|  |  |
| --- | --- |
|  | **2012** |
|  | Zhejiang University  State Key Lab. of CAD&Graphics  Jing Xu  Email: victorxujing@gmail.com |

|  |
| --- |
| **[Function Block开发文档以及Magicdraw插件开发指南]** |
|  |

Contents

[MagicDraw OpenAPI UserGuide解读 1](#_Toc336681482)

[1 PLUG-INS 1](#_Toc336681483)

[1.1 How plug-ins work 1](#_Toc336681484)

[1.2 Writing plug-in 2](#_Toc336681485)

[1.3 本节小结及补充 3](#_Toc336681486)

[2 ADDING NEW FUNCTIONALITY 4](#_Toc336681487)

[2.1 Invoking Actions 4](#_Toc336681488)

[2.2 Creating a new action for MagicDraw 5](#_Toc336681489)

[2.2.1 Create new action class 5](#_Toc336681490)

[2.2.2 Specify action properties 6](#_Toc336681491)

[2.2.3 Describe enabling/disabling logic 7](#_Toc336681492)

[2.2.4 Configure Actions 8](#_Toc336681493)

[2.2.5 Register configurator 11](#_Toc336681494)

[2.3 Actions hierarchy 13](#_Toc336681495)

[2.4 本节小结及补充 13](#_Toc336681496)

[3 UML MODEL 15](#_Toc336681497)

[3.1 Project 15](#_Toc336681498)

[3.2 Changing UML Model 16](#_Toc336681499)

[3.2.1 SessionManager 16](#_Toc336681500)

[3.2.2 ModelElementsManager 16](#_Toc336681501)

[3.2.3 Create new model element 17](#_Toc336681502)

[3.2.4 Editing model element 17](#_Toc336681503)

[3.2.5 Adding new model element or moving it to another parent 18](#_Toc336681504)

[3.2.6 Removing model element 18](#_Toc336681505)

[3.2.7 Creating Diagram 19](#_Toc336681506)

[3.2.8 Creating new Relationship object 20](#_Toc336681507)

[3.3 Working with Stereotypes and Tagged Values 20](#_Toc336681508)

[3.4 本节小结及补充 22](#_Toc336681509)

[4 PRESENTATION ELEMENTS 23](#_Toc336681510)

[4.1 Presentation Element 23](#_Toc336681511)

[4.2 Presentations Elements Manager 25](#_Toc336681512)

[4.2.1 Creating shape element 25](#_Toc336681513)

[4.2.2 Creating path element 25](#_Toc336681514)

[4.2.3 Reshaping shape element 25](#_Toc336681515)

[4.2.4 Changing path break points 26](#_Toc336681516)

[4.2.5 Deleting presentation element 26](#_Toc336681517)

[4.2.6 Changing properties of presentation element 26](#_Toc336681518)

[4.3 本节小结及补充 27](#_Toc336681519)

[5 NEW DIAGRAM TYPES 28](#_Toc336681520)

[5.1 Diagram Types Hierarchy 28](#_Toc336681521)

[5.2 Adding a new diagram type for MagicDraw 28](#_Toc336681522)

[5.2.1 Override abstract DiagramDescriptor class 28](#_Toc336681523)

[5.2.2 Register new diagram type in the application 31](#_Toc336681524)

[5.3 本节小结及补充 32](#_Toc336681525)

[6 MagicDraw插件开发总结 33](#_Toc336681526)

[Function Block开发文档 34](#_Toc336681527)

[1 Function Block开发目的 34](#_Toc336681528)

[2 Function Block开发内容 35](#_Toc336681529)

[3 Function Block文件结构 36](#_Toc336681530)

[4 Function Block的设计 37](#_Toc336681531)

[4.1 插件主类与辅助类设计 37](#_Toc336681532)

[4.1.1 插件主类FunctionBlockPlugin 37](#_Toc336681533)

[4.1.2 辅助类FunctionBlockHelper 37](#_Toc336681534)

[4.1.3 常数类FunctionBlockConstants 37](#_Toc336681535)

[4.2 自定义Diagram设计 38](#_Toc336681536)

[4.2.1 Function Block Diagram设计 38](#_Toc336681537)

[4.2.2 ECC Diagram设计 39](#_Toc336681538)

[4.3 Actions设计 40](#_Toc336681539)

[4.3.1 DrawShapeDiagramAction 40](#_Toc336681540)

[4.3.1.1 DataInPortAction, DataOutPortAction, EventInPortAction, EventOutPortAction 40](#_Toc336681541)

[4.3.1.2 ECCInitialStateAction， ECCStateAction 41](#_Toc336681542)

[4.3.1.3 BasicFunctionBlockAction， CompositeFunctionBlockAction 41](#_Toc336681543)

[4.3.1.4 AlgorithmAction 43](#_Toc336681544)

[4.3.2 DrawPathDiagramAction 44](#_Toc336681545)

[4.3.2.1 ECCTransitionAction 44](#_Toc336681546)

[4.4 Renders设计 44](#_Toc336681547)

[4.4.1 ShapeDecorator 45](#_Toc336681548)

[4.4.1.1 FunctionBlockRender 46](#_Toc336681549)

[4.4.1.2 PortRender 47](#_Toc336681550)

[4.4.1.3 InitialStateRender 48](#_Toc336681551)

[4.4.1.4 AlgorithmRender 49](#_Toc336681552)

[4.4.1.5 EventOutputRender 49](#_Toc336681553)

[4.4.2 PathDecorator 52](#_Toc336681554)

[4.4.2.1 DependencyRender 52](#_Toc336681555)

[5 Function Block范例 52](#_Toc336681556)

# MagicDraw OpenAPI UserGuide解读

《MagicDrwa OpenAPI UserGuide》是MagicDraw官方提供的开发手册，内容有22章。

## 1 PLUG-INS

### 1.1 How plug-ins work

这一节简单介绍了插件（Plug-in）的工作原理，即MagicDraw是如何载入Plug-in的。

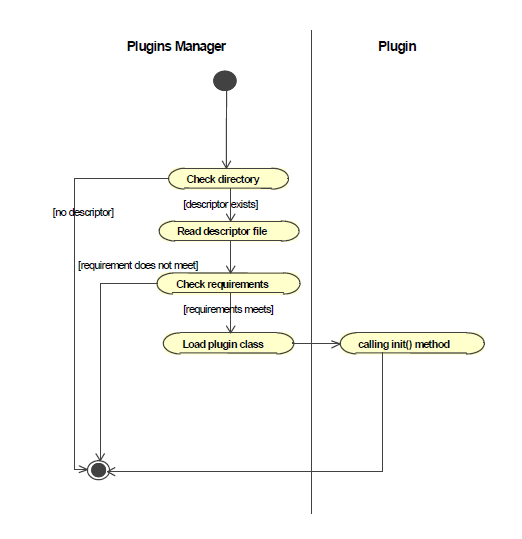


图1-1 Plug-in的载入原理图

MagicDraw启动时，调用Plugins Manager检查/plugins目录下的子目录。进入每一个子目录，读取plugin.xml。

plugin.xml描述了插件的详细信息。

### 1.2 Writing plug-in

建立好工程之后，需要将/lib目录下除了md\_commontw.jar和md\_commontw\_api.jar之外的所有jar文件导入你的工程中。然后才能着手插件编写工作。

编写插件大致分为四步：

* 第一步：创建插件的目录

由上一节介绍可知，你需要在/plugins目录下创建子目录用来存放你的插件；

* 第二步：编写插件代码

你的插件中至少有一个类是继承自com.nomagic.magicdraw.plugins.Plugin，这个类必须实现三个方法init()、close()、isSupported()。

当MagicDraw启动时Plugins Manager调用init()，MagicDraw关闭时调用close()。

* 第三步：编译代码并将其打包成jar格式

将所有代码编译并打包成jar格式置入你创建的子目录中。

* 第四步：编写plugin.xml

在子目录中编写plugin.xml。

plugin.xml的格式如图1-2所示：

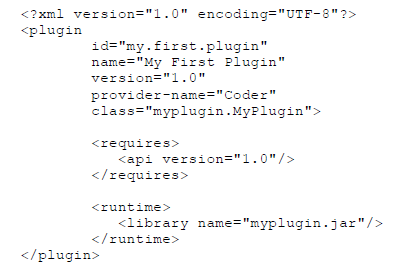


图1-2 Plugin.xml的格式说明

其中最重要的是class信息，class信息填写的是你的插件中继承自com.nomagic.magicdraw.plugins.Plugin的那个类的详细名称（包括包路径）。

### 1.3 本节小结及补充

启动MagicDraw之后，点选Options->Environment->Plugins，查看插件的详细信息，注意当Loaded和Enabled都为true时，才表示你的插件成功载入并成功启用。

Loaded表示插件的载入情况，false表示你的程序有问题，插件载入失败。

Enabled表示插件是否启用，你可以人为控制关闭或启用某些插件。

在/openapi/examples目录下有许多样例，可供参考。

## 2 ADDING NEW FUNCTIONALITY

### 2.1 Invoking Actions

《MagicDraw OpenAPI UserGuide》指出有六种方式可以调用Action。它们分别是：

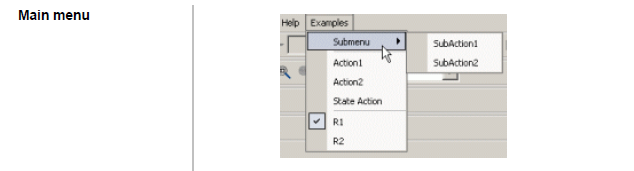


图1-3 Main menu

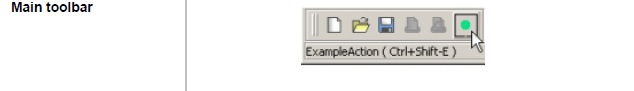


图1-4 Main toolbar

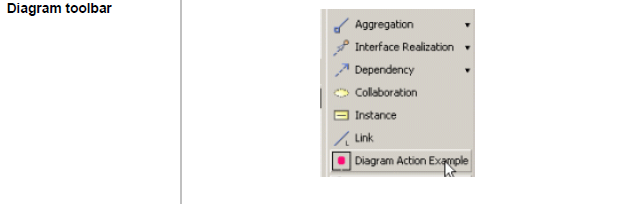


图1-5 Diagram toolbar

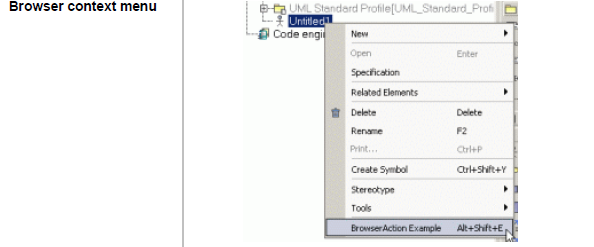


图1-6 Browser context menu

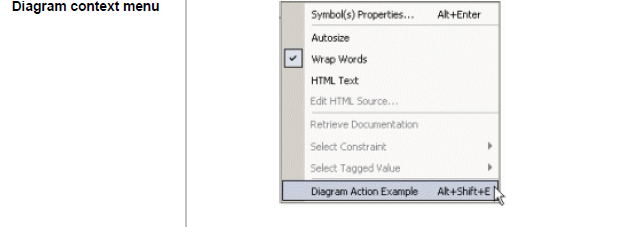


图1-7 Diagram context menu



图1-8 Keyboard shortcuts

### 2.2 Creating a new action for MagicDraw

在官方文档中，Action的创建分为五步。

#### 2.2.1 Create new action class

在MagicDraw中所有的Action都必须是MDAction的子类。

MagicDraw以MDAction为基类创建了三个类，你可以继承它们来自定义新Action。它们分别是：

* DefaultBrowserAction——如果你要实现的Action与点选Browser node有关，那么你可以继承这个类；
* DefaultDiagramAction——如果你要实现的Action与点选Diagram Elements有关，那么你可以继承这个类；
* PropertyAction——如果你要实现的Action是改变元素或者应用的属性，那么你可以继承这个类。

继承了这三个类中的任意一个之后，你需要重载actionPerformed()方法，这个方法描述了自定义Action的行为。

#### 2.2.2 Specify action properties

在《MagicDraw OpenAPI UserGuide》中，有两个很简单的例子：

class SimpleAction extends MDAction

{

public SimpleAction(String id, String name)

{

super(id, name, null, null);

}

/\*\*

\* Shows message.

\*/

public void actionPerformed(ActionEvent e)

{

JOptionPane.showMessageDialog(Application.getInstance().getMainFrame().getDialogParent(), "This is:" + getName());

}

}

public class BrowserAction extends DefaultBrowserAction

{

/\*\*

\* Creates action with name "ExampleAction"

\*/

public BrowserAction()

{

super("", "ExampleAction", null, null);

}

public void actionPerformed(ActionEvent e)

{

//…

}

}

值得关注的是它们基类的构造函数（super()方法），基类的构造函数包含四个参数，依次是：ID、Name、Shortcut and mnemonics、Icon。

这四个参数加上Description构成了Action的五项属性。

* ID：

ID是Action的唯一编号，像我们的身份证一样，不过MagicDraw里面的ID不是数字串，而是字符串。

MagicDraw默认的所有Action的ID定义在了ActionsID类中。你可以利用这些ID在自己的插件中调用默认的功能。

* Name：

显示在GUI上的名字。

* Shortcut and mnemonics：

键盘的快捷键设置，具体使用参考/openapi/examples目录中的实例。该值可以为空（null）。

* Icon

功能的图标，图标分为两类，一类是Small icon，另外一类是Large icon。在菜单中显示的是Small icon，在工具栏中显示的是Large icon。当参数为空（null）时，会显示该功能的Name。

* Description

以tool tip text的形式显示。

#### 2.2.3 Describe enabling/disabling logic

进入MagicDraw之后，Action会根据外部条件自动关闭（Disable）或者启用（Enable）。这一节介绍的是如何实现启用和关闭Action。

有两种机制实现Action状态（Enabled/Disabled）的控制：

* 第一种是将你自定义的Action加入到MagicDraw预先定义好的Action组（Action Group）里面，这个组里的所有Action将会处于同种状态（Enabled or Disabled）。
* 第二种是自己重载updateState()方法，在这个方法中描述功能开启的条件，以及关闭的条件。

当然，你也可以两者都不选，那么你自定义的功能将在MagicDraw启动之后默认开启（Enable）。

#### 2.2.4 Configure Actions

MagicDraw里面用ActionsManager表示目录容器，ActionsCategory表示目录。

因此你需要将自己定义的Action先加入ActionsCategory中，然后再将ActionsCategory加入到ActionsManager中。另外，目录可以嵌套目录。

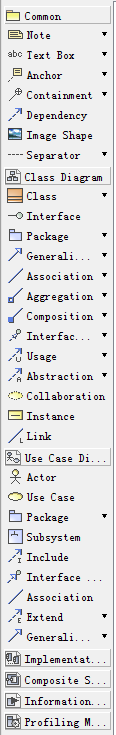


图2-1 All Toolbars

图2-1是一个目录容器，即一个ActionsManager。在它里面包含了许多Toolbar，例如：Common、Class Diagram等。除了All Toolbars这种形式以外，ActionsMananger还可以是menu bar或者context menu。

ActionsCategory是目录，它由一组功能组成。如图2-1，Common是一个ActionsCategory，它由功能Note、Text Box、Anchor、 Containment、 Dependency、 Image Shape、 ----Separator等组成。

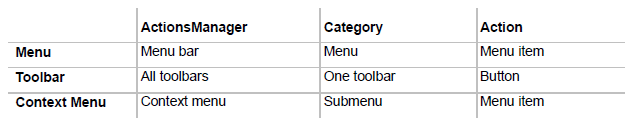


图2-2 ActionsManager、ActionsCategory（Category）、Action与GUI的映射关系

你需要用配置器（Configurator）将Action配置（configure）到ActionsManager中。

MagicDraw提供三种类型的Configurator，它们分别是：

* AMConfigurator

通用的Configurator，用于Menus、Toolbars、Browsers以及Diagram shortcuts。

* BrowserContextAMConfigurator

主要用于处理Browser context（popup） menu。通过这个配置器，你可以读取Browser tree以及node。

* DiagramContextAMConfigurator

主要用于处理Context menus in a diagram。通俗地说就是你右键点选到图中的元素之后出现的菜单。

在2.2.2中你定义好Action之后，需要使用自己定义的Configurator将Action配置到ActionsManager中，这样才能让Action在GUI中显示。

《MagicDraw OpenAPI UserGuide》在这一小节有两个简单的例子，可以很好地帮助你了解如何编写Configurator。

//browserAction是你自己编写好的Action

final DefaultBrowserAction browserAction = …

//创建自定义的BrowserContextAMConfigurator

BrowserContextAMConfigurator brCfg = new BrowserContextAMConfigurator()

{

// implement configuration.

// Add or remove some actions in ActionsManager.

// tree is passed as argument, provides ability to access nodes.

//必须重载的方法configure()

public void configure(ActionsManager mngr, Tree browser)

{

// actions must be added into some category.

// so create the new one, or add action into existing category.

//将自定义的Action加入到ActionsCategory中

MDActionsCategory category = new MDActionsCategory("", "");

category.addAction(browserAction);

// add category into manager.

// Category isn’t displayed in context menu.

//将ActionsCategory加入到ActionsManager中

mngr.addCategory(category);

}

/\*\*

\* Returns priority of this configurator.

\* All configurators are sorted by priority before configuration.

\* This is very important if one configurator expects actions from

\* other configurators.

\* In such case configurator must have lower priority than others.

\* @return priority of this configurator.

\* @see AMConfigurator.HIGH\_PRIORITY

\* @see AMConfigurator.MEDIUM\_PRIORITY

\* @see AMConfigurator.LOW\_PRIORITY

\*/

public int getPriority()

{

return AMConfigurator.MEDIUM\_PRIORITY;

}

};

// create some action.

//someAction是你自己编写好的Action

final MDAction someAction = …

//创建自定义的AMConfigurator

AMConfigurator conf = new AMConfigurator()

{

//重载方法configure()

public void configure(ActionsManager mngr)

{

// searching for action after which insert should be done.

//利用ActionsID.NEW\_PROJECT找到MagicDraw默认的打开新工程的Action

NMAction found= mngr.getActionFor(ActionsID.NEW\_PROJECT);

// action found, inserting

if( found != null )

{

// find category of "New Project" action.

//找到New Project所属的目录（Category）

ActionsCategory category = (ActionsCategory)mngr.getActionParent(found);

// get all actions from this category (menu).

//读取该目录（Category）下的所有功能（Actions）

List actionsInCategory = category.getActions();

//add action after "New Project" action.

//找到New Project的位置

int indexOfFound = actionsInCategory.indexOf(found);

//在New Project后面插入自定义的someAction

actionsInCategory.add(indexOfFound+1, someAction);

// set all actions.

//再将更新后的所有功能置入目录中

category.setActions(actionsInCategory);

}

}

public int getPriority()

{

return AMConfigurator.MEDIUM\_PRIORITY;

}

}

#### 2.2.5 Register configurator

完成了前四步之后，你还需要将自定义的Configurator注册到MagicDraw中。MagicDraw中所有的Configurator都由ActionsConfiguratorsManager管理。因此，你需要把你的Configurator添加到ActionsConfiguratorsManager中。

代码很简单：

ActionsConfiguratorsManager.getInstance().add<configuration\_name>Configurator(configurator);

ActionsConfiguratorsManager.getInstance()获取了MagicDraw中唯一的ActionsConfiguratorsManager。

<configuration\_name>Configurator是许多组预先定义好的Configurators集合，你需要把你的configurator加入到里面。

如图2-3，如果你要把你的configurator加入到CONTAINMENT\_BROWSER\_CONTEXT中，那么上述语句应该写为：

ActionsConfiguratorsManager.getInstance().addContainmentBrowserContextConfigurator(configurator);

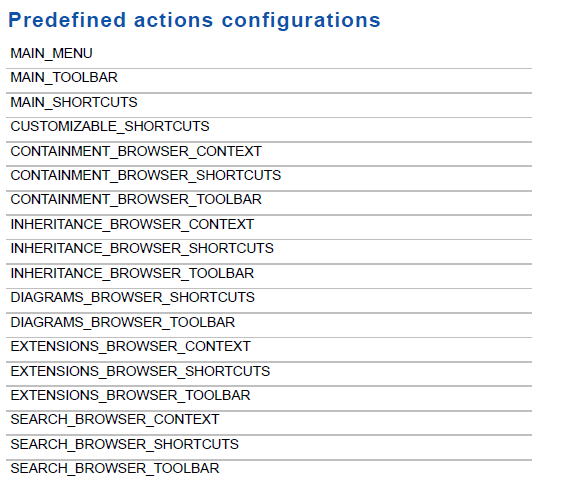


图2-3 预先定义的Configurations

### 2.3 Actions hierarchy

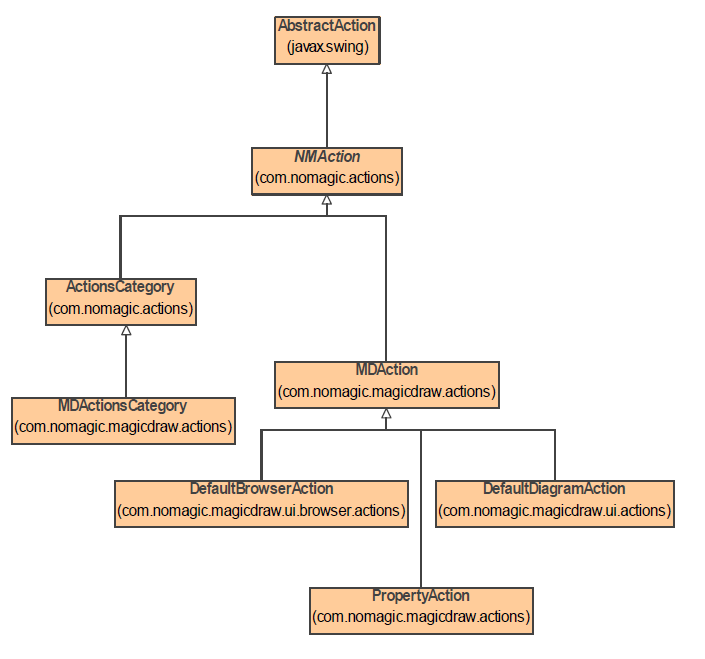


图2-4 Actions hierarchy

### 2.4 本节小结及补充

本节介绍了如何创建新的Action。

* 第一步，确定Action所处的位置：

是在menu中、toolbar中还是context中加入Action。

* 第二步，选择合适的Action基类进行继承：

根据Action所处的位置，选择合适的基类。同时，确定好Action的参数：唯一的ID、显示的名字、键盘映射的快捷键以及显示的图标。

* 第三步，编写Action主要代码：

在actionPerformed()方法中编写Action的代码。

* 第四步，编写Configurator：

根据Action的位置，继承合适的Configurator基类。并将Action加入到ActionsCategory中，再将ActionsCategory加入到ActionsManager中。

* 第五步，注册Configurator

在/openapi/examples目录中有一个例子SelectActions可供参考。

MagicDraw插件编写的调试非常麻烦，我所知道的方法是在程序中调用JOptionPane.showMessageDialog(null, message)方法查看信息。

message是你需要查看的信息，例如有一个Element实例名为a，那么调用如下方法查看a的名称：

JOptionPane.showMessageDialog(null, a.getSimpleName());

## 3 UML MODEL

### 3.1 Project

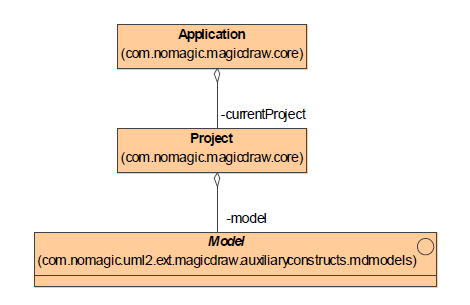


图3-1 MagicDraw的层次结构

Applicatin是MagicDraw程序，Project是你在MagicDraw程序里面打开的Project，Model是Project下所包含的元素模型。

Project project = Application.getInstance().getProject();

Project project = Project.getProject(element);

这两种方法都可以获取project。前者是根据当前运行的MagicDraw程序获取它所指向的Project，后者是根据一个元素获取了它所属的Project。

Project记录了它的根Model，也记录了它所有的Diagrams。如图3-2，方法getDiagrams()获取它所有的Diagrams，方法getActiveDiagram()获取当前你在MagicDraw中激活的Diagram，方法getModel()获取根Model。

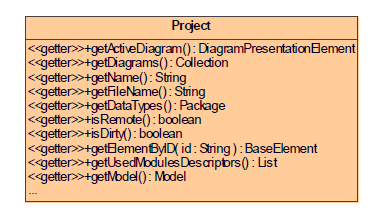


图3-2 Project的部分方法

### 3.2 Changing UML Model

#### 3.2.1 SessionManager

如果要对Element进行修改（增加、删除属性，修改属性等）必须先调用SessionManager.createSession(sessionName)方法创建Session， 修改完毕之后，再调用SessionManager.closeSession()方法关闭创建的Session。

例如：

SessionManager.createSession(yourSessionName);

//edit element

SessionManager.closeSession();

yourSessionName是Session的名字，可以为空（null）。

#### 3.2.2 ModelElementsManager

ModelElementsManager是MagicDraw提供的增加、删除或者修改model的工具类。

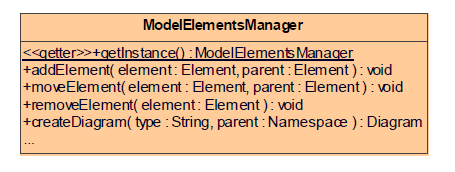


图3-3 ModelElementsManager部分方法

切记，在使用ModelElementsManager之前一定要保证Session已经创建，并且没有关闭。

ModelElementsManager在对model修改之前会自动进行许多检查，例如该model是否只读，该model父亲是否合法等。

#### 3.2.3 Create new model element

你可以使用ElementsFactory创建MagicDraw中任何模型元素的实例。

例如，创建Class：

elementsFactory.createClassInstance();

创建package：

elementsFactory.createPackgeInstance();

任何模型元素，只要你知道它的英文名字，就可以调用create<name>Instance()方法新建它的实例。

当然，做这一切之前，确保已经创建了Session。

《MagicDraw OpenAPI UserGuide》中的例子：

// creating new session

SessionManager.getInstance().createSession("Edit package A");

ElementsFactory f = Application.getInstance().getProject().getElementsFactory();

Package packageA = f.createPackageInstance();

…

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

#### 3.2.4 Editing model element

非常简单，《MagicDraw OpenAPI UserGuide》中的例子：

// creating new session

SessionManager.getInstance().createSession("Edit class A");

if (classA.isEditable())

{

classA.setName(newName);

}

SessionManager.getInstance().closeSession();

#### 3.2.5 Adding new model element or moving it to another parent

3.2.2一节介绍的工具类ModelElementsManager终于派上用场了。

将新的元素A加入到另元素B中，就是给B添加一个儿子A。

在《MagicDraw OpenAPI UserGuide》的例子中，A是Class，B是Package。

ElementsFactory f =Application.getInstance().getProject().getElementsFactory();

Class classA = f.createClassInstance();

// create new session

SessionManager.getInstance().createSession("Add class into package");

try

{

// add class into package

ModelElementsManager.getInstance().addElement(package, classA);

}

catch (ReadOnlyElementException e)

{

}

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

如果将A加入到B中是不合法的操作，系统将会根据不合法的具体情况抛出不同的异常。

当然，你也可以用另一种方式将A添加到B中：

Element parent = …;

ElementsFactory f =Application.getInstance().getProject().getElementsFactory();

Class classA = f.createClassInstance();

// create new session

SessionManager.getInstance().createSession("Add class into parent");

if (parent.canAdd(classA))

{

classA.setOwner(parent);

}

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

#### 3.2.6 Removing model element

这个操作也很简单，《MagicDraw OpenAPI UserGuide》中的例子很好理解：

Class classA = …;

// create new session

SessionManager.getInstance().createSession("Remove class");

try

{

// remove class

ModelElementsManager.getInstance().removeElement(classA);

}

catch (ReadOnlyElementException e)

{}

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

同样，你也可以不用通过ModelElementsManager删除element：

Class classA = …;

// create new session

SessionManager.getInstance().createSession("Remove class");

if (classA.isEditable()))

{

classA.dispose();

}

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

#### 3.2.7 Creating Diagram

通过我对《MagicDraw OpenAPI UserGuide》中的例子的解释，你应该能够大致了解创建Diagram的过程：

Project project = Application.getInstance().getProject();

Namespace parent = project.getModel();

// create new session

SessionManager.getInstance().createSession("Create and add diagram");

try

{

//class diagram is created and added to parent model element

Diagram diagram = ModelElementsManager.getInstance().createDiagram(DiagramTypeConstants.UML\_CLASS\_DIAGRAM, parent);

//open diagram

project.getDiagram(diagram).open();

}

catch (ReadOnlyElementException e)

{}

// apply changes and add command into command history.

SessionManager.getInstance().closeSession();

DiagramTypeConstants记录了MagicDraw默认的所有Diagram类型，例如：DiagramTypeConstatns.UML\_STATECHART\_DIAGRAM是State Machine Diagram。

它们像是Action里面的ID参数一样，是唯一的字符串，ModelElementsManager通过字符串找到Diagram，并创建新的实例。

所以，如果你要创建你自己的Diagram新实例的话，只需要把createDiagram()方法的第一个参数改为自己的Diagram的ID。

createDiagram()方法的第二个参数是新实例的父亲。

#### 3.2.8 Creating new Relationship object

Relationship object指的是连线，一个线的有两个端点，因此它有两个Element：Supplier和Client。

方法ModelHelper.isRelationship(element)判断Element是否是Relationship Object。

ModelHelper.getSupplierElement(), ModelHelper.getClientElement()分别获取两端的

Element。

创建Relationship包括三个步骤：

1. 新建Relationship model element；

2. 利用方法ModelHelper.setSupplierElement(), ModelHelper.setClientElement()指定两端的Element；

3. 将Relationship model element加入到model中（例如diagram）。

### 3.3 Working with Stereotypes and Tagged Values

MagicDraw提供了工具类StereotypesHelper处理Stereotype。StereotypesHelper中包含了许多方法：创建新Stereotype，给Element添加Stereotype，删除Stereotype，为Stereotype创建TaggedValues等。

值得注意的是，在代码中，TaggedValues被写为了Slots。

《MagicDraw OpenAPI UserGuide》用一个非常详细的例子说明了如何创建Stereotype，并为元素添加了该Stereotype，同时设置了tag：

ElementsFactory elementsFactory = project.getElementsFactory();

// create profile

//用profile存放所有新建的Stereotype

Profile profile = elementsFactory.createProfileInstance();

profile.setName("myProfile");

//把profile加入到根Model中

ModelElementsManager.getInstance().addElement(profile, project.getModel());

// get metaclass "Class"

com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class metaClass =StereotypesHelper.getMetaClassByName(project, "Class");

// create stereotype, stereotypes will be applicable to classes

//这个函数包含了三个参数

//第一个参数是这个新建的Stereotype的父亲

//第二个参数是这个新建的Stereotype的名字

//第三个参数是这个新建的Stereotype可应用的元素

Stereotype stereotype = StereotypesHelper.createStereotype(profile,"myStereotype", Arrays.asList(metaClass));

// create tag definition

//为泛型添加property

Property property = elementsFactory.createPropertyInstance();

ModelElementsManager.getInstance().addElement(property, stereotype);

// tag name

property.setName("myTag");

// tag type is String

//说明property的类型

Classifier typeString = ModelHelper.findDataTypeFor(project, "String");

property.setType(typeString);

//为element添加这个Stereotype

if (StereotypesHelper.canApplyStereotype(element, stereotype))

{

// apply stereotype

StereotypesHelper.addStereotype(element, stereotype);

// set tag value

//为element的Stereotype的Tag赋值

//这个函数包含了四个参数

//第一个参数是element

//第二个参数是element的stereotpe（element可能包含多个Stereotype）

//第三个参数是stereotype的Property的名字

//第四个参数是要赋给Property的值

StereotypesHelper.setStereotypePropertyValue(element, stereotype,"myTag", "myTagValue");

}

可以看出，创建Stereotype最关键的是调用createStereotype()方法，这个方法的三个参数说明可以看上例的具体解释。

同时，《MagicDraw OpenAPI UserGuide》还给出了一个抽取tag values的例子：

// find profile

Profile profile = StereotypesHelper.getProfile(project, "myProfile");

// find stereotype

//通过名称找到Stereotype

Stereotype stereotype = StereotypesHelper.getStereotype(project,"myStereotype", profile);

// get stereotyped elements

//调用getExtendedElements(stereotype)获取所有添加了stereotype的element

List stereotypedElements = StereotypesHelper.getExtendedElements(stereotype);

for (int i = stereotypedElements.size() - 1; i >= 0; --i)

{

// stereotyped element

Element element = (Element) stereotypedElements.get(i);

if (stereotype.hasOwnedAttribute())

{

// get tags - stereotype attributes

List<Property> attributes = stereotype.getOwnedAttribute();

for (int j = 0; j < attributes.size(); ++j)

{

Property tagDef = attributes.get(j);

// get tag value

List value = StereotypesHelper.getStereotypePropertyValue(element, stereotype, tagDef.getName());

for (int k = 0; k < value.size(); ++k)

{

// tag value

Object tagValue = (Object) value.get(j);

}

}

}

}

### 3.4 本节小结及补充

本节介绍了Element的操作。它的大部分操作都能通过工具类ModelElementsManager实现。

除此之外，还介绍了Stereotype的相关操作：创建Stereotype，为Element添加Stereotype等。

切记，任何对于模型的修改都需要打开Session，修改完毕之后需要关闭Session。

## 4 PRESENTATION ELEMENTS

### 4.1 Presentation Element

MagicDraw将所有的元素都看做是Element，而它们的图形化表示是Presentation Element。对于一个Element，它可能会有几个Presentation Element。



图4-1 Element示例

在图4-1中有三个Element：A， classDiagram， classDiagram2。

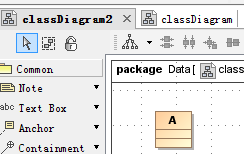
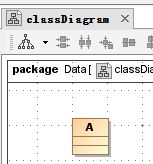


图4-2 Presentation Element示例

对应图4-1中的三个Element，在图4-2中有四个Presentation Element。这是因为A在classDiagram和classDiagram2中都有一个PresentationElement。

为了说明它们是同一个Element，我们可以对A进行修改，为A添加属性a。

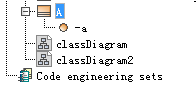


图4-3 Class A添加属性a

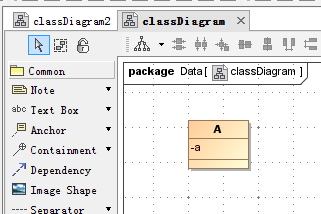
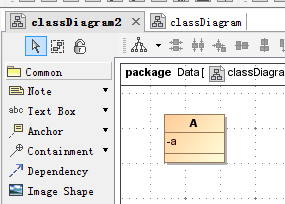


图4-4 Presentation Element的变化

从图4-4中可以看出，在classDiagram和classDiagram2中的两个Presentation Element是同一个Element。

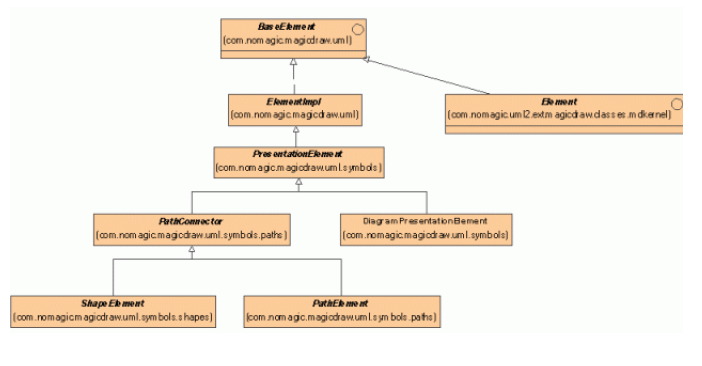


图4-5 PresentationElement的继承结构

图形化表示分为三种：Shape、Path以及Diagram。

顾名思义，Shape表示的是类似于Class、Package等图形；Path表示的是Dependency、Connector等连线，也就是3.2.8中描述的Relationship Object的图形化表示；Diagram是所有图的图形化表示。

### 4.2 Presentations Elements Manager

你需要通过PresentationsElementsManager修改Presentation element的属性、创建新的Presentation element。

在使用PresentationsElementsManager之前记得先创建Session，使用完毕止之后记得关闭Session。

#### 4.2.1 Creating shape element

你可以调用方法createShapeElement创建shape element，默认位置为（0，0）。类似于：

com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class clazz = …;

DiagramPresentationElement diagram = …;

SessionManager.getInstance().createSession("Test");

ShapeElement shape =PresentationsElementsManager.getInstance().createShapeElement(clazz,diagram);

SessionManager.getInstance().closeSession();

通过上述代码，你可以在diagram中为Element：clazz创建Shape Element：shape。

#### 4.2.2 Creating path element

com.nomagic.uml2.ext.magicdraw.classes.mddependencies.Dependency link = …;

PresentationElement clientPE = …;

PresentationElement supplierPE = …;

SessionManager.getInstance().createSession("Test");

PathElement path =PresentationsElementsManager.getInstance().createPathElement(link,clientPE,supplierPE);

SessionManager.getInstance().closeSession();

创建Path Element时没有指定diagram，它会自动在clientPE和supplierPE所属的diagram中创建。

#### 4.2.3 Reshaping shape element

ShapeElement element = …;

Rectangle newBounds = new Rectangle(100,100,80,50);

SessionsManager.getInstance().createSession(“Test”);

PresentationsElementsManager.getInstance().reshapeShapeElement(element,newBounds);

SessionsManager.getInstance().closeSession();

在newBounds指定的矩阵中，重新绘制shape element。

Rectangle的四个参数分别是x坐标、y坐标、宽、长。

每一个Shape都有preferred size，你指定的新size不能小于这个size。

#### 4.2.4 Changing path break points

PathElement element = …;

ArrayList points = new ArrayList();

points.add(new Point(100, 100));

points.add(new Point(100, 150));

SessionsManager.getInstance().createSession(“Test”);

PresentationsElementsManager.getInstance().changePathBreakPoints(element,points);

SessionsManager.getInstance().closeSession();

Break points值的就是线段的两个端点。

上述代码示范了如何改变两个端点的位置，端点的顺序是先supplier后client。

#### 4.2.5 Deleting presentation element

PresentationElement element = …;

SessionsManager.getInstance().createSession(“Test”);

PresentationsElementsManager.getInstance().removePresentationElement(element);

SessionsManager.getInstance().closeSession();

#### 4.2.6 Changing properties of presentation element

像Action一样，你需要将新创建的Property加入到PropertyManager中，然后再将PropertyManager加入到Presentation Element。

PropertyManager是一组Property的集合，PresentationElement包含了许多PropertyManager。

ShapeElement element = …;

//新建PropertyManager

PropertyManager properties = new PropertyManager();

//添加新的Property，类型是Boolean，默认值是true

properties.addProperty(new BooleanProperty(PropertyID.AUTOSIZE, true));

SessionsManager.getInstance().createSession(“Test”);

//通过工具类PresentationsElementsManager将PropertyManager置于ShapeElement中

PresentationsElementsManager.getInstance().setPresentationElementProperties(element, properties);

SessionsManager.getInstance().closeSession();

值得一提的是，必须新建PropertyManager实例，而不能修改原有的PropertyManager实例。

### 4.3 本节小结及补充

如果需要自定义Property，可以参考《MagicDraw OpenAPI UserGuide》中“Properties”一节。

第三章介绍的是Element相关的内容，这一章介绍的是与Element息息相关的PresentationElement。

PresentationElement是Element的图形化表示，PresentationElement的大部分操作可以通过工具类PresentationElementsManager实现。

切记，任何对于模型的修改都需要先打开Session，修改完毕之后需要关闭Session。

## 5 NEW DIAGRAM TYPES

### 5.1 Diagram Types Hierarchy

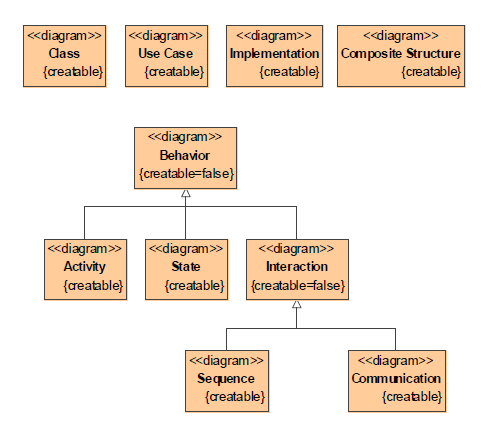


图5-1 Diagram继承结构

图5-1中，creatable=false的diagram type是不能创建的，只能作为基类使用。

如果你要创建新的diagram type，只能继承自图5-1中某一个图，具体的步骤接下来详细介绍。

### 5.2 Adding a new diagram type for MagicDraw

新建diagram type分为两步，第一步继承DiagramDescriptor类，第二步注册diagram type。

#### 5.2.1 Override abstract DiagramDescriptor class

新建的diagram type需要继承DiagramDescriptor类，这个类包含以下方法：

**•** getDiagramTypeId() –返回Diagram Type ID，唯一的ID；

**•** getSingularDiagramTypeHumanName() –返回单数形式的名字；

**•** getPluralDiagramTypeHumanName() –返回复数形式的名字；

**•** getSuperType() –返回继承的diagram type；

**•** isCreatable() – 是否可创建；

**•** getLargeIcon() – LargeIcon的介绍在2.2.2节；

**•** getSmallIconURL() – SmallIcon的介绍在2.2.2节；

**•** getDiagramActions() –返回diagram的ActionsManager，ActionsManager的介绍在2.2.4节；

**•** getDiagramToolbarConfigurator() – 返回diagram的toolbar configurator，configurator的介绍在2.2.4节；

**•** getDiagramShortcutsConfigurator() – 返回diagram的shortcuts configurator，configurator的介绍在2.2.4节；

**•** getDiagramContextConfigurator() –返回diagram的context configurator，configurator的介绍在2.2.4节

《MagicDraw OpenAPI UserGuide》的例子非常清晰地说明了如何创建自定义的digram type：

/\*\*

\* Descriptor of specific diagram.

\*/

//继承DiagramDescriptor

public class SpecificDiagramDescriptor extends DiagramDescriptor

{

public static final String SPECIFIC\_DIAGRAM = "Specific Diagram";

/\*\*

\* Let this diagram type be a sub type of class diagram type.

\*/

//定义基类，这里使用的是class diagram

public String getSuperType()

{

return DiagramType.UML\_CLASS\_DIAGRAM;

}

/\*\*

\* This is creatable diagram.

\*/

public boolean isCreatable()

{

return true;

}

/\*\*

\* Actions used in this diagram.

\*/

//diagram的ActionsManager，也就是ActionsCategory的集合，ActionsCategory是Actions的集合

//这里在另外的文件中定义了SpecificDiagramActions

public MDActionsManager getDiagramActions()

{

return SpecificDiagramActions.ACTIONS;

}

/\*\*

\* Configurator for diagram toolbar.

\*/

//Toolbar的configurator

//自定义新的ToolBarConfigurator

public AMConfigurator getDiagramToolbarConfigurator()

{

return new SpecificDiagramToolbarConfigurator();

}

/\*\*

\* Configurator for diagram shortcuts.

\*/

//Shortcuts的configurator

//直接使用class diagram的shortcuts configurator

public AMConfigurator getDiagramShortcutsConfigurator()

{

return new ClassDiagramShortcutsConfigurator();}

/\*\*

\* Configurator for diagram context menu.

\*/

//diagram context menu的configurator

//直接使用base diagram的configurator

public DiagramContextAMConfigurator getDiagramContextConfigurator()

{

return new BaseDiagramContextAMConfigurator();

}

/\*\*

\* Id of the diagram type.

\*/

//diagram type的唯一ID

public String getDiagramTypeId()

{

return SPECIFIC\_DIAGRAM;

}

/\*\*

\* Diagram type human name.

\*/

//diagram的单数名字

public String getSingularDiagramTypeHumanName()

{

return "Specific Diagram";

}

/\*\*

\* Diagram type human name in plural form.

\*/

//diagram的复数名字

public String getPluralDiagramTypeHumanName()

{

return "Specific Diagrams";

}

/\*\*

\* Large icon for diagram.

\*/

//设置LargeIcon

public ImageIcon getLargeIcon()

{

return new ImageIconProxy(newVectorImageIconControler(getClass(), "icons/specificdiagram.svg",VectorImageIconControler.SVG));

}

/\*\*

\* URL to small icon for diagram.

\*/

//设置SmallIcon

public URL getSmallIconURL()

{

return getClass().getResource("icons/specificdiagram.svg");

}

}

说明：

可以看到LargeIcon使用的是SVG格式的图片，SVG中文名称为可缩放矢量图形（Scalable Vector Graphics，SVG）。

它是基于XML，用于描述二维矢量图形的一种图形格式。

#### 5.2.2 Register new diagram type in the application

这一步很简单，在Plugin类中的init()方法中写入如下代码即可：

Application.getInstance().addNewDiagramType(new SpecificDiagramDescriptor());

SpecificDiagramDescriptor是你在第一步中自定义的类名。

### 5.3 本节小结及补充

本节介绍了如何创建新的diagram type。

创建新的diagram type只需要继承DiagramDescriptor，并注册即可。

本节例子的全部代码可以在/openapi/examples/specificdiagram中找到。

## 6 MagicDraw插件开发总结

第一章介绍了插件的工作原理和基本格式，简单地说就是一个文件夹，文件夹里面抱包含了一个.jar文件和一个plugin.xml。plugin.xml有一定的格式要求，具体在1.2节。

第二章介绍了如何在MagicDraw中添加新的功能（Action），以及如何注册新的功能。MagicDraw提供了三个简单的Action基类，你可以继承它们中某一个。

编写完Action代码，不能直接调用Action，需要用一定的机制将其注册。

你需要用Configurator将Action配置到ActionsManager中才能使用。ActionsManager像一个大的容器，例如：All toolbars、Menu bar等。它里面包含了一个个具体的目录，例如All toolbars包含了许多Toolbar，Menu bar包含了许多Menu。这些目录是由ActionsCategory表示的。所以Configurator的任务是将Action添加到ActionsCategory中，再把ActionsCategory放置到ActionsManager中。

最后，还需要把configurator注册到MagicDraw中。

用图形表示：

YourConfigurator{

YourAction->YourActionsCategory->ActionsManager

}

YourConfigurator->Application

第三章介绍了MagicDraw中Element相关的操作，如果你是要实现修改Element的插件，那么你需要在第二章中Action的actionPerformed()方法中，添加你的代码。

第四章介绍了MagicDraw中PresentationElement相关的操作，如果你是要实现修改PresentationElement的插件，同样你需要在第二章中Action的actionPerformed()方法中，添加你的代码。

第五章介绍了如何在MagicDraw中创建新的Diagram Type，你的Diagram Type需要继承自DiagramDescriptor，并实现它的所有抽象方法。

除此之外，《MagicDraw OpenAPI UserGuide》中还包括了：Distributing Resources、JPython Scripting、Patterns、Properties、Projects Management、Selections Management、Creating Images、Creating Metrics、Configuring Element Specification、Custom diagram painters、Validation、Teamwork、Code Engineering、Oracle DDL generation and customization、Running MagicDraw in batch mode等内容。

# Function Block开发文档

## 1 Function Block开发目的

在MagicDraw中形象地表现IEC 61499的Function Block，方便工程师在MagicDraw中进行Function Block的设计。

## 2 Function Block开发内容

* 创建了新的Stereotype并保存在名为FunctionBlock’s Stereotypes的Profile中：

BasicFunctionBlock，

CompositeFunctionBlock，

EventInPort，

EventOutPort，

DataInPort，

DataOutPort，

ECCInitialState，

ECCState，

ECCTransition，

Algorithm，

EventOutput；

* 创建了新的Diagram type：

Function Block Diagram，

ECC Diagram；

* 在Function Block Diagram中创建了基本的Actions：

BasicFunctionBlockAction，

CompositeFunctionBlockAction，

DataInPortAction，

DateOutPortAction，

EventInPortAction，

EventOutPortAction；

* 在ECC Diagram中创建了基本的Actions：

AlgorithmAction，

ECCInitialStateAction，

ECCStateAction，

ECCTransitionAction；

* 为了形象化表示Function Block中的元素，创建了一系列render：

AlgorithmRender，

DependencyRender，

EventOutputRender，

FunctionBlockRender，

InitialStateRender，

PortRender。

## 3 Function Block文件结构

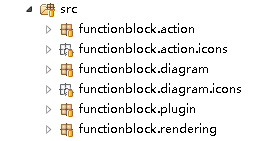


图3-1 Function Block文件结构

/src/functionblock/action——存放了所有的actions源代码，

/src/functionblock/action/icons——存放了与action有关的图标，

/src/functionblock/diagram——存放了自定义diagram的源代码，

/src/functionblock/diagram/icons——存放了与diagram有关的图标，

/src/functionblock/plugin——存放了插件主类以及辅助类，

/src/functionblock/rendering——存放了所有的render源代码。

## 4 Function Block的设计

### 4.1 插件主类与辅助类设计

#### 4.1.1 插件主类FunctionBlockPlugin

该类继承自Plugin，重载了Plugin的三个方法。

在init()方法中，注册了自定义的renders、自定义的FunctionBlockDiagram、自定义的ECCDiagram。并调用辅助类FunctionBlockHelper的静态方法完成了Stereotype的创建。

#### 4.1.2 辅助类FunctionBlockHelper

辅助类包含三个静态方法：createSession()、closeSession(flag)和createStereotypes()。

设计createSession()和closeSession(flag)的初衷来源于我经常需要修改Element和PresentationElement。

SessionsManager.getInstance().createSession();

//Section 1

SessionsManager.getInstance().closeSession();

//…

SessionsManager.getInstance().createSession();

//Section 2

SessionsManager.getInstance().closeSession();

在我编写代码的过程中，经常会碰到这种情况：在Section 1中调用了某个方法，转到Section 2。这时Session又被创建了一次，但是只会存在一个Session，此时并不会报错。但是当Section 2执行完毕，关闭Session。回到Section 1之后，程序会尝试再次关闭Session，这时将会产生错误。

因此，我设计了这两个静态方法，createSession()会返回一个Boolean类型的变量，表示是否当前代码段创建的Session。

closeSession(flag)会根据flag的值决定是否关闭Session。当flag为true时，表示是当前代码段创建的Session；当flag为false时，表示不是由当前代码段创建的Session。

#### 4.1.3 常数类FunctionBlockConstants

FunctionBlockConstants中包含了Function Block中所有的常量，包括三大部分：DiagramsConstants， ActionsConstants， StereotypesConstants。

DiagramsConstants保存了与FunctionBlockDiagram和ECCDiagram相关的常量，包括：ID、名字、图标路径等。

ActionsConstants保存了所有Actions的ID、名字、图标路径等。

StereotypesConstants保存了所有的Stereotypes的名字。

### 4.2 自定义Diagram设计

#### 4.2.1 Function Block Diagram设计

Function Block Diagram通过类FunctionBlockDiagramDescriptor实现。

FunctionBlockDiagramDescriptor继承自DiagramDescriptor，基类选择的是Class diagram。自定义了Toolbar和Actions。

值得一提的是，如果要使用MagicDraw默认的Actions，要通过ActionsID调用。

例如：

//ActionsID类中保存是MagicDraw默认的Actions的ID

category.addAction(actions.getActionFor(ActionsID.*ADD\_DEPENDENCY*));

//DataInPortAction.DRAW\_DATA\_IN\_PORT\_ACTION保存了DataInPortAction的ID

category.addAction(actions.getActionFor(DataInPortAction.*DRAW\_DATA\_IN\_PORT\_ACTION*));

前者是MagicDraw默认的Action，添加一个Dependency；

后者是自定义的一个Action。

FunctionBlockDiagramToolbarConfigurator是自定义的ToolBarConfigurator，

FunctionBlockDiagramActions是自定义的Actions集合。

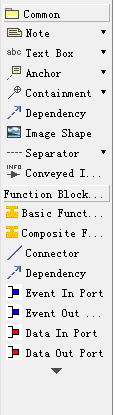


图4-1 Function Block Diagram的Toolbar

#### 4.2.2 ECC Diagram设计



图4-2 ECC Diagram的Toolbar

ECC Diagram是通过类ECCDiagramDescriptor实现。

ECCDiagramDescriptor继承自DiagramDescriptor，基类选择的是State Machine Diagram。自定义了Toolbar和Actions。

ECCDiagramToolbarConfigurator是自定义的ToolBarConfigurator，

ECCDiagramActions是自定义的Actions集合。

### 4.3 Actions设计

我需要设计的显然是DiagramAction，那么我应该继承BaseDiagramAction。

但是实际上，存在着层次更高的类DrawShapeDiagramAction和DrawPathDiagramAction供我使用。

根据类名，DrawShapeDiagramAction是在Diagram上面绘制Shape的Action，DrawPathDiagramAction是在Diagram上绘制Path的Action。

如此一来，我设计的Action可以分为两组：

AlgorithmAction， BasicFunctionBlockAction， CompositeFunctionBlockAction， DataInPortAction， DataOutputPortAction， ECCStateAction， EventInPortAction， EventOutputPortAction继承DrawShapeDiagramAction。

ECCTransitionAction继承DrawPathDiagramAction。

#### 4.3.1 DrawShapeDiagramAction

继承自该类的Action，必须实现createElement()方法。

需要调用基类的构造函数，传递ID、名字、键盘快捷键映射。

同时在自己的构造函数中调用setLargeIcon(icon)方法，为Action设置图标。

##### 4.3.1.1 DataInPortAction, DataOutPortAction, EventInPortAction, EventOutPortAction

这四个Action比较类似，都是在createElement()中创建Port，并添加Stereotype：

protected Element createElement() {

//创建Port实例

com.nomagic.uml2.ext.magicdraw.compositestructures.mdports.Port port = Application.*getInstance*().getProject().getElementsFactory().createPortInstance();

//调用自己编写的函数为Port添加Stereotype

applyStereotype(port);

return port;

}

这四个Action唯一的区别在于Event和Data的图标不同，因此，DataInPortAction和DataOutPortAction共用基类DataPortAction；EventInPortAction和EventOutputPortAction共用基类EventPortAction。

##### 4.3.1.2 ECCInitialStateAction， ECCStateAction

、 两者都是创建State实例，只是添加的Stereotype不相同，前者是添加ECCInitialState，后者是添加ECCState。

以ECCStateAction为例：

protected Element createElement() {

//创建State实例

State state = Application.*getInstance*().getProject().getElementsFactory().createStateInstance();

//添加Stereotype

//这个函数包含两个参数

//第一个参数是需要添加Stereotype的Element

//第二个参数是Stereotype的名称

StereotypesHelper.*addStereotypeByString*(state, FunctionBlockConstants.StereotypesConstants.*ECC\_STATE*);

return state;

}

##### 4.3.1.3 BasicFunctionBlockAction， CompositeFunctionBlockAction

两者都是创建了Class，并添加Stereotype。同时，添加一个DataInPort，一个DataOutPort，一个EventInPort，一个EventOutPort。

BasicFunctionBlockAction在Class中还嵌入了一个ECCDiagram，表示Basic Function Block中的ECC：

protected Element createElement() {

//创建Class实例

com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class clazz = Application.*getInstance*().getProject().getElementsFactory().createClassInstance();

clazz.setActive(true);

//添加Stereotype

StereotypesHelper.*addStereotypeByString*(clazz, FunctionBlockConstants.StereotypesConstants.*BASIC\_FUNCTION\_BLOCK*);

//添加四个Port

addBasicPorts(clazz);

//在clazz中添加element，这个element的类型是StateMachine

//StateMachine中包含了ECCDiagram

addECC(clazz);

return clazz;

}

private void addECC(Class functionblock) {

//创建Session

boolean sessionFlag = FunctionBlockHelper.*createSession*();

try {

//创建状态机ECC

StateMachine ecc = Application.*getInstance*().getProject().getElementsFactory().createStateMachineInstance();

//把ecc添加到functionblock中，并命名为“ECC”

ecc.setOwner(functionblock);

ecc.setName("ECC");

//为ecc创建ECCDiagram

ModelElementsManager.*getInstance*().createDiagram(ECCDiagramDescriptor.*ECC\_DIAGRAM*, ecc);

} catch (Exception ex) {

JOptionPane.*showMessageDialog*(null, "BasicFunctionBlockAction.addECC(Class) " + ex);

}

//关闭Session

FunctionBlockHelper.*closeSession*(sessionFlag);

}

CompositeFunctionBlockAction在Class中嵌入了CompositeStructureDiagram，表示Composite Function Block内部的结构图：

protected Element createElement() {

com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class clazz = Application.*getInstance*().getProject().getElementsFactory().createClassInstance();

clazz.setActive(true);

StereotypesHelper.*addStereotypeByString*(clazz, FunctionBlockConstants.StereotypesConstants.*COMPOSITE\_FUNCTION\_BLOCK*);

addBasicPorts(clazz);

//在clazz中添加element，这个element的类型是CompositeStructureDiagram

addCompositeStructureDiagram(clazz);

return clazz;

}

private void addCompositeStructureDiagram(Class functionblock) {

boolean sessionFlag = FunctionBlockHelper.*createSession*();

try {

//为functionblock创建CompositeStructureDiagram

ModelElementsManager.*getInstance*().createDiagram(DiagramType.*UML\_COMPOSITE\_STRUCTURE\_DIAGRAM*, functionblock);

} catch (Exception ex) {

JOptionPane.*showMessageDialog*(null, "CompositeFunctionBlockAction.addCompositeStructureDiagram(Class) " + ex);

}

FunctionBlockHelper.*closeSession*(sessionFlag);

}

##### 4.3.1.4 AlgorithmAction

创建了State，并添加Algorithm的Stereotype。同时在其内部创建了Comment，添加EventOutput的Stereotype。

两者表示ECC中的Action，Action由Algorithm和Event Output组成。

我们可以在Algorithm里面添加Constraints，表示算法。

Event Output是当算法结束时，输出的事件。所以它必须紧紧跟随Algorithm。为了之后能够方便绘制Event Output，让它作为Algorithm的子Element。

protected Element createElement() {

// TODO Auto-generated method stub

//创建State实例

State algorithm = Application.*getInstance*().getProject().getElementsFactory().createStateInstance();

//添加Stereotype

StereotypesHelper.*addStereotypeByString*(algorithm, FunctionBlockConstants.StereotypesConstants.*ALGORITHM*);

//创建Comment实例

Comment eventOutput = Application.*getInstance*().getProject().getElementsFactory().createCommentInstance();

//添加Stereotype

StereotypesHelper.*addStereotypeByString*(eventOutput, FunctionBlockConstants.StereotypesConstants.*EVENT\_OUTPUT*);

//把eventOutput加入algorithm中

eventOutput.setOwner(algorithm);

return algorithm;

}

#### 4.3.2 DrawPathDiagramAction

继承自该类的Action，必须实现createElement()方法。

需要调用基类的构造函数，传递ID、名字、键盘快捷键映射。

同时在自己的构造函数中调用setLargeIcon(icon)方法，为Action设置图标。

##### 4.3.2.1 ECCTransitionAction

创建了Transition，并添加了ECCTransition的Stereotype，表示ECC中的Transition。我们可以在Transition上添加Constraints，表示ECC中的Transition Condition：

protected Element createElement() {

//创建Transition实例

Transition transition = Application.*getInstance*().getProject().getElementsFactory().createTransitionInstance();

//添加Stereotype

StereotypesHelper.*addStereotypeByString*(transition, FunctionBlockConstants.StereotypesConstants.*ECC\_TRANSITION*);

return transition;

}

### 4.4 Renders设计

所有的Renders都通过RendererProvider提供。

PresentationElement是MagicDraw中用来形象化表示Element的，其实，在PresentationElement之上还有更为具体的表示类。那就是各种View类。

例如Class的Element，Element映射到PresentationElement，ClassView继承PresentationElement，

又例如Port的Element，Element映射到PresentationElement，PortView继承PresentationElement。

在MagicDraw中每一个元素都有一个View，具体可以在com.nomagic.magicdraw.uml.symbols.shapes.\*和com.nomagic.magicdraw.uml.symbols.path.\*下寻找。这些类都是以<name>View的形式命名的。

如果我们想要改变某个element的外形，那么我们只要改变它的View的绘制方法即可。

但是，我只是要改变带有特殊Stereotype的element的外形。

因此，只要当检测到可能需要修改的View时，进行详细判断，然后重新绘制

。

例如：对于Class，当检测到需要绘制ClassView时，那么它有可能是Basic Function Block或者Composite Function Block。

这时，我们通过getter方法，获取到它的Element，判断该Element是否添加了BasicFunctionBlock或CompositeFunctionBlock的Stereotype：

//判断PresentationElement是不是ClassView

if (presentationElement instanceof ClassView) {

//获取实际的Element

Element element = presentationElement.getActualElement();

//判断Element是否添加了BasicFunctionBlock或CompositeFunctionBlock的Stereotype

//StereotypesHelper.getStereotypes(element)方法获取添加在element上的所有Stereotype

if (StereotypesHelper.*getStereotypes*(element).contains(StereotypesHelper.*getStereotype*(project, *BASIC\_FUNCTION\_BLOCK*)) || StereotypesHelper.*getStereotypes*(element).contains(StereotypesHelper.*getStereotype*(project, *COMPOSITE\_FUNCTION\_BLOCK*))) {

//为PresentationElement添加自己的renderer

obj = functionBlockRender;

addToCash(presentationElement, ((SymbolDecorator) (obj))); }

}

obj是ShapeDecorator或者PathDecorator类型。ShapeDecorator绘制Shape，PathDecorator绘制Path。

我设计的AlgorithmRender，EventOutputRender，FunctionBlockRender，InitialStateRender， PortRender继承自ShapeDecorator。

DependencyRender继承自PathDecorator。

#### 4.4.1 ShapeDecorator

所有继承自ShapeDecorator的类，如果要实现自己的绘制方法，需要重载draw(g, presentationelement)方法。

##### 4.4.1.1 FunctionBlockRender

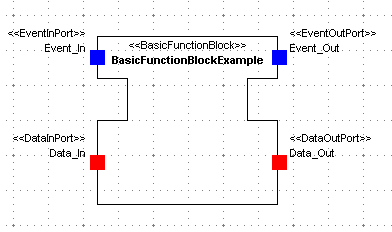


图4-3 Basic Function Block与四个Port

如图，FunctionBlockRender的工作是将原来的Class绘制成Function Block的形状。

public class FunctionBlockRenderer extends ShapeDecorator {

//重载draw方法，绘制特殊图形

@Override

public void draw(Graphics g, PresentationElement presentationelement) {

// TODO Auto-generated method stub

//获取getBounds()方法，获取矩形外框

Rectangle r= presentationelement.getNotCopyBounds();

int x0 = r.x;

int y0 = r.y;

int xStep = r.width / 6;

int yStep = r.height / 4;

//计算得出构成Function Block形状的端点坐标

int xPoints[] = { x0, x0 + 6 \* xStep, x0 + 6 \* xStep, x0 + 5 \* xStep,

x0 + 5 \* xStep, x0 + 6 \* xStep, x0 + 6 \* xStep, x0, x0,

x0 + xStep, x0 + xStep, x0 };

int yPoints[] = { y0, y0, y0 + yStep, y0 + yStep, y0 + 2 \* yStep,

y0 + 2 \* yStep, y0 + 4 \* yStep, y0 + 4 \* yStep, y0 + 2 \* yStep,

y0 + 2 \* yStep, y0 + yStep, y0 + yStep };

//绘制Function Block

g.drawPolygon(xPoints, yPoints, 12);

for (PresentationElement pe : presentationelement.getPresentationElements()) {

//对于ClassView中的其他View进行操作

if (pe instanceof ClassHeaderView) {

//ClassHeaderView包含了class的许多信息，抽取其中必要的进行绘制

ClassHeaderView classHeaderView = (ClassHeaderView) pe;

classHeaderView.getNameLabel().shapeSpecificDraw(g);

classHeaderView.getStereotypeLabel().shapeSpecificDraw(g);

} else {

//绘制class中其他可见的PresentationElement

if (pe instanceof PortView) {

pe.setVisible(true);

}

if (pe.isVisible()) {

pe.draw(g);

}

}

}

//绘制class的装饰部分

presentationelement.paintAdornments(g, presentationelement);

}

}

值得一提的是ClassView中的ClassHeaderView，Class的许多信息都包含在了ClassHeaderView中，例如Class包含的Stereotype名称，Class的名称，Class包含的Attribute名称等。

因为Function Block并不需要Attribute和Operation等，因此我只选择了必要的部分进行绘制。

##### 4.4.1.2 PortRender

PortRender比较复杂，我需要绘制三种Port：普通Port，EventPort，DataPort。

对于普通Port，直接调用presentationelement的shapeSpecificDraw(g)方法进行默认绘制；

由于EventPort和DataPort在Function Block上有一定的位置限制，因此需要对位置不合法的Port进行调整。同时，用蓝色填充EventPort，用红色填充DataPort以示区别。

以绘制EventOutPort为例：

public void drawEventOutPort(Graphics g, PresentationElement presentationelement) {

//记录当前颜色

Color c = g.getColor();

//设置填充颜色

g.setColor(Color.*BLUE*);

//获取Port所属的Function Block边框

Rectangle FBrec = presentationelement.getParent().getBounds();

//获取Port的边框

Rectangle rec = presentationelement.getBounds();

//设置Port的x坐标到合法位置

//对于Out Port，x坐标在Function Block边框的最右边

//对于In Port，x坐标在Function Block边框的最左边

//任何PresentationElement的Bounds都是以左上角的坐标作为（x，y）值的。

presentationelement.setLocation(FBrec.x + FBrec.width - rec.width / 2, rec.y);

//检测Port的y坐标是否合法

//对于Event Port，y坐标在Function Block边框的上部分

//对于Data Port，y坐标在Function Block边框的下部分

if (out(rec.y, FBrec.y - rec.height / 2, FBrec.y + FBrec.height / 4 - rec.height / 2)) {

presentationelement.setLocation(FBrec.x + FBrec.width - rec.width / 2, FBrec.y + FBrec.height / 8 - rec.height / 2);

}

//获取更新后的Port边框

rec = presentationelement.getBounds();

//进行填充绘制

g.fillRect(rec.x, rec.y, rec.width, rec.height);

//还原颜色

g.setColor(c);

}

##### 4.4.1.3 InitialStateRender

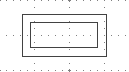


图4-4 ECC中的Initial State

InitialStateRender的工作是将添加了ECCInitialState的Stereotype的State绘制成Function Block中的形状。

这个“回”字形，实际上由两个矩形构成。

public void draw(Graphics g, PresentationElement presentationelement) {

// TODO Auto-generated method stub

//获取presentationelement的边框

Rectangle r = presentationelement.getNotCopyBounds();

//将宽设置为高的两倍，保持矩形的形状

presentationelement.setBounds(r.x, r.y, r.height \* 2, r.height);

r = presentationelement.getBounds();

//绘制外层的矩形

g.drawRect(r.x, r.y, r.width, r.height);

//绘制内层的矩形

g.drawRect(r.x + r.width / 10, r.y + r.height / 5, r.width \* 8 / 10, r.height \* 3 / 5);

//绘制presentationelement的装饰部分

presentationelement.paintAdornments(g, presentationelement);

}

##### 4.4.1.4 AlgorithmRender

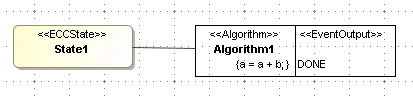


图4-5 ECC中的State以及它的Action

AlgorithmRender的任务很简单，将添加了Algorithm的Stereotype的State形状绘制成矩形。

public void draw(Graphics g, PresentationElement presentationelement) {

Rectangle r = presentationelement.getNotCopyBounds();

g.drawRect(r.x, r.y, r.width, r.height);

for (PresentationElement pe : Presentationelement.getPresentationElements()) {

if (pe.isVisible()) {

pe.draw(g);

}

}

presentationelement.paintAdornments(g, presentationelement);

##### 4.4.1.5 EventOutputRender

Event Output显示时要紧紧贴着Algorithm，因此EventOutputRender需要知道谁是跟它关联的Algorithm。

由于之前AlgorithmAction中，Event Output是作为Algorithm的子Element保存。因此，找到Event Output所属的DiagramPresentationElement，然后遍历DiagramPresentationElement内所有的PrensentationElement，直到找到Event Output的父亲为止。

public void draw(Graphics g, PresentationElement presentationelement) {

//得到presentationelement所属的diagramPresentationElement

DiagramPresentationElement diagramPresentationElement = presentationelement.getDiagramPresentationElement();

StateView algorithm = null;

Element element = presentationelement.getActualElement();

//遍历diagramPresentationElement中的所有PresentationElement

for (PresentationElement pe : diagramPresentationElement.getPresentationElements()) {

//判断是否为Event Output的父亲

if (pe.getActualElement().getOwnedElement().contains(element)) {

//找到Algorithm

algorithm = (StateView) pe;

break;

}

}

//调整Stereotype标签显示的位置

handleStereotypeLabel(algorithm, (CommentView) presentationelement);

//调整Event Output的边框

handleBounds(algorithm, (CommentView) presentationelement);

//获取Event Output更新后的边框

Rectangle r = presentationelement.getBounds();

//绘制矩形

g.drawRect(r.x, r.y, r.width, r.height);

//绘制Event Output的可视的子PresentationElement

for (PresentationElement pe : presentationelement.getPresentationElements()) {

if (pe.isVisible()) {

if (pe instanceof TextAreaView) {

} else {

pe.draw(g);

}

}

}

presentationelement.paintAdornments(g, presentationelement);

}

//调整Stereotype标签显示的位置

private void handleStereotypeLabel(StateView algorithm, CommentView eventOutput) {

//获取Algorithm的Stereotype标签

TextAreaView algStereo = algorithm.getStereotypeLabel();

//获取Event Output的Stereotype标签

TextAreaView cmtStereo = eventOutput.getStereotypeLabel();

//Algorithm的Stereotype标签的边框

Rectangle r = algStereo.getBounds();

//设置Event Output的Stereotype标签的边框

cmtStereo.setBounds(cmtStereo.getBounds().x, r.y, r.width, r.height);

//设置Event Output的Stereotype标签的边框最小值和最优值

cmtStereo.setMinimumSize(algStereo.getMinimumSize().width, algStereo.getMinimumSize().height);

cmtStereo.setPreferredSize(algStereo.getPreferredSize().width, algStereo.getPreferredSize().height);

}

//调整Event Output的边框

private void handleBounds(StateView algorithm, CommentView eventOutput) {

//获取Algorithm的边框

Rectangle r = algorithm.getBounds();

//设置Event Output的边框紧贴Algorithm的边框，同时大小相同

eventOutput.setBounds(r.x + r.width, r.y, r.width, r.height);

//设置Event Output的边框最小值和最优值

eventOutput.setMinimumSize(algorithm.getMinimumSize().width, algorithm.getMinimumSize().height);

eventOutput.setPreferredSize(algorithm.getPreferredSize().width, algorithm.getPreferredSize().height);

}

#### 4.4.2 PathDecorator

##### 4.4.2.1 DependencyRender

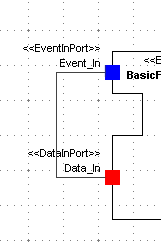


图4-6 Data Port和Event Port的依赖关系



图4-7 ECC Action和ECC State的依赖关系

在Function Block中Dependency主要用于两个地方：

表示Data Port和Event Port的依赖关系；

表示ECC Action和ECC State的依赖关系。

只需要简单地将MagicDraw中原本带箭头的虚线改为不带箭头的实线。

## 5 Function Block范例

Calculator是一个简单的计算器，它能够对两个数进行加、减、乘、除四种运算。

Calculator有四个事件输入端口 ，一个事件输出端口，两个数据输入端口和一个数据输出端口，两个数据输入端口与四个事件输入端口都有关联，数据输出端口与时间输出端口有关联。

Calculator的ECC除了初始状态之外有四个状态，分别是add、sub、mul、div。它们分别关联着四个不同的动作（ECC Action）。

add的Action由加法算法与DONE事件输出组成；

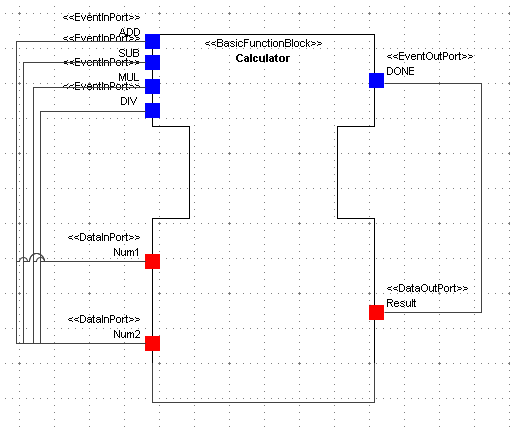


图5-1 Calculator的外部结构

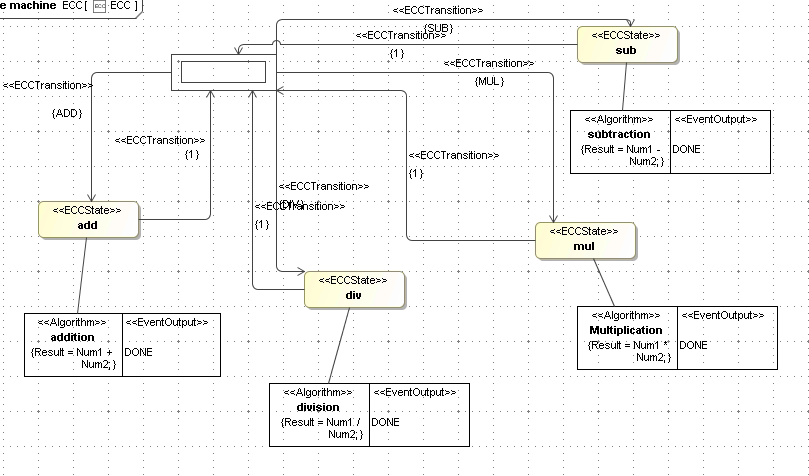


图5-2 Calculator的内部ECC构造

sub的Action由减法算法与DONE事件输出组成；

mul的Action由乘法算法与DONE事件输出组成；

div的Action由除法算法与DONE事件输出组成；

当不同事件发生时，初始状态会跳转到某一个状态。状态的算法完成之后，无条件跳转回初始状态。