RunaWFE. Business process file-archive structure

Version 3.0

Introduction

RunaWFE is a free solution for enterprise business process management. It is delivered under LGPL licence. RunaWFE consists of JBOSS-JBPM workflow core and a set of additional components. The task of these components is to provide convenience for the end-user.

Business processes in RunaWFE system

The business process definition language in RunaWFE is based on jPDL language (from JBOSS jBPM project). RunaWFE extends jPDL in several ways. And some of the jPDL features (such as loading up classes from archive-file of business process) are not supported in RunaWFE yet. jPDL description can be found here: http://www.jbpm.org/jpdl.html [2]

In this document a language for RunaWFE business processes definitions is described.

When the export command is chosen from GPD menu the current process definition is placed into jar-archive with .par extension.

Note. Archive file must not be compressed. This file may be created by "jar cvf0 <archiveFileName>.jar"

A business process definition that is placed in a .par file is a set of xml-files and files that define the forms of business process. The archive may also contain an image of process graph and an icon for a business process.

A developed process definition can be deployed to the RunaWFE system via its web-interface.

After a business process is deployed it can be seen in a list of process definitions. Now the permission for this process may be set and it can be run.

Process archive structure

The .par file-archive structure:

- processdefinition.xml
- forms.xml
- variables.xml
- processimage.jpg
- start.png
- forms files
- form validation files

processdefinition.xml file is located in the root of process archive and contains a process graph and swimlane definitions.

forms.xml file in the root of archive defines process variables and lists process forms.

Optional processimage.jpg file is graphical representation of process graph displayed in properties of deployed process definition.

All form files that are defined in forms.xml must be present in process archive.

In variables.xml file all the business process variables and their types are described.

In the files of form validation there are the description of validation for the variables values entered in the form.

Business processes versions

The version system for business processes is based on the following principles:

- When a new business process version if deployed it is saved in the database. This business process version receives a subsequent version number. Previous versions are not deleted from database.
- The system considers business process definition as a new version of existing one if there is a definition with the same name already deployed.
- When a new business process instance is started it uses the definition with the largest version number from all currently deployed for the process being started. If after the process is started a new definition for this process is deployed the already started instance is not influenced at any way.

Elements of RunaWFE process definition language description

Description of processdefinition.xml its attributes

In order to be able to work with RunaWFE user groups and organization structure functions you should define a swimlane elements in a particular way. As it is described in details below.

Swimlanes usage in RunaWFE system

Swimlane is a special type of business process variables. Every activity has a swimlane associated with it. Swimlane determines who can execute this activity. In RunaWFE before the activity starts a swimlane must have a corresponding initializer that returns a user or a group of users. The swimlane initialization means that swimlane business process variable is assigned with ID of a user or a group of users. If swimlane isn't initialized yet it will be when the process comes to the corresponding activity. If a swimlane is initialized with a group of users then after the activity is done this swimlane is additionally initialized. And swimlane variable value is changed from group ID to the ID of the user that performed the activity.

If a process comes to an activity and the corresponding swimlane is already initialized then it's being checked if the swimlane reinitialization is needed. The reinitialization option can be set during creating/editing process definition in gpd. On activity node in a popup menu choose role sub-menu and check/uncheck "reinitialize" option. If the reinitialize is checked then on this activity swimlane is initialized as if it wasn't initialized before. If the reinitialize is unchecked then on this activity swimlane variable value is used as it was set previously and wouldn't be changed.

RunaWFE uses the following algorithm to initialize a swimlane:

- If the swimlane is not initialized yet, the activity is displayed in tasklists for all users from orgfunction output. When the first user executes the activity, swimlane is initialized with this user id.
- If the swimlane is initialized with user id, and "reinitialize" option is not set, the activity is displayed in tasklist for this user only.
- If the swimlane is initialized with user id, and "reinitialize" option is set, the activity is initialized with the initial group id.

If a swimlane is not set explicitly as a variable (in form, by bot or in start-state) then it's definition in processdefinition.xml must contain delegation tag and a class that implements org.jbpm.delegation.AssignmentHandler (as a rule that is org.jbpm.delegation.AssignmentHandler class) and a string that initializes a swimlane. This string must have the following form:

<Initializer Java class (a class of particular type)>(<parameter>,<parameter>,...)

Initializer Java class can be one of the following already existing classes:

- ru.runa.af.organizationfunction.ExecutorByNameFunction. Parameter is name of executor. The initializator returns user or group of users.
- ru.runa.af.organizationfunction.DemoChiefFunction. Parameter is user id. The initializator returns the chief of this user (demo specific class)

Parameters can be either string constants or process variable values. Variable values must be enclosed in special braces -- ${} (e.g. ${variable name}). In this case during execution orgfunction class receives corresponding variable value.

Note. In jPDL there's no specification for the swimlane initialization.

File structure description

Processdefinition.xml file is an XML document with process-definition tag as root tag.

Tag process-definition is shown below.

An example of processdefinition.xml:

<?xml version="1.0" encoding="UTF-8"?>

<process-definition name="MainProcess" xmlns="urn:jbpm.org:jpdl-3.2">

<description><![CDATA[MainProcess for subprocess demonstration (no start form proveded)]]></description>

<swimlane name="a_role">

<assignment class="ru.runa.wf.jbpm.delegation.assignment.AssignmentHandler" config-type="configuration-property"><![CDATA[]]></assignment>

</swimlane>

<start-state name="start">

<task name="start" swimlane="a_role"/>

<transition name="tr1" to="before subprocess"/>

</start-state>

<task-node name="before subprocess">

<description><![CDATA[before subprocess]]></description>

<task name="before subprocess" swimlane="a_role"/>

<transition name="tr1" to="process-state"/>

</task-node>

<task-node name="after subprocess">

<description><![CDATA[after subprocess]]></description>

<task name="after subprocess" swimlane="a_role"/>

<transition name="tr1" to="end"/>

</task-node>

<event type="subprocess-created">

<action class="ru.runa.wf.jbpm.delegation.action.SetSubProcessPermissionsActionHandler" config-type="configuration-property"><![CDATA[]]> <nowiki></action>

</event>

<process-state name="process-state">

<transition name="tr1" to="after subprocess"/>

<sub-process binding="late" name="SubProcess"/>

<variable access="read" mapped-name="role1" name="a_role"/>

<variable access="read,write" mapped-name="n" name="number"/>

<variable access="read,write" mapped-name="d" name="date"/>

</process-state>

<end-state name="end"/>

</process-definition>

Description of process-definition tag

The first tag element is an optional "description" element. Then follows a block with swimlanes tags, a block with type tags and a start-state tag which is a first action-node in the business process. Start-state tag must contain a transition tag to the next node of the business process and a swimlane which initializes with the user that starts the process instance.

And then follows a list of tags that correspond to the rest of business process nodes. Their types are:

- state (action-node)
- decision (exclusive choice)
- fork (parallel split)
- join (synchronization)
- process-state (sub-process)

Every action-node must contain an "assignment" tag with a "swimlane" parameter. The value of this parameter defines the executor that will receive the task when a business process instance reaches this action-node. Every node must contain one or several links to the next business process node(s).

Then follows an "end-state" tag that corresponds to a business process finish. When business process comes to this state it ends.

Swimlane element

Swimlane is a special type of business process variables. It is used to determine which users can perform a certain task. A swimlane corresponds to a task.

Start-state element

Start-state element corresponds to a process starting point. There must be only one such element in a business process definition. Unlike the rest of action-nodes start-state contains a swimlane parameter. The value of the parameter is an id of the user who starts the process. There might be a graphical form associated with a start-state. If such form exists it is shown right after a process start button is pressed. The business process instance will start only after a correctly filled start-state form is submitted.

State element

State element corresponds to an activity node (in UML terminology).

Assignment element

This element is used to define the users who can perform a certain task. The swimlane parameter is used to link a particular swimlane to an action-node. If this swimlane was not initialized by a user id earlier, after the user performs the task his/her user id is assigned to corresponding swimlane variable.

Embedded action element

You can attach actions (implemented as a special type of java-class) to a state element. These actions will take place if some events occur. Possible events are:

- Business process instance reached an action-node
- Business process instance left an action-node
- etc

More specifically action element is described below in this document.

Embedded transition element

This element indicates the next node of the process.

More specifically transition element is described below in this document.

Milestone element

Milestone element corresponds to milestone pattern. Current version of RunaWFE jPDL doesn't support this element.

Process-state element

Process-state element starts subprocess. The main process waits in this state until the subprocess instance finishes.

Decision element

Decision element corresponds to "exclusive choice" node. It chooses one of the several possible transition on the basis of the current values of business process instance variables. The rules that define the choice are set with the help of delegation tag. RunaWFE class ru.runa.wf.jbpm.delegation.decision.BSFDecisionHandler makes a choice based on script from BeanShell tag body (see http://www.beanshell.org/intro.html [3]). You can have access to all business process variables from the configuration (It's recommended to cast variables explicitly to the right type). The script should return a String value that matches with one of the transitions' names.

Example of tag usage:

```
<decision name="check business trip type">
<delegation class="ru.runa.wf.jbpm.delegation.decision.BSFDecisionHandler">
<![CDATA[
if( (String) businessTripType.equals("local"))
return "local";
else
return "toAnotherRegion";
]]>
</delegation>
```

<transition name="local" to="done"/>

<transition name="toAnotherRegion" to="Make an order"/>

</decision>

Fork element

Fork element corresponds to a "parallel split" node. When token comes to a fork for each transition that comes out of the fork a new token is generated. All generated tokens are running simultaneously. A closing "join" element is obligatory. Join gathers all fork generated tokens back together. Fork and join comprise a so called "parallel block" with one token at the start and one token at the end of the block, no "sideways" tokens are allowed.

Note. Fork element behavior can be redefined via delegation. In order to do it you should implement ForkHandler interface.

Join element

Join element corresponds to a "synchronization" node. It has only one outgoing transition. This node gathers all tokens that were generated in fork element. After all tokens comes to "join" a single token comes out and goes to the next business process node.

Note. Join element behavior can be redefined via delegation. In order to do it you should implement JoinHandler interface.

End-state element

End-state element corresponds to the process end. There must be only one element of this type in business process definition. When token comes to this node business process completely finishes.

Transition element

Transition element defines a transition between business process nodes.

Action element

Action element defines java code that is executed by WF-system core when particular events occur during business process execution.

Action element behavior can be defined via delegation. In order to do it you should implement ActionHandler interface.

Delegation element

Delegation is a way to enable a developer to incorporate his/her own Java classes to a business process. There is a specific class loader that allows to load these classes to the system core.

A java class must implement a specific interface depending on in which tag delegation tag is used. For example, in case of action tag this must be an ActionHandler interface, in case of decision a DecisionHandler interface, etc. A delegation class also always implements interface Configurable.

Delegation is set via:

1. A name of a class used for delegation – class attribute (required)
2. A configuration for delegation – #PCDATA in the tag's body (optional)

Description of forms.xml

In JBOSS JBPM forms.xml file structure is not formalized and this file is not required. Not so in RunaWFE. In RunaWFE it is required and strictly defined. File consists from one "forms" tag. In the "forms" tag there's a set of "form" tags. Every "form" tag corresponds to a node with a graphical form or to a node where business process variables validation is used.

Tag has 3 required attributes:

- state – is a name of business process node
- file – file name of a form with a graphical form representation.
- type – тип формы ("html" (a vartag form), "ftl" (a freemarker form), "xsn" (an infopath form))

The complete version of XML scheme for forms.xml can be found in the resource directory of Runa WFE distribution.

Description of *.html/*.ftl files

Every file contains the activity form description in HTML. The reference form parsing mechanism uses HTML with additional customtag tag or freemarker tags. You cannot use customtag tag and freemarker tags simultaneously in one form. These tags are used to display process variable values in the form.

A customtag tag have the following attributes:

- var – process variable name
- delegation – name of Java class, responsible for the variable value rendering (must implements ru.runa.wf.web.html.VarTag interface)

Examples of process archives development.

HelloWorld process.

The process scenario:

- At start HelloWorld form appears.
- When the button “complete” is pressed the process ends.

This process has two nodes:

- Start-state
- Stop-state

The content of processdefinition.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- process-definition tag beginning  -->
<process-definition name="Hello World" xmlns="urn:jbpm.org:jpdl-3.2">
    <!--  Swimlane definition -->
  <swimlane name="requester" />
```

```
    <!-- Process start point -->
  <start-state name="Hello World state" swimlane="requester">
    <!-- Transition to the next state -->
     <transition to="done"/>
  </start-state>
    <!--  The state, where process ends -->
```

```
  <end-state name="done" />
```

```
<!-- process-definition tag end -->
</process-definition>
```

The content of forms.xml file:

```
<?xml version="1.0"?>

<forms xmlns="http://runa.ru/xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://runa.ru/xml forms.xsd">
```

```
  <!--  This tag links state with form file -->
  <form state="Hello World state"  file="forms/HelloWorld.form" type="html">
      <!--  No variables are initialized -->
  </form>
</forms>
```

The graph.gif file may contain the following picture:

The HelloWorld.form file content:

```
<b>Hello World!</b> <br> <br> <br>
```

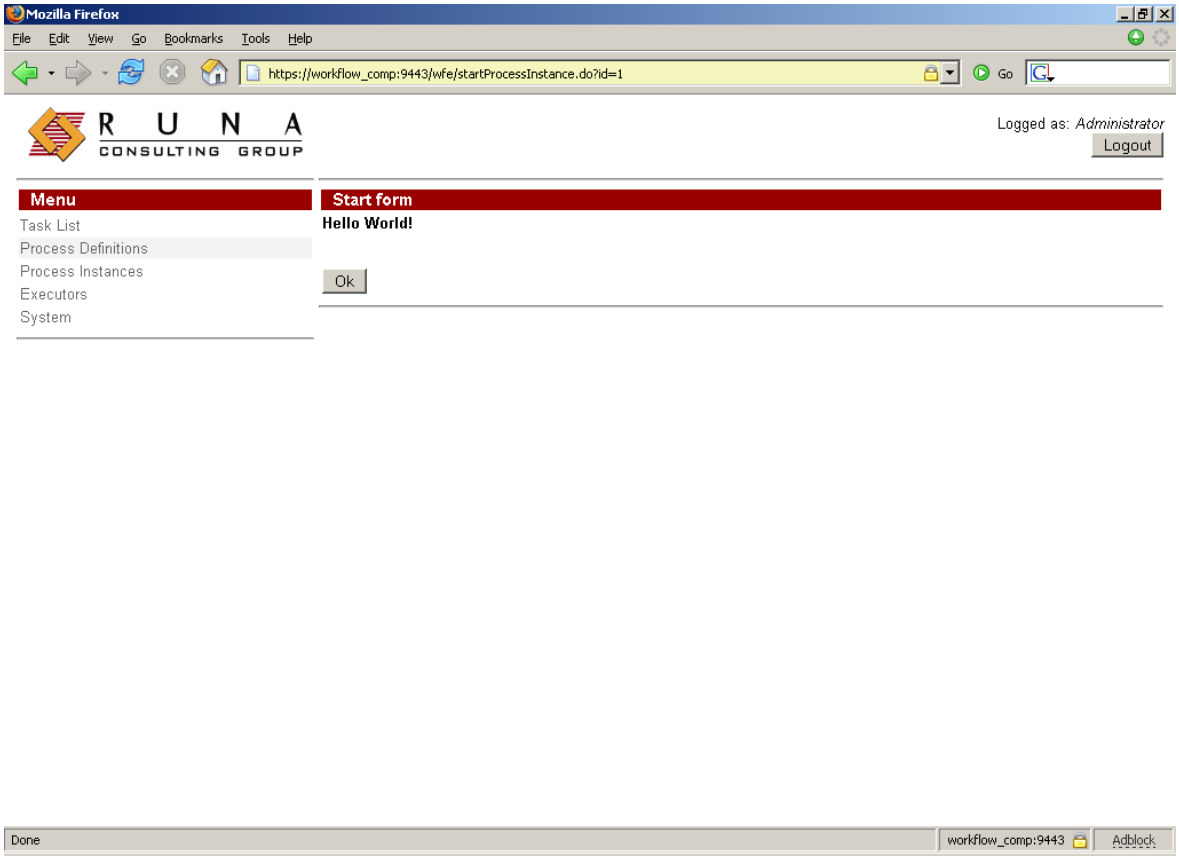
The structure of HelloWorld.par is:

File HelloWorld.par

- processdefinition.xml
- forms.xml
- graph.gif
- form "start"
- validation file start.validation.xml

Now process archive HelloWorld.par can be deployed into Runa WFE.

Process executing will produce the following task:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://workflow_comp:9443/wfe/startProcessInstance.do?id=1
Go
R U N A
CONSULTING GROUP
Logged as: Administrator
Logout
Menu
Task List
Process Definitions
Process Instances
Executors
System
Start form
Hello World!
Ok
Done
workflow_comp:9443
Adblock

OverTime process.

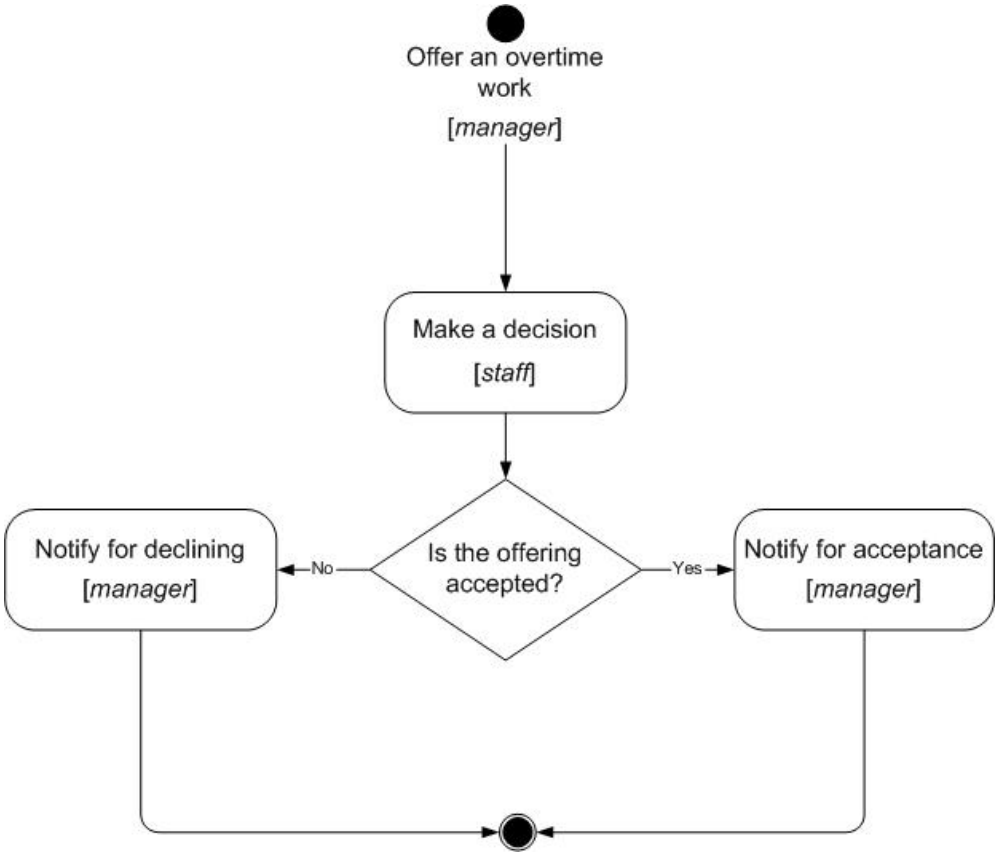
Short description:

Manager offers an overtime work to an employee. Employee accepts or rejects the offer. Then manager receives a corresponding notification.

All managers are members of group “manager” and all employees are members of group “staff”.

Process designing

Step 1. Process graph designing



Step 2. Variables setup

Establish the following variables for the OverTime process:

Variable	Description	Type	Activity, where initialization take place
staff	Employee’s ID	ID	Offer an overtime work
since	Date-time since…	Date-time	Offer an overtime work
till	Date-time till…	Date-time	Offer an overtime work
reason	OverTime reason	String	Offer an overtime work
comment	comment	Text	Offer an overtime work
staff person decision	employee decision	Boolean	Make a decision
staff person comment	employee comment	Text	Make a decision

Step 3. Task executors setup

Establish following swimlanes:

- staff person - employee

- manager – manager (chief of “staff person”)

Swimlanes initialization:

Swimlane	Who initialize swimlane
manager	Person, who starts process. (It is implied that only members of the group “manager” have permission to start this process.)
staff	Members of group “staff”.

Swimlane – activity mapping:

Activity	Swimlane
Offer an overtime work	Manager
Make a decision	Staff
Notify of rejection	Manager
Notify of acceptance	Manager

Process archive development

File processdefinition.xml

<?xml version="1.0" encoding="UTF-8"?>

<process-definition name="Overtime Work" xmlns="urn:jbpm.org:jpdl-3.2">

<description><![CDATA[Participants of this process are members of manager and staff groups]]></description>

<swimlane name="staff">

<assignment class="ru.runa.wf.jbpm.delegation.assignment.AssignmentHandler" config-type="configuration-property"><![CDATA[]]></assignment>

</swimlane>

<swimlane name="manager">

<assignment class="ru.runa.wf.jbpm.delegation.assignment.AssignmentHandler" config-type="configuration-property"><![CDATA[]]></assignment>

</swimlane>

<start-state name="Offer an overtime work">

<task name="Offer an overtime work" swimlane="manager"/>

<transition name="tr1" to="Make a decision"/>

</start-state>

<decision name="Is accepted?">

<handler class="ru.runa.wf.jbpm.delegation.decision.BSFDecisionHandler" config-type="configuration-property">

<![CDATA[if(Boolean.valueOf(staffPersonDecision).booleanValue()) return "accepted"; else return "rejected";]]></handler>

<transition name="accepted" to="Notify of acceptance"/>

<transition name="rejected" to="Notify of rejection"/>

</decision>

<task-node name="Make a decision">

<description><![CDATA[Accept or decline the offering]]></description>

<task name="Make a decision" swimlane="staff"/>

<transition name="tr1" to="Is accepted?"/>

</task-node>

<task-node name="Notify of acceptance">

<description><![CDATA[Notify that an overtime work is accepted]]></description>

<task name="Notify of acceptance" swimlane="manager"/>

<transition name="tr1" to="end"/>

</task-node>

<task-node name="Notify of rejection">

<description><![CDATA[Notify that an overtime work is rejected]]></description>

<task name="Notify of rejection" swimlane="manager"/>

<transition name="tr1" to="end"/>

</task-node>

<end-state name="end"/>

</process-definition>

File forms.xml

<?xml version="1.0" encoding="UTF-8"?>

<forms xmlns="http://runa.ru/xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://runa.ru/xml forms.xsd">

<form file="Offer an overtime work" jsValidation="false" state="Offer an overtime work" type="html" validationFile="Offer an overtime work.validation.xml"/>

<form file="Make a decision" jsValidation="false" state="Make a decision" type="html" validationFile="Make a decision.validation.xml"/>

<form file="Notify of acceptance" jsValidation="false" state="Notify of acceptance" type="html" validationFile="Notify of acceptance.validation.xml"/>

<form file="Notify of rejection" jsValidation="false" state="Notify of rejection" type="html" validationFile="Notify of rejection.validation.xml"/>

</forms>

File graph.gif

graph.gif is the picture from the section “Step 1. Process graph designing”

variables.xml file

<?xml version="1.0" encoding="UTF-8"?>

<variables xmlns="http://runa.ru/xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://runa.ru/xml variables.xsd">

<variable format="org.jbpm.web.formgen.format.DefaultFormat" name="staff"/>

<variable format="ru.runa.wf.web.forms.format.DateTimeFormat" name="since"/>

<variable format="ru.runa.wf.web.forms.format.DateTimeFormat" name="till"/>

<variable format="org.jbpm.web.formgen.format.DefaultFormat" name="reason"/>

<variable format="org.jbpm.web.formgen.format.DefaultFormat" name="comment"/>

<variable format="org.jbpm.web.formgen.format.DefaultFormat" name="staff person comment"/>

<variable format="ru.runa.wf.web.forms.format.BooleanFormat" name="staffPersonDecision"/>

</variables>

Files *.html (or no extention)

File OfferAnOvertimeWork:

```
<table cellspacing="0">
  <tr>
    <td valign="top">
```

```
     <table cellspacing="0" bgcolor="#eeeeee" style="border-style:solid;
                                                    border-width:1px;border-color:black;">
```

```
    <tr>
          <th colspan="2">
            <h3>Offer an overtime work</h3>
            <hr>
          </td>
        </tr>
```

```
    <tr title="staff">
        <td align="right">
        Employee:
        </td>
        <td>
 <!-- Special tag, it corresponds to choice, containing the list of
group which name is defined with the help of “var=...” construction . The
 tag  returns ID of chosen group member. Tag uses delegation mechanizm
-->
    <customtag var="staff" delegation =

"ru.runa.wf.web.html.vartag.GroupMembersComboboxVarTag" />
        </td>
    </tr>
    <tr title="since">
          <td align="right">
            DateTime since (dd.mm.yyyy hh:mm):
          </td>
          <td>
            <!--Special tag for working with dates-->
            <customtag var="since"
delegation="ru.runa.wf.web.html.vartag.DateTimeInputVarTag" />
          </td>
        </tr>
```

```
   <tr title="till">
         <td align="right">
           DateTime till (dd.mm.yyyy hh:mm):
         </td>
         <td>
           <customtag var="till" delegation="ru.runa.wf.web.html.vartag.DateTimeInputVarTag" />
```

```
        </td>
      </tr>
```

```
        <tr title="reason">
          <td align="right">
            Reason :
          </td>
          <td>
            <INPUT TYPE="text" NAME="reason">
          </td>
        </tr>
```

```
        <tr title="comment">
          <td align="right">
            Comment :
          </td>
          <td>
            <textarea name="comment"> </textarea>
          </td>
        </tr>
</table>
```

Content of files

- MakeaDecision.form
- NotifyForAcceptance.form
- NotifyForDeclining.form

is similar to the content of OfferAnOvertimeWork

Validation files for form variables

File "Offer an overtime work.validation.xml"

<validators>

<field name="since">

<field-validator type="required">

<message>Field is required</message>

</field-validator>

</field>

<field name="reason">

<field-validator type="stringlength">

<message>Length cannot be more than 100 symbols</message>

<param name="maxLength">100</param>

</field-validator>

<field-validator type="required">

<message>Value is required</message>

</field-validator>

</field>

<field name="staff">

<field-validator type="required">

<message>Field is required</message>

</field-validator>

</field>

<field name="comment">

<field-validator type="stringlength">

<message>Length cannot be more than 255 symbols</message>

<param name="maxLength">255</param>

</field-validator>

</field>

<field name="till">

<field-validator type="required">

<message>Field is required</message>

</field-validator>

</field>

<validator type="expression">

<message>Till should be later that since</message>

<param name="expression">till.getTime() > since.getTime()</param>

</validator>

</validators>

Files to validate variables in other forms are similar to one above.

Process archive structure

File overTimeDemo.par

- processdefinition.xml
- forms.xml
- graph.gif
- form files
  - OfferAnOvertimeWork
  - MakeaDecision
  - NotifyForAcceptance
  - NotifyOfRejection
- validation files
  - Offer an overtime work.validation.xml
  - Make a decision.validation.xml
  - Notify of acceptance.validation.xml
  - Notify of rejection.validation.xml

After archiving it is possible to deploy process into Runa WFE.

You can find more demo processes in the Runa WFE distributive:

- VacationDemo.par – vacation
- BusinessTripDemo – business trip

- TimerDemo – timer usage example

References

[1] http://www.gnu.org/licenses/lgpl.html
[2] http://www.jbpm.org/jpdl.html
[3] http://www.beanshell.org/intro.html