RunaWFE. Developer guide

Version 3.0

Introduction

See About [2]

Basic components and used technologies

The following general architecture has been chosen for WF-system. It's basicly coherent with WfMC architectural proposal.

System components

- System core. (Based on JBOSS JBPM)
  - Contains a set of business processes definitions
  - Contains a set of running business processes instances
- A component that assigns executors for tasks
- Client
  - Task list. (A set of graphical forms, that contains the list of incoming tasks, sorting and filter configurations)
  - Form player (Renders forms that were developed in process editor (GPD))
  - Administrative interface
    - Shows the process state, allows to filter and stop processes
    - Allows to deploy/redeploy/remove process definitions
    - Allows to create/edit/remove users
    - Allows to set and configure permissions
  - Editor and substitutions assignment.
- Graphical processes editor (GPD).
- Graphical forms constructor (a part of GPD).
- Bot stations that contain bots. (Bots are applications of a particular type, they execute tasks just as regular system users)
- Subsystems of permissions management (authorization and authentication)

'Core technologies:'

- Hibernate 2.1 – ORM
- JAAS – authentication
- JBoss jBPM
- EJB 2.0 stateless session beans – for remoting and transaction demarcation
- JSP 2.0, Servlet 2.3, Struts 1.2 – for web client UI
- Eclipse RCP – this platform in used for GPD development

ORM Hibernate allows easily to switch the WFE-system to other DBMSes.

Programming platform and the development software used

The programming platform is J2EE.

And the development tools software is:

1. **Application server** - JBOSS (http://www.jboss.org [3]).
2. **IDE -** Eclipse от IBM (http://www.eclipse.org [4]).
3. **Code and doclets generator** – xdoclet (http://xdoclet.sourceforge.net [5]).
4. **Version Control System** – subversion (http://subversion.tigris.org/ [6]).
5. **Application Builder** – ant (http://ant.apache.org [7]).
6. **Database Server** – the following DS are supported:
   - MS SQL Server (http://www.microsoft.com/sql/evaluation/default.mspx [8])
   - MySQL (http://www.mysql.com [9])
   - HSSQLDB (http://hsqldb.org [10])
   - Oracle (http://www.oracle.com) [11]
   - Postgres (http://postgresql.com [12])
7. **Operating System —** the following OSes are supported:
   - Windows (Server 2000-2008, XP, Vista, 7)
   - ALTLinux
   - Mandriva Linux
   - Fedora
   - Debian/Ubuntu

A detailed description of the current system architecture

The project consists of several subprojects each of which contain logically connected components

Subprojects' structure

- wfe - the main subproject of RunaWFE system
- customization – additional components that can be developed by the during system customization for a particular enterprise needs.
  - organization functions
  - VarTags
  - FTLTags
  - DecisionHandlers
  - TaskHandlers
  - Validators
  - Variables Formatters
  - ActionHandlers
- web – system web-interface
- bots – bots and bot-stations
- gpd – business process graphic editor
- rtn – task notifier - notifies user when a new task arrives
- abs – files that are used to automate a full RunaWFE system building and tests running.
- os-specific - defines the targets and properties that are necessary for correct RunaWFE building on different OSes.
- docs - project documentation.

Main subsystems

- jbpm - workflow system core (is adopted from JBOSS JBPM, and a lot of enhancements are made to it) (it is located in wfe project)
- af - authorization and authentication subsystem. It is not depended on other subsystem and can be used separately, apart from RunaWFE (located in wfe subproject)
- wf – the workflow subsystem. It depends on af and jbpm. (located in wfe subproject)
- web interface (located in web subproject)

RunaWFE - Workflow-environment

The main functionality of RunaWFE is concentrated in 2 subsystems:

- af - authorization and authentication sub systems.
- wf – main workflow subsystem. Depends on af.

System architecture layers

Each subsystem contains several logical components that are relevant to each other. Each component consists of several layers.

The layers are:

- delegate
- service
- logic
- dao
- hibernate

Delegate Layer

Delegate-interfaces and DelegateImpl-classes that implement them are designed according to BusinessDelegate programming pattern and they simplify access to the server side workflow API. Client application or bot interacts with workflow system only via Delegate-classes. DelegateFactory provides access to the Delegate interfaces based on the configuration used. Delegate classes are a part of client API.

The main Delegate-interfaces are:

- af
  - AuthenticationServiceDelegate – contains methods for users authentication in the system (via entering login/password, Kerberos and etc). It also allows to get the current user from the Subject object.
  - AuthorizationServiceDelegate — contains methods for work with users' permissions (assigning and verifying permissions for users etc).
  - BotsServiceDelegate — contains methods for work with bots and botstations.
  - ExecutorServiceDelegate — contains methods for work with executors. Allows to create/delete/edit executors and to add them to the groups.
  - ProfileServiceDelegate — contains methods for work with users profiles.
  - SubstitutionServiceDelegate — contains methods for substitution rules management.
  - SystemServiceDelegate — contains methods for user log in and log out.
- wf
  - DefinitionServiceDelegate — contains methods for work with process definitions.
  - ExecutionServiceDelegate — contains methods for work with process instances

All currently existing Delegate-classes implement the required functionality with the help of EJB (stateless session beans)

Service Layer

Service Layer – is a server side API that provides system access point. Delegate interface implementation address directly to this layer. Every Delegate has a corresponding Service class (one-to-one correspondence). Currently all developed service classes and interfaces are oriented on EJB technology. However it's possible to make other implementation in the future. Service classes implementations are Stateless Session Bean EJB, that support transactional calls via declaration and delegates the calls to the corresponding classes from Logic layer.

Thus, classes "Delegate – Service - Logic" form a communication between client and server.

Logic Layer

Logic Levei – is a system business logic implementation. This layer is connected with JBOSS JBPM core interface and DAO layer classes to access persisted data.

Dao Layer

Dao Layer – contains classes and interfaces that provide access to the persisted data (from DS). This layer is implemented only for af-system. In case of wf-system the access is carried out differently (via JBOSS JBPM interfaces). Currently all Dao classes of the system are implemented with the help of ORM Hibernate.

Hibernate Usage

Hibernate is an ORM (Object/Relational Mapping) implementation. It maps an object oriented architecture to the relational data structure. It allows to switch between a wide variety of RDBMS without changing the application code. Among supported DBMS are:

- MySql
- HSQLDB
- Oracle
- MS SQL Server
- etc

Hibernate supports distributed transactions, automatically creates tables for new classes etc. All data manipulations within RunaWFE system is carried out only via Hibernate.

Interfaces that are accessible as web-services

Currently only a limited functionality is accessible via web-services. It's sufficient only for the basic interaction with RunaWFE system. But the work to extend the number of accessible interfaces is in progress.

Wsdl of web services can be found here: /wf.webservice/ExecutionBean?wsdl and /af.webservice/AuthenticationBean?wsdl

Components' location in the file system

The list of all components modules:

- runa-common.jar
- af.core.jar
- af.logic.jar
- af.service.jar
- af.delegate.jar
- af.webservice.jar

- wf.core.jar
- wf.logic.jar
- wf.service.jar
- wf.delegate.jar
- wf.webservice.jar
- wfe.war
- wfe-bot.jar
- wfe-botstation.jar
- jbpmdelegation.jar

Additional modules (jbpm library with "out" patches)

- jbpm.core.jar

A more detailed modules description.

- runa-common.jar

Classes that are used by all other components

Dependences: none

Authorization and authentication system (af):

- af.core.jar

Core classes of af-system (Actor, Group etc)

Dependences:

  - runa.commons.jar

- af.logic.jar

Implements logic and Dao layers for af-system

Dependences:

  - runa.commons.jar
  - af.core.jar

- af.service.jar

Implements services layer for af-system

Dependences:

  - runa.commons.jar
  - af.core.jar
  - af.logic.jar

- af.delegate.jar

Implements Delegate layer for af-sysetem

Dependences:

  - runa.commons.jar
  - af.core.jar

- af.webservice.jar

Implements web services for af-system

Dependences:

  - runa.commons.jar
  - af.core.jar
  - af.logic.jar

Workflow subsystem (wf):

- wf.core.jar

Core classes for wf

Dependences:

  - runa.commons.jar

- wf.logic.jar

Implements logic layers for wf

Dependences:

  - runa.commons.jar
  - wf.core.jar
  - af.logic.jar
  - jbpm2.core.jar

- wf.service.jar

Implements services layer for wf

Dependences:

  - runa.commons.jar
  - wf.core.jar
  - wf.logic.jar

- wf.delegate.jar

Implements Delegate layer for wf

Dependences:

  - runa.commons.jar
  - wf.core.jar

- af.webservice.jar

Implements web services for wf-system

Dependences:

  - runa.commons.jar
  - wf.core.jar
  - wf.logic.jar

- wfe-botstation.jar

Implements functionality that is necessary for bot's support

Dependences:

  - runa.commons.jar
  - af.core.jar
  - wf.core.jar
  - af.delegate.jar
  - wf.delegate.jar

Web interface:

- wfe.war

Implements tags and graphical forms, etc.

Dependences:

- runa.commons.jar
- af.core.jar
- wf.core.jar
- af.delegate.jar
- wf.delegate.jar

Bots:

- wfe-bot.jar

Implements functionality that is related to bots (TaskHandler, BotInvoker и т.д.)

Dependences:

- runa.commons.jar
- af.core.jar
- wf.core.jar
- af.delegate.jar
- wf.delegate.jar

jbpm-delegation:

- jbpmdelegation.jar

Implements mechanisms jbpm-delegation that are used for the whole bundle (see RunaWFE. The Business process developer guide).

Dependences:

- runa.commons.jar
- af.core.jar
- wf.core.jar
- af.delegate.jar
- wf.delegate.jar

jbpm:

- jbpm.core.jar

jbpm is a workflow core of jBoss jBPM project. It is being used as a library. RunaWFE releases contain a binary core code.

Dependences: none

A description of main ant tasks

The most important and often used tasks are defined in the root:

- install.wfe — builds and installs RunaWFE.
- install.simulation — builds and installs RunaWFE simulator.
- install.remote.bots — builds and installs a RunaWFE remote botstation.
- test.wfe — tests RunaWFE by running the tests from wfe and web. Jboss would be started automatically before the tests run.

Main tasks in subprojects.

1 wfe

- dist – builds libraries of wfe subproject.
- deploy – copies RunaWFE libraries and configuration to jboss. The libraries are built if needed
- install – copies RunaWFE libraries and configuration to jboss. Also necessary third-parties libraries are copied and jboss files are configured to work with RunaWFE.

- test – tests system, running the tests from af.build and wf.build. When run this task jboss with RunaWFE must be already running.

2 web

- copy.libs – synchronizes RunaWFE libraries that are necessary to build the subproject with what is on the jboss.
- dist – builds web subproject libraries
- deploy – copies RunaWFE libraries and configurations to jboss. The libraries are built if needed.
- test – tests the system by running tests from af.build and wf.build. When run this task jboss with RunaWFE must be already running.

3 bots

- copy.libs – synchronizes RunaWFE libraries that are necessary to build the subproject with what is on the jboss.
- dist – builds bots subproject libraries
- deploy – copies RunaWFE libraries and configurations to jboss. The libraries are built if needed.

4 rtn

- copy.libs – synchronizes RunaWFE libraries that are necessary to build the subproject with what is on the jboss.
- dist – builds rtn subproject libraries

5 gpd

- copy.libs – synchronizes RunaWFE libraries that are necessary to build the subproject with what is on the jboss.
- dist – builds rtn subproject libraries. The build process is carried out for several platforms with the help of delta-pack.

6 abs

- build – runs an automated build process.

Subprojects description

Wfe subproject

Subproject contains a workflow core. The main components that subproject contains are:

- shared files
- authorization subsystem (af)
- bots and botstations support
- jbpm with all necessary patches
- workflow subsystem (wf)

The main subproject folders are:

- resources — contains subproject resources
- samples — contains demo processes
- src — contains source code
- lib – contains libraries that are necessary for building

Source code folders (packages) structure

- af – authorization subsystem. It's being compiled into af.core.jar, af.logic.jar, af.service.jar, af.delegate.jar, af.weservice.jar. It also contains tests for the authorization subproject (in test subfolder).
- bot – bots and botstation support. It's being compiled into wfe-botstation.jar.
- common – shared files. They are being compiled into runa-common.jar.
- jbpm – the (jbpm) system core. Compiled into в jpm.core.jar.
- wf – workflow subsystem. Compiled into wf.core.jar, wf.logic.jar, wf.service.jar, wf.delegate.jar, wf.weservice.jar. It also contains tests for the workflow subsystem.

Resources structure. Only the most interesting settings are described.

- adminkit – administrative scripts that are placed into JBOSS_HOME/adminkit folder during the build process.
- af — settings for af subsystem
- af_delegate.properties — setting for af system delegates
- af_logic.properties — sets the default properties for a system administrator as well as a list of actions that must be done during the user login and user status change.
- kerberos_module.properties — settings for the kerberos authentication via RMI.
- login_module.properties — login modules login. Permits or prohibits authentication via some module.
- ntlm_support.properties — settings for ntlm authentication via RMI.
- bot — botstation settings.
- bot_invoker.properties — sets the schedule for the botstation calls and a class for bot's invocation.
- distr-build — files that are used for different distributives build.
- jboss-configuration — files that are used for jboss configuration.
- remote-bots — files that are used to build a remote botstation.
- wf — settings for workflow subsystem.
- graph.properties — process graph generation settings.
- runa_loading.properties — an order in which RunaWFE libraries are loaded during the system start.
- wf_logic.properties — the main setting is the period of time between each transfer of finished processes logs to the separate tables.

All the components that provide client API include the layers described above. To optimize performance caching is used in some components, it's implemented either on logic or on DAO layer. The main cache management part is in CachingLogic, WFRunaHibernateInterceptor classes and in corresponding to classes cache classes. Changes in cached objects are intercepted in WFRunaHibernateInterceptor and are reported by CachingLogic to all classes that are subscribed to cached objects change event. A transaction that changes cached is considered committed after the ejb call is processed.

During the development sometimes it's necessary to change data base scheme. In order to do it you will need to make a data base patch. A data base patch is being applied at the system start time and in general case it can change not only the DB structure but also make any other actions. In order to add a patch:

1. Write a patch that implements ru.runa.commons.dbpatch.DBPatch interface. In the apply method patch should make all the work and return true if the patch is successfully applied to the system.
2. Add the newly created patch to the end of the array dbPatches in ru.runa.af.logic.InitializerLogic class.

JBPM logs what happens with business process instance in logging table JBPM_LOG. The size of this table grows fast and it may lead to a system performance degradation (during operations that are followed up by inserting a log record to the logging table). To mend this situation the a special feature is added. That helps to transfer finished processes logs record from JBPM_LOG to separate tables. It allows to make JBPM_LOG table smaller. The time period between checks for the log records that should be transferred is set in wf_logic.properties with the help of ru.runa.wf.logrotation.period property. If a negative value is set the logs are not transferred automatically. (In this case you should manage the logs transfer in some other ways).

This subproject also contains a part of client RunaWFE API implementation in a form of web-services. This implementation is located in src/af/webservice and src/wf/webservice folders. Currently a part of client RunaWFE API that is implemented via web-services continues to grow.

Customization subproject

Customization subproject contains RunaWFE extensions. The main components that comprise the subproject are:

- actionHandlers — handlers that are used in business processes.
- decisionHandlers — components that are used to "make a decision" in the decision element.
- formatters — are used to convert values to string and back.
- Org-functions — are used to assign swimlanes and in substitution rules to find a substitutor.
- validators — are used to validate variable values in forms.
- varTags — generates html code to represent a variable in a web interface.
- ftlTags (under varTags folder in customization project) - generates html code to represent a variable in a web interface.

The main subproject folders:

- resources — contains subproject resources
- src — contains source code
- lib – contains libraries that are necessary for building

Resources structure. Only the most interesting settings are described.

- ActionHandlers – folder contains settings for various ActionHandlers.
- autoShowForm.properties — after an action is performed if there's a form in the current process that should be shown to the user it will be shown immediately (instead of opening the user tasks list) if auto.show.form is set to true.
- confirmationPopup.properties — settings for confirmation dialogs.
- emailTaskNotifier.properties — settings for notifying users about new tasks by email.

Web subproject

Contains RunaWFE web-interface.

Web interface is based on JSP and Struts technologies.

Main folders of subproject:

- resources — contains subproject resources
- src — contains source code
- lib – contains libraries that are necessary for building

Resources structure. Only the most interesting settings are described.

- merge tooltip web — are used during .war file creation
- portlet — contains files that are used for building an interface in portlets form.
- kerberos_web_support.properties ntlm_support.properties — settings for kerberos and ntlm authentication via browser.

Web interface is coded in such a manner that portlet interface version is built from the same source that it used for a regular one. In order to keep the source code in a form that allows correct portlet-version interface built it's necessary:

- to form URLs in jsp pages with the help of html:rewrite (See main_layout.jsp).
- to use methods from ru.runa.common.web.Commons to form URL in java code.

In order to setup RunaWFE with portlets it's necessary:

1. Use jboss-portal for a jboss server
2. Install RunaWFE (ant install.wfe from the project root)
3. Build web subproject using portlet.dist (ant portlet.dist).

4. Replace old wfe.war file in в JOSS_HOME/server/default/deploy to a newly built wfe.war from deploy folder (target folder of portlet.dist).
5. Copy jbpm.core.jar from server/default/deploy to server/default/lib
6. Remove jbpm-*.jar from server/default/deploy/jboss-portal.sar

Bots subproject

Contain bots and standard bot invokers for RunaWFE system.

Main folders of subproject:

- resources — contains subproject resources
- src — contains source code
- lib – contains libraries that are necessary for building

Resources structure. Only the most interesting settings are described.

Mainly the default configurations are placed in bots configurations. Most interesting are files:

- bot_invoker.properties — bot invoker setting (set a class and a value for invocation period)
- botstation.xml — botstation settings. User name for botstation and the number of the threads that could be used by bots to perform tasks.

Rtn subproject

Contains RunaWFE task notifier client.

Main folders of subproject:

- resources — contains subproject resources
- src — contains source code
- lib – contains libraries that are necessary for building

Gpd subproject

Contains graphical process editor. See Process editor developer guide for documentation.

Os-specific subproject

Contains ant properties and tasks that are specific for different platforms.

Abs subproject

Contains files for automated build.

Handler Interfaces

It's possible to trigger java-code execution in RunaWFE when certain events happen, for instance when process execution passes a certain transition. Java code must be placed in execute() method of a class that implements a handler interface. This class must be loaded in RunaWFE.

The list of the Handler interfaces:

- ActionHandler – allows to make custom Action implementations
- TaskHandler - allows to implement tasks for bots
- AssignmentHandler – allows to make custom swimlane implementations
- DecisionHandler – allows to make custom Decision element implementations
- ForkHandler – allows to make custom Fork element implementations
- JoinHandler – allows to make custom Join element implementations
- ProcessInvocationHandler – allows to implement customized ways to start subprocesses

BSHActionHandler

BSHActionHandler is implemented in RunaWFE and is used for recalculating and reassigning variable values in business processes.

This handler configuration must be a well defined BeanShell code (www.beanshell.org [13]),

whose syntax is very similar to Java and it is possible to make Jave calls from it.

Configuration example:

```
My_date = new java.util.Date();
My_rnd = new java.util.Random(1000).nextInt();
My_time = java.lang.System.currentTimeMillis();
```

The variables whose values were modified in the script will change values in business process. Only those new variables that were declared in business process variables list will be saved, creating new undeclared variables is not available.

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 GPD. All implemented ActionHandlers in RunaWFE are a part of **customization** subproject.

ActionHandler must implement org.jbpm.graph.def.ActionHandler interface. ActionHandler instance in RunaWFE is created with the help of default constructor and with passing parameters via setConfiguration(String) method call.

When writing a new ActionHandler it is recommended to inherit it from BaseActionHandler abstract class (it is also a part of customization subproject). In setConfiguration method handler saves its parameters if it expects it, then in execute() method the execution of necessary routine happens.

If during ActionHandler execution an exception occurs the transaction will be rolled back and the business process will be returned to its previous state which caused the ActionHandler execution.

ActionHandlers are useful for simple operations (for example to send sms notification if a business process has reached some important state). But they also have several limitations. First of all, ActionHandlers 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 ActionHandler tries to send an e-mail, but the SMTP-server is not reachable, the ActionHandler 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 wakens up, checks its task list and performs found tasks.

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

- CancelProcessTaskHandler
- CancelThisProcessInstanceTaskHandler
- DatabaseTaskHandler
- DoNothingTaskHandler
- EmailTaskHandler

- MSWorkReportTaskHandler

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 to resume the business process.

Please note, that task handlers are executed in a security context of the bot and may perform any operation (if allowed by the bot 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 positive that 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 positive, that the SMTP-server has accepted the message for delivery.

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

Decision Handler

Desicion handler is a java class that implements org.jbpm.delegation.DecisionHandler interface. There is only one method in this interface, that is decide(ExecutionContext executionContext). This method returns the name of transition to be taken (one of all that exits this decision).

Decision handler accepts ExecutionContext as an input parameter. Actually a decision handler has to make a decision based on the business process variables values and conditions from the configuration. Like action handlers decision handlers can not use secured method, they can not handle their errors and postpone the decision until the good times.

Currently there are the following interface implementations:

- ru.runa.wf.jbpm.delegation.decision.BSFDecisionHandler – a handler based on BSF script, (it is part of customization subproject)
- org.jbpm.delegation.decision.ExpressionDecisionHandler – unfinished class from JBOSS jBpm project

This classes may be configured with the help of constructor that accepts String as parameter or via default constructor and method setConfiguration(). The selection of configuration method depends on business process in question and it is recommended to implement both ways.

In business process (see processdefinition.xml file) Decision handler is placed in <decision> tag body in handler subelement. As class parameter for the <delegation> tag the class with decision handler is indicated, config-type parameter sets the way of creating and configuring the handler class instance (via setConfiguration() by default), and inside <handler> tag the configuration is placed.

Organization Functions

Organization Function in Runa WFE operates over an enterprise organizational hierarchy domain. Organization functions are used in RunaWFE primarily to initialize business process swimlanes. They are also used in a substitution engine.

In order to create a new orgfunction 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;
}
```

The getExecutorIds() function requires an array of objects as its parameter. This array is created from a list of actual arguments specified during business process development in GPD.

An orgfunction must return a non-null array of executor ids or throw an OrganizationFunctionException.

In a business process (see processdefinition.xml file) orgfunction is used to define swimlanes. It is in the <swimlane> tag inside <assignment> tag. The ru.runa.wf.jbpm.delegation.assignment.AssignmentHandler class is indicated as a "class" parameter of the <assignment> tag .

Inside <assignment> tag there is a configuration for the AssignmentHandler class. It is the name of the class that implements OrganizationFunction interface and a list of parameters in parenthesis. A parameter may be a definite value or a name of business process variable in curly braces with dollar sign in front of it (for instance ${variableName}).

Variable Format

Variable formats are used to declare typed variables in jpdl. Technically variable formats are subclasses of ru.runa.commons.format.WebFormat.

There are two methods that should be implemented:

```
public Object parse(String[] source) throws ParseException;
public String format(Object object);
```

Variable format class transforms data from HTTP request string into Java class instance (which represents a business process variable)

In RunaWFE variable format classes are used to define business process variables in <variable> tag of forms.xml file.

Currently the are the following variable format classes in RunaWFE:

- ru.runa.wf.web.forms.format.BooleanFormat – format for variables of boolean type
- ru.runa.wf.web.forms.format.DateFormat – format for variables of Date type
- ru.runa.wf.web.forms.format.DateTimeFormat – format for variables of DateTime type
- ru.runa.wf.web.forms.format.TimeFormat – format for variables of Time type
- org.jbpm.web.formgen.format.DefaultFormat – default format for variables of String type
- org.jbpm.web.formgen.format.DoubleFormat – format for numeral variables with floating point
- ru.runa.wf.web.forms.format.StringFormat – format for variables of String type
- ru.runa.wf.web.forms.format.ArrayListFormat – format for variables of java.util.ArrayList type
- ru.runa.wf.web.forms.format.StringArrayFormat – format for variables of (String[]) type

- ru.runa.wf.web.forms.format.LongFormat – format for variables of integer type (java Long type)
- ru.runa.wf.web.forms.format.FileFormat – format for file variables of FileVariable type

Variable format class should be serializable.

Variable Tags

This way of developing custom tags for RunaWFE is depricated. The ftl tags are recommended instead. 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;
}
```

Interface contains one method getHtml(Subject subject, String varName, Object varValue, PageContext pageContext) where varName is the name of business process variable and varValue is the variable value. This method (in a class implementing VarTag) will return HTML code that will be inserted into corresponding place of business process HTML form.

The graphic element of the data input or output in a business process form in RunaWFE is designated with <customtag> tag.

Tag parameters are:

- var – business process variable name
- delegation – the name of a class that implements ru.runa.wf.web.html.VarTag interface

Currently there are the following VarTag interface implementations in RunaWFE:

- ActorFullNameDisplayVarTag - a class for the output of Actor's full name
- ActorNameDisplayVarTag - a class for the output of Actor's name
- SubordinateAutoCompletingComboboxVarTag – a class for the output of the subordinates hierarchy
- AbstractDateTimeInputVarTag – an abstract class for the date and time input
- DateInputVarTag – a class for the date input
- DateTimeInputVarTag – a class for the date and time input
- ComboBoxVarTag - an abstract class for choosing Actors from a list
- ActorComboboxVarTag - a class for choosing Actors from a list
- AutoCompletionComboBoxVarTag
- GroupMembersComboboxVarTag - a class for choosing Actors from a list of certain group member
- DateTimeValueDisplayVarTag - a class for the output of date and time
- DateValueDisplayVarTag - a class for the output of date
- FileVariableValueDownloadVarTag - a class for the output of file name available for download

- HiddenDateTimeInputVarTag
- TimeValueDisplayVarTag
- VariableValueDisplayVarTag - a class for the output of string variables

FreeMarker tags

Classes that extend abstract class ru.runa.commons.ftl.FreemarkerTag in RunaWFE system are used in development of FTL forms for business processes.

It is necessary to implement method:

```
protected abstract Object executeTag() throws TemplateModelException;
```

Initialized variables of type Subject, PageContext are available in this method as well as a list of form variables. The return value might be any object that can be further subjected to any operations with freemarker expression language.

An example:

```
public class GroupMembersTag extends FreemarkerTag {

    @Override
    protected Object executeTag() throws TemplateModelException {
        String varName = getParameterAs(String.class, 0);
        String groupVarName = getParameterAs(String.class, 1);
        String view = getParameterAs(String.class, 2);
        String groupName = (String) variables.get(groupVarName);

        List<Actor> actors = getActors(subject, groupName);
        if ("all".equals(view)) {
            return getHtml(actors, varName);
        } else if ("raw".equals(view)) {
            return actors;
        } else {
            throw new TemplateModelException("Unexpected value of VIEW parameter: " + view);
        }
    }

    @Override
    protected int getParametersCount() {
        return 3;
    }

}
```

How to add Ajax support to a form

Classes that extend abstract class ru.runa.wf.web.ftl.AjaxFreemarkerTag (while it extends ru.runa.commons.ftl.FreemarkerTag) are used in RunaWFE system for user-server interaction on a form with the help of Ajax technologies.

Methods to implement are:

```
protected abstract String renderRequest();
```

```
public void processAjaxRequest(HttpServletRequest request, HttpServletResponse response);
```

renderRequest is called during initial process of tag output in the form, while **processAjaxRequest** is called with the help of JavaScript from the form (either periodically or triggered by user actions).

See ru.runa.wf.web.ftl.tags.AjaxGroupMembersTag class for example. In this class **renderRequest** renders 2 combo boxes (a list of groups and a list of users of group), while **processAjaxRequest** is called when user makes a choice of group in the first combo box.

Adding links on RunaWFE mainpage

In order to add a link to some web page from the mainpage of RunaWFE web interface do the following:

1. Create a class similar to this example:

```
package ru.runa.wf.web;
public class RunaMainPageLinks {
   public static String getAdditionalLinks() {
  return "<table class='box'>"
               + "<th class='box'>Some Header"
               + "<tr><td class='tab'><A HREF=\"http://somesite\">Some Site</A>"
               + "";
   }
}
```

2. Put the name of this class into main_page.properties file. For the example class the file contains will look like this:

```
    ru.runa.web.additional_links=ru.runa.wf.web.RunaMainPageLinks
```

3. Put main_page.properties file to the server/default/conf folder on jboss RunaWFE server.

Authentication and Authorization systems

Authentication

Authentication in RunaWFE system is based on JAAS (while authorization is a part of RunaWFE code). According to JAAS for each kind of authentication there must be a class that implements javax.security.auth.spi.LoginModule.

LoginModule classes that are developed in RunaWFE system are placed in ru.runa.af.authentication. There are the following login modules:

- for authentication in inner database (installed by default in system distributive)
- a stub for LDAP authentication
- for AD authentication
- for Kerberos authentication

- for NTLM authentication

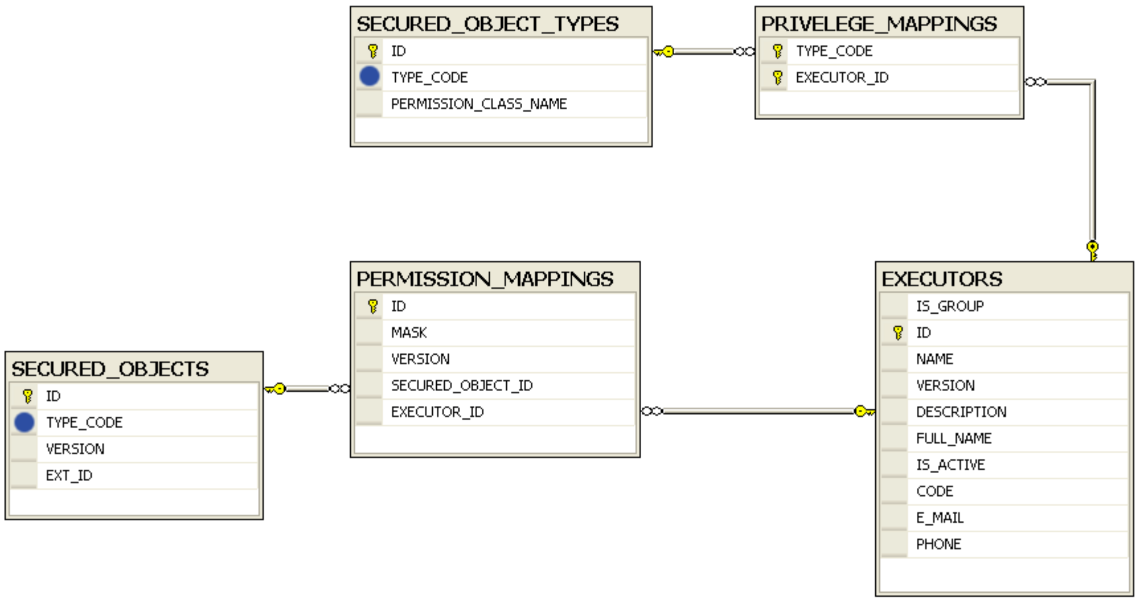
Authorization

The description of authorization logic is placed in «RunaWFE. Administrator guide» document.

The implementation of authorization system is in ru.runa.af folder

Description of crucial entities:

Class (DB table)	**Description**	**Note**
ru.runa.af.Identifiable (Interface)	It is implemented by all classes for whose instances permissions can be given	
ru.runa.af.Permission (Is not present in DB)	The instances set is a set of singleton-objects of all permissions	Implementation is not transparent: there's no enum support in Java1.4.
ru.runa.af.SecuredObject (SECURED_OBJECTS)	It is created for every protected object in the system	The protected by security system object and the SecuredObject to it are created in one transaction
ru.runa.af.dao.impl.SecuredObjectType (SECURED_OBJECT_TYPES)	It is created for every protected object in the system	
ru.runa.af.dao.impl.PermissionMapping (PERMISSION_MAPPINGS)	It represents a certain right of access for a certain user on a certain object	The MASK field bijectively defines the right of access
ru.runa.af.dao.impl.SecuredObjectType.privelegedExecutors (PRIVELEGE_MAPPINGS)	It defines a set of privileged users and groups for a given type	It is used to create default permissions for given users and groups when SecuredObject of given type is created



2 TYPE_CODE field marked by blue pegs contains the type of protected object (ru.runa.af.Identifiable.identifiableType()).

How it works

If some class implements Identifiable interface then this class objects may be protected:

- a set of permissions is defined for the given class (SecuredObjectType is created with indication of Permission child class)
- when creating objects of given class in DB it is necessary to create corresponding SecuredObject in DB
- users and groups are being given permissions on this class objects (PermissionMappings are created)

Available API

On the Delegate level (for remote call, with authentication)

- ru.runa.af.delegate.AuthorizationServiceDelegate. With its help it's possible to request and to set permissions on objects.

On the DAO level (from the same JVM, without authentication)

- ru.runa.af.dao.SecuredObjectDAO. Management of SecuredObject and SecuredObjectType.
- ru.runa.af.dao.PermissionDAO. With its help it's possible to request and to set permissions on objects.

Swimlane initialization and substitutors system

It is possible to redirect task of a user to other users in some special cases (say if the user is sick). It is implemented with the help of substitution rules and user's status.

Swimlanes and their initialization

There is a set of swimlanes (specialized local variables) in business process, each swimlane is linked to initializer (a specialized operator).

Each state in business process corresponds to one of the swimlanes.

Initializer initializes swimlane with a set of users.

Swimlane is narrowed to one user when task is taken for performing by one of users assigned to swimlane.

Initializator algorithm is defined by an initialization on users formula, business process variables and functions over executors.

Initialization formula is an assigning operator with the swimlane name in the left part and with executor, business process variable or function over executors with parameter set in the right part.

Task lists

For every user there is a list of tasks that consists of:

- Tasks assigned to the user or to the groups that this user is a member of
- Tasks assigned to other users but redirected to this user by substitution

List of redirected tasks is defined in Substitution rules subsection.

If a swimlane is initialized with a group of users then any group member may perform the task. After one member of the group performs the task the swimlane is reinitialized with this user and s/he is responsible for performing all the following up tasks for this swimlane. If it is necessary to reinitialize the swimlane each time it enters new task (so that the new task could be performed not only by the user who performed previous task for that swimlane) then an option "reinitialize swimlane" must be checked on the follow up tasks while developing business process in GPD.

User status

User can have one of the following statuses:

- Active (present)
- Inactive (absent)

Substitution rules

In some cases a task can be redirected to another user (a substitutor).

The substitutor is chosen with the help of a set of user substitution rules. This set is an ordered list of rules.

Generally a rule is a orgfunction that returns a substitutor.

Also there is a standard type of rules with the parameters that can be set via the system graphical interface. The list of parameters for this type of rule:

- A substituted user (User)
- A substitutor (User, organization stucture function that returns a user)
- Criteria if the rule can be applied (function that returns boolean)

An example of rule:

- Ivanov
- Petrov
- (swimlane == «HRinspectors») && (businessProcess == «Sicklist»)

An order of substitution rules application

In case if a user is inactive (and the state swimlane is initialized with the user not with the group that this user is a member of), then from a list of all rules the rules linked to this user are selected. Then from that subset the first rule that matches the criteria and with active substitutor is taken. The task will be shown in this user (substitutor) task list.

Note. It is possible that there are no substitutors found for a user.

Note. When user becomes inactive swimlane values are not altered. This user tasks appear in corresponding task lists of the substitutors.

Note. A swimlane that corresponds to a state is shown on the business process diagram (in the upper part of the state rectangular in parenthesis).

Note. A swimlane that corresponds to a user who started the process is indicated on the business process diagram (above the start point in parenthesis).

References

[1] http://www.gnu.org/licenses/lgpl.html
[2] http://wf.runa.ru/About
[3] http://www.jboss.org/
[4] http://www.eclipse.org/
[5] http://xdoclet.sourceforge.net/
[6] http://www.cvshome.org/
[7] http://ant.apache.org/
[8] http://www.microsoft.com/sql/evaluation/default.mspx
[9] http://www.mysql.com/
[10] http://hsqldb.org/
[11] http://www.oracle.com/
[12] http://postgresql.com/
[13] http://www.beanshell.org/

Article Sources and Contributors

RunaWFE. Developer guide *Source*: http://wf.runa.ru/doc/index.php?oldid=819 *Contributors*: Dofs, Natkinnat, WikiSysop

Image Sources, Licenses and Contributors

Image:permissions.png *Source*: http://wf.runa.ru/doc/index.php?title=File:Permissions.png *License*: unknown *Contributors*: Natkinnat