RunaWFE. Graphical business process designer. Developer guide

Version 3.0

INTRODUCTION

RUNA WFE Graphical Process Designer (GPD) is based on the JBoss jBPM Graphical Process Designer, modified according to the requirements of RUNA. Technologically, RUNA WFE is based on the Graphical Editing Framework (GEF) that is part of the Eclipse platform. Eclipse implements the OSGi (OSGi Framework) services model on a Java platform.

OSGi Framework provides a unified environment for applications (“bundles”), connecting:

– a bundle execution environment;

– modules that complement Java class loading policies with private classes for a module and controlled module binding;

– application module lifecycle management, allowing to dynamically install, start, stop, update, and delete modules;

– registration services, allowing for dynamic sharing of objects by applications.

The Eclipse platform is a set of subsystems, implemented with a small run-time kernel and a number of modules (plug-ins), extending the functionality of the platform. For purposes of this document, the terms "module" and "plug-in" are equivalent and interchangeable. The use of these terms is basically determined by style considerations.

The running Eclipse kernel dynamically discovers, configures, and starts the plug-ins of the platform. Eclipse supports dynamic connection of plug-ins, described by plug-in descriptors (in MANIFEST.MF and plugin.xml files). To extend the functionality, the platform plug-ins define extension points in plug-in descriptors. An extension point is an xml description of the interface of the extended plug-in component. Extending plug-ins use extension points to add functionality. The Eclipse platform does not distinguish between user plug-ins and those native to the platform.

The Eclipse platform is implemented in Java, which makes developed applications portable to different platforms with different operating systems.

GEF (Graphical Editing Framework) provides an environment for development of graphical designers. GEF is implemented as a set of plug-ins, extending the plug-ins of the Eclipse platform. GEF connects the elements of the application model with their graphical views that are created, using graphical components from the Draw2d library. GEF controllers support visual representation of model elements in the MVC (Model-View-Controller) architecture. For each element of the view, the controller for this view interprets the events of the user interface and converts them into processing commands for the corresponding element of the model.

A general overview of the GEF architecture is shown in Figure 1. General view of the GEF architecture. A description of GEF components is presented in Table 1. GEF architecture components.

Table 1. GEF architecture components.

Component	Description
Model	Holds data. Must have a mechanism to notify about changes.
View	A visual representation of the model. Consists of figures representing the elements of the model. A model can be represented both graphically and as a hierarchical (tree) structure.
Controller	Controllers connect model elements with corresponding view elements. Controllers can be graphical or hierarchical depending on the type of view they provide. They are responsible for editing model elements through a view and also for displaying changes in model elements in a view. Controllers use editing policies – elements performing the majority of editing tasks.
Action	Elements that process data input. Convert user interface events into requests that use controller APIs.
Request	Requests are elements, encapsulating user interface events. Allow to abstract from the source of the event.
Command	Commands encapsulate data on changes in the model. Returned by controllers in response to requests. Also contain information on possibility of interaction.
Event	Events are changes in the user interface, causing changes in a view or model.

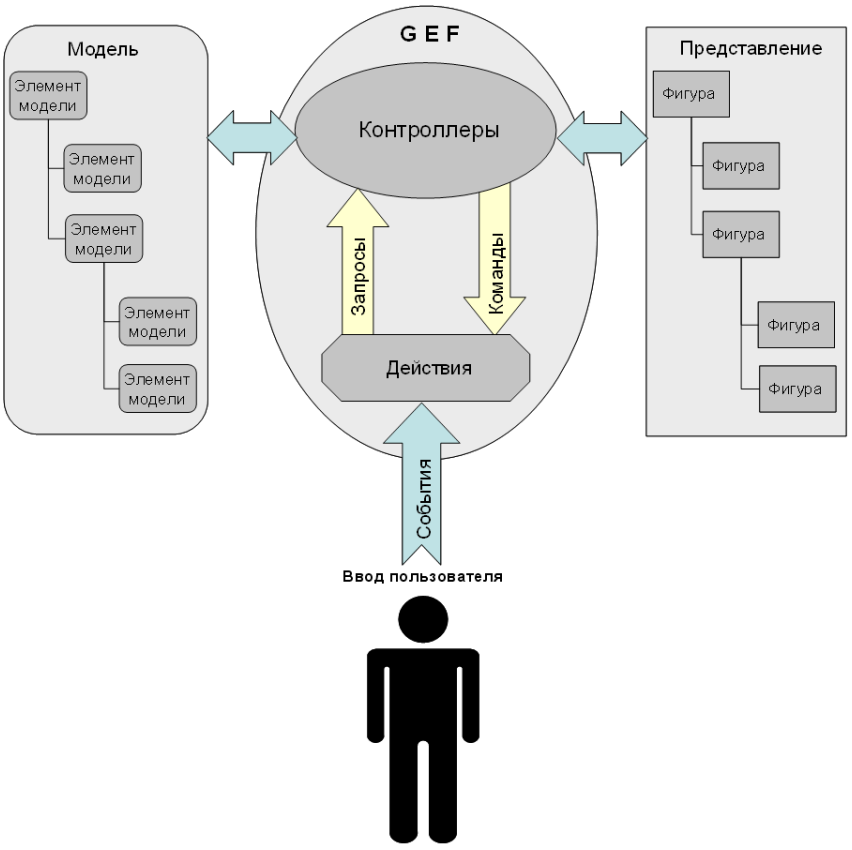


Figure 1. General view of the GEF architecture.

Graphical designer modules

The graphical designer is based on the JBOSS JBPM engine whose main module jbpm.core loads and unloads business process definitions, creates business process instances and execution flows and stops them. Other modules of the graphical designer use the services of the jbpm.core engine for their functions.

The graphical designer modules and their descriptions are shown in Table 2. Graphical designer modules. The relationships of the modules are shown in Figure 2. Relationships between RUNA WFE modules..

Table 2. Graphical designer modules.

Module	Description
org.jbpm.core	Contains JBOSS JBPM engine libraries as well as interfaces to work with the engine.
org.jbpm.db	Not used in the current implementation.
org.jbpm.feature	Arranges designer modules into a group.
org.jbpm.help	GPD help subsystem. Contains no help data in the current implementation.
org.jbpm.ui	Contains JBOSS JBPM graphical designer packages, including GEF, model elements and graphical views. JBOSS JBPM packages are used in RUNA WFE Graphical Process Designer.
ru.runa.jbpm.ui	RUNA WFE Graphical Process Designer module. Based on org.jbpm.ui.
tk.eclipse.plugin.htmleditor	HTML editor module.
tk.eclipse.plugin.wysiwyg	Visual (WYSIWYG) editor module. Extends the functionality of tk.eclipse.plugin.htmleditor.

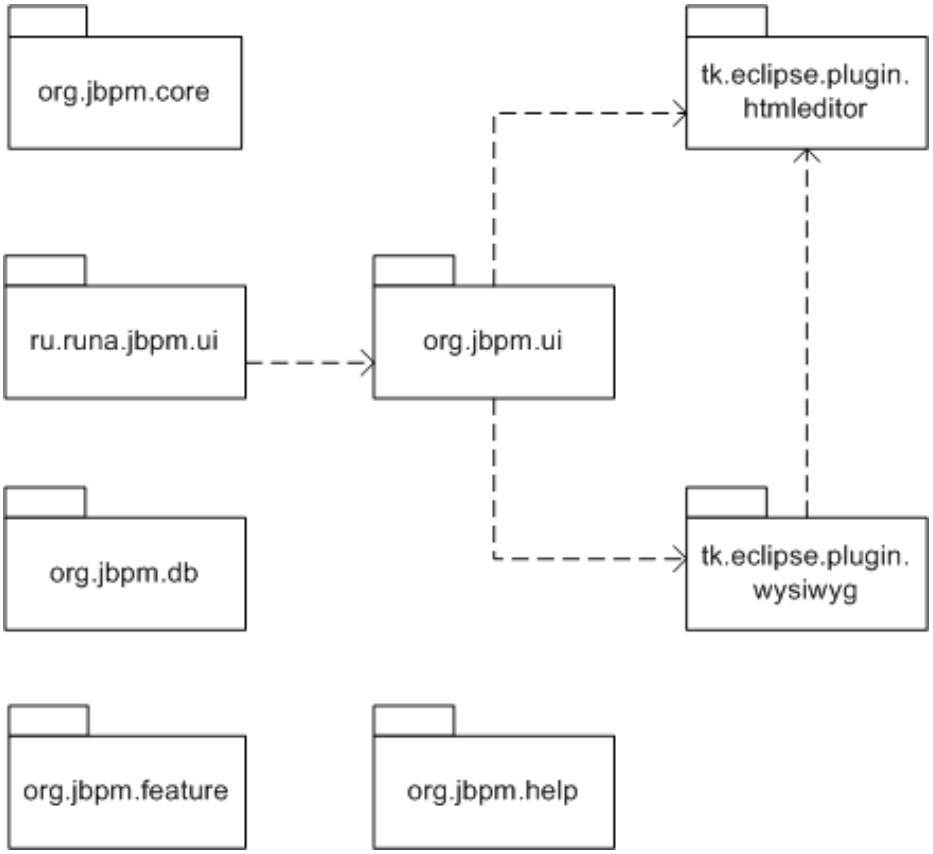


Figure 2. Relationships between RUNA WFE modules.

Currently two jPDL versions are supported 2.0 and 3.2. Saving model to XML and reading model from XML is done by a serializer class (one for each jPDL version). jPDL versions migration for processes might be done iteratively. You can identify the currently used jPDL version by the structure of the processdefinition.xml. If you want to add another jPDL version you should understand code structure. You should use org.jbpm.ui.elements extension point (described in elements.xsd). Each org.jbpm.ui.elements extension (2.0 and 3.2 for now) contains a list of model elements and a content provider for this jPDL version. Content provider consists of a serializer and a converter from a previous jPDL version.

CONFIGURING IDE

Eclipse configuration

Eclipse (Java EE, with GEF) 3.4 is recommended. Or the following plug-in must be installed in Eclipse: GEF (with Draw2d)

Localization

Eclipse localization plug-ins are to be installed (with version matching Eclipse's version) and GEF localization installation is also a must.

For eclipse: http://download.eclipse.org/technology/babel/babel_language_packs/.

For GEF http://www.eclipse.org/gef/translations/translation.html (Group 2)

After plug-ins installation restart eclipse.

Check out gpd project from the sourceforge SVN repository https:/ / runawfe. svn. sourceforge. net/ svnroot/ runawfe/RunaWFE-3.x/trunk/gpd [2]. Use "import project" option in Eclipse to create 6 separate projects, one per every gpd plug-in (except ru.runa.specific plug-in).

In org.jpbm.ui plug-in project open gpd.product > Configuration. Remove all listed plug-ins and then choose to add 6 plug-ins that were imported as projects on previous step. The gpd is ready to be built.

RCP application assembly in GPD

GPD is designed to work as a separate RCP (Rich Client Platform) application/product. RCP application assembly is required after making changes to GPD modules.

RCP application is exported as a file with the .product extension. A product file can be created automatically by Eclipse at the time, when a module project is created, based on an RCP application template. A product file can also be created at the time when the product configuration is set up. A product file can be viewed and edited in the product file editor of the Eclipse development environment. The product file editor contains the tabs “Overview”, “Configuration” and “Brand”.

On the Overview tab of the product file editor the following is specified:

– product ID;

– the application to run when the product is launched;

– the name displayed in the application header.

The “Synchronize” link in the “Testing” section is used to update the plugin.xml descriptor of the main module of the product to reflect the changes made to the product modules in the development environment. The “Launch the Product” and “Launch the product in Debug Mode” options allow to test an RCP application without exporting it.

In the “Exporting” section, the Eclipse Product Export Wizard allows to set up export parameters and export an RCP application, based on the configuration defined on the Configuration tab.

On the Configuration tab in the “Modules and Fragments” section, the modules constituting the RCP application are specified. After the main module is specified, the rest of the required set of modules can be defined automatically. The way in which the RCP application is to be launched is specified in the “Configuration File” and “Launch Arguments” sections.

To assemble an RCP application of the Graphical process Designer, do the following:

1. Open the org.jbpm.ui.gpd.product product file. Set the following parameters on the Overview tab in the “Product Definition” section:

– Product ID: org.jbpm.ui.RUNA;

– Application: ru.runa.jbpm.ui.bp editor;

– Product Name: Runa WFE GPD;

– The product configuration is based on a plug-in.

1. In the “Testing” section click “Synchronize” to synchronize the changes with the main module of the product.
2. To add newly created modules (if any) to the set, click Add on the Configuration tab and select the necessary modules from the list.

Eclipse allows to recreate the list of the required modules for the RCP application. To do this:

– delete all the modules by choosing Delete All;

– add the main module of the RCP application to an empty list;

– select “Add Required Modules”.

1. On the “Overview” tab in the “Exporting” section select Eclipse Product Export Wizard.
2. In the wizard window:

– specify the full path to the target export catalog;

– specify compatibility with the target Java machine in the compiler options;

– press Finish.

The RCP application will be exported using the path specified.

To export RCP applications to Linux, Mac OS X and Solaris platforms, the Eclipse-RCP-delta-pack plug-in must be installed in the Eclipse development environment. The Eclipse-RCP-delta-pack plug-in can be installed by choosing Help > Software Updates > Manage Configuration in the main menu to update the installed Eclipse platform and by choosing the plug-in from a plug-in list.

Once Eclipse-RCP-delta-pack is installed in the product file editor, a tab is added to launch the RCP application on the selected platform.

Extending Designer Features

GPD allows to extend its features with the help of eclipse extension points.

(http://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points [3]

http://www.vogella.de/articles/EclipseExtensionPoint/article.html [4])

At first you should create a new plug-in project and add all necessary plug-ins to its dependencies (at least those, where the extension points defined, those are org.jbpm.ui, tk.eclipse.plugin.wysiwyg).

Also it could be done in an existing project (e.g. in org.jbpm.ui), but it is not recommended. See how to create new project:

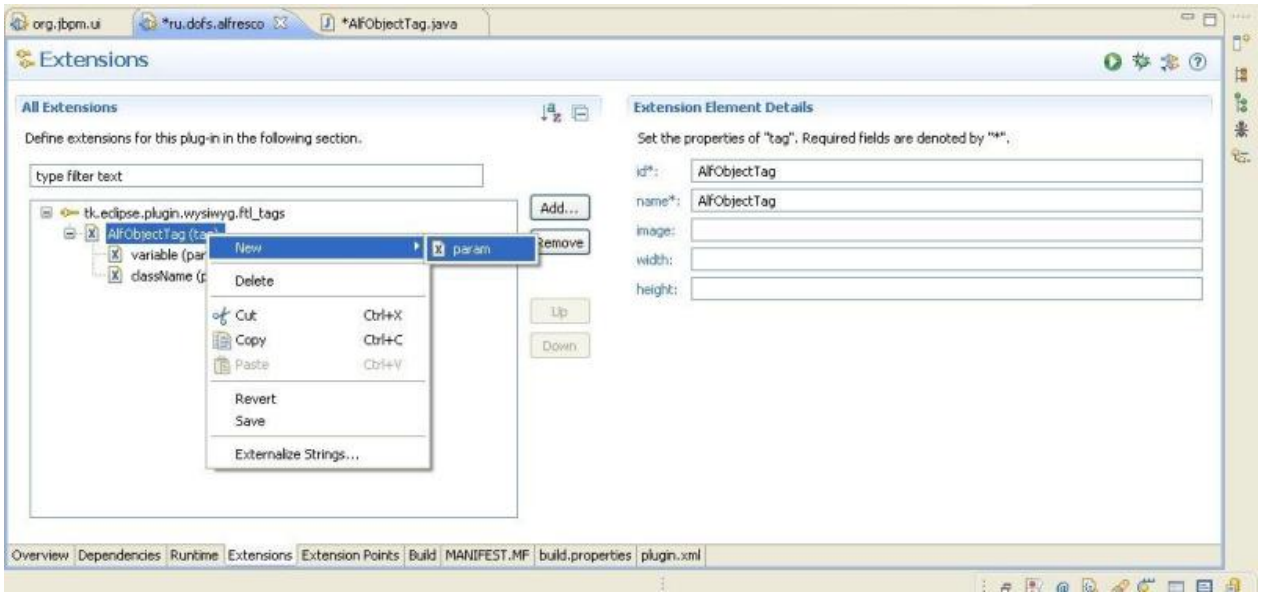
(http://www.ibm.com/developerworks/library/os-ecplug/ [5]

http://www.eclipse.org/articles/Article-Your%20First%20Plug-in/YourFirstPlugin.html [6])

A standard way of creating and editing extensions is used.

Consider an example of adding a new FTL tag.

Open plugin.xml file, choose Extensions tab, choose add and select tk.eclipse.plugin.wysiwyg.ftl_tags (if it is not in the list then check if the plug-in that declares this extension point is present in the dependencies of the plug-in being developed)



Use context menu to add/remove a tree element (except for the root element which represents the link to the extension point). The snapshot above shows how to add a parameter to a tag.

The tree structure is defined by the scheme of the extension point. (See org.jbpm.ui/schema for the .xsd files).

In order to build GPD with a new plug-in included you can either in configuration tab of the org.jbpm.ui/gpd.product file add the plug-in and build a new version of GPD or

use Overview -> Export wizard of plugin.xml file to build the plug-in independently and put the resultant .jar file to the /plugins folder of the existing built version of GPD. Then it's recommended to restart GPD with -clean option.

Forms types

Currently there are 3 supported forms' types:

FTL (HTML + Freemarker (www.freemarker.org))

VarTag (HTML + VarTags (simple template engine))

InfoPath (forms are created with the help of MS Infopath)

This list can be extended by your own type of forms

The scheme of the extension point: org.jbpm.ui.formtype

- element[@name] – form type name

- element[@contributor] – class that extends org.jbpm.ui.forms. FormType

- element[@type] – file name extention for the files with forms (e.g. .ftl)

org.jbpm.ui.forms. FormType methods:

- IEditorPart openForm(IFile formFile, FormNode formNode)

Opens a form for edit

- String getFormFileName(IFile definitionFile, FormNode formNode)

Returns the initial file name

- Map<String, Integer> getFormVariableNames(IFile formFile, FormNode formNode)

Returns the list of the form variables as a “name – usage type” list. The type of usage value can be read, write, not defined

- void validate(IFile formFile, FormNode formNode)

Validates the form

jPDL version

The editor can support more than one jPDL versions simultaneously (now it's versions 2.x and 3.x). It's possible to add another jPDL version, but it requires a significant amount of efforts.

There's no extension point scheme. A slightly different approach is used. In order to add a new version it is necessary to add an extension org.jbpm.ui.elements, and indicate in its properties

- @ID – a unique name

- @Name – jPDL version

It's possible to extend (or modify) the model in different jPDL versions:

1) to create context menus – filter by EditParts

2) to create new properties (in properties view)

jPDL elements

These are the elements that can be accessed from the editor palette.

The extension point scheme is org.jbpm.ui.elements

- element[@name] – a unique (in this jPDL version namespace) element name

- element[@model] — a model class (must extend org.jbpm.ui.common.model.GraphElement)

- element[@graphicalEditPart] – responsible for graph representation

- element[@treeEditPart] – responsible for representation in the scheme tree

- element[@figure] – responsible for graph rendering

- element/entry – an optional element that can be displayed in the palette (if absent the element is not accessible from the palette)

- element/entry[@id] – a unique element name in the palette. This field is used to sort within the group when palette is rendered.

- element/entry[@category] – group name

- element/entry[@label] – text in the palette

- element/entry[@type] – element type (entity or transition)

- element/entry[@icon] – a picture in the palette

- contentProvider[@serializerClass] – provides storing and loading in given jPDL version

- contentProvider[@converterClass] – provides process update from the previous jPDL verion

ActionHandlers

Action handlers are used for code execution during the process instance run. They can be added to many places of the process definition.

The extension point scheme is: org.jbpm.ui.delegablePropertyDescriprors

- delegable[@type] – the type {actionHandler|decisionHandler|assignmentHandler}

- delegable[@name] – a name that is used in the editor only to represent the action handler

- delegable[@className] – a full class name

- delegable[@cellEditorProvider] – a class that extends org.jbpm.ui.custom.DelegableProvider. It allows to create UI to edit action handler configuration.

Also see appendix A.

DecisionHandlers

Decision handlers are used for Decision elements of jPDL. The interface defines a method that uses the process variables values to take a decision which transition to take next.

The extension point scheme is org.jbpm.ui.delegablePropertyDescriprors

The attributes meaning is the same as for ActionHandler.

Note: delegable[@cellEditorProvider] can implement org.jbpm.ui.custom.IDecisionProvider, bringing in new features.

Also see appendix A

Organization functions

All organization functions implement ru.runa.af.organizationfunction.OrganizationFunction. There is a single method in interface: long[] getExecutorIds(Object[] parameters), that returns a list of the WF executors codes, the parameters are defined by the implementation.

They are used for role initialization.

The extension point scheme is org.jbpm.ui.orgfunctions

- orgfunction[@name] – a name that is used to represent orgfunction in the editor

- orgfunction[@className] – a full name of the orgfunction class

- orgfunction/parameter – orgfunction parameters definition

- orgfunction/parameter [@name] – a parameter name that is used to represent orgfunction in the editor

- orgfunction/parameter [@type] – a type of the variables that can be used as a dynamic value of the parameter.

- orgfunction/parameter [@value] – an initial value of the parameter

- orgfunction/parameter [@multiple] – if set to true then there might be several such parameters in a row.

Note: orgfunction/parameter [@multiple] can be set to true only for the last parameter.

Roles constructors

Open GPD, click on roles tab and push “change” button. You will see a dialog where you can modify role initialization. It has a tree structure with two depth levels (the root level is the tabs selection, and the second level is the blocks with the contents of the tabs). GUI elements that open in blocks can be added in this dialog.

See the extension point scheme org.jbpm.ui.swimlaneelements.

Validators

Validators are used for process variables input validation. There are two kinds of validators: global and variable specific.

Note: currently it's impossible to add a new global validator (now there is only one global validator based on the BeanShell and it interpreters most of the java code)

The extention point scheme is: org.jbpm.ui.validators

- validator[@name] – a validator name, same as listed in WF in validators.xml

- validator[@displayName] – a name that is displayed

- validator[@description] — a displayed description

- validator[@applicable] – variable types that this validator is applicable to. The types must be comma separated e.g. (long,double). An empty value means all types.

- validator/param – validator parameters

- validator/param[@name] – a parameter name. In a validator class a value of the parameter is accessible by this name.

- validator/param[@displayName] – a displayed name

- validator/param[@type] – a type of parameter value

Variable formats

Variable formats are used for 2 purposes:

- to define a variable type in WFE

- to define the procedure (with java code) of data parsing for the web forms in WFE

Note: WFE allows to assign a variable of type X a value of type Y in the editor (with the help of action handler for example). Currently there's no extension point in the editor that would allow to define a new format.

See appendix A.

FTL: tags

Ftl tags in freemarker context can be understood as methods that are used in WFE and provide HTML code as return values.

The extension point scheme is tk.eclipse.plugin.wysiwyg.ftl_tags

- tag[@id] – a tag name, it must be declared in WF in freemarker-tags.xml file

- tag[@name] – a displayed tag name

- tag[@image] – path to the tag picture that will be displayed in GPD

- tag[@width], tag[@height] – picture size

- tag/param[@name] — a tag displayed name

- tag/param[@type] – a type of parameter display (e.g. text – a variable value input field , combo – a choice from a list)

- tag/param[@variableAccess] – is variable connected to the parameter and in which mode (write, read). It used for form variables search.

- tag/param[@values] – a list of variables of a given type in combo, “any” stands for all variables.

- tag/param/paramValue – a static value of a combo parameter. It is used if the tag/param[@values] value is empty and tag/param[@type] set to combo.

- tag/param/paramValue[@name] – a displayed name.

- tag/param/paramValue[@value] – a value of chosen option

FTL: variables display

It is $test?format in freemarker context.

The extension point scheme is tk.eclipse.plugin.wysiwyg.ftl_formats

- type[@name] – a variable type name. The available types are: string|double|long|date|time|boolean

- type/format – freemarker format for this type

- type/format[@name] – displayed name

- type/format[@value] – a name that follows a question mark (?) during FTL code generation

Var: tags (deprecated)

See scheme tk.eclipse.plugin.wysiwyg.var_tags

Adding a new menu item

The RunaWFE GPD menu is contained in plugin.xml descriptor of the ru.runa.jbpm.ui plug-in. The ru.runa.jbpm.ui plug-in defines the GPD menu items via adding the features to the extension point org.eclipse.ui.actionSets of org.eclipse.ui plug-in.

It is convenient to add and edit menu items on the Extensions tab of the manifest editor in Eclipse IDE module. The menu elements serve for a sole purpose to structure functional elements (actions) that are leaf-elements on the menu hierarchy.

In order to add a menu element:

1. Open plugin.xml file of ru.runa.jbpm.ui plug-in in manifest module editor.

2. To create a menu element from a context menu of the main menu element of the org.eclipse.ui.actionSets extension point choose “New” then “menu”. Manifest editor will create a new extension element “menu” and the new element properties will be displayed at the right part of the window:

- id – a unique element identificator;

- label – a label displayed on the element;

- path – a path to localization for the menu item in the menu structure.

3. To add a separator element to the created menu element. In order to make the new menu element available for the extensions from other plug-ins the separator name must be «additions».

To add actions:

1. From a context menu of the main menu element of the org.eclipse.ui.actionSets extension point choose “New” then “action”. Manifest editor will create a new extension element “action” and the new element properties will be displayed at the right part of the window:

- id – a unique element identificator;

- label – a label displayed on the element;

- accelerator – Deprecated;

- definitionId – an identificator of a command linked to this action;

- menubarPath – a path to localization for the action item in the menu structure.

- toolbarPath – a path to localization for the action item in the tools panel;

- icon – a relative path to the file with an icon for the action element;

- disabledIcon – a relative path to the file with an icon for the disabled action element;

- hoverIcon - a relative path to the file with an icon for the mouse hovered over action element;

- tooltip – tool tip text;

- helpContextId – an identifier of the context help;

- style – an attribute of action representation (push, radio, toggle, pulldown);

- state – optional attribute of the initial state;

- pulldown – Deprecated;

- class – an absolute path to the class of task handler. The class must implement org.eclipse.ui.IWorkbenchWindowActionDelegate or org.eclipse.ui.IWorkbenchWindowPulldownDelegate. If retarget attribute set to true, then this attribute is ignored;

- retarget – if this attribute set to true, a global action handler is used;

- allowLabelUpdate – is used if retarget is set to true. If this attribute set to true, then label and tooltip of this action will be substituted by global action handler attributes;

- enablesFor – if this attribute is not set it is ignored. It defines the number of elements that has to be selected in order to executed the given action.

2. After the absolute path to the action handler class is set (in class attribute), choose the link to the class. Eclipse IDE will generate a stub for an action handler class in the package matching the specified path. The stub class contains stub methods that should be implemented in order to define the action handler behaviour.

Appendix A

Locating extensions by loading the classes that implement interfaces:

```
ActionHandler, DecisionHandler, AssignmentHandler, WebFormat
```

You should put a .jar file, that contains class(es) implementing one of the enumerated above interfaces to a folder{GPD}/plugin/org.jbpm.core/lib and restart GPD.

Note 1: if a class has dependencies to other libraries, you should put all the required libraries to the same folder with class' .jar.

Note 2: Sometimes JDT cache prevents new elements to appear. In order to cope with it, close GPD and delete the cache folder {GPD}/workspace/.metadata/.plugins/org.eclipse.jdt.core. Then start GPD again.

INTERACTION WITH INFOPATH (windows ONLY)

Architecture

Microsoft InfoPath 2007, included in the Microsoft Office 2007 distribution disk, has been selected as a graphical designer. It allows to create form templates, using a standard and a customizable form element palette. To use the form elements, specific to RUNAWFE, we will use elements that will be added to the InfoPath palette.

Attention! For the system to work correctly, the RunaGPDInfoPathSupport.dll is required, stored in C:\WINDOWS\SYSTEM32. This library allows to work with archive file forms (**XSN**), stored in the **CAB format**.

Standard InfoPath Elements Supported by RUNAWFE

InfoPath element	Description
Text Box	Single-line text input
Drop-Down List Box	A drop-down list
Check Box	A checkbox
Rich Text Box	A field for multi-line text input
Date Picker	Selecting a date with the help of a calendar
File Attachment	Uploading and downloading a file
Image	An image on a form
Hyperlink	Opens in a new window

In addition to these standard elements, InfoPath allows to add other elements based on TemplatePart (templates) or ActiveX (COM components).

Creation of InfoPath ActiveX Elements

Microsoft Visual Studio 2007 development environment is used.

The added InfoPath elements are ActiveX objects, implementing UserControl, IObjectSafety and ICOMControl interfaces. They must all gave a unique GUID in the COM model. The utility to create a GUID is called GUIDGEN.EXE and delivered with MS Visual Studio. Net Framework will be used for simplicity, though it is not necessary.

A new project of type **Windows Class Library** is created in MS Visual Studio (Visual C# projects).

In the properties of the created object on the tab Configuration Properties > Build select **Register for COM Interop = true**.

Create a new **GUID**.

Make a copy of class “template”, implementing the InfoPath element, and change the GUID

Change the **OnPaint** method (optional). This method is called each time an element is displayed on an InfoPath form. To give the element a beautiful appearance, draw it here.

Note the **OnSizeChanged** method of the class. It is called each time an InfoPath user tries to change the size of an element on a form. Thus, you can explicitly control the size of an element on a form.

Having done all this, press Project Build and wait for compilation and creation of a new DLL, ready to be installed on a local computer.

For installation on other computers, additional steps are required. Sign DLL (**Strong Name**) – generate a key file (sn.exe –k keyfile.snk) with **sn.exe** (a VS utility), add it into the project and write the values into the AssemblyInfo.cs file [assembly: AssemblyKeyFile("..\\..\\keyfile.snk")], [assembly:

AssemblyKeyName("ActorFullNameDisplay")] instead of default values.

The element should then be added into GAC by **gacutil.exe** (a VS utility) and registered as COM by **regasm.exe (**a .Net utility**)**.

The component business logic can access RUNAWFE and is not limited in functionality.

From the MS InfoPath 2007 interface on the Controls tab, we can register ActiveX components as elements by using Add or Remove Custom Controls, but not ActiveX components in the .Net category. For this category, the user has to write component descriptions manually and place them in a certain location.

The path to the description files of customizable InfoPath elements:

${USER_HOME}\Local Settings\Application Data\Microsoft\InfoPath\Controls

Example:

C:\Documents and Settings\dofs\Local Settings\Application Data\Microsoft\InfoPath\Controls

Element descriptors must have an **.ict** extension for InfoPath to read them correctly while loading.

A convenient name for a file in this folder is

{GUID}.ict (for example, {C4134657-1B43-4968-9913-FAE952F685A0}.ict), where GUID is the GUID of the corresponding InfoPath element.

The file must be in UTF-8 coding.

The file format is as follows:

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<ict:control name="CONTROL_NAME">

<ict:designTimeProperties>

<ict:iconBitmap>CONTROL_ICON</ict:iconBitmap>

</ict:designTimeProperties>

<ict:deployment>

<ict:controlPackage classid="{ACTIVEX_GUID}"/>

</ict:deployment>

</ict:control>

where

CONTROL_NAME = element name in the palette

CONTROL_ICON = an icon in the palette; represented in the Base64 format in the file

ACTIVEX_GUID = GUID ActiveX.

If the file has been created incorrectly, InfoPath will output an error message with details of the error at the next run. If there is no message and the element is not added into the palette, check that the .ict file is in the right folder.

Use existing components as examples.

Creation of InfoPath Template Parts Elements

These components are limited by the functionality of the DataSource, that will be processed at form initialization in the run mode. This element type is useful to output different lists (used to output actors).

Publication consists of the registration of **XTP** template files in InfoPath from the “update/delete elements” menu.

Use existing templates as an example.

REFERENCES

1. Almost exhaustive information on the technologies mentioned in this document is available at http://www.eclipse.org/ [7]. Besides, the Eclipse development environment has a well-developed help system that includes reference information on the technologies used, links to information resources and examples. Update plug-ins, that can be loaded from http://www.eclipse.org/ [7], usually include help data that is added into the Eclipse help system automatically.

3. The book “Building Commercial Quality Eclipse Plug-ins” by Eric Clayberg and Dan Rubel is recommended to plug-in developers. Publisher: Addison WesleyProfessional.ISBN: 032142672X; Published: Mar 22, 2006; Copyright 2006; Dimensions 7x9-1/4; Pages: 864; Edition: 2nd..

2. Information on OSGi Framework can be obtained from the OSGi alliance site at:

http://www.osgi.org/osgi_technology/index.asp?section=2 [8] .

3. GEF documentation is available at: http://www.eclipse.org/gef/reference/articles.html [9]. Useful information for developers is contained in http://wiki.eclipse.org/index.php/GEF_Developer_FAQ [10] and http://wiki.eclipse.org/index.php/GEF_Troubleshooting_Guide#Draw2D_common_mistakes [11]. An example of using GEF components to create a database schema editor can be found in at http://www.eclipse.org/articles/Article-GEF-editor/gef-schema-editor.html [12].

1. GEF tutorials:

http://www-128.ibm.com/developerworks/opensource/library/os-gef/ [13]

http://eclipsewiki.editme.com/GefDescription [14]

References

[1] http://www.gnu.org/licenses/lgpl.html
[2] https://runawfe.svn.sourceforge.net/svnroot/runawfe/RunaWFE-3.x/trunk/gpd
[3] http://wiki.eclipse.org/FAQ_What_are_extensions_and_extension_points
[4] http://www.vogella.de/articles/EclipseExtensionPoint/article.html
[5] http://www.ibm.com/developerworks/library/os-ecplug/
[6] http://www.eclipse.org/articles/Article-Your%20First%20Plug-in/YourFirstPlugin.html
[7] http://www.eclipse.org/
[8] http://www.osgi.org/osgi_technology/index.asp?section=2
[9] http://www.eclipse.org/gef/reference/articles.html
[10] http://wiki.eclipse.org/index.php/GEF_Developer_FAQ
[11] http://wiki.eclipse.org/index.php/GEF_Troubleshooting_Guide#Draw2D_common_mistakes
[12] http://www.eclipse.org/articles/Article-GEF-editor/gef-schema-editor.html
[13] http://www-128.ibm.com/developerworks/opensource/library/os-gef/
[14] http://eclipsewiki.editme.com/GefDescription

Article Sources and Contributors

RunaWFE. Graphical business process designer. Developer guide *Source*: http://wf.runa.ru/doc/index.php?oldid=671 *Contributors*: Natkinnat, WikiSysop

Image Sources, Licenses and Contributors

Image:Process-editor_Developer_guide_ris1.png *Source*: http://wf.runa.ru/doc/index.php?title=File:Process-editor_Developer_guide_ris1.png *License*: unknown *Contributors*: WikiSysop
Image:Process-editor_Developer_guide_ris1a.png *Source*: http://wf.runa.ru/doc/index.php?title=File:Process-editor_Developer_guide_ris1a.png *License*: unknown *Contributors*: WikiSysop
Image:Process-editor_Developer_guide_ris3.jpg *Source*: http://wf.runa.ru/doc/index.php?title=File:Process-editor_Developer_guide_ris3.jpg *License*: unknown *Contributors*: Natkinnat
Image:Process-editor_Developer_guide_ris4.jpg *Source*: http://wf.runa.ru/doc/index.php?title=File:Process-editor_Developer_guide_ris4.jpg *License*: unknown *Contributors*: Natkinnat