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
1 PLUG-INS 1
1.1 How plug-ins work 1
1.2 Writing plug-in 2
1.3 本节小结及补充 3
2 ADDING NEW FUNCTIONALITY 4
2.1 Invoking Actions 4
2.2 Creating a new action for MagicDraw 5
2.2.1 Create new action class 5
2.2.2 Specify action properties 6
2.2.3 Describe enabling/disabling logic 7
2.2.4 Configure Actions 8
2.2.5 Register configurator 11
2.3 Actions hierarchy 13
2.4 本节小结及补充 13
3 UML MODEL 15
3.1 Project 15
3.2 Changing UML Model 16
3.2.1 SessionManager 16
3.2.2 ModelElementsManager 16
3.2.3 Create new model element 17
3.2.4 Editing model element 17
3.2.5 Adding new model element or moving it to another parent 18
3.2.6 Removing model element 18
3.2.7 Creating Diagram 19
3.2.8 Creating new Relationship object 20
3.3 Working with Stereotypes and Tagged Values 20
3.4 本节小结及补充 22
4 PRESENTATION ELEMENTS 23
4.1 Presentation Element 23
4.2 Presentations Elements Manager 25
4.2.1 Creating shape element 25
4.2.2 Creating path element 25
4.2.3 Reshaping shape element 25
4.2.4 Changing path break points 26
4.2.5 Deleting presentation element 26
4.2.6 Changing properties of presentation element 26
4.3 本节小结及补充 27
5 NEW DIAGRAM TYPES 28
5.1 Diagram Types Hierarchy 28
5.2 Adding a new diagram type for MagicDraw 28
5.2.1 Override abstract DiagramDescriptor class 28

5.2.2 Register new diagram type in the application 31
5.3 本节小结及补充 32
6 MagicDraw 插件开发总结 33
Function Block 开发文档 34
1 Function Block 开发目的 34
2 Function Block 开发内容 35
3 Function Block 文件结构 36
4 Function Block 的设计 37
4.1 插件主类与辅助类设计 37
4.1.1 插件主类 FunctionBlockPlugin 37
4.1.2 辅助类 FunctionBlockHelper 37
4.1.3 常数类 FunctionBlockConstants 37
4.2 自定义 Diagram 设计 38
4.2.1 Function Block Diagram 设计 38
4.2.2 ECC Diagram 设计 39
4.3 Actions 设计 40
4.3.1 DrawShapeDiagramAction 40
4.3.1.1 DataInPortAction, DataOutPortAction, EventInPortAction, EventOutPortAction 40
4.3.1.2 ECCInitialStateAction， ECCStateAction 41
4.3.1.3 BasicFunctionBlockAction， CompositeFunctionBlockAction 41
4.3.1.4 AlgorithmAction 43
4.3.2 DrawPathDiagramAction 44
4.3.2.1 ECCTransitionAction 44
4.4 Renders 设计 44
4.4.1 ShapeDecorator 45
4.4.1.1 FunctionBlockRender 46
4.4.1.2 PortRender 47
4.4.1.3 InitialStateRender 48
4.4.1.4 AlgorithmRender 49
4.4.1.5 EventOutputRender 49
4.4.2 PathDecorator 52
4.4.2.1 DependencyRender 52
5 Function Block 范例 52

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。

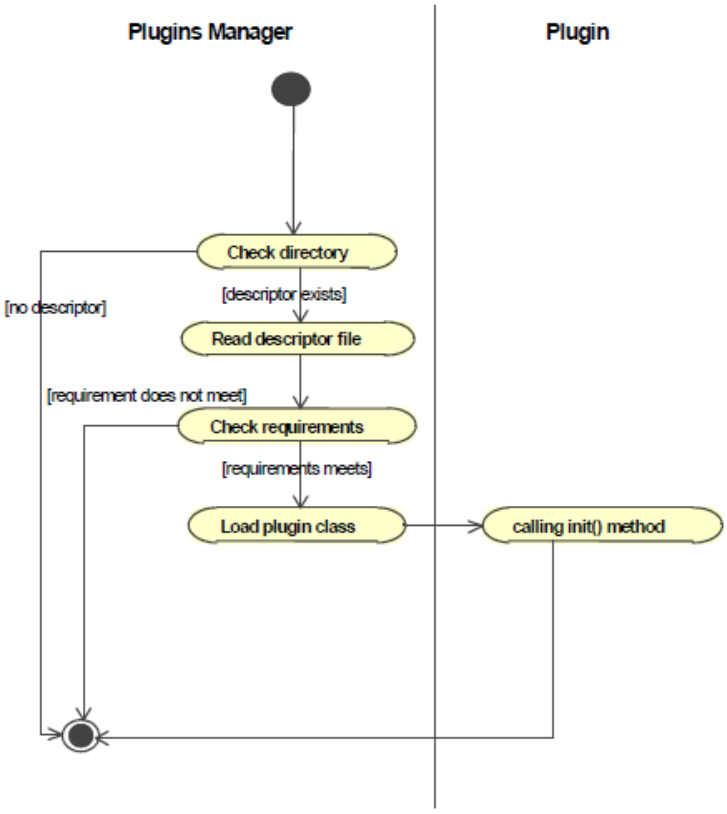

图 1-1 Plug-in 的载入原理图

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：

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
        id="my.first.plugin"
        name="My First Plugin"
        version="1.0"
        provider-name="Coder"
        class="myplugin.MyPlugin">

        <requires>
           <api version="1.0"/>
        </requires>

        <runtime>
           <library name="myplugin.jar"/>
        </runtime>
</plugin>
```

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：

Main menu

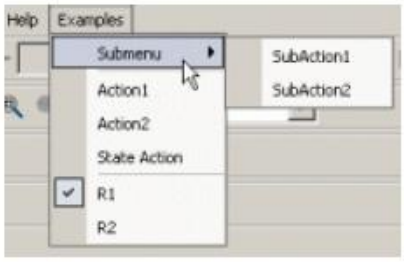

图 1-3 Main menu

Main toolbar

图 1-4 Main toolbar

Diagram toolbar

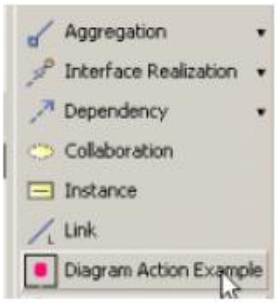

图 1-5 Diagram toolbar

Browser context menu

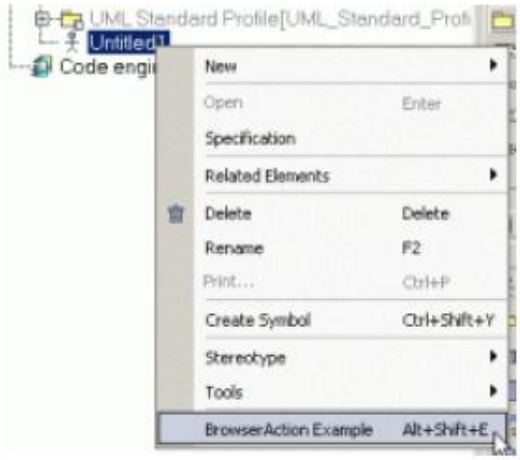

图 1-6 Browser context menu

Diagram context menu

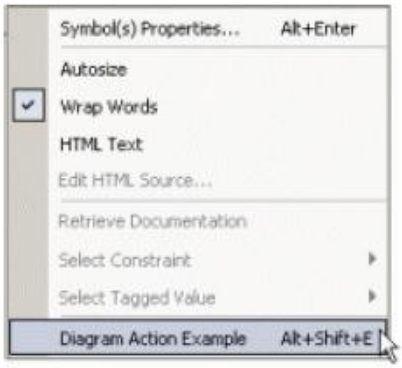

图 1-7 Diagram context menu

Keyboard shortcuts

Action can be not represented in GUI. Create a new action and assign some keyboard shortcut for invoking it.

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().getDialogPare
    nt(), "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。

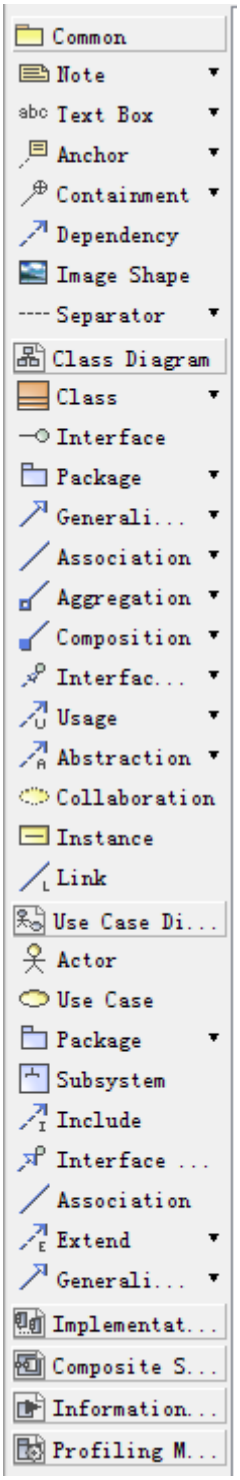

图 2-1 All Toolbars

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。

	ActionsManager	Category	Action
Menu	Menu bar	Menu	Menu item
Toolbar	All toolbars	One toolbar	Button
Context Menu	Context menu	Submenu	Menu item

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(configurato
r);
```

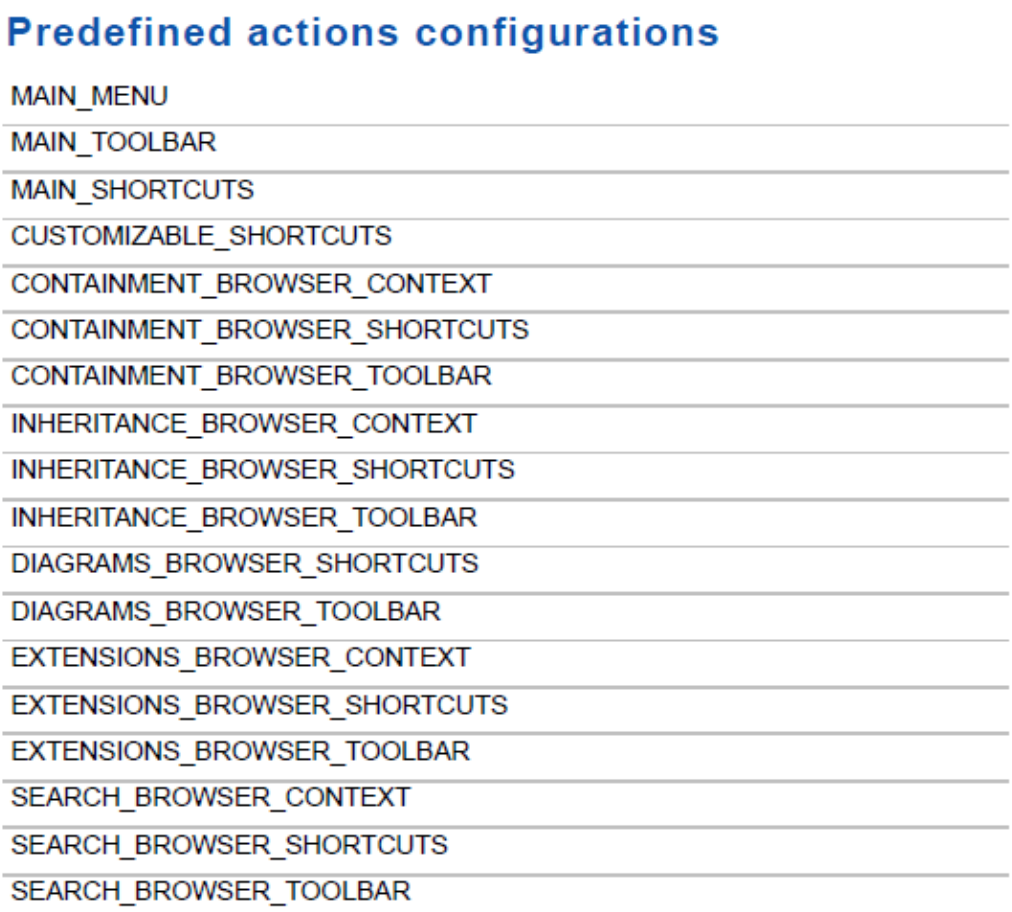

图 2-3 预先定义的 Configurations

2.3 Actions hierarchy

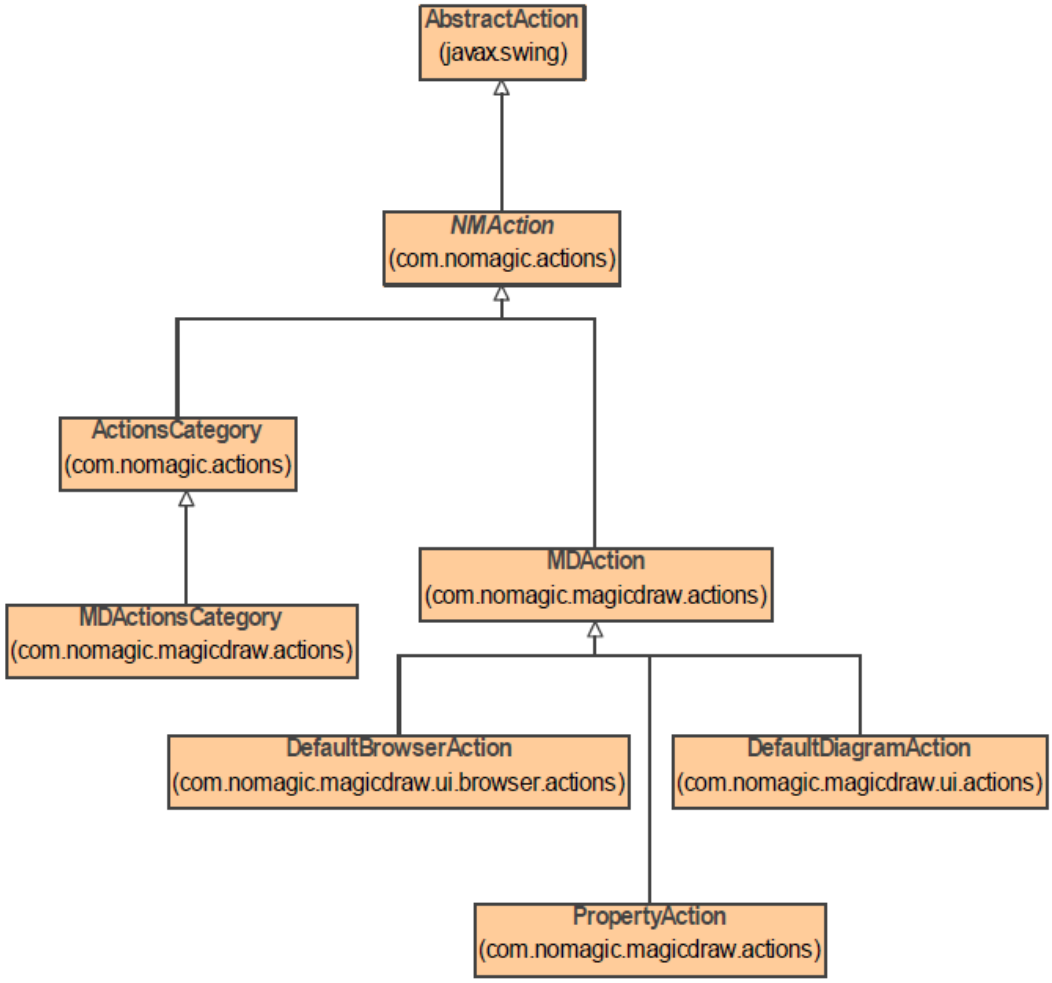

图 2-4 Actions hierarchy

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

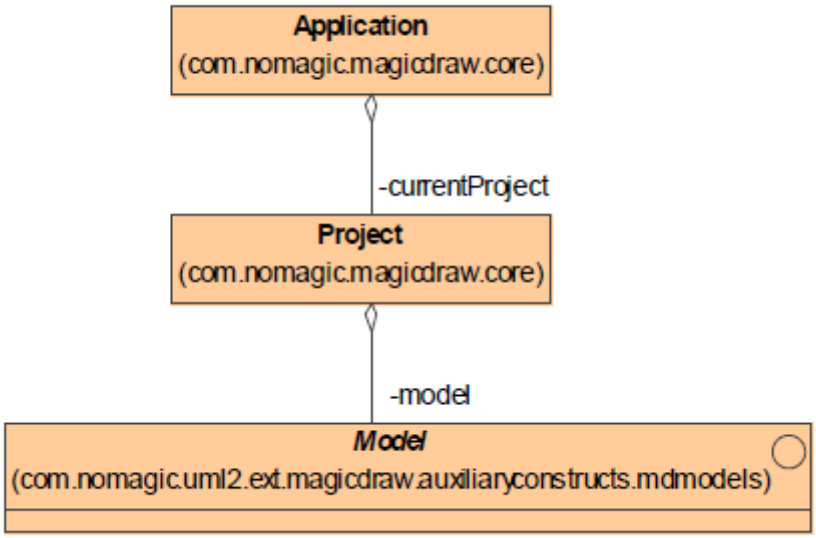

图 3-1 MagicDraw 的层次结构

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。

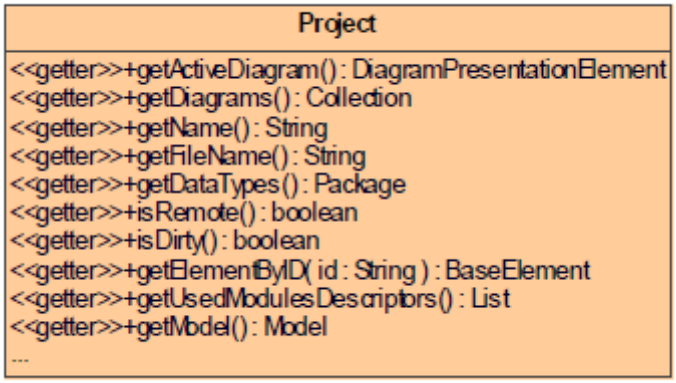

图 3-2 Project 的部分方法

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。

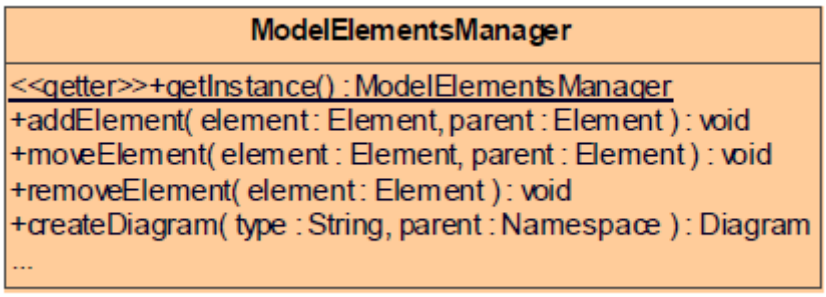

图 3-3 ModelElementsManager 部分方法

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_DIAGR
AM, 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。

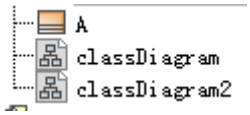

图 4-1 Element 示例

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

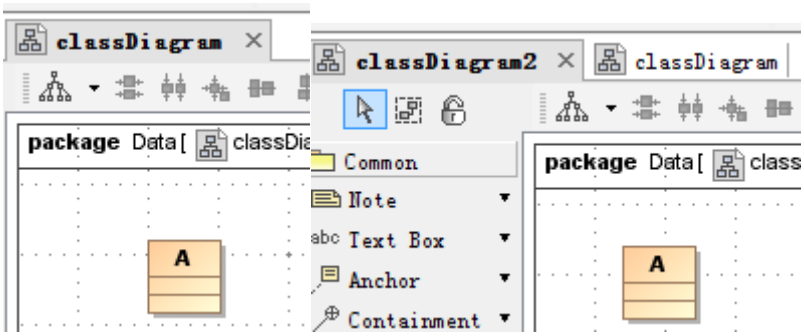

图 4-2 Presentation Element 示例

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。

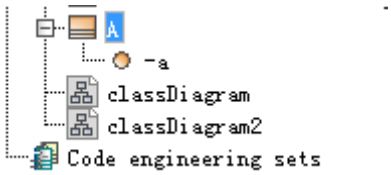

图 4-3 Class A 添加属性 a

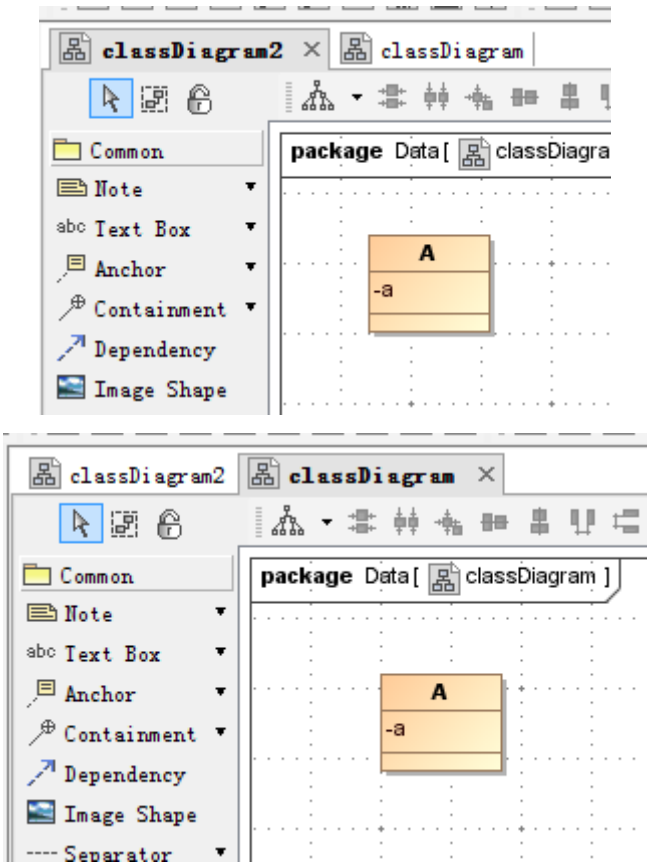

图 4-4 Presentation Element 的变化

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

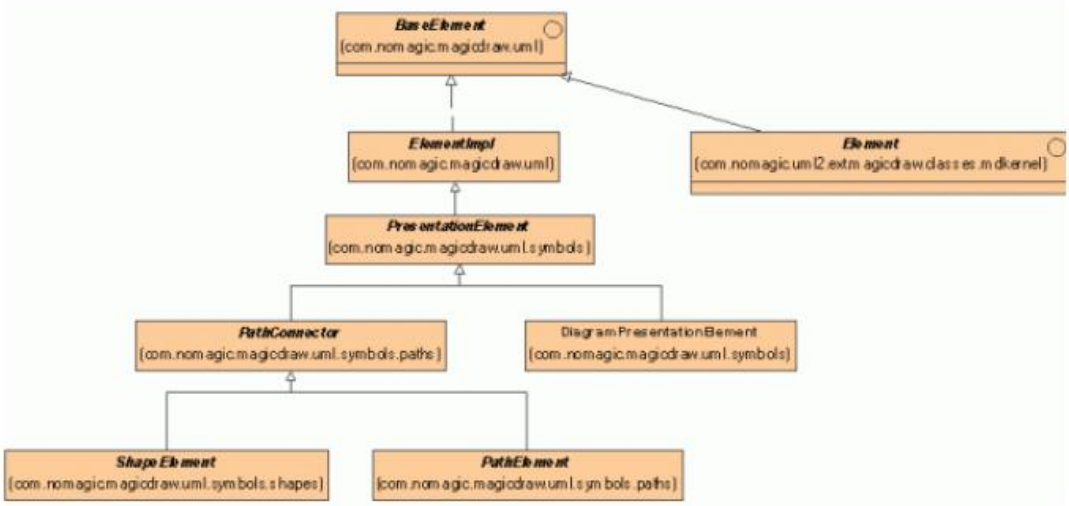

图 4-5 PresentationElement 的继承结构

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

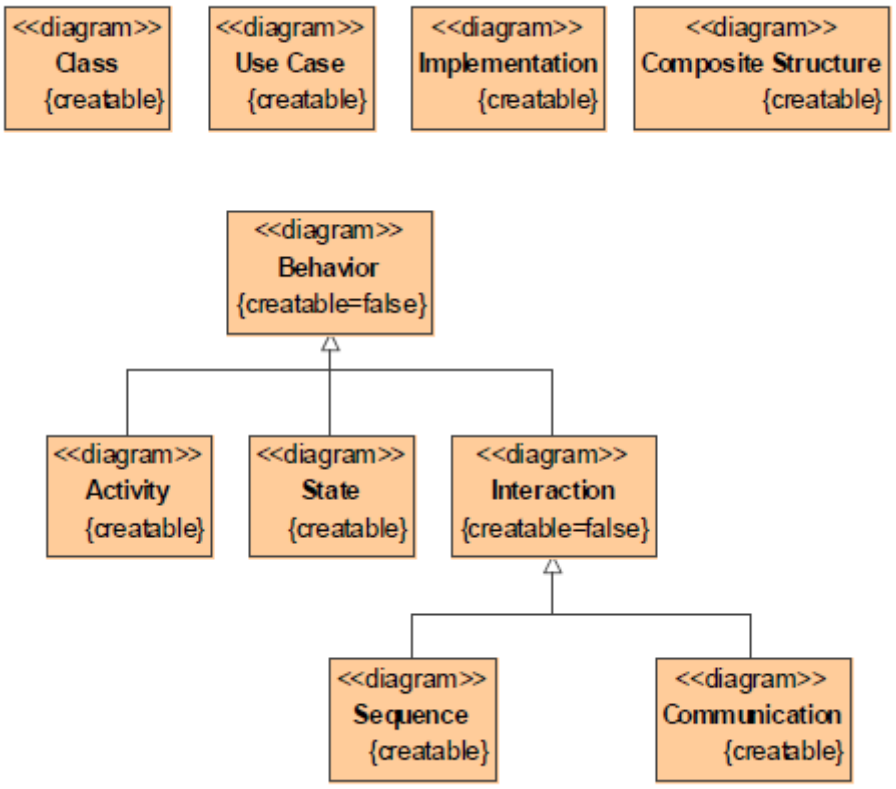

图 5-1 Diagram 继承结构

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

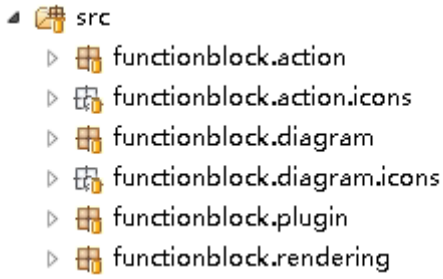

图 3-1 Function Block 文件结构

/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。

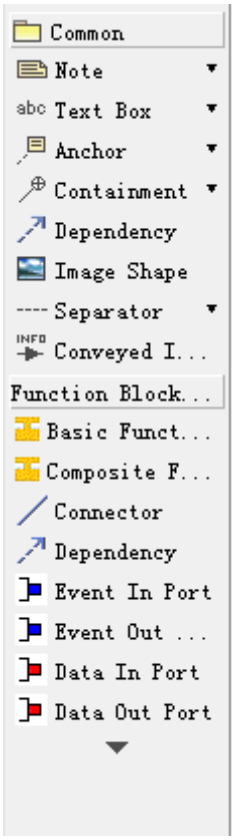

图 4-1 Function Block Diagram 的 Toolbar

4.2.2 ECC Diagram 设计

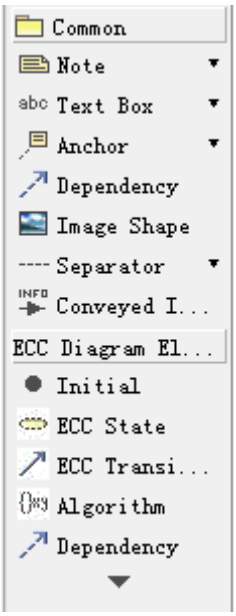

图 4-2 ECC Diagram 的 Toolbar

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

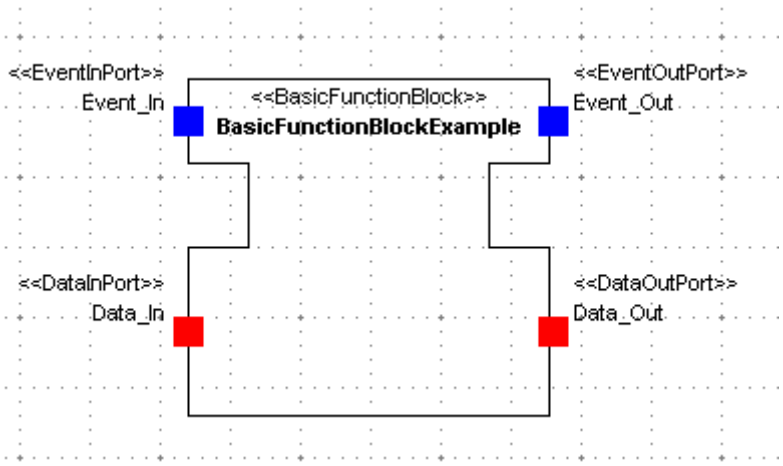

图 4-3 Basic Function Block 与四个 Port

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

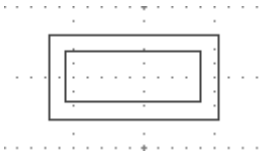

图 4-4 ECC 中的 Initial State

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

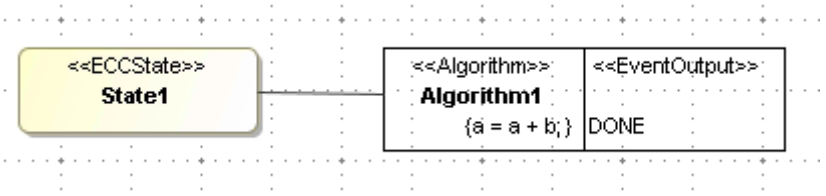

图 4-5 ECC 中的 State 以及它的 Action

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

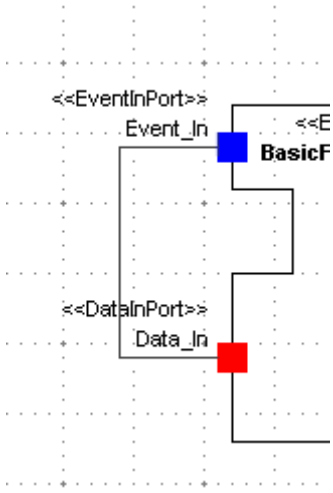

图 4-6 Data Port 和 Event Port 的依赖关系

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；

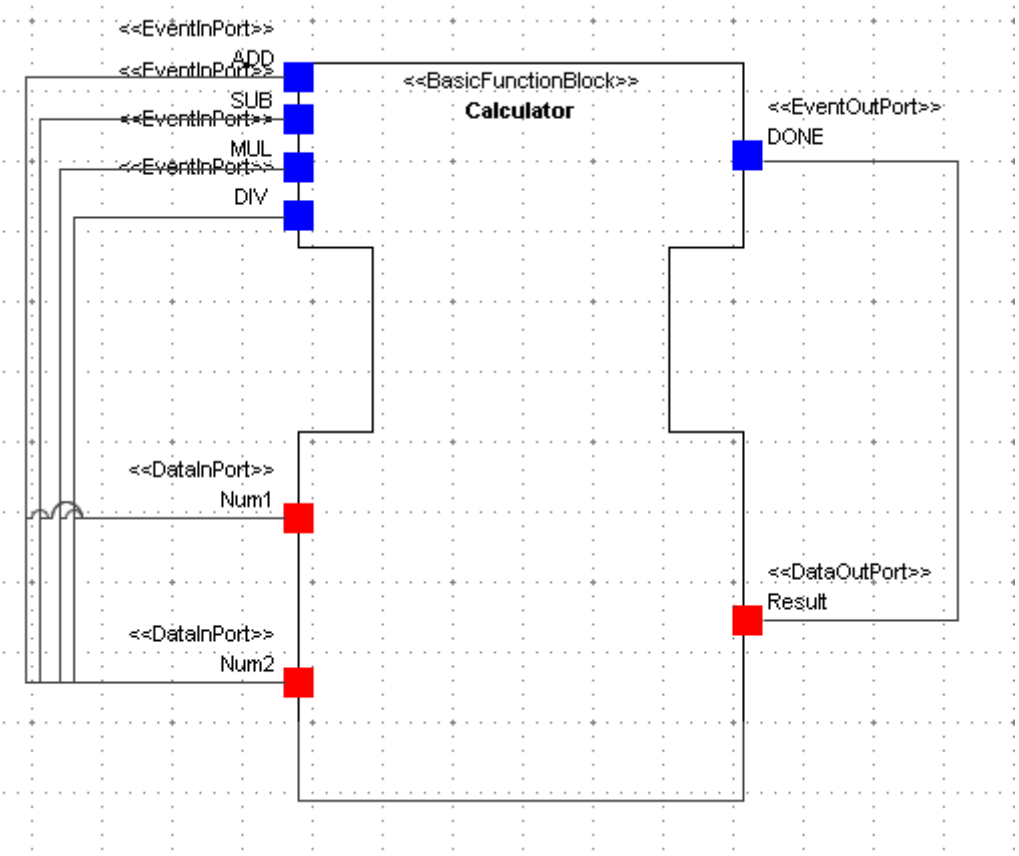

图 5-1 Calculator 的外部结构

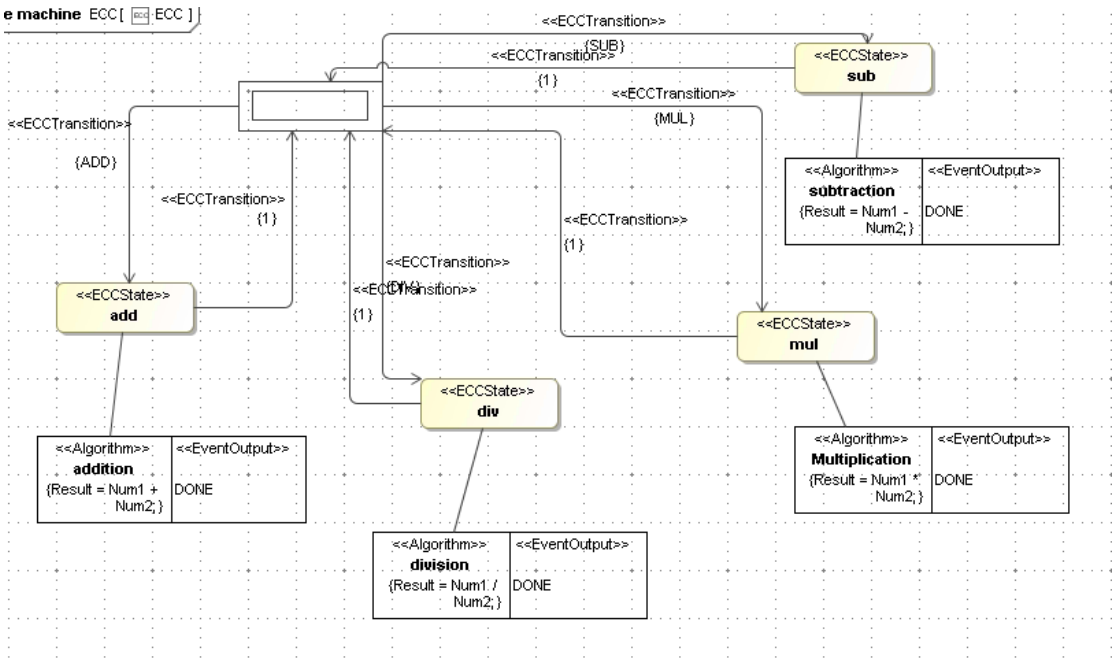

图 5-2 Calculator 的内部 ECC 构造

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。