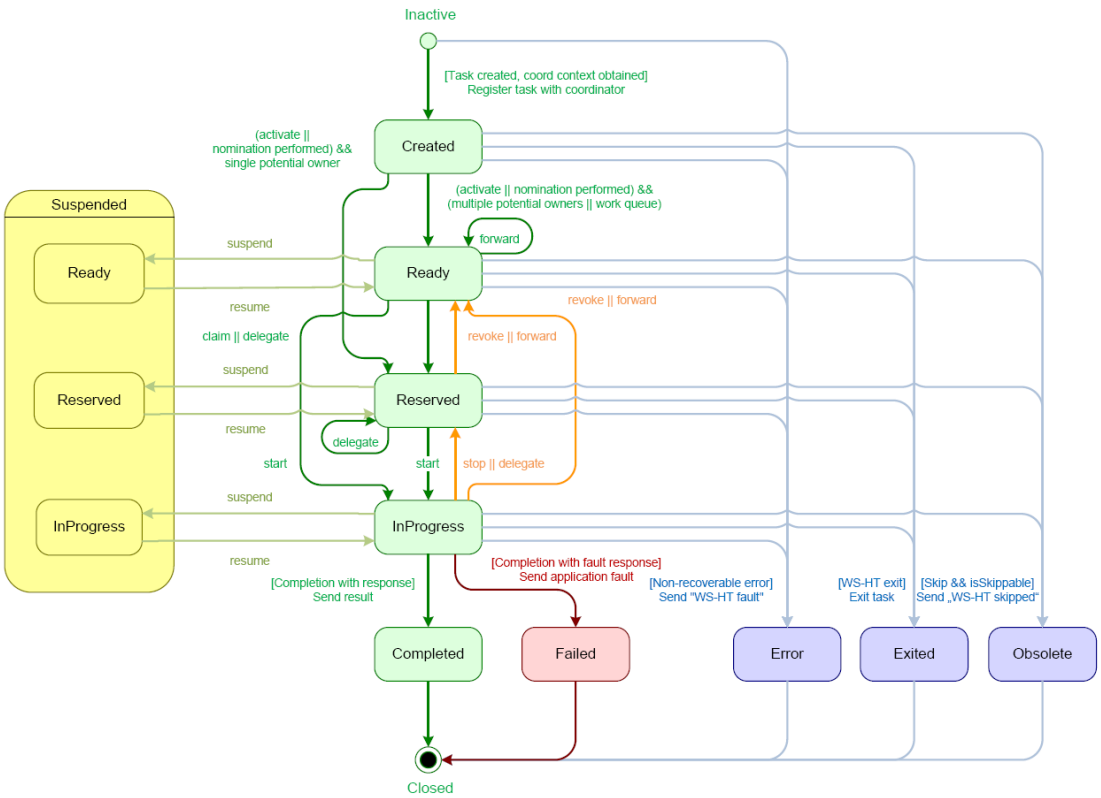
swimlanes 的人工任务无论什么时候被完成，swimlanes 的 actorid 都会被设定为执行人工任务的 actorid。这就允许将人工任务分配给一组用户，并且可以分配 swimlanes 未来的任务给用户。如果在任务的某个点再次被分配给用户，这将会直接自动的更改任务的分配。

将人工任务添加到 swimlanes，只需指定用户任务节点的 Swimlanes 参数为 swimlane 的值即可。当然，这些 swimlane 必须已经在流程中定义了。

12.2 Human task 管理组件

12.2.1 任务生命周期

从流程的视图可以看到，在流程实例的执行过程中，用户任务节点无论什么时候被触发，人工任务都会被创建。当人工任务完成时流程将仅仅从这一点继续执行或终止。无论怎样，人工任务一般都有自己单独的生命周期。现在我们将简短的介绍一下生命周期，像下面展示的图一样。更多的信息请查看 WS-Human Task 规范。



无论一个任务什么时候被创建，它都是从“Created”开始。它会自动转换到“Ready”状态，在这一点的时候，任务将会被展示在所有参与者的任务列表上，并允许参与者执行任务。这里是正在等待这些参与者中一个来认领任务，这就表示他正在执行任务。一旦一个用户认领了一个任务，状态就会更改为“Rserved”。注意：一个任务仅仅有一个潜在的参与者，在自动分配给参与者之前就创建了任务。任务认领任务之后，用户能够在某一点开始执行任务，这种情况下任务的状态会被更改为“Inprigress”。最后，一旦任务完成，用户必须完成任务，这种情况下状态被更改为“Completed”。如果任务没有完成，用户也显示使用了错误的响应，那么执着情况下状态就被更改为“Failed”。

上面介绍的生命周期是正常的生命周期。这个服务还允许很多其他的生命周期方法：

•在委托或者推进一个任务的情况下，任务会被分配给其他的参与者

•撤销一个任务，这个任务不会再被特定的用户认领，但是会在所有潜在的参与者的任务列表中出现

•暂时挂起和重新开始一个任务

•在流程中停止一个任务

•跳过一个任务，这种情况下这个任务将不会被执行

12.2.2 将任务组件链接到 jBPM 引擎

任务管理组件和其他的外部服务一样，需要和 jBPM 引擎结合，通过这次一个工作条目 handler，这个 handler 是用来转换抽象的工作条目到一个特定的调用。在 org.jbpm.process.workitem.wsht.WSHumanTaskHandler 中我们已经实现了 work item handler，所以你可以像下面一样很容易链接到 work item handler：

```
StatefulKnowledgeSession ksession = ...;
ksession.getWorkItemManager().registerWorkItemHandler("Human
Task", new WSHumanTaskHandler());
```

默认的，这个 handler 将会在本地的机器 9123 端口上连接人工任务管理组件，但是你能够简单的通过调用 WSHHuman TaskHandler 上的 setConnection（ipAddress，port）方法来更改。如果你使用持久化 session，你应该使用 org.jbpm.process.workitem.wsht.CommandBasedWSHumanTaskHandler 来确保在和流程引擎链接之后，流程实例的状态能够被正确的持久化。

在人工任务服务和流程引擎或任何任务客户端之间的信息传达都是使用 messages 来在客户端和服务器端直接传递。这个工具允许不同的转换机制被嵌入，但是，默认的，Mina *(http://mina.apache.org/)* [http://mina.apache.org/]被用来客户端/服务器端的信息发送。

一个任务客户端提供了以下方法来管理人工任务的生命周期：

```
public void start( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void stop( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void release( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void suspend( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void resume( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void skip( long taskId, String userId, TaskOperationResponseHandler
responseHandler )
public void delegate( long taskId, String userId, String targetUserId,
TaskOperationResponseHandler responseHandler )
public void complete( long taskId, String userId, ContentData outputData,
TaskOperationResponseHandler responseHandler )
```

你能够直接使用这些方法，也可以使用 GUI 的一些类型，最终用户能够使用 GUI 来查看执行分配给他们的任务。你能够发现几乎所有的方法都有以下的参数：

•taskid：我们正在执行的任务 id。这一般是通过用户接口从用户任务列表中所选的任务获得的。

•userId：正在执行动作的用户 id。这一般就是登陆到应用的用户的 id

•responseHandler：和任务服务的信息交互一般都是异步的，所有你应该在结果是可用的时候使用 response handler 来通报。

你可以想象一下所有的方法创建一个消息并发送到服务器端，然后服务器端将会执行正确动作的逻辑。

12.2.3 启动任务管理组件

任务管理组件是和流程引擎交互的完全独立的服务。因此我们建议将它作为一个单独的服务来启动。installer 包含一个启动任务服务的命令（这种情况下使用 Mina 传输协议），或者你也可以使用下面的代码片段：

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("org.jbpm.task");
TaskService taskService = new TaskService(emf,
SystemEventListenerFactory.getSystemEventListener());
MinaTaskServer server = new MinaTaskServer( taskService );
Thread thread = new Thread( server );
thread.start();
```

任务管理组件使用 Java Persistence API（JPA）来存储所有的任务信息在一个持久化样式中。为了配置持久化，你需要编辑 persistence.xml 配置文件。我们提到了 JPA 文档对于如何去做。下面的片段展现了怎样使用任务管理组件、Hibernate 和一个 H2 数据库：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence
version="1.0"
xsi:schemaLocation=
"http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
http://java.sun.com/xml/ns/persistence/orm

http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/persistence">
```

```
<persistence-unit name="org.jbpm.task">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<class>org.jbpm.task.Attachment</class>
<class>org.jbpm.task.Content</class>
<class>org.jbpm.task.BooleanExpression</class>
<class>org.jbpm.task.Comment</class>
<class>org.jbpm.task.Deadline</class>
<class>org.jbpm.task.Comment</class>
<class>org.jbpm.task.Deadline</class>
<class>org.jbpm.task.Delegation</class>
<class>org.jbpm.task.Escalation</class>
<class>org.jbpm.task.Group</class>
<class>org.jbpm.task.I18NText</class>
<class>org.jbpm.task.Notification</class>
<class>org.jbpm.task.EmailNotification</class>
<class>org.jbpm.task.EmailNotificationHeader</class>
<class>org.jbpm.task.PeopleAssignments</class>
<class>org.jbpm.task.Reassignment</class>
<class>org.jbpm.task.Status</class>
<class>org.jbpm.task.Task</class>
<class>org.jbpm.task.TaskData</class>
<class>org.jbpm.task.SubTasksStrategy</class>
<class>org.jbpm.task.OnParentAbortAllSubTasksEndStrategy</class>
<class>org.jbpm.task.OnAllSubTasksEndParentEndStrategy</class>
<class>org.jbpm.task.User</class>
<properties>
<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
<property name="hibernate.connection.driver_class" value="org.h2.Driver"/>
<property name="hibernate.connection.url" value="jdbc:h2:mem:mydb" />
<property name="hibernate.connection.username" value="sa"/>
<property name="hibernate.connection.password" value="sasa"/>
<property name="hibernate.connection.autocommit" value="false" />
<property name="hibernate.max_fetch_depth" value="3"/>
<property name="hibernate.hbm2ddl.auto" value="create" />
<property name="hibernate.show_sql" value="true" />
</properties>
</persistence-unit>

</persistence>
```

在你第一次启动任务管理组件时，你需要确保所有必需的用户和组都已经被添加到了数据库。在尝试分配任务给用户或组时，我们的工具要求所有的用户和组已经被预定义了。所以你需要确保使用 askSession.addUser(user)和 taskSession.addGroup(group)方法来向数据库添加必需的用户和组。注意：你最后需要添加一个“Administrator”用户来作为管理员角色，而所有的任务都会自动的分配给这个用户。

jbpm-human-task 模块在 src/test/java 源文件文件夹中包含一个 org.jbpm.task.RunTaskService 类，能够被用来启动一个任务服务。它将自己添加在 LoadUsers.mvel 和 LoadGroups.mvel 配置文件中中定义的用户和组。

12.2.4 和任务管理组件的交互

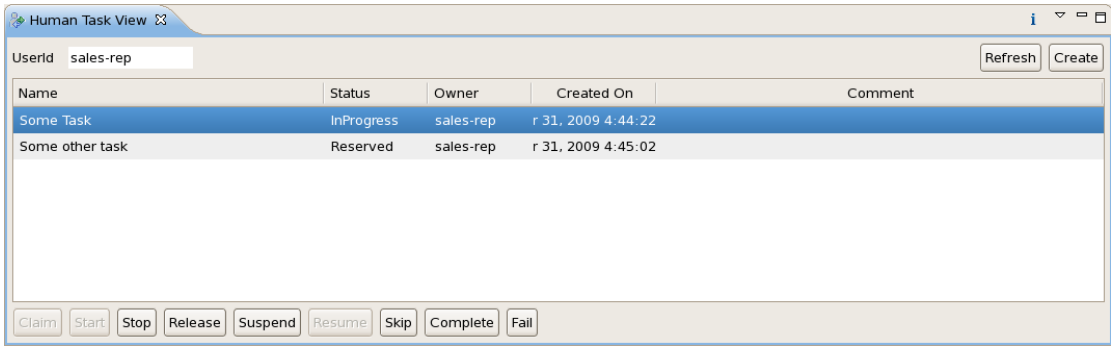
任务管理组件有很多的方法来通过 Java API 管理任务的生命周期。这就允许客户端和任务管理组件结合。注意：最终用户有可能不能够和低级的 API 直接交互，但是可以使用客户端任务列表中一个。这些客户端可以使用 API 和任务管理组件交互。下面的示例代码展示了如何去创建一个任务客户端、和任务服务交互来创建、启动和完成一个任务。

```
TaskClient client = new TaskClient(new MinaTaskClientConnector("client 1",
new MinaTaskClientHandler(SystemEventListenerFactory.getSystemEventListener())));
client.connect("127.0.0.1", 9123);
// adding a task
BlockingAddTaskResponseHandler addTaskResponseHandler = new
BlockingAddTaskResponseHandler();
Task task = ...;
client.addTask( task, null, addTaskResponseHandler );
long taskId = addTaskResponseHandler.getTaskId();
// getting tasks for user "bobba"
BlockingTaskSummaryResponseHandler taskSummaryResponseHandler =
new BlockingTaskSummaryResponseHandler();
client.getTasksAssignedAsPotentialOwner("bobba", "en-UK",
taskSummaryResponseHandler);
List<TaskSummary> tasks = taskSummaryResponseHandler.getResults();
// starting a task
BlockingTaskOperationResponseHandler responseHandler =
new BlockingTaskOperationResponseHandler();
client.start( taskId, "bobba", responseHandler );
// completing a task
responseHandler = new BlockingTaskOperationResponseHandler();
client.complete( taskId, users.get( "bobba" ).getId(), null, responseHandler );
```

12.3 人工任务管理接口

12.3.1 Eclipse integration

Drools IDE 包含一个 org.drools.eclipse.task 插件，允许你使用人工任务来测试或调试流程。人工任务视图能够连接运行中的任务管理组件，请求特定的用的相关任务。这些任务的生命周期能够被执行，认领或发布一个任务，启动或停止一个任务的执行，完整一个任务。下面是一个人工任务视图的截屏。你能够配置任务管理组件连接到 Drools Task(select Window -> Preferences and select Drools Task).，这里你能够指定 url 和端口号（默认=127.0.0.1:9123）



12.3.2 基于 Web 的任务视图

jBPM 控制台包含一个任务视图，用来查看任务列表并管理这些任务的生命周期。更多的信息请查看 jBPM 控制台章节。

第十三章 特定领域的流程

13.1 介绍

jBPM 的一个目标就是允许用户用特定领域扩展来扩展默认的流程构造，特定的领域扩展就是简单的在一个特定领域的应用中开发。这个指南描述了开发特定领域的流程的第一步。注意：你不需要是一个 jBPM 专家就能够定义你自己的特定领域节点，这应该是一个综合的代码，只要是有一些 jBPM 开发的经验就能够自己完成。

很多的流程语言提供一般的动作（节点）建造，这允许嵌入在传统的用户动作。然而这些动作一般都是低等级的，用户需要去写传统的代码来实现在流程中合为一起的工作。代码也必须紧紧链接到特定的目标环境，这就是使在不同的上下文中中重用流程变的非常困难。

特定领域语言的目标是针对于特定应用领域，因此能够提供和用户尝试解决的问题紧紧相关的构造。这就使得流程更容易理解。我们将展现给你如何定义特定领域工作条目，这代表需要被执行的工作原子单元。这些服务接待将指定在发布的行为流程的上下文中将有执行测工作，并指出了在高一级需要执行的内容并隐藏了实现细节。

所有我们想要的服务节点有：

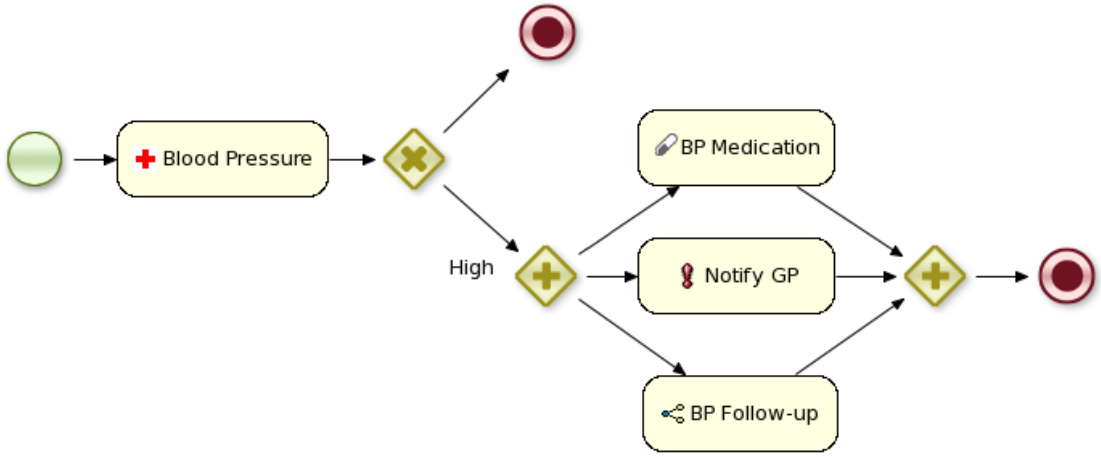
1. domain-specific

2. declarative (what, not how)

3. high-level (no code)

4. customizable to the context

用户能够很容易的定义他们自己的特有领域服务节点并和我们的流程语言相结合。例如，下面的图就展现了一个在卫生保健上下文中一个流程示例。这个流程包括特有领域的服务节点，这些节点用来分配看护任务，处方药物处理和通知病患家属。



13.2 示例：通告

让我们通过展现怎样包括一个用来发送通告的工作条目来开始。一个工作条目代表着一个在公布的路线中的原子工作单元。它通过唯一的名称和额外的参数来定义，额外的参数是用来描述工作的细节。工作条目能够在他们执行之后返回信息指定为结果。我们的工作条目就是使用一个工作定义和四个参数并没有返回结果：

```
Name: "Notification"
Parameters
From [String]
To [String]
Message [String]
Priority [String]
```

13.2.1 创建工作定义

所有的工作定义必须指定在项目的 classpath 中的一个或多个配置文件，所有的属性都是作为名-值对来定义。参数和结果是相互映射的，每一个参数名都和所期望的数据类型想映射。注意：这个配置文件包含一些额外的用户接口信息，像图标和工作条目的显示名称。

在我们的示例中我们将使用 MVEL 来读取配置文件，这允许我们去做更高级的配置文件。宰割文件必须放在项目的类路径的 META-INF 的文件夹中，我们的 MyWorkDefinitions.wid 文件：

```
import org.drools.process.core.datatype.impl.type.StringDataType;
[
// the Notification work item
[
"name" : "Notification",
"parameters" : [
"Message" : new StringDataType(),
"From" : new StringDataType(),
"To" : new StringDataType(),
"Priority" : new StringDataType(),
],
"displayName" : "Notification",
"icon" : "icons/notification.gif"
]
]
```

项目的目录结构像下面这样：

```
project/src/main/resources/META-INF/MyWorkDefinitions.wid
```

现在你可能想创建你自己的图标来附和你的工作定义。你需要像素为 16×16 的.gif 或者.png 的图形文件。他们的存放地址：

```
project/src/main/resources/icons/notification.gif
```

13.2.2 注册你的工作定义

配置 API 使用 drools.workDefinitions 属性来为你的项目注册工作定义文件，这个代表了包含工作定义的文件列表。例如，在你项目的 META-ING 文件夹下包含一个 drools.rulebase.conf 文件并添加下面的一行：

```
drools.workDefinitions = MyWorkDefinitions.wid
```

这将会用你的流程编辑器中新定义的通告节点来代替默认的领域 EMAIL 和 LOG 节点。你应该希望添加最新创建的节点定义到已经存在的画板节点中，调用 drools.workDefinitions 属性来设置配置文件。

```
drools.workDefinitions = MyWorkDefinitions.conf WorkDefinitions.conf
```

13.2.3 在你的流程中使用你最新的工作条目

一旦我们的工作定义已经创建并注册，我们就能够在我们的流程中使用了。在流程编辑器的画板中包含一个分离的片段，对于项目中出现的不同服务的节点已经被定义了。

通告拖拽就可以在你的流程中使用 notification 节点了。它的属性可以在属性视图中填写。

处理工作条目属性的定义，所有的工作条目也应该有三个属性：

1. Parameter Mapping:

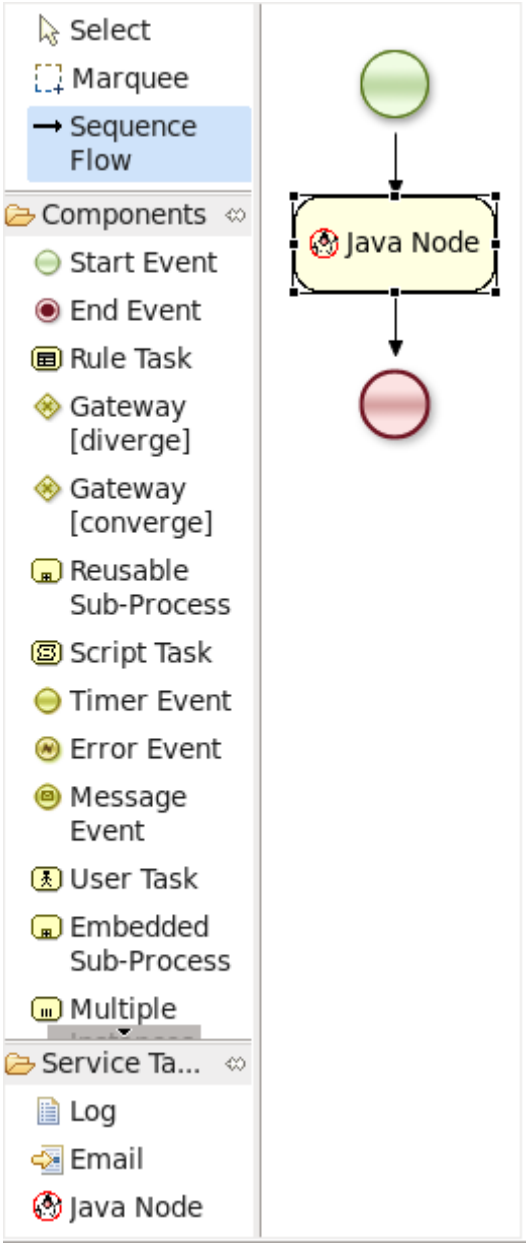
2. 2. Result Mapping

3. Wait for completion

这里有一个示例，创建了一个特定领域节点来执行 Java，访问类和方法参数。它包括传统的 java.gif 图标和下面的文件和结果组成：

```
import org.drools.process.core.datatype.impl.type.StringDataType;
[
// the Java Node work item located in:
// project/src/main/resources/META-INF/JavaNodeDefinition.conf
[
"name" : "JavaNode",
"parameters" : [
"class" : new StringDataType(),
"method" : new StringDataType(),
],
"displayName" : "Java Node",
"icon" : "icons/java.gif"
]
]
```

```
// located in: project/src/main/resources/META-INF/drools.rulebase.conf
//
drools.workDefinitions = JavaNodeDefinition.conf WorkDefinitions.conf
// icon for java.gif located in:
// project/src/main/resources/icons/java.gif
```



13.2.4 执行服务节点

jBPM 引擎包含一个在需要时执行 work items 的 WorkItemManager。WorkItemManager 是用来将 work items 委托给 WorkItemHandler 来执行 work item 并当 work item 完成的时候通知 WorkItemManager。为了能够通告执行的 work items，应该创建一个 NotificationWorkItemHandler。

```
package com.sample;
import org.drools.runtime.process.WorkItem;
import org.drools.runtime.process.WorkItemHandler;
import org.drools.runtime.process.WorkItemManager;
public class NotificationWorkItemHandler implements WorkItemHandler {
public void executeWorkItem(WorkItem workItem, WorkItemManager manager) {
// extract parameters
String from = (String) workItem.getParameter("From");
String to = (String) workItem.getParameter("To");
String message = (String) workItem.getParameter("Message");
String priority = (String) workItem.getParameter("Priority");
// send email
EmailService service = ServiceRegistry.getInstance().getEmailService();
service.sendEmail(from, to, "Notification", message);
// notify manager that work item has been completed
manager.completeWorkItem(workItem.getId(), null);
}
public void abortWorkItem(WorkItem workItem, WorkItemManager manager) {
// Do nothing, notifications cannot be aborted
}
}
```

这个 WorkItemHandler 以 email 的形式发送一个通告，并在 work item 完成的时候直接通告给 WorkItemManager。注意：并不是所有 work item 都能够直接完成。这种情况下执行一个 work item 就要花费一点时间了，以异步的形式继续执行，并且 work itemmanager 也在稍后执行。这种情况可能导致在 work item 没有执行完成之前就中止了。中止方法指定了如何中止这种工作条目。

使用下面的 API，将 WorkItemHandlers 注册到 WorkItemManager：

```
ksession.getWorkItemManager().registerWorkItemHandler(
"Notification", new NotificationWorkItemHandler());
```

流程本身执行 work item 的装置有以下优势：

1. 流程更加易于描述，指定应该执行什么
2. 改变环境只需通过采用适合的 work item handler 来执行。而流程本身不需要改变。在不同的环境中使用同一个流程也是可能的，work item handler 用来负责与正确的服务相结合。
3. 通过流程和项目是很容易将 work item handler 共享

4. 不同的 work item handler 需要依赖上下文。例如，在测试或模拟当中，可能不需要实际的去执行 work item。这种情况下，在测试当中应该指定一个虚拟的 work item handler。

第十四章 测试和调试

即使业务流程没有执行代码，但是他们还是有生命周期的。并去由于业务流程能够被动态的升级，这就使测试他们变得非常重要。

14.1 单元测试

当进行单元测试的时候，你要测试在特定的使用情况下流程的行为是否按照期望的那样，例如，输出已经输入的数据。对于特定的单元测试，jBPM 包括一个叫做 bpmJUnitTestCase 的帮助类，你能够通过下面所提供的非常简单的进行单元测试：

•helper 方法基于给定的流程创建一个新的 knowledge base 和 session

•你能够选择是否使用持久化

•通过断言来核对

•流程实例的状态

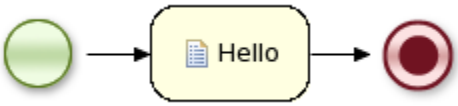
•节点实例当前是否为激活状态

•哪些节点被触发

•得到变量的值

•等待

例如，下面的 hello world 流程包含一个开始事件，一个脚本任务和一个终止事件。接下来的单元测试将会创建一个新的 session，启动一个流程然后核查流程实例是否成功完成并且这三个节点是否已经执行。



```
public class MyProcessTest extends JbpmJUnitTestCase {
public void testProcess() {
// create your session and load the given process(es)
StatefulKnowledgeSession ksession = createKnowledgeSession("sample.bpmn");
// start the process
ProcessInstance processInstance = ksession.startProcess("com.sample.bpmn.hello");
// check whether the process instance has completed successfully
assertProcessInstanceCompleted(processInstance.getId(), ksession);
// check whether the given nodes were executed during the process execution
assertNodeTriggered(processInstance.getId(), "StartProcess", "Hello", "EndProcess");
}
}
```

14.1.1 使用 Helper 方法创建你的 session

提供的几个方法能很简单的创建一个 knowledge base 和 session 来和引擎交互。

• *createKnowledgeBase(String... process)*:返回一个新的 knowledge base 包含了给出的文件名的所有流程

• *createKnowledgeBase(Map<String, ResourceType> resources)*：返回一个新的 knowledge base 包含了给出的文件名的所有的资源

• *createKnowledgeBaseGuvnor(String... packages)*: 返回一个新的 knowledge base 包含所有从 Guvnor 加载的流程

• *createKnowledgeSession(KnowledgeBase kbase)*: 创建一个 knowledge session 基于给出的 knowledge base

• *restoreSession(StatefulKnowledgeSession ksession, boolean noCache)*：将 session 完全的存储在数据库中，能够用来重建一个 session 来模拟可鉴定的错误和测试恢复，如果 noCache 是 true 的话，存在的 persistence cache 将不会用来存储数据。

14.1.2 断言

下面被添加的断言能简单的对流出实例的当前状态进行测试：

• *assertProcessInstanceActive(long processInstanceId, StatefulKnowledgeSession ksession)*: check whether the process instance with the given id is still active

• *assertProcessInstanceCompleted(long processInstanceId, StatefulKnowledgeSession ksession)*: check whether the process instance with the given id has completed successfully

• *assertProcessInstanceAborted(long processInstanceId, StatefulKnowledgeSession ksession)*:

check whether the process instance with the given id was aborted

• *assertNodeActive(long processInstanceId, StatefulKnowledgeSession ksession, String... name)*: check whether the process instance with the given id contains at least one active node with the given node name (for each of the given names)

• *assertNodeTriggered(long processInstanceId, String... nodeNames)* : check for each given node name whether a node instance was triggered (but not necessarily active anymore) during the execution of the process instance with the given

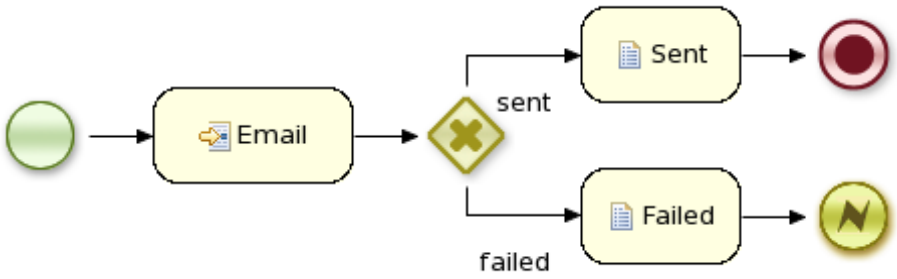
• *getVariableValue(String name, long processInstanceId, StatefulKnowledgeSession ksession)*:retrieves the value of the variable with the given name from the given process instance, can then be used to check the value of process variables

14.1.3 测试与外部服务的结合

真正的业务流程的典型代表就是要包含对外部服务的调用。我们的特定领域流程方法的优点之一就是你能够指定怎样实际的去执行你自己的特定领域的节点，不过需要你注册一个 handler。这个 handler 依赖于你的上下文而定，handler 允许你使用测试 handlers 来测试你的流程。当你对你的业务流程进行单元测试的时候，你能够注册一个测试 handlers 来核实特定服务是否被正确的请求，并对这些服务提供测试响应。例如，想象一下你有一个 email 节点或一个人工任务来作为你流程的一部分。当单元测试时，你不能够发出一个实际的 email，但是可以测试被请求的 email 是否包含正确的信息。

默认的 TestWorkItemHandler 能够被注册用来收集所有指定类型的 work item。这个测试 handler 可以用来查询，在在流程的执行中，测试检查特定的工作是否实际上被请求和和工作相关的数据是否正确。

下面的试了描述了一个向外发送 email 的流程能够被测试。这个测试用例将详细的测试在 email 不能发送的时候是否有异常发声。使用了 handler 的测试用例在 email 被请求的时候会很简单侧重。一旦引擎已经通告 email 不能发送，测试单元就会通过登录并产生一个错误来成功的验证流程处理这种情况，在这种情况下中止流程实例。




```
public void testProcess2() {
// create your session and load the given process(es)
StatefulKnowledgeSession ksession = createKnowledgeSession("sample2.bpmn");
// register a test handler for "Email"
TestWorkItemHandler testHandler = new TestWorkItemHandler();
ksession.getWorkItemManager().registerWorkItemHandler("Email", testHandler);
```

14.1.4 配置持久化

你能够配置在执行单元测试是否使用持久化。默认的，单元测试使用持久化，也就意味着所有流程实例的状态都将被存储在数据库中，并且历史日志将被用来检查和断言相关的执行历史。当持久化不能使用时，流程实例仅仅存在于内存中，并且内存 logger 被用来存储历史断言。

默认的，持久化是配开始的。要关闭持久化，只需在创建一个测试示例时向上一级的构造器传递一个布尔值即可。

```
public class MyProcessTest extends JbpmJUnitTestCase {
public MyProcessTest() {
// configure this tests to not use persistence in this case
super(false);
}
```

14.2 调试

这一部分描述了怎样使用 Eclipse 插件来调试流程。这就意味着你运行中的当前状态能够被查看并在执行中可视化。注意：我们目前不允许你在一个节点上添加断点。但是你能够将断点添加在任何的 java 代码中或者 rule 中。在这些段子中，你可以查看你流程实例的内部状态。

当调试应用时，你可以使用下面的视图来跟踪流程的执行：

1. The process instances view, showing all running process instances (and their state). When double-clicking a process instance, the process instance view visually shows the current state of that process instance at that point in time.

2. The human task view, showing the task list of the given user (fill in the user id of the actor and click refresh to view all the tasks for the given actor), where you can then control the life cycle of the task, for example start and complete it.

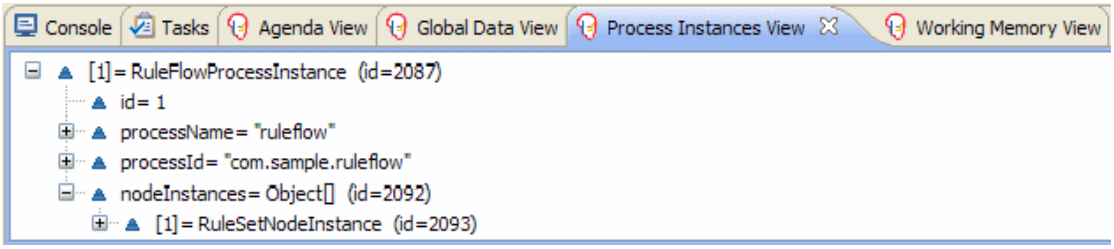
3. The audit view, showing the audit log (note that you should probably use a threaded file logger if you want to session to save the audit event to the file system on regular intervals, so the audit view can be update to show the latest state).

4. The global data view, showing the globals.

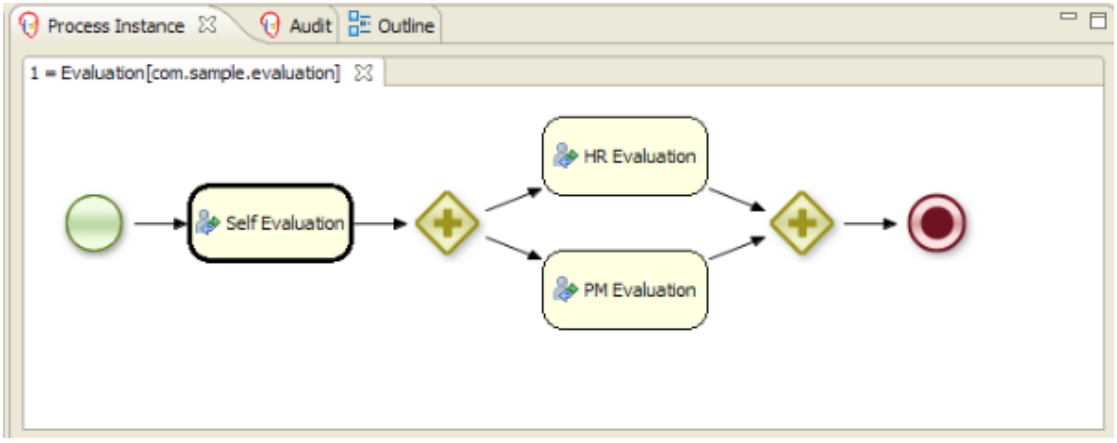
5. Other views related to rule execution like the working memory view (showing the contents (data) in the working memory related to rule execution), the agenda view (showing all activated rules), etc.

14.2.1 流程实例视图

流程实例视图展现了当前运行中的流程实例。示例展现了当前一个正在运行的流程实例，当前正在执行的一个节点实例，业务规则任务。当双击一个流程实例时，流程实例视图将会以图形化的形式展现流程实例的前进。示例中，流程实例正在等待一个人工角色来处理一个自我评估任务。



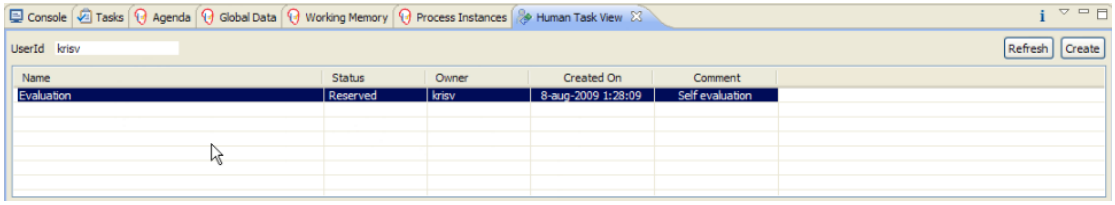
当你双击在流程实例视图中的一个流程实例时，流程实例视图报出不能找到流程，这就意味着插件不能够在解析流程定义的缓存中找到所选的流程实例的流程定义。为了解决这个问题，只需更改流程的定义并重新保存，或者重构包含流程定义问题的项目。



14.2.2 人工任务视图

人工任务视图能连接运行中的人工任务服务并请求和特定用户相关的任务。这些任务的生命周期稍后就被执行，认领或发布任务，启动或停止一个任务的执行，完成一个任务。

下面所展示的是人工任务视图的截屏。你能够配置任务服务连接 Drools Task 属性页。(select Window -> Preferences and select DroolsTask).这里你能够指定 url 和端口号 (default = 127.0.0.1:9123)。



14.2.3 核查视图

核查视图展现了核查日志，是所有事件的日志，记录来自于 session。创建一个 logger，要使用 KnowledgeRuntimeLoggerFactory 来创建一个新的 logger 并配属于一个 session。注意：如果你想要用 session 来每隔固定的时间就保存核查事件到文件系统，你应该使用线程文件 logger，所以核查视图能够被更新来展示最新的状态。当创建一个线程文件 logger 时，你能够指定核查日志的名字，之后事件应该被保存在文件中。要确保在记录完之后 关闭 logger。

```
KnowledgeRuntimeLogger logger = KnowledgeRuntimeLoggerFactory
.newThreadedFileLogger(ksession, "logdir/mylogfile", 1000);
// do something with the session here
logger.close();
```

打开核查视图的核查树，在核查视图中选择日志文件或者简单的将文件拖进核查视图。一个基于树的视图将在核查日志的基础上创建。如果子事件被父事件引发，那么一个事件就作为另一个事件的子节点。下面就是一个例子。

```
RuleFlow started: ruleflow[com.sample.ruleflow]
  RuleFlow node triggered: Start in process ruleflow[com.sample.ruleflow]
    RuleFlow node triggered: Hello in process ruleflow[com.sample.ruleflow]
      RuleFlow node triggered: End in process ruleflow[com.sample.ruleflow]
        RuleFlow completed: ruleflow[com.sample.ruleflow]
```

第十五章 流程知识库

如果你想在你的应用中启动多个业务流程，特别是你想要有能力动态更新他们，一个流程知识库将是你的 BPM 体系的重要组成部分。流程知识库是用来存储和管理你的业务流程。因为他们不能够作为你应用的一部分，因为他们有自己的生命周期，意味着你能够动态的更新业务流程， 不能改变应用代码。

注意：流程知识库不仅仅是一个用来存储你流程定义的数据。它几乎扮演着源代码管理系统、内容管理系统、协作组件、开发和测试环境的结合体的角色。下面是来自流程知识库的你所期望的特性种类：

• Persistent storage of your processes so the latest version can always easily be accessed from anywhere, including versioning

• Build and deploy selected processes

• User-friendly (web-based) interface to manage, update and deploy your processes (targeted to business users, not just developers)

• Authentication / authorization to make sure only people that have the right role can see and/or edit your processes

• Categorization and searching

• Scenario testing to make sure you don't break anything when you change your process

• Collaboration and other social features like comments, notifications on change, etc.

• Synchronization with your development environment

实际上，knowledge 知识库不仅仅作为存储流程定义的知识库来讨论要更好一点，还应该和流程表，任务表，流程图像结合来讨论。无论任何必需的时候，控制台都会被配置从 Guvnor 得到所有的信息并展示他们在控制台上。

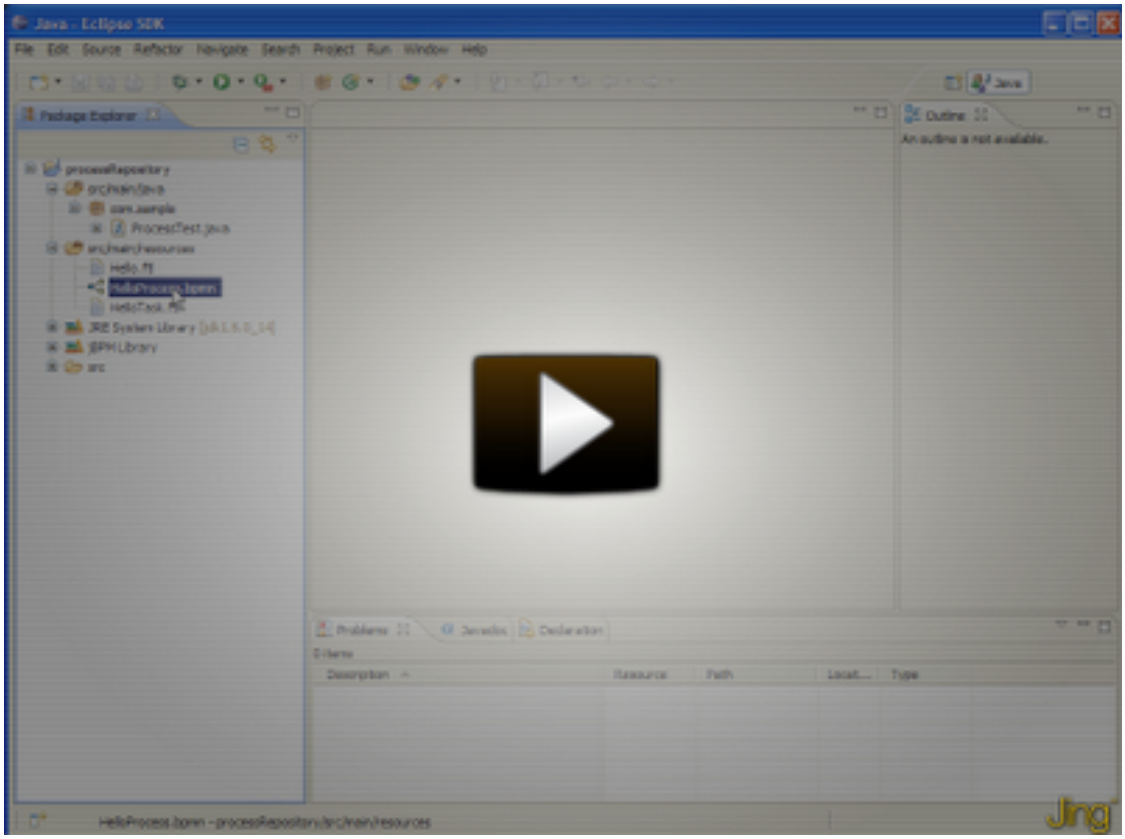


图 15.1 [http://people.redhat.com/kverlaen/jBPM5-guvnor-integration.swf]

如果你使用 installer，它会自动下载并安装最新的 Guvnor 版本。简单配置你的资源到 Guvnor，并创建包和启动控制台。

当前和 jbpm-console 的结合使用下面的命名规范来找到它所需要的 artefacts(不知道怎么译，理解为流程图)：

•所有的流程都应该配置到 Guvnor 上的"defaultPackage"

•一个流程应该定义在包名为“defaultPackage”的包中，不然你不能够在 Guvnor 上创建你的包

•在打开控制台之前不要忘了在 Guvnor 上创建包，一旦你创建了包，Guvnor 将仅仅发布拟流程的最新版本。

•通常的，控制台将会在控制台第一次请求流程列表的时候加载流程定义。在这一点上，包被重建或被关闭，默认的会自动的从 Guvnor 更新，所以你将必须配置这个或者重启应用服务来得到最新的版本。

•任务表应该和特定的流程定义相关联，流程定义应该命名为"{processDefinitionId}.ftl"

•一个特定人工任务的任务表应该命名为"{taskName}.ftl"

•对于特定流程的流程图应该命名为"{processDefinitionId}-image.png"

如果你遵循了这些规则，你的流程，表和图像都应该展现在控制台上，而不会出现任何问题。

第十六章 业务活动监视器

你需要积极的监控你的流程来确保你能够察觉到任何的异常，并对不期望的事件尽可能快的的做出反应。业务活动监视器和你的流程监视器与干预选项有直接的关系，甚至可能自动的，基于这些事件来分析。

jBPM 允许用户通过流程引擎并基于事件的产生来定义报告，并且使用复杂事件流程规则可能直接的介入到特定的情形中，就像下面两部分介绍的一样。未来jBPM 平台的发布将包括对所有业务活动监视器需求的支持，包括能够被用来和运行中流程引擎交互，查看它的状态，产生报告等等的基于 web 的应用。

16.1 报告

通过向流程引擎添加一个历史日志，所有的相关事件都被存储在数据库中，这个日志能够被用来监视和分析你流程的执行。我们正在使用 Eclipse BIRT 来创建报告来展示关键的执行显示器。你自己可以很容易定义你自己的报告，通过使用包含所有流程历史信息和其他任何你想要添加的数据资源的预定于数据集。

Eclipse BIRT 框架允许你定义数据集，创建报告，包括表，预览，你的报告并输出它们到一个 web 页。下面的截图展示了如何去创建一个表的示例。

火龙果•整理
uml.org.cn

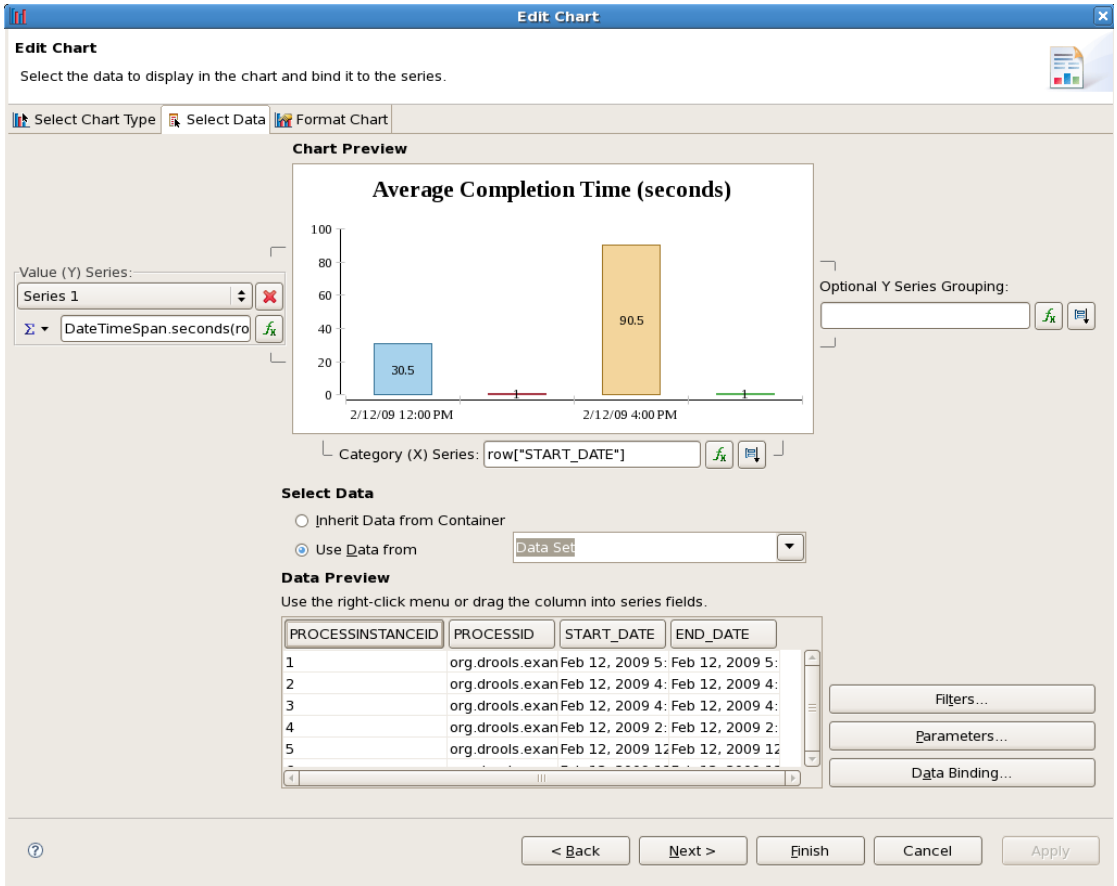


图 16.1 使用 Eclipse BIRT 创建一个报告

下面的图展示了一个基于历史数据的一个简单报告，展示了每小时请求的数量和在那一个小时内完成请求的平均时间。这些表能够被用来检查不期望的下降或生气请求，平均处理时间的上升等等。这些表也能够在情形真正的脱离控制之前对可能的问题发出信号。

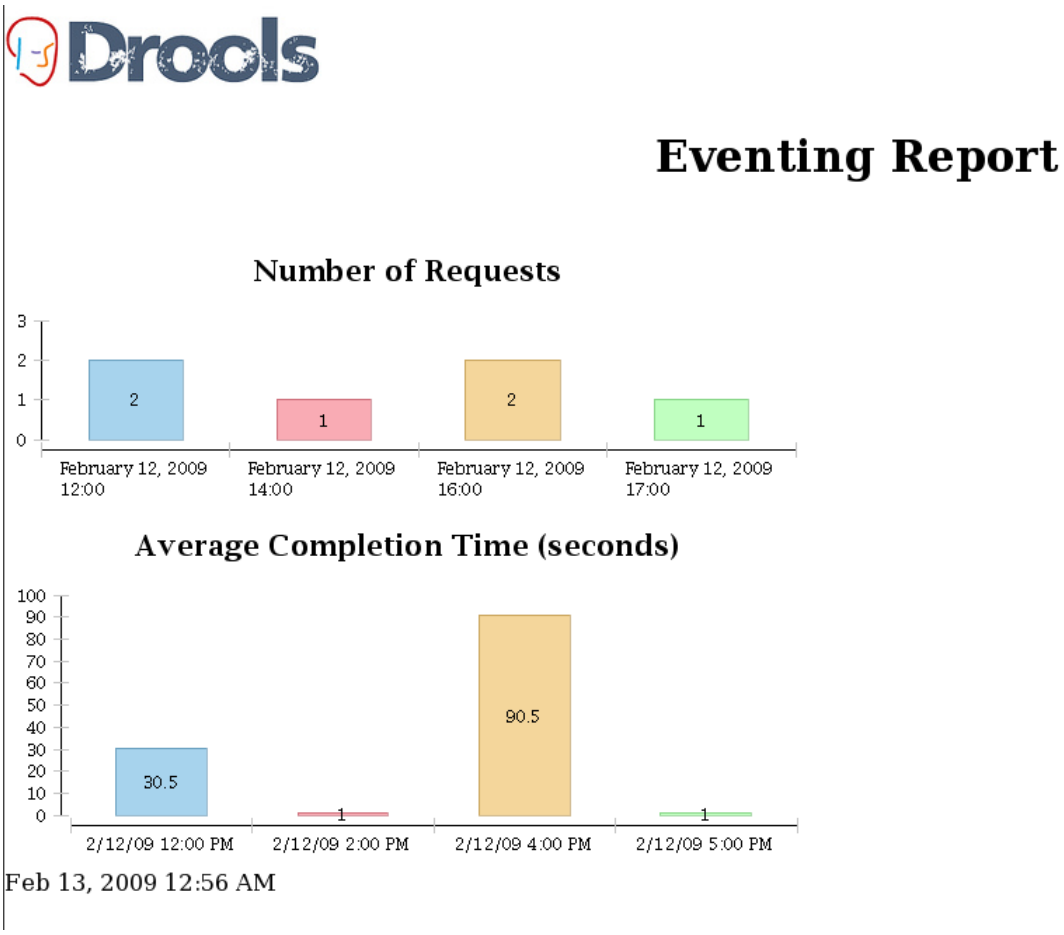


图 16.2 事件报告

16.2 直接接入

报告能够被用来显示你流程的当前状态的总的情况，但是它们要依赖一个人工角色，人工角色基于这些表中信息来采取行动。无论怎样，我们允许用户对于特定的情况来定义自动的响应。

Drools Fusion 提供了大量的特性，这些特性使得处理大量的事件集变得很容易。这个能够用来监视流程引擎的自身。这可以通过添加一个监视器到引擎来获得，包括引擎向前的所有相关流程事件，例如，开始和流程实例的完成，或者特定节点的触发，负责处理这些事件的 session。这可能是同一个 session 来执行这些流程，也可能是单独的 session 来执行每一个流程。Complex Event Processing（CEP）复杂事件流程规则用来指定怎样处理这些事件。例如，这些规则能够基于一个特定的低级流程事件的发生来产生一个高级的业务事件。规则也可以指定对于特定的情形如何响应。

下面的部分展示了一个简单的规则，这个规则积聚一个特定订单的开始流程事件，并使用"sliding window"（滑动窗口）来在最后一个小时进行处理。如果在最后一个小时有多余 1000 个流程实例被启动，这个规则就打印出一个错误消息。注意：在一个现实的设置中，这可以通过发送一个 email 或其他的通告表来代替消息。

```
declare ProcessStartedEvent
@role( event )
end
dialect "mvel"
rule "Number of process instances above threshold"
when
Number( nbProcesses : intValue > 1000 )
from accumulate(
e: ProcessStartedEvent( processInstance.processId ==
"com.sample.order.OrderProcess" )
over window:size(1h),
count(e) )
then
System.err.println( "WARNING: Number of order processes in the last hour above 1000: " +
nbProcesses );
end
```

这些规则甚至能够被用来在运行时来自动的改变流程的行为，这要通过引擎并基于事件的产生。例如，一个特定的情形无论什么时候被发觉，额外的规则能够被添加到 Knowledge Base 来编辑流程的处理。例如，在一个特定的时间框内有大量的用户请求的情况无论什么时候被发现，一个额外的验证都能够添加到那个流程，并强迫一些流程控制的类型减少进入请求的频率。通过配置额外的日志规则来作为发现问题的结果也是可行的。当这种情形回到了正常之后，这个规则就会被再次删除。

第十七章 复杂流程

在今天，事例管理和它相关的问题对于 BPM 来说是一个热门的话题。在最终用户之间对更加灵活和适应的业务流程的需求似乎有明确的增长，以极度复杂的情形结束的除外。每一个人似乎都同意：使用一个接近于很多事例的流程中心将导致复杂的流程会难以维持。"knowledge workers"不再想被锁入一个严格的流程当中，但是它想有能力和机动性来重新恢复对流程自身的更多控制。

条件事例管理经常被用在 context（上下文中）。没有尝试着对于什么可能或不可能而给出明确的定义，这种情况已经成为一个热门的主题，它的基础观点来自于在现实世界中许多的应用都不可能从开始到结束被完全的描述出来。事例管理有点点接近：代替了尝试建模从开始到结束什么应该发生，我们让最终用户拥有灵活性，在运行时自己决定什么应该发生。在许多的极端形式中，事例管理甚至不需要任何的流程定义。无论一个新的事例什么时候发生，最终用户能够基于所有的实例数据来决定下面应该做什么。

healthcare 卫生保健就是一个典型的例子，照顾计划被用来描述在特定的情况下患者应该怎样被对待，但是人们所喜欢的普通从业者仍然需要拥有灵活性来添加额外的步骤并和先前的计划相背离，作为每一个事例都是唯一的。并且在权限管理，帮助支持中都有相似的例子。

所有，我们应该扔掉我们的 BPM 系统吗？答案当然是 NO！即使在很多的极端情况下，我们仍然需要 BPM 系统所提供的很多其他特性：对于审核日志，监视器，协调各种服务，人工交互，分析等等仍然有明确的需求。更重要的是，学到事例处于这中间的某一处，或者随着时间的流逝，事例管理甚至有可能进化到更具结构性的业务流程。如果我们能够使我们的流程具有灵活性，那我们能否让用户来决定他们所喜欢的怎样和在哪应用它。

让我用两个示例来给你展示一下你能够添加很多灵活性到你的流程。第一个示例展示了一个照料计划，展示了一个病人在高血压的时候应该执行什么任务。然而流程的很大一部分还是拥有良好的结构的，一个普通的从业者自己就能决定作为子流程的一部分的哪一个任务应该被执行。在这时期，它也有能力添加新的任务，而添加的任务在流程中没有定义，或者任务重复多次等等。这种流程使用一种 ad-hoc 的子流程来建模这种的灵活性，在决定哪一个片段要执行中，也可能扩展规则或事件处理到帮助中。

图 17.1

第二个示例实际上比上面那个使用了更多的特性。在这个示例中，一个网络提供者能够定义关于网络连接问题的事例通过网络提供者将会怎么被处理。这里有很多的动作，而事例工作者能够从这里面选择，但是这都是些小的简单流程片段。事例工作者负责选择下一步要做什么，甚至能够动态的添加新任务。作为你所看到的，从开始到结束再也没有流程，但是用户负责选择哪一个处理片段来执行。

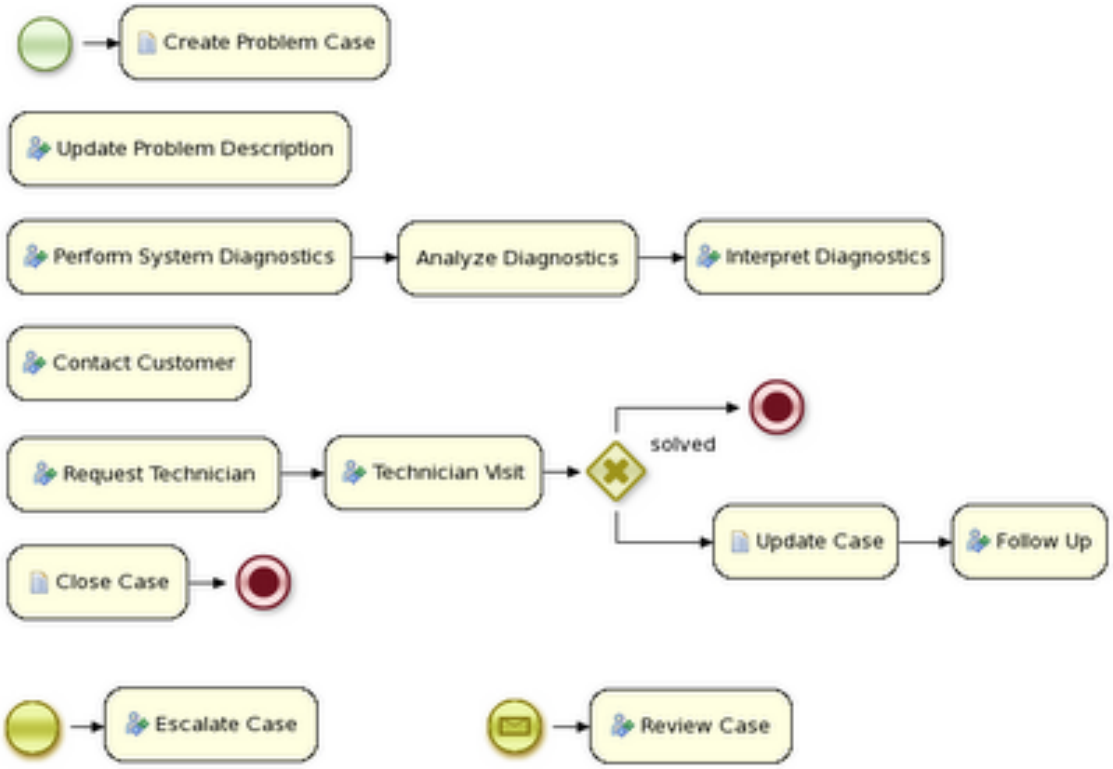


图 17.2

在许多的极端情况下，我们甚至允许你在没有流程定义的情况下创建一个事例实例，在运行时，需要什么被执行将被直接的选择。但是这并不意味着你能够不描绘出实际中所发生的事情。例如，会议是 adhoc 和动态的，但是我们经常需要一个实际中讨论的内容的日志。下面的截屏展示了我们的常规审核视图仍然在这些事例中使用，之后最终用户能够得到很多和实际中发生的事情相关的信息，这要通过查看和这些步骤的每一个相关的数据。或者随着时间的流程，我们甚至能够通过使用一个半结构流程成为流程的一部分并自动化。

图 17.3

第十八章 和 Maven，OSGi，Spring 的整合

jBPM 能够和其他很多技术整合。这一章给出部分的概览。这些当中的很多模块都是作为 droolsjbpm-intergration 模块的一部分被开发，所有他们不仅仅为你的业务流程而且也为业务规则和复杂的事件处理。

18.1 Maven

通过使用 Maven 配置文件 pom.xml 来定义你项目的依赖，你能够让 maven 为你得到你的依赖。下面的 pom.xml 文件就是一个能够被用来创建支持执行 BPMN2 流程的 Maven 项目：

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
maven-v4_0_0.xsd">

<modelVersion>4.0.0</modelVersion>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-maven-example</artifactId>
<name>jBPM Maven Project</name>
<version>1.0-SNAPSHOT</version>
```

```
<repositories>
<!-- use this repository for stable releases -->
<repository>
<id>jboss-public-repository-group</id>
<name>JBoss Public Maven Repository Group</name>
<url>https://repository.jboss.org/nexus/content/groups/public/</url>
<layout>default</layout>
<releases>
<enabled>true</enabled>
<updatePolicy>never</updatePolicy>
</releases>
<snapshots>
<enabled>false</enabled>
</snapshots>
</repository>
<!-- use this repository for snapshot releases -->
<repository>
<id>jboss-snapshot-repository-group</id>
<name>JBoss SNAPSHOT Maven Repository Group</name>
<url>https://repository.jboss.org/nexus/content/repositories/snapshots/</url>
<layout>default</layout>
<releases>
<enabled>false</enabled>
</releases>
<snapshots>
<enabled>true</enabled>
<updatePolicy>never</updatePolicy>
</snapshots>
</repository>
</repositories>
<dependencies>
<dependency>
<groupId>org.jbpm</groupId>
<artifactId>jbpm-bpmn2</artifactId>
<version>5.0.0</version>
</dependency>
</dependencies>
</project>
```

使用这个配置文件来作为 Eclipse 中项目的基础，也可以使用 M2Eclipse 或者使用"mvn eclipse:eclipse"来基于这个 pom 产生 eclipse .project 和 .classpath 文件

18.2 OGSi