PROJETO FRAMEWORK – CELEPAR

Definição de Motor de Workflow para Sistemas Corporativos

Grupo de Trabalho FRAMEWORK

Novembro/2005

Sumário de Informações do Documento		
Tipo do Documento: Definição		
Título do Documento: Definição de Motor de Workflow para Sistemas corporativos		
Estado do Documento: EB (Elaboração)		
Responsável: Renan Marcel Cardoso Baggio		
Palavras-Chaves: workflow; jbpm; motor		
Resumo:		
Número de páginas: 28		
Software utilizados: OpenOffice Writer1.1.3		
Versão	**Data**	**Mudanças**
1.0	24/08/05	Versão inicial por Renan Marcel Cardoso Baggio
	21/11/05	Inclusão de maiores informações sobre jBPM.

Sumário

1.Introdução....4

2 Motores de Workflow disponíveis....4

2.1 JBPM....5

2.1.1 Definição....5

2.1.2 Características....6

2.1.3 Funcionamento....7

2.1.4 Script....12

2.1.5 Superstates....13

2.1.6 Instâncias de tarefas....14

2.1.7 Exceções....15

2.1.8 Processos compostos....16

2.1.9 jBPM Process Definition Language (JPDL)....17

2.1.10 Versionamento de Processo....18

2.1.11 Classe de carregamento jBPM....18

2.1.12 JPDL xml schema....18

3.Referências....28

1.Introdução

Este documento visa a apresentar uma série de motores de Workflow para Sistemas Corporativos existentes, definir o mais promissor entre eles e verificar suas capacidades e funcionamento, para se avaliar a viabilidade para inclusão no framework Pinhão.

2 Motores de Workflow disponíveis

As seguintes soluções Open Source em Java foram encontradas:

- **Enhydra Shark** – (http://shark.objectweb.org/doc/index.html).
- **Werkflow** – (http://werkflow.codehaus.org).
- **OpenSymphony OSWorkflow** – (http://www.opensymphony.com/osworkflow/).
- **jBpm** – (http://docs.jboss.com/jbpm/).
- **wfmOpen** – (http://wfmopen.sourceforge.net/).
- **OFBiz Workflow Engine** – (http://www.ofbiz.org/docs/workflow.html).
- **ObjectWeb Bonita** – (http://bonita.objectweb.org/).
- **Bigbross Bossa** – (http://www.bigbross.com/bossa/).
- **Taverna** – (http://taverna.sourceforge.net/).
- **Jfolder** – (http://www.jfolder.com/).
- **Open Business Engine –** (http://www.openbusinessengine.org/index.html).
- **Open WFE –** (http://openwfe.sourceforge.net/).
- **Freefluo –** (http://freefluo.sourceforge.net/).
- **Zbuilder –** (http://www.tripodsoft.com/products/intro.htm).
- **Micro-Workflow –** (http://micro-workflow.com/Downloads.phtml).
- **concern** – (http://concern.sourceforge.net/).
- **Twister** – (http://www.smartcomps.org/twister/).
- **YAWL** – (http://www.citi.qut.edu.au/yawl).
- **Zebra** – (http://zebra.tigris.org/).
- **ActiveBPEL** – (http://www.activebpel.org/).
- **Xflow2** – (http://www.kgionline.com/xflow2/).
- **Apache Agila** – (http://incubator.apache.org/projects/agila/index.html).
- **AntFlow** – (http://antflow.onionnetworks.com/).
- **MidOffice BPEL Engine** – (http://mobe.objectweb.org/).
- **PXE** – (http://www.fivesight.com/pxe.shtml).
- **Beexee** – (http://bexee.sourceforge.net/).
- **Syrup** – (http://syrup.sourceforge.net/).

2.1 JBPM

A escolha do JBPM sobre os outros motores de workflow Java disponíveis decorre de:

- Disponibilidade de ferramenta gráfica para design e deploy de definições de processos de negócio (jBPM Graphical Process Designer), através de um plugin Eclipse.
- Utiliza Hibernate para camada de persistência.
- Separação forte entre modelagem e implementação de processos.
- Utiliza base de dados para armazenar definições de processos, estados de instâncias de processos e informações de log.
- Suporte a vários bancos de dados.
- Disponibilidade de documentação elaborada.
- A consagrada JBoss é responsável pelo projeto. Provê suporte, treinamento e consultoria.
- Pode ser integrado a uma aplicação Java standalone, ao JBoss ou a outros servidores de aplicação.
- Suporte a clusterização.
- Logging, que gera históricos sobre a execução dos processos.
- Versionamento, que permite alterar as definições do processo sem prejudicar as instâncias (execuções) do mesmo.
- Se encontra num estágio maduro quanto à implementação.
- Interface web para interação com usuários.
- Possibilita a apresentação de tarefas de usuários de outras aplicações juntamente com as tarefas do processo jBPM.

2.1.1 Definição

O jBPM é definido como uma engine de Workflow e BPM (Business Process Management) que permite a criação de processos de negócios que coordenam o fluxo entre pessoas, aplicações e serviços.

Um Workflow está mais relacionado ao gerenciamento de tarefas para pessoas. Num processo de Workflow, tarefas realizadas por um indivíduo se relacionam à realização de algum objetivo do processo.

Business Process Management combina a função do workflow com a tecnologia Enterprise Application Integration (EAI), permitindo integração com sistemas externos para realizar tarefas e recuperar os resultados das mesmas. Um BPM especifica uma combinação de tarefas humanas com softwares, e como os mesmos se relacionam para a realização de um processo de negócio.

O jBPM é uma biblioteca Java. Seus componentes principais são um software Java para gerenciar as definições de processo e um ambiente para a execução de instâncias dos mesmos.

O jBPM utiliza Workflow Patterns, que são uma série de padrões definidos numa pesquisa acadêmica que considerou um conjunto de Workflow's comerciais.

2.1.2 Características

O jBPM possui uma interface centralizada ao usuário através de uma aplicação web (jBPM console web application), permitindo interagir com suas tarefas geradas pela execução de processos, além do gerenciamento e monitoramento de instâncias de processos.

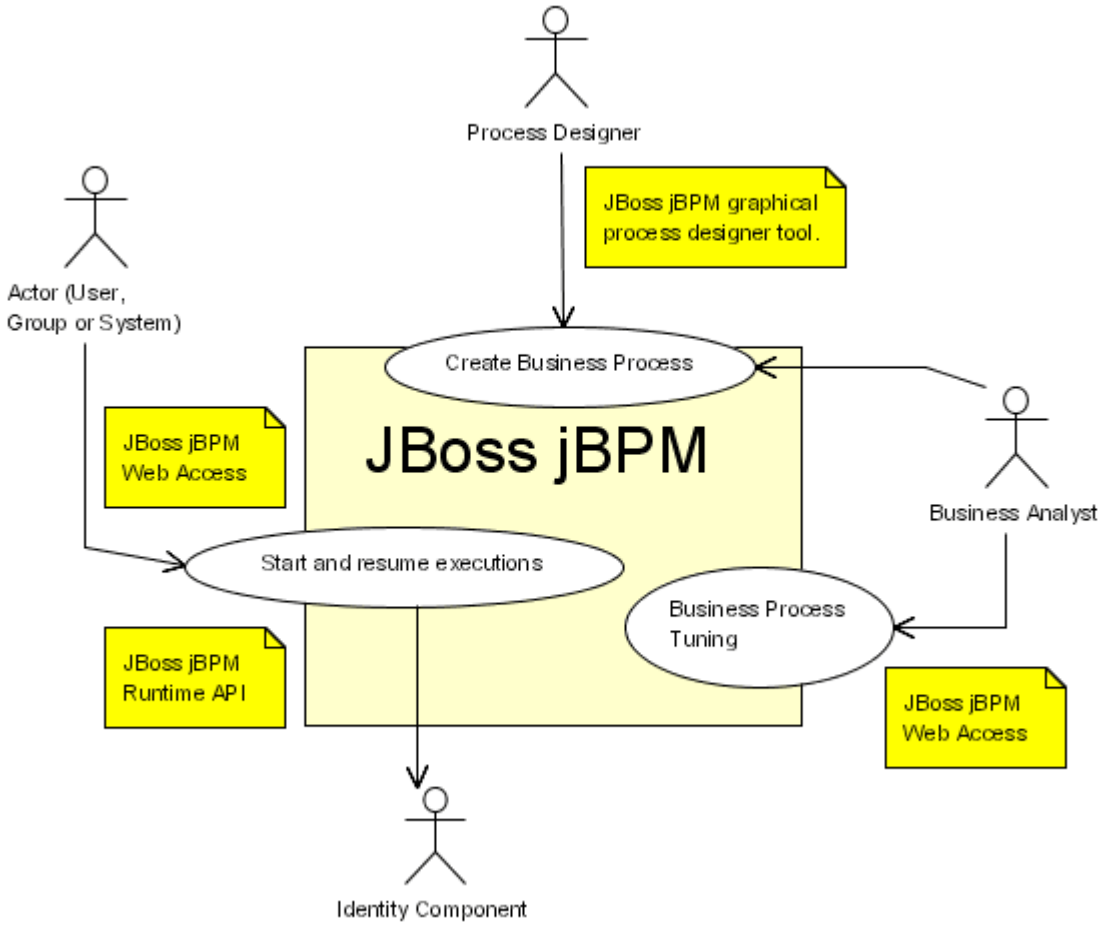

Ilustração 1Componentes do JBoss jBPM

Utiliza o banco de dados para desenvolvimento Hypersonic HSQLDB por padrão, mas é

possível definir outros bancos de dados usando o componente Database Extensions. O HSQLDB possui um modo in-memory que é particularmente conveniente para testes, por permitir a criação das tabelas do workflow em memória (não armazena em arquivo do disco local) através do gerador de Schema do Hibernate 3.0.

O banco de dados jBPM contem a definição do processo, as execuções de processo e o logging. Os dados de definição do processo é estático (não mudam). As execuções de processo referem-se em dados na definição do processo. E os logs de processo contem informações sobre todas as mudanças nas execuções de processo.

2.1.3 Funcionamento

Cada definição de processo é representada através de um grafo direcionado, com nodos (nós) e transições, contendo um estado inicial e um estado final, podendo existir mais de um estado final. Transições possuem direção e conectam dois nodos em um processo gráfico. As definições de processos são feitas num arquivo XML (processdefinition.xml), e podem ser persistidas no banco de dados ou representadas como um objeto Java.

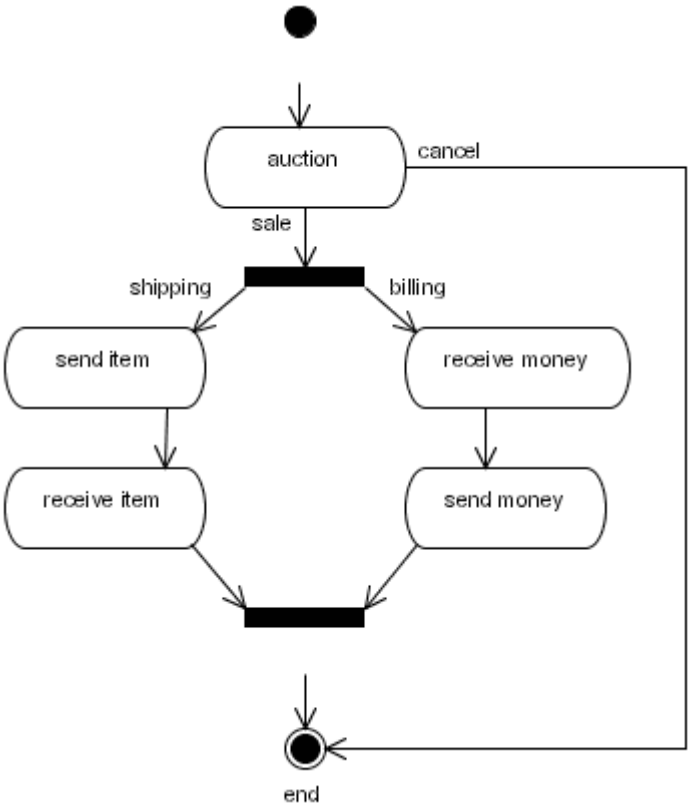

Ilustração 2Processo Exemplo: Leilão

Segue representação xml do processo gráfico acima:

```
<process-definition>

  <start-state>
    <transition to="auction" />
  </start-state>

  <state name="auction">
    <transition name="auction ends" to="salefork" />
    <transition name="cancel" to="end" />
  </state>

  <fork name="salefork">
    <transition name="shipping" to="send item" />
    <transition name="billing" to="receive money" />
  </fork>

  <state name="send item">
    <transition to="receive item" />
  </state>

  <state name="receive item">
    <transition to="salejoin" />
  </state>

  <state name="receive money">
    <transition to="send money" />
  </state>

  <state name="send money">
    <transition to="salejoin" />
  </state>

  <join name="salejoin">
    <transition to="end" />
  </join>

  <end-state name="end" />

</process-definition>
```

Uma instância de um processo é uma execução específica de uma definição de processo.

Um nodo pode realizar atividades de negócio relacionadas ao seu tipo (e.g. enviar uma mensagem, atualizar um banco de dados, criar tarefas para um usuário...) e prosseguir execução a um ou mais nodos através de uma transição. Cada nodo tem um tipo específico (state, decision, fork, join...) e uma transição de saída, que deve ter um nome distinto no processo.

Um processo permite múltiplos caminhos de execução no grafo, também conhecidos como tokens. Cada token mantém um ponteiro para um nodo do grafo. Quando uma instância de um processo é criada, um token denominado root token é criado e posicionado no estado inicial. Este token conterá o caminho principal de execução do processo.

Assim que um token aponta para um nodo, o mesmo é executado. Então, os nodos podem prosseguir a execução do processo (i.e. transição para outro nodo), ou entrar num estado de espera.

Um nodo representa um estado de espera quando não propaga execução. Neste estado, o token aguarda a chegada de um sinal para prosseguir execução, que pode indicar quais das possíveis transições deverão ser executadas. Neste caso, o estado deve ser persistido num banco de dados para se poder prosseguir a execução posteriormente.

A execução do processo gráfico termina quando todos os tokens tiverem entrado em estado de espera.

Os nodos possuem as seguintes opções quanto ao prosseguimento da execução do processo:

1) **Propagar a execução através de uma das possíveis transições a um nodo.** Atualiza o ponteiro do token para um novo nodo.
2) **Não propagar a execução.** Neste caso o nodo se comporta como um estado de espera.
3) **Criar novos caminhos de execução.** Um nodo pode criar novos tokens, que representam um caminho de execução diferente. E.g. nodo do tipo fork.
4) **Finalizar o caminho de execução.** Finaliza o token e termina a execução.
5) **Geralmente, um nodo pode modificar a estrutura toda de execução da instância do processo.** A estrutura de execução é uma instância de processo que contem uma árvore de tokens. Cada token representa um caminho de execução. Um nodo pode criar e finalizar tokens, colocar cada token em um nodo do processo e disparar tokens sobre transições.

Os tipos de nodos mais importantes, definidos na JDPL, são:

1) **Task.** Refere-se a uma ou mais tarefas a serem realizadas por indivíduos. Um nodo task pode ter zero ou mais tarefas, que são descrições estáticas na definição do processo. Quando um task node é executado, instâncias da tarefa são criadas na lista de tarefas dos participantes do workflow. Então, o nodo entra num estado de espera. Quando o usuário termina sua tarefa, um sinal é passado ao token e a execução do processo pode prosseguir.
2) **State.** Representa um estado de espera. É particularmente útil quando um processo deve aguardar por algum sistema externo. E.g. na entrada do nodo, poderia-se gerar uma ação que mandaria uma mensagem a um sistema externo. Após isso, o processo aguardaria no estado de espera. Quando o sistema externo enviasse uma mensagem de resposta, o token poderia ser sinalizado e o processo poderia seguir execução. A diferença entre task nodo é que não serão criadas instâncias de tarefas em nenhuma lista de tarefas dos participantes do workflow.
3) **Decision.** É possível modelar decisões de fluxo de duas formas distintas, dependendo de quem

será responsável pela decisão: o processo (i.e. especificado na definição do processo) ou uma entidade externa.

a) Quando a decisão for feita pelo processo, um nodo de decisão deve ser usado. Existem duas formas de se especificar os critérios de decisão. O mais simples é adicionar elementos condicionais (conditions) nas transições. Condições são expressões BeanShell script que retornam um valor booleano. Em tempo de execução, o nodo de decisão fará um loop nas suas transições de saída, na ordem especificada na definição do processo, e executará cada condição e avaliará o resultado. A primeira transição cuja condição retornar verdadeiro será a escolhida. Alternativamente, pode se especificar um DecisionHandler. Então, a decisão de fluxo ficaria a cargo de uma classe Java.

b) Quando a decisão ficar a cargo de uma entidade externa (não constando na definição de processo), deve-se usar múltiplas transições de saída ou nodos de estado de espera. Neste caso, a transição de saída pode ser provida a partir de uma trigger externa que resume execução após o estado de espera terminar.

4) **Fork.** Divide um caminho de execução em múltiplos caminhos concorrentes. O comportamento padrão do fork é criar um token filho para cada transição que deixa o fork, criando uma relação pai-filho com o token que entrou no fork.
5) **Join.** Convergência de nodos. O join padrão assume que todos os tokens que convergiram a ele são filhos de um mesmo token pai. Essa situação é criada quando o fork é utilizado da forma descrita acima e quando todos os tokens criados pelo fork chegam ao mesmo join. Quando todos os tokens filhos chegarem ao join, o token pai será propagado pela única transição de saída. Enquanto algum token filho ainda estiver ativo, o join se comportará como um estado de espera.
6) **Node**. Permite criar um código customizado dentro de um nodo. Este tipo de nó precisa de um subelemento action, que é executado quando o nodo ganha execução. O código no action handler deve obrigatoriamente ser responsável por propagar a execução. Este nodo pode utilizar uma JavaAPI para implementar alguma lógica funcional importante à representação gráfica do processo. Este nodo é visível na representação gráfica do processo.

Em contraste a nodos do tipo Node, as ações (Actions) permitem criar códigos invisíveis na representação gráfica do processo, no caso da lógica não ser importante ao modelo do processo. É possível definir ações – código Java – para eventos relacionados à execução do processo. Exemplos de eventos são a entrada ou saída de um nodo ou a entrada num estado de transição. Ações são um

modo de prover detalhes de implementação fora da modelagem gráfica do processo. A diferença entre uma ação incluída em um evento e ação incluída em um nodo é que ação incluída em evento é executada com o disparo do evento e não influenciam no fluxo de controle do processo. É similar ao pattern Observer. Por outro lado uma ação incluída em um nodo tem a responsabilidade de propagação da execução.

Exemplo de uma ação (action) em um evento: Suponha que queremos fazer um update na base em um dada transição. O update da base é tecnicamente vital mas não importante para a modelagem do negócio.

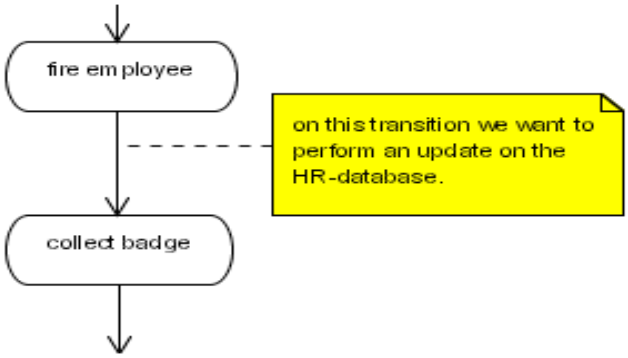

Ilustração 3Processo Exemplo: Action em uma transição

```
public class RemoveEmployeeUpdate implements ActionHandler {
  public void execute(ExecutionContext ctx) throws Exception {
    // get the fired employee from the process variables.
    String firedEmployee = (String) ctx.getContextInstance().getVariable("fired
employee");

    // by taking the same database connection as used for the jbpm updates, we
    // reuse the jbpm transaction for our database update.
    Connection connection =
ctx.getProcessInstance().getJbpmSession().getSession().getConnection();
    Statement statement = connection.createStatement("DELETE FROM EMPLOYEE WHERE
...");
    statement.execute();
    statement.close();
  }
}
```

A configuração da chamada da action acima no arquivo .xml será:

```
<process-definition name="yearly evaluation">

  ...
  <state name="fire employee">
    <transition to="collect badge">
      <action class="com.nomercy.hr.RemoveEmployeeUpdate" />
    </transition>
  </state>

  <state name="collect badge">
  ...

</process-definition>
```

Operações de persistência podem ser obtidas através de uma JbpmSession, que contém a conexão JDBC com um banco de dados. Esta, por sua vez, é criado através de um JbpmSessionFactory. A melhor forma de configurar o hibernate `SessionFactory` é colocando o arquivo `hibernate.cfg.xml` na raiz do classpath. O arquivo `hibernate.cfg.xml` contem a informação de como obter conexão jdbc e também a localização dos arquivos de mapeamento do hibernate.

2.1.4 Script

Uma instância de um processo pode conter variáveis, que podem ser persistidas juntamente com o processo ou não (variáveis transientes). Cada variável é relativa a um token, e são armazenadas em VariableInstances, que são mapeadas para campos no banco de dados pelo hibernate.

Um script é uma ação que executa um script beanshell. Por padrão, todas as variáveis de processo são disponibilizadas como variáveis de script, porém nenhuma variável de script será gravada como uma variável de processo. As seguintes variáveis de script são disponibilizadas: executionContext, token, node, task e taskInstance.

```
<process-definition>
  <event type="node-enter">
    <script>
      System.out.println("this script is entering node "+node);
    </script>
  </event>
  ...
</process-definition>
```

Para customizar o comportamento padrão de carregar e salvar variáveis no script, a variável elemento pode ser usada como um sub-elemento do script. Neste caso, o script deve ser colocado em um sub elemento do script: `expression`.

```
<process-definition>
  <event type="process-end">
    <script>
      <expression>
        a = b + c;
      </expression>
      <variable name='XXX' access='write' mapped-name='a' />
      <variable name='YYY' access='read' mapped-name='b' />
      <variable name='ZZZ' access='read' mapped-name='c' />
    </script>
  </event>
  ...
</process-definition>
```

Antes de iniciar o script, as variáveis de processo `YYY` e `ZZZ` ficarão disponíveis para o script como variáveis de script `b` e `c` respectivamente. Assim que o script for finalizado, o valor da variavel de script `a` é salvo na variável de processo `XXX`.

Se o atributo de acesso (`access`) da variável (`variable`) conter '`read`', a variável de processo será carregada como variável de script antes da execução do script. Se o atributo de acesso conter '`write`', a variável de script será salva como uma variável de processo após a execução do script. O atributo `mapped-name` pode tornar a variável de processo disponível sob outro nome no script. Isto é útil quando o nome de sua variável de processo contem espaços ou outros caracteres literais de script inválidos.

2.1.5 Superstates

Superstates são conjuntos de nodos que podem ser agrupados recursivamente e permitem adicionar hierarquia na definição do processo. Um token poderá, então, apontar para um conjunto de nodos num dado momento, neste caso o token será direcionado para o primeiro nodo do superstate.

Ações podem ser associadas a eventos de superstates.

Todos os nodos dentro do superstate podem executar qualquer transição de saída associada ao superstate, e qualquer nodo externo pode executar uma transição para qualquer nodo interior ao superstate. Superstates podem ser auto-referenciados (um nodo dentro do superstate aponta uma transição para o próprio superstate).

Existem dois eventos associados a superstate: `superstate-enter` e `superstate-leave`. Esses eventos serão acionados não importando qual a transição em que o nodo é acessado ou deixado. Esses eventos não são disparados por transições de tokens internos a um superstate.

Então existem tipos de eventos separados para states e superstates, isso para facilitar a distinção entre eventos de superstate e eventos de nodos que podem ser propagados dentro de um superstate.

Os nomes dos nodos tem que ser únicos em seu escopo. O escopo do nodo é sua coleção de nodos. A definição do processo e o superstate sao coleções de nodos. Para se referir a nodos em superstates, deve ser especificado a relação com uma barra (/) separando os nomes do superstate e seu nodo. Utilize '..' para se referir a um nível acima. Vejamos como se referir a um superstate vizinho:

```
<process-definition>
  ...
  <state name="preparation">
    <transition to="phase one/invite murphy"/>
  </state>
  <super-state name="phase one">

  </super-state>
  ...
</process-definition>
```

Vejamos como subir na hierarquia do superstate:

```
<process-definition>
  ...
  <super-state name="phase one">
    <state name="preparation">
      <transition to="../phase two/invite murphy"/>
    </node>
  </super-state>
  <super-state name="phase two">

  </super-state>
  ...
</process-definition>
```

2.1.6 Instâncias de tarefas

Uma instância de uma tarefa pode ser associada a um usuário através de um actorId (String).

Todas as instâncias são armazenadas em uma tabela do banco de dados (JPBM_TASKINSTANCE). Consultando essa tabela por actorId, se consegue a lista de tarefas do usuário.

Instâncias de tarefas são tipicamente criadas pela execução de um nodo task (com o método `TaskMgmtInstance.createTaskInstance(...)`). Uma interface para o usuário poderá apresentar as listas de tarefas de um usuário (com o método `TaskMgmtSession.findTaskInstancesByActorId(...)` e receber entradas sobre o andamento das mesmas.

Instâncias de tarefas concluídas são mantidas na base de dados, mas se diferenciam de tarefas correntes por contar com uma data de conclusão.

No caso de mais de uma instância de uma tarefa estar associada a um task node, é possível definir o comportamento da continuidade do processo quanto a completude das tarefas. Por padrão uma instância de tarefa é signalling (envia sinal para seu token continuar o processo) e não-bloqueável (permite que o token deixe o nodo da tarefa antes do término da instância de tarefa). Os seguintes valores são possíveis na propriedade signal de um nodo task:

- **last**: Valor padrão. Procede execução quando a **última** instância de tarefa terminar execução. Mesmo se nenhuma tarefa for criada na entrada deste nodo, a **execução prossegue.**
- **last-wait**: Procede execução quando a **última** instância de tarefa terminar execução. Enquanto nenhuma tarefa for criada na entrada deste nodo, o nodo fica em **estado de espera** até a criação de alguma tarefa.
- **first**: Procede execução quando a **primeira** instância de tarefa terminar execução. Mesmo se nenhuma tarefa for criada na entrada deste nodo, a **execução prossegue.**
- **first-wait**: Procede execução quando a **primeira** instância de tarefa terminar execução. Permanece em **estado de espera** até a criação de alguma tarefa.
- **unsynchronized**: Execução sempre continua, independentemente do estado das tarefas.
- **never**: Execução nunca prossegue.

Uma classe assignmentHandler pode ser criada para se definir um ou mais responsáveis por realizar uma tarefa, chamada quando a instância da tarefa é criada. Nela, deve ser informado um objeto que implementa a interface Assignable: TaskInstance ou SwimlaneInstance. Em ambas as classes é possível informar quem é o responsável pela tarefa e um grupo e usuários candidatos para a mesma.

```
public interface Assignable {
  public void setActorId(String actorId);
  public void setPooledActors(String[] pooledActors);
}
```

Um Swinlane é um papel (role) de processo. É um mecanismo que especifica que múltiplas tarefas num processo podem ser realizadas pelo mesmo ator. Então, quando a primeira instância de uma tarefa é criada para uma certa swimlane, o ator será memorizado para todas as tarefas subseqüentes que referenciam o mesmo swimlane.

2.1.7 Exceções

Exceções não são geradas pela execução do grafo em si, mas pela execução de classes Java delegadas.

Na definição de processos, nodos e transições, uma lista de exception-handlers podem ser especificados. Cada exception-handler indica uma lista de classes de ação para gerenciar a exceção.

Exceções no jBPM não podem modificar o fluxo de execução do processo.

2.1.8 Processos compostos

Processos compostos tem suporte no jBPM via process-states. O process state é um estado associado a uma definição de um processo externo. Quando a execução de um grafo alcança um process state, uma nova instância do sub-processo é criada e se associa com o caminho de execução no process state. O caminho de execução no processo pai, ou super processo, aguardará a conclusão da instância do sub-processo. Quando da conclusão, o super processo deixará o process state e continuará a execução do grafo no super processo. É possível o intercâmbio de variáveis entre sub e super processo.

```
<process-definition name="hire">
  <start-state>
    <transition to="initial interview" />
  </start-state>
  <process-state name="initial interview">
    <sub-process name="interview" />
    <variable name="a" access="read,write" mapped-name="aa" />
    <variable name="b" access="read" mapped-name="bb" />
    <transition to="..." />
  </process-state>
  ...
</process-definition>
```

O processo 'hire' contem um `process-state` que comporta o processo 'interview' (sub-processo). Quando a execução chega em 'first interview', uma nova execução (=instância de processo) para a última versão do processo 'interview' é criada. Então a variável 'a' do processo 'hire' é copiada para a variável 'aa' do processo 'interview'. Da mesma forma, a variável 'b' do processo 'hire' é copiada para a variável 'bb' de 'interview'. No término do processo 'interview', somente a variável 'aa' do processo 'interview' é copiada para a variável 'a' do processo 'hire' (write).

No geral, quando um sub-processo inicia, todas `variables` com `read` access são lidas do super processo e carregadas para o novo sub-processo criado antes do sinal ser dados para sair do estado inicial (start state). Quando a instância do sub-processo for finalizada, todas `variables` com `write` access serão copiadas do sub-processo para o super processo. O atributo `mapped-name` do elemento `variable` permite especificar os nomes de variáveis que devem ser utilizadas no sub-processo.

2.1.9 jBPM Process Definition Language (JPDL)

JPDL especifica um esquema xml (linguagem) e o mecanismo para empacotar todos os arquivos de definições de processo em um arquivo de processo.

O arquivo de processo é um arquivo zip geralmente com a extensão .par. O arquivo central no arquivo de processo é `processdefinition.xml`. A principal informação no arquivo é o fluxo do processo. O arquivo `processdefinition.xml` também contem informações sobre actions e tasks. Um arquivo de processo também pode conter outros arquivos de processos relativos tal como classes, ui-forms (telas de interface de usuário) para tarefas, entre outros.

jBpm reconhece três tipos de dados requeridos na descrição do processo:

- **uma descrição declarativa do processo de negócio:** no jPdl, isso é representado no arquivo processdefinition.xml.
- **lógica de programação**: para conectar lógica de programação ao processo como descrito no arquivo processdefinition.xml, classes java podem ser incluídas no arquivo. Todas as classes no arquivo de processo devem estar localizadas na subpasta /classes.
- **outros arquivos de recursos:** como um cliente de motor de workflow, você pode querer incluir uma variedade de arquivos de recursos no processo para ser utilizado em tempo de execução, como descrições de telas que são associadas com tarefas a serem executadas por um usuário. jBpm não impõe restrições de tipos de arquivos de recursos que se queira incluir na definição de processo.

Pode ser feito deploy de arquivos de processos de 3 maneiras: com a ferramenta de desenho de processo (em construção), com uma tarefa ant or via programação.

O deploy com uma tarefa ant pode ser feito como segue:

```
<target name="deploy.par">
  <taskdef name="deploypar"
classname="org.jbpm.jpdl.par.ProcessArchiveDeployerTask">
    <classpath --make sure the jbpm-[version].jar is in this classpath--/>
  </taskdef>
  <deploypar par="build/myprocess.par" />
</target>
```

O deploy via programação pode ser feito com a classe `org.jbpm.jpdl.par.ProcessArchiveDeployer`.

2.1.10 Versionamento de Processo

Definições de processos dificilmente devem ser alterados pois seria extremamente difícil prever todos os efeitos colaterais possíveis com a mudança das definições. Uma nova versão da definição do processo é gerada a cada deploy do arquivo de processo.

O versionamento contorna esse problema no jBPM, permitindo que múltiplas definições com o mesmo nome existam na base de dados. Uma instância de processo pode ser iniciada com a última versão disponível e será mantida em execução até ter seu ciclo completado. Quando uma nova versão surgir, novas instâncias serão criadas com as mesmas, enquanto que processos já em andamento continuarão sendo executados com as definições já existentes na versão da ocasião de seu início.

2.1.11 Classe de carregamento jBPM

É a classloader que tem a biblioteca `jbpm-3.x.jar` em seu classpath. Para deixar classes visiveis ao jBPM classloader, coloque-as em um arquivo jar e coloque esse arquivo jar juntamente com o classloader (por exemplo, na pasta WEB-INF/lib no caso de aplicações web).

2.1.12 JPDL xml schema

O esquema xml do JPDL é o esquema utilizado no arquivo de processo `processdefinition.xml`.

Maiores detalhes a respeito podem ser encontrados em http://www.jboss.com/products/jbpm/docs/jpdl#otherprocessarchivefiles.

2.1.12.1 process-definition

Nome	*Type*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome do processo
swimlane	element	[0..*]	os swimlanes usados no processo. o swimlanes representam papéis do processo e são usados para atribuição de tarefas.
start-state	element	[0..1]	o estado inicial do processo. Um processo sem estado inicial é válido mas não é executável.
{end-state|state|node|task-node|process-state|super-state|fork|join|decision}	element	[0..*]	os nodos de definição do processo. Um processo sem nodos é válido mas não é executável.
event	element	[0..*]	os eventos do processo que servem de conteiner para actions.
{action|script|create-timer|cancel-timer}	element	[0..*]	actions globais que podem ser referenciadas por eventos e transições.. Essas actions devem especificar um nome para que possa ser referenciada.
task	element	[0..*]	tarefas globais que podem ser usadas nas actions.
exception-handler	element	[0..*]	uma lista de manipuladores de exceção que aplica para todas exceções lançadas por classes de delegação definidas no processo.

2.1.12.2 node

Nome	*Tipo*	*Multiplicidade*	*Descrição*
{action|script|create-timer|cancel-timer}	element	1	action customizada que reflete o comportamento do nodo.
common node elements			Veja em common node elements

2.1.12.3 common node elements

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	required	o nome do nodo
transition	element	[0..*]	a transição de saída. Cada transição de saída do nodo deve ter um nome distinto. Somente uma transição de saída pode ficar sem nome. A primeira transição especificada é a transição padrão. A transição padrão é assumida quando o nodo é deixado e não tem uma transição especificada.
event	element	[0..*]	tipo de eventos suportados: {node-enter|node-leave}
exception-handler	element	[0..*]	lista de manipuladores de exceção aplicáveis para todas as exceções lançadas por classes de delegação deste nodo do processo.
timer	element	[0..*]	tempo que monitora a duração de uma execução do nodo.

2.1.12.4 start-state

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome do nodo
event	element	[0..*]	tipos de eventos suportados: {node-leave}
transition	element	[0..*]	transições de saída. Cada transição deve ter nome distinto.
exception-handler	element	[0..*]	lista de manipuladores de exceção aplicáveis para todas as exceções lançadas por classes de delegação deste nodo do processo.

2.1.12.5 end-state

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	required	nome do end-state
event	element	[0..*]	tipos de eventos suportados: {node-enter}
exception-handler	element	[0..*]	lista de manipuladores de exceção aplicáveis para todas as exceções lançadas por classes de delegação deste nodo do processo.

2.1.12.6 state

Nome	*Tipo*	*Multiplicidade*	*Descrição*
common node elements			Veja em common node elements

2.1.12.7 task-node

Nome	*Tipo*	*Multiplicidade*	*Descrição*
signal	attribute	optional	{unsynchronized|never|first|first-wait|last|last-wait}, o padrão é `last`. signal especifica o efeito da tarefa na continuação da execução do processo.
create-tasks	attribute	optional	{yes|no|true|false}, o padrão é `true`. pode ser setado para false quando um cálculo em tempo de execução determina qual tarefa a ser criada. neste caso, adiciona uma action no `node-enter`, cria as tarefas na action e seta `create-tasks` para `false`.
task	element	[0..*]	as tarefas que devem ser criadas quando a execução alcançar este task node.

Nome	*Tipo*	*Multiplicidade*	*Descrição*
common node elements			Veja em common node elements

2.1.12.8 process-state

Nome	*Tipo*	*Multiplicidade*	*Descrição*
sub-process	element	1	o sub processo que está associado a este nodo.
variable	element	[0..*]	especifica como os dados devem ser copiados do super processo para o sub processo no início e do sub processo para o super processo no final de execução do sub processo.
common node elements			Veja em common node elements

2.1.12.9 super-state

Nome	*Tipo*	*Multiplicidade*	*Descrição*
{end-state|state|node|task-node|process-state|super-state|fork|join|decision}	element	[0..*]	os nodos do superstate. superstates podem ser agrupados.
common node elements			Veja em common node elements

2.1.12.10 fork

Nome	*Tipo*	*Multiplicidade*	*Descrição*
common node elements			Veja em common node elements

2.1.12.11 join

Nome	*Tipo*	*Multiplicidade*	*Descrição*
common node elements			Veja em common node elements

2.1.12.12 decision

Nome	*Tipo*	*Multiplicidade*	*Descrição*
handler	element	tanto um elemento 'handler' ou condições nas transições devem ser especificadas	o Nome de uma implementação `org.jbpm.jpdl.Def.DecisionHandler`

Nome	*Tipo*	*Multiplicidade*	*Descrição*
transition	element	[0..*]	as transições de saída. As transicões de saída de decisão pode ser extendida com uma condição. A decisão verificará pela primeira transição na qual a condição ficará verdadeira. Um transição sem uma condição é considerada evoluindo para true. (to model the 'otherwise' branch). See the condition element
common node elements			See common node elements

2.1.12.13 event

Nome	*Tipo*	*Multiplicidade*	*Descrição*
type	attribute	required	o tipo de evento que é expressado relativo ao elemento no qual o evento está.
{action|script|create-timer|cancel-timer}	element	[0..*]	a lista de actions que devem ser executadas neste evento.

2.1.12.14 transition

Nome	*Tipo*	*Multiplicidade*	*Descrição*
Nome	attribute	optional	o Nome da transição. Cada transição saindo de uma transição deve ter nomes distintos.
to	attribute	required	o nome hierárquico do nodo destino. Para mais informações veja em "Hierarchical names"
{action|script|create-timer| cancel-timer}	element	[0..*]	as actions a executar ao assumir esta transição. As actions de uma transição não necessita ser colocada em um evento.
exception-handler	element	[0..*]	uma lista de manipuladores de exceção aplicáveis para todas as exceções lançadas por classes delegadas definidas no nodo do processo.

2.1.12.15 action

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome da action. Quando se dá nome as action, elas podem ser vistas pela definição do processo. Isso é útil para actions de tempo de execuçao e declarando action em um única vez.
class	attibute	either this or ref-name	o nome da classe totalmente qualificada que implementa a interface `org.jbpm.graph.def.ActionHandler`.
ref-name	attibute	either this or class	o nome da action referenciada. O conteúdo desta action não é processado a não ser que uma action refenciada é especificada.
accept-propagated-events	attribute	optional	{yes|no|true|false}. O padrão é yes|true. Se setado para false, a action somente executará em eventos disparados no elemento desta action. Veja mais em "Event propagation"

Nome	*Tipo*	*Multiplicidade*	*Descrição*
config-type	attribute	optional	{field|bean|constructor|configuration-property}. Especifica como o objeto action deve ser construído e como o conteúdo deste elemento deve ser utilizado como informação de configuração para o objeto action.
	{content}	optional	o conteúdo da action pode ser utilizado como informação de configuração para sua implementação de action. Isto possibilita a criação de classes de delegação reusáveis. Veja mais sobre classes de delegação em "Configuration of delegations".

2.1.12.16 script

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome do script action. Quando houver nomes para actions, elas podem ser vistas a partir da definição do processo. Isso é útil para actions de tempo de execução e declarando actions somente uma vez.
accept-propagated-events	attribute	optional [0..*]	{yes|no|true|false}. O padrão é yes|true. Se false, a action somente executará em eventos que foram disparados neste elemento da action, veja mais em "Event propagation"
expression	element	[0..1]	o script beanshell. Se não especificar elementos variable, pode ser escrita a expressão com o conteúdo do elemento de script (omitindo a tag do elemento expression).
variable	element	[0..*]	variáveis internas do script. Caso não especificada, todas as variáveis do token corrente serão carregadas na carga do script. Utilize as variáveis internas se quiser limitar o número de variáveis a carregar.

2.1.12.17 expression

Nome	*Tipo*	*Multiplicidade*	*Descrição*
	{content}		um script bean shell.

2.1.12.18 variable

Nome	*Tipo*	*Multiplicidade*	*Descrição*
Nome	attribute	required	o nome da variável do processo.
access	attribute	optional	padrão é `read,write`. é uma lista separada por vírgula de especificação de acesso.
mapped-name	attribute	optional	padroniza o nome da variável. Especifica um nome para mapear a variável. o significado do mapped-name é dependente do contexto no qual este elemento é utilizado. Para um script, este será o nome da variável de script. Para um controlador de tarefa, será o label do parametro da tarefa e para um estado de processo, será o nome da variável usada no sub-processo.

2.1.12.19 handler

Nome	*Tipo*	*Multiplicidade*	*Descrição*
class	attibute	either this or ref-Nome	nome da classe totalmente qualificada da classe que implementa a interface `org.jbpm.graph.node.DecisionHandler`.
config-type	attribute	optional	{field|bean|constructor|configuration-property}. Especifica como o objeto action deve ser contruído e como o conteúdo deste elemento deve ser utilizado como informação de configuração para o objeto action.
	{content}	optional	o conteúdo do handler pode ser utilizado como informação de configuração para as implementações de handler customizadas. Isso permite a criação de classes de delegação reusáveis. Veja informações sobre configuração de delegações em "Configuration of delegations".

2.1.12.20 timer

Nome	*Tipo*	*Multiplicidade*	*Descrição*
Nome	attribute	optional	o nome do timer. Sem especificar o nome, o nome do nodo onde se encontra é assumido. Todo timer deve ter um nome único.
duedate	attribute	required	a duração (opcionalmente expressao em "business hours") que especifica o período de tempo entre a criação do timer e a execução do timer. Verifique sintaxe em "Duration".
repeat	attribute	optional	{duration | 'yes' | 'true'}após a execução de um timer no duedate, 'repeat' opcionalmente especifica a duração entre a repetição de execução até o nodo ser deixado. Se `yes` ou `true` é especificado, a mesma duração the de tempo é assumida para a repetição. Verifique sintaxe em "Duration".
transition	attribute	optional	o nome da transição a assumir quando o timer executa, após disparo do evento timer e executar a action (se existir).
cancel-event	attribute	optional	somente é utilizado em timers. especifica o evento em que o timer será cancelado. por padrão, é o evento `task-end`, mas pode ser setado em `task-assign` ou `task-start`. Os tipos de `cancel-event` podem ser combinados especificando-os separados por vírgulas.

2.1.12.21 create-timer

Nome	*Tipo*	*Multiplicidade*	*Descrição*
Nome	attribute	optional	o nome do timer. O nome pode ser usado para cancelamento do timer com a action cancel-timer.
duedate	attribute	required	duração que especifica o período de tempo entre a criação do timer e execução do mesmo. Veja sintaxe em "Duration".
repeat	attribute	optional	{duration | 'yes' | 'true'}após after a timer ser executado, 'repeat' especifica a duração entre a repetição da execução do timer até o nodo ser deixado. Se `yes` or `true` é especificado, a mesma duração da execução é assumida para a repetição repeat. Veja sintaxe em "Duration".
transition	attribute	optional	o nome da transição a ser assumida na execução do timer, após disparado o evento timer e executada a action (se existir).

2.1.12.22 cancel-timer

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome do timer a ser cancelado.

2.1.12.23 task

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	optional	o nome da task. Tasks nomeadas podem ser referenciadas e vistas através do `TaskMgmtDefinition`.
blocking	attribute	optional	{yes|no|true|false}, padrão é false. Se true, o nodo não pode ser deixado quando a task não está finalizada. Se false (default) um signal no token é autorizado a continuar a execução e deixar o nodo. O padrão é false porque o bloqueio é normalmente forçado pela interface do usuário.
duedate	attribute	optional	é a duração. Verifique melhor em *Business calendar*
swimlane	attribute	optional	referencia a um swimlane. Se um swimlane é especificado na tarefa, a atribuição é ignorada.
priority	attribute	optional	um de {highest, high, normal, low, lowest}. alternativamente, qualquer número inteiro pode ser especificado para a prioridade. exemplo: (highest=1, lowest=5).
assignment	element	optional	descreve uma delegation que atribui a tarefa a um ator quando a tarefa é criada.
event	element	[0..*]	tipos suportados de eventos: {task-create|task-start|task-assign|task-end}. Especialmente para a `task-assign` tem uma propriedade não persistente adicionada `previousActorId` para a `TaskInstance`.
exception-handler	element	[0..*]	uma lista de manipuladores de exceção aplicáveis a todas exceções lançadas por classes de delegação deste nodo do processo.
timer	element	[0..*]	um timer que monitora a duração de uma execução nesta tarefa. Especial para task timers, o `cancel-event` pode ser especificado. por padrão o `cancel-event` é `task-end`, mas pode ser customizado para , por exemplo `task-assign` ou `task-start`.
controller	element	[0..1]	especifica como as variáveis de processo são transformadas em parâmetros no formulário de tarefa. Os parâmetros do formulário de tarefa são utilizados pela interface do usuário para conceder a tarefa ao usuário.

2.1.12.24 swimlane

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	required	o nome do swimlane. Swimlanes são referenciados e visíveis via `TaskMgmtDefinition`
assignment	element	[1..1]	atribuição deste swimlane. Será atribuído quando a primeira instância de tarefa é criada neste swimlane.

2.1.12.25 assignment

Nome	*Tipo*	*Multiplicidade*	*Descrição*
class	attribute	required	o nome da classe inteiramente qualificada de uma implementação de `org.jbpm.taskmgmt.def.AssignmentHandler`
config-type	attribute	optional	{field|bean|constructor|configuration-property}. Especifica como o objeto manipulador de atribuição deve ser construído e como o conteúdo deste elemento deve ser usado como informação de configuração para o objeto manipulador/gerenciador de atribuição.
	{content}	optional	o conteúdo do elemento de atribuição pode ser usado como informação de configuração para implementação do AssignmentHandler. Isso permite a criação de classes de delegação reusáveis. Para maiores informações verifique em "Configuration of delegations".

2.1.12.26 controller

Nome	*Tipo*	*Multiplicidade*	*Descrição*
class	attribute	optional	o nome da classe inteiramente qualificada de uma implementação de `org.jbpm.taskmgmt.def.TaskControllerHandler`.
config-type	attribute	optional	{field|bean|constructor|configuration-property}. Especifica com o objeto gerenciador de atribuição deve ser construído e como o conteúdo deste elemento deve ser usado como informação de configuração para o objeto gerenciador de atribuição.
	{content}		o conteúdo de um controller é a configuração de um gerenciador de controlador de tarefa (se o atributo de classe é especificado, se não especificado o gerenciador controlador de tarefa, o conteúdo deve ser uma lista de variáveis).
variable	element	[0..*]	no caso de não especificar um task controller handler pelo atributo de classe, o conteúdo do elemento controller deve ser uma lista de variáveis.

2.1.12.27 sub-process

Nome	*Tipo*	*Multiplicidade*	*Descrição*
name	attribute	required	nome do sub processo. Verifique teste de sub-processo em "Testing sub processes"
version	attribute	optional	versão do sub processo. Se não especificada, a última versão de um processo existente será assumido.

2.1.12.28 condition

Nome	*Tipo*	*Multiplicidade*	*Descrição*
	{content}	required	o conteúdo de um elemento de condição é uma expressão beanshell que deve evoluir em boolean. Uma decisão assume a primeira transição (ordenada em processdefinition.xml) no qual a expressão seja `true`.

2.1.12.29 exception-handler

Nome	*Tipo*	*Multiplicidade*	*Descrição*
exception-class	attribute	optional	Especifica o nome da classe java lançável inteiramente qualificada que deve combinar com esse gerenciador de exceção. se não especificado, combina com todas as exceções (`java.lang.Throwable`).
action	element	[1..*]	uma lista de actions a ser executada quando uma exceção está sendo manipulada por este gerenciador de exceção.

3.Referências

- http://docs.jboss.com/jbpm/v3/userguide/
- http://www.jboss.com/products/jbpm/docs