Runa WFE 2.1

Developer's Guide

Legal Notice

Table of Contents

Overview of system architecture

Architecture Levels

Runa WFE contains 5 levels.
This levels are: DAO, Logic, Service, Delegate, Application.

1. DAO level does all persistent operations. (it could be substituted via Factory mechanism). You can substitute DAO implementation with EJB3/Hibernate3/JDO2 Oracle Toplink or even plain JDBC.
2. Logic level does all business logic operations (it can't be substituted).
3. Service level provides means to access Logic. Runa WFE has Session Beans implementation of service level, WEB services and plain method invocation (for performance issue) implementations are in plans.
4. Delegate level provides API for all Application level classes (it hides complexity of service invocation from developer). Each delegate implementation calls corresponding Service.
5. Application level consists of web components (vartags), bots,action and task handlers, script api and other useful stuff.

Core technologies

- Hibernate 2.1 – ORM (DAO Level)
- JAAS – authentication (logic)
- JBoss jBPM – workflow engine (logic)
- EJB 2.0 stateless session beans – for remoting and transaction demarcation (service/delegate level)
- JSP 2.0, Servlet 2.3, Struts 1.2 – for web client UI (application level)

Main packages

You will find two packages ru.runa.af and ru.runa.wf:

af – stands for authorization framework.
wf – stands for workflow.

Workflow Process Engine

Runa WFE uses JBoss JBPM as the workflow engine.
jbpm specific files names are ru.runa.wf.logic.jbpm*.

Extending RunaWFE

Runa WFE supports customization in a number of different ways:

- Action Handlers
- Task Handlers
- Organization Functions
- VarTags
- Variable Formats
- Business Calendars
- Decisions

Let's discuss these “extension points” one by one.

Action Handlers

Actions are triggered on different events from jbpm engine. For example, an action can be triggered on state-enter or state-leave event. Actions are configured in GDP. When jbpm triggers an action the specified action handler is called.

Action handler is a class that implements a simple interface

```
public interface ActionHandler {

  void execute (ExecutionContext executionContext);

}
```

`ExecutionContext` contains the process variables, links to process definition and process instance etc. It is also used to create new variables, and set their values.

Action handlers are useful for simple operations (for example to send sms-notification if the business process has reached some important state). But they also have several limitations. First of all, action handlers do not operate in a associated security context. This means they can not execute any secured methods (start processes, complete tasks, create users etc). They may not block business process execution (for example with `Thread::Sleep(...)` method). For example, if an action handler tries to send an e-mail, but the SMTP-server is not reachable, the action handler is not allowed to “delay” its execution.

Task Handlers (bots)

Bot is an abbreviation of “robot”. In Runa WFE bots are a special case of actors. The system does not distinguish bots from humans. Each bot has its credentials to login to the system, it periodically wakes up, checks its task list and performs found tasks.

Each bots tasks is processes with a corresponding task handler. Runa WFE provides several task handlers out of the box:

- `CancelProcessTaskHandler`
- `CancelThisProcessInstanceTaskHandler`
- `DatabaseTaskHandler`
- `DoNothingTaskHandler`
- `EmailTaskHandler`
- `MSWorkReportTaskHandler`
- `NotWorkingTaskHandler`
- `SleepingTaskHandler`

Implementing new task handler is as simple as implementing `ru.runa.wf.logic.bot.TaskHandler` interface:

```
public interface TaskHandler {

      public void configure(String configurationName)
                  throws TaskHandlerException;

      public void handle(Subject subject, TaskStub taskStub)
                  throws TaskHandlerException;

}
```

Once a task handler has finished its task it calls `ExecutionServiceDelegate->completeTask(...)` method to mark the task as completed and resume the business process.

Please note, that task handlers are executed in a security context of the bot-user and may perform any operation (if allowed by the security configuration).

Task handlers can efficiently block the execution of the business process. The only thing a task handler has to do is not to call `completeTask` until it is sure it has finished the task.

For example, an e-mail sending task handler is allowed to “block” the execution of the business process this way until it is sure, that the SMTP-server has accepted the message for delivery.

Task handlers are invoked periodically when the bot checks its task list.

Organization Functions

Organization Function in Runa WFE is a function object, that operates over an enterprise organizational hierarchy domain. A function that searches a database for a boss of a given employer is a simple example of an organization function.

Organization Functions are used in Runa WFE primarily to initialize business process swimlanes. They are also used in a substitution engine.

Runa WFE currently can not provide a complete set of organization functions because of the great variety of relations between employers in different organizations. This way Runa WFE provides a number of “built-in” functions:

- `ru.runa.af.organizationfunction.ChiefFunction`
- `ru.runa.af.organizationfunction.ChiefRecursiveFunction`
- `ru.runa.af.organizationfunction.DemoChiefFunction`
- `ru.runa.af.organizationfunction.ExecutorByNameFunction`
- `ru.runa.af.organizationfunction.GroupFunction`
- `ru.runa.af.organizationfunction.SQLFunction`
- `ru.runa.af.organizationfunction.SubordinateFunction`
- `ru.runa.af.organizationfunction.SubordinateRecursiveFunction`

and a possibility to write your own functions.

Organization Function is not treated in a pure mathematical sense. This way it can have side effects, return different values given the same arguments and it is not suitable for memoizing (caching of the returned values). The latter drives us to the requirement that in heavily loaded installations all used organization functions should have efficient implementation.

Let's discuss how to create a new organization function.

In order to create a new organizational function you need to implement a

`ru.runa.af.organizationfunction.OrganizationFunction` interface:

```
package ru.runa.af.organizationfunction;

public interface OrganizationFunction {

      public long[] getExecutorIds (Object[] parameters)
                  throws OrganizationFunctionException;

}
```

Once you've implemented a function you need place the compiled class under the jboss CLASSPATH. After that, you may deploy and run business processes that depend on this function.

Let's discuss the signature of the `getExecutorIds` function. It requires an array of objects as an input. This array is created from the list of actual arguments specified during business process development in GPD.

The function must return a non-null array of executor ids or throw an `OrganizationFunctionException`.

Variable Tags

Classes that implement `ru.runa.wf.web.html.VarTag` interface are used to develop business process task forms. They are used to render business process variables values in HTML.

```
package ru.runa.wf.web.html;

import javax.security.auth.Subject;

import javax.servlet.jsp.PageContext;

import ru.runa.af.AuthenticationException;

public interface VarTag {

      public String getHtml (
                  Subject subject
                  , String varName
                  , Object varValue
                  , PageContext pageContext)
                  throws WorkflowFormProcessingException
                        , AuthenticationException;

}
```

```
Derived classes must implement getHtml method to return valid HTML code.
```

Runa WFE provides a number of built-in variable tags that are located in `ru.runa.wf.web.html.vartag` package.

It is possible of course to develop custom tags. All you have to do is to implement `VarTag` interface and put the compiled code under jboss CLASSPATH as usually.

Variable Format

Variable formats are used to declare typed variables in jpdl. Technically variable formats are subclasses of `java.text.Format` (there is a move currently to make formats subclasses of a custom type `ru.runa.Format`) and implement methods to serialize/deserialize variables of supported types into/from string representation.

There are two methods that should be implemented:

```
Object parseObject(String source, ParsePosition pos)

StringBuffer format(Object obj, StringBuffer toAppendTo, FieldPosition pos)
```

Please, see `java.text.Format` documentation for details.

Business Calendar

Business Calendar is a concept needed to implement scheduler, deadlines and other features, that need to distinguish business time and real time.

Business Calendar takes care about business hours, holidays, coffee breaks etc. Runa WFE provides a default implementation of Business Calendar that is adopted from jboss jbpm3.

In case if the default implementation is not suitable one can implement its own Business Calendar by subclassing `org.jbpm.calendar.BusinessCalendar` interface.

```
package org.jbpm.calendar;

import java.util.Date;

public interface BusinessCalendar {

      Date add(Date date, String duration);

      Date findStartOfNextDay(Date date);

      boolean isHoliday(Date date);

      boolean isInBusinessHours(Date date);

}
```

It is also required to set `jbpm.business.calendar` property to point to your new implementation.

Decision Handler

Decision handlers are used in a “Decision” jpdl-element to select a single execution path among several possible. A decision handler is an implementation of

```
package org.jbpm.delegation;

public interface DecisionHandler {

  String decide (ExecutionContext executionContext);

}
```

Currently Runa WFE provides a single working decision handler `ru.runa.wf.jbpm.delegation.decision.BSFDecisionHandler` that is based on BSF scripts.

Its is easy to notice that decision handler accepts `ExecutionContext`. Actually a decision handler has to make a decision based on the business process variables values. Like action handlers decision handlers can not use secured method, they can not handle their errors and postpone the decision until good times.

A decision handler must return the name of the transition to accept.