S201407016 敖菁

S201407075 刘倚天

S201425004 郑玉

多层数据流图可视化编辑系统

目录

[1 使用说明书 2](#_Toc407229885)

[1.1 系统概述 2](#_Toc407229886)

[1.2 界面说明 2](#_Toc407229887)

[1.3 功能及使用方法介绍 3](#_Toc407229888)

[1.3.1 创建图元 3](#_Toc407229889)

[1.3.2 连接图元 3](#_Toc407229890)

[1.3.3 释放图元连接 4](#_Toc407229891)

[1.3.4 移动图元 4](#_Toc407229892)

[1.3.5 编辑图元名称 4](#_Toc407229893)

[1.3.6 删除图元 4](#_Toc407229894)

[1.3.7 进入子图编辑 5](#_Toc407229895)

[1.3.8 结束子图编辑 5](#_Toc407229896)

[1.3.9 查找路径 5](#_Toc407229897)

[1.3.10 退出查找路径 6](#_Toc407229898)

[2 系统概述 7](#_Toc407229899)

[2.1 解决方案及设计思想 7](#_Toc407229900)

[2.1.1 实现方式概述 7](#_Toc407229901)

[2.1.2 类设计思想概述 7](#_Toc407229902)

[2.2 系统框架概述 8](#_Toc407229903)

[2.2.1 窗口主框架MainFrame与图形管理类（器）DiagramEditor 8](#_Toc407229904)

[2.2.2 文档视图与图形间的关联方式 8](#_Toc407229905)

[2.2.3 系统总体操作状态转换 9](#_Toc407229906)

[2.2.4 工厂模式的使用 10](#_Toc407229907)

[3 案例分析 11](#_Toc407229908)

[3.1 新建图形 11](#_Toc407229909)

[3.1.1 案例描述 11](#_Toc407229910)

[3.1.2 案例文档 12](#_Toc407229911)

[3.2 编辑图形 12](#_Toc407229912)

[3.2.1 案例描述 13](#_Toc407229913)

[3.2.2 案例文档 13](#_Toc407229914)

[3.3 查找路径 15](#_Toc407229915)

[3.3.1 案例描述 15](#_Toc407229916)

[3.3.2 案例文档 15](#_Toc407229917)

[4 静态模型 17](#_Toc407229918)

[4.1 图形类（Diagram） 18](#_Toc407229919)

[4.2 图形管理器类（DiagramEditor） 20](#_Toc407229920)

[4.3 工具类 22](#_Toc407229921)

[4.4 Tool 22](#_Toc407229922)

[4.5 CreateTool 23](#_Toc407229923)

[4.6 RectangleTool 24](#_Toc407229924)

[4.7 LineTool 24](#_Toc407229925)

[4.8 EllipseTool 24](#_Toc407229926)

[4.9 StreamTool 24](#_Toc407229927)

[4.10 EditTool 25](#_Toc407229928)

[4.11 Lookuptool 25](#_Toc407229929)

[4.12 图元类 26](#_Toc407229930)

[4.13 Element 27](#_Toc407229931)

[4.14 DFDDataStorage 27](#_Toc407229932)

[4.15 DFDProcess 28](#_Toc407229933)

[4.16 DFDSource 28](#_Toc407229934)

[4.17 Stream 28](#_Toc407229935)

[5 动态模型 30](#_Toc407229936)

[5.1 各操作的序列图模型 30](#_Toc407229937)

[5.1.1 图元创建过程序列图 30](#_Toc407229938)

[5.1.2 移动图元序列图 32](#_Toc407229939)

[5.1.3 流图元Stream伸缩过程 33](#_Toc407229940)

[5.1.4 图元改名过程 34](#_Toc407229941)

[5.1.5 删除图元过程 35](#_Toc407229942)

[5.1.6 创建/打开子图操作过程 37](#_Toc407229943)

[5.1.7 退出子图返回上层图过程 38](#_Toc407229944)

[5.1.8 主图查找路径过程 39](#_Toc407229945)

[5.1.9 跨子图查找路径过程 40](#_Toc407229946)

[5.1.10 退出查找路径过程 41](#_Toc407229947)

# 使用说明书

## 系统概述

本系统实现了一个简单的多层数据流图可视化编辑系统，系统界面提供了新建图元和查找路径的菜单栏及图形编辑区域，用户可以通过点选菜单栏中的选项后在图形编辑区创建数据流图的基本元素，并通过在编辑区进行鼠标单击/双击/右击/拖动等操作编辑该数据流图，并可以根据自己的需求查找出所有到达指定源的路径。本系统提供创建数据流图中四种图形元素，移动图元，连接图元，编辑图元名称，删除指定图元，添加/编辑图元的子图，以及给定搜索终点搜索并显示出所有起点到达该终点的路径的功能，同时支持跨子图的路径高亮显示，用户可以通过本系统实现简单的多层数据流图的编辑。

## 界面说明

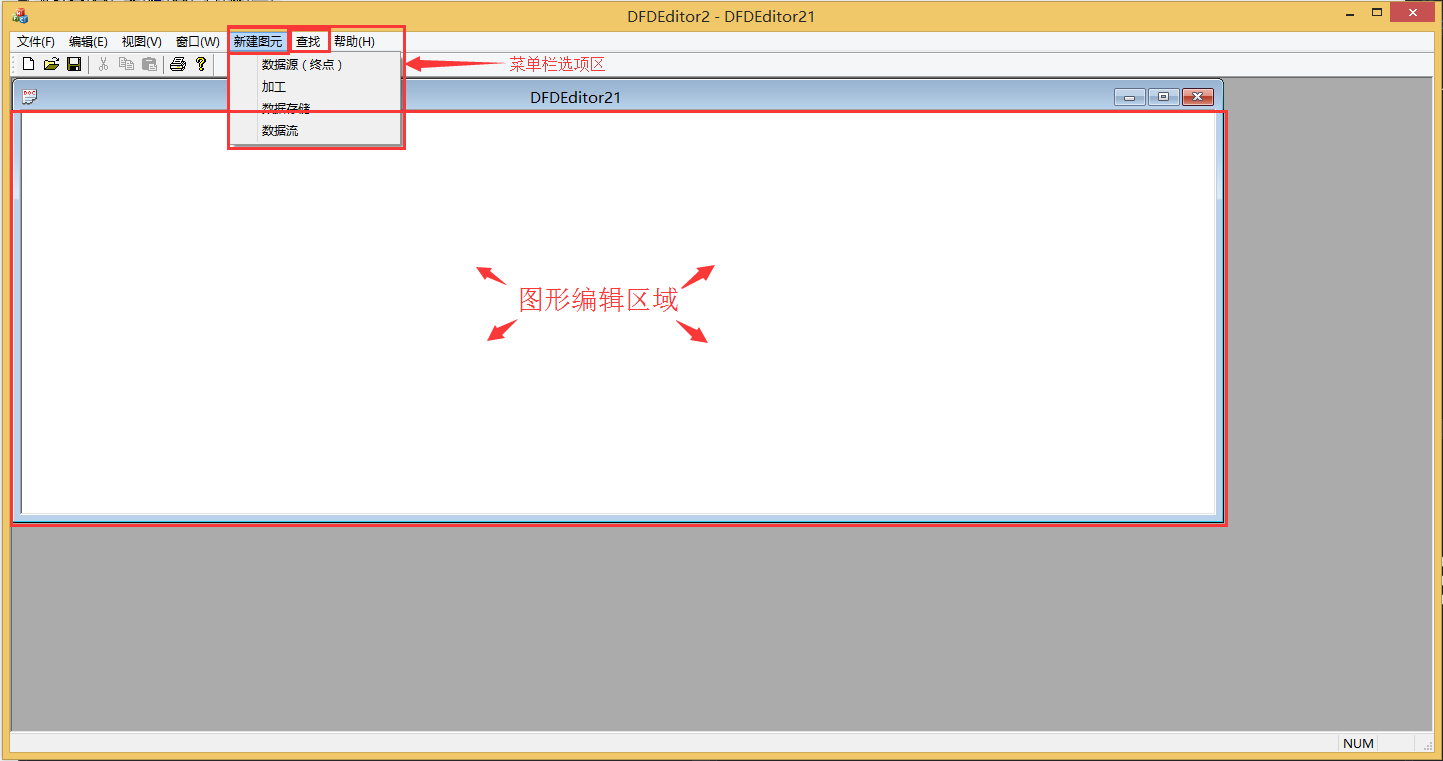


图 1‑1 可视化编辑系统分区

如图 1‑1所示，本系统包括两个区域：菜单栏选项区和图形编辑区域。

菜单栏选项区：提供两个菜单项，“新建图元”和“查找”，“新建图元”中包括四个菜单选项，数据源（终点）/加工/数据存储/数据流，为用户提供选择创建图元的类型的选项，”查找”菜单栏中包含查找路径项，为用户提供查找路径的功能。

图形编辑区域：为用户提供可视化界面，可在菜单栏选择相应操作后开始在图形编辑区域可以进行创建/编辑/查找数据流图的工作。

## 功能及使用方法介绍

### 创建图元

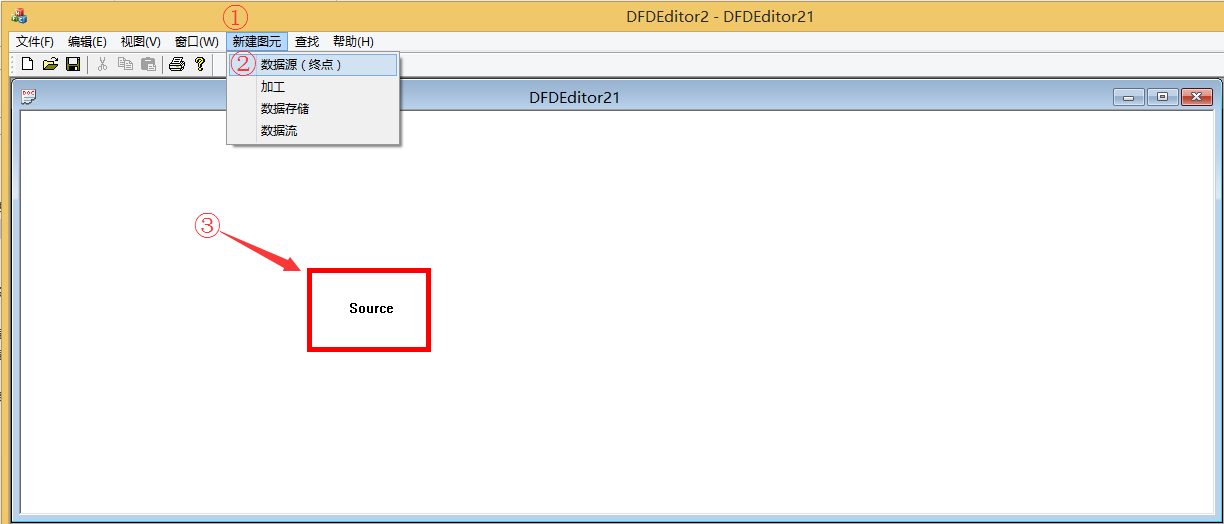


图 1‑2 创建图元过程

如图 1‑2所示，创建一个图元分为三个步骤，(1)点选菜单栏的“新建图元”菜单项(2)在下拉菜单中四个选项中选择要新建的图元类型(3)在图形编辑区域中左击鼠标选择任一点后将出现图元的轮廓线，移动鼠标，在确定区域释放鼠标左键，图元创建完成。

### 连接图元

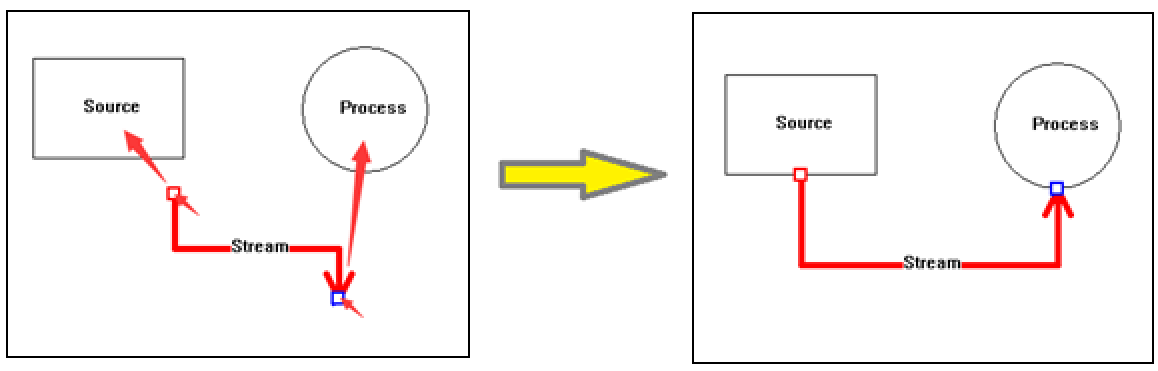


图 1‑3 连接图元过程

图 1‑3描述了图元连接的过程，图示仅为图形编辑区域，首先按下鼠标左键选中连接线的端点，移动鼠标，将端点拖入图元中，释放鼠标，连接线自动识别连接位置，将首端末端都设置好后两图元完成连接。

### 释放图元连接

图 1‑3实现了图元的连接，按下鼠标左键选中连接线的首端末端进行拖动离开连接图元所在区域即可释放图元的连接。

### 移动图元

首先，点击待移动的图元区域，图元变为红色高亮后移动鼠标即可实现图元的移动。

### 编辑图元名称

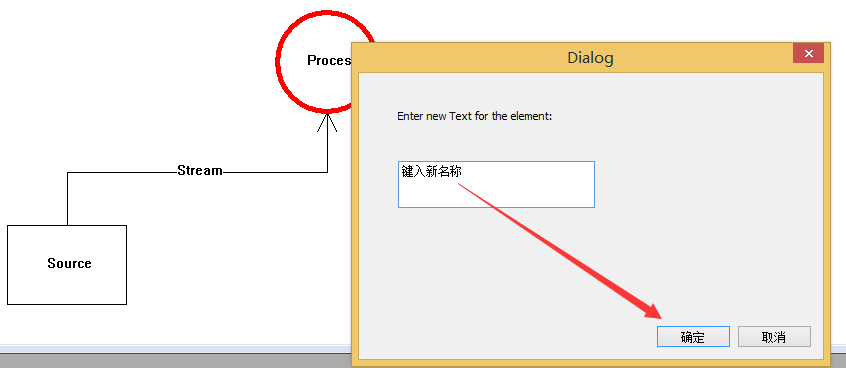


图 1‑4 编辑图元名称图示

右击该图元所在区域，图元会被高亮为红色并弹出改名的对话框，如图 1‑4所示，点击白色区域出现光标后输入新名称后点击确定按钮或按下键盘上的Enter回车键，图元名称改变。

### 删除图元

首先使用鼠标左键点击要删除的图元所在区域，待该图元变为红色高亮后，按下键盘上的Delete键，图元被删除。

### 进入子图编辑

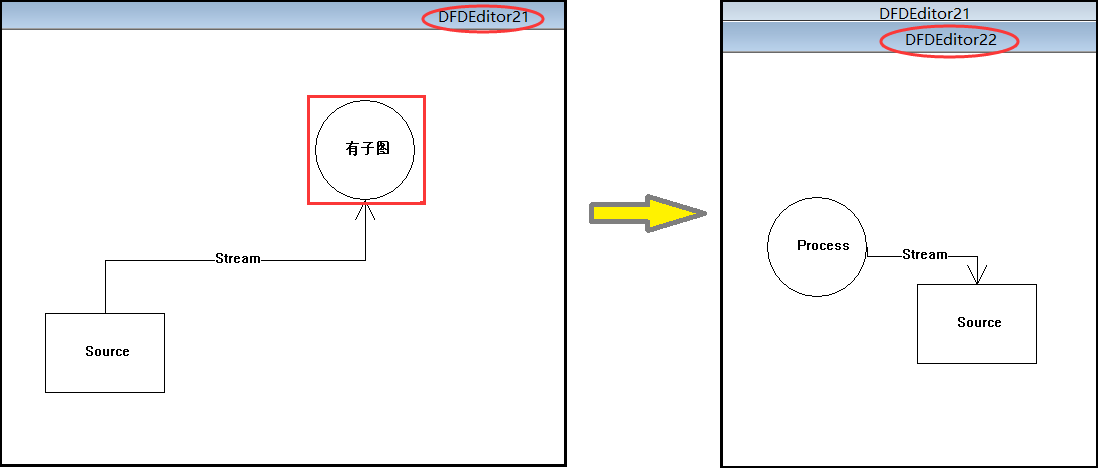


图 1‑5 进入子图编辑区进行编辑

如图 1‑5所示，因为在数据流图中，只有加工图元拥有子图，所以要进入子图编辑，首先要确定一个待有子图待编辑的加工图元，之后将鼠标移动到该加工图元所在区域，双击鼠标左键，系统会弹出子图编辑窗口，在窗口中进行创建/删除/连接图元等操作及可完成子图的创建。

### 结束子图编辑

在子图窗口按下键盘上的ESC键即可结束子图编辑回到上一层。

### 查找路径

#### 查找主图路径

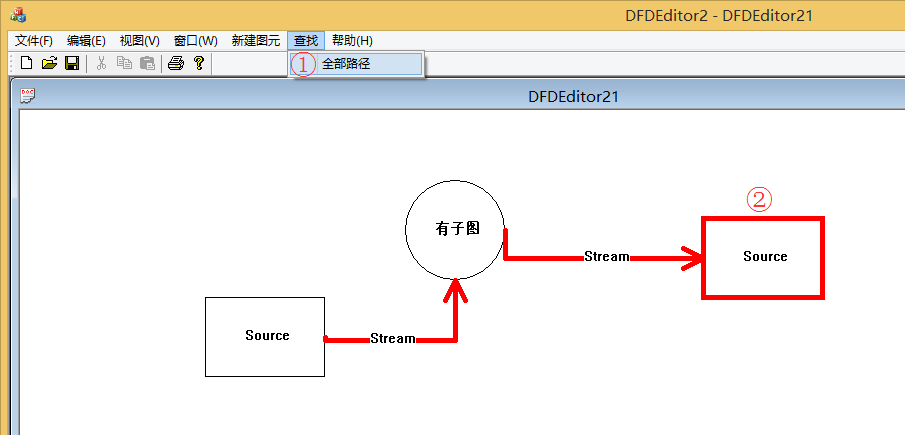


图 1‑6 查找主图路径

如图 1‑6所示，首先在菜单栏中选择查找->全部路径，之后选择查找路径的结束图元（只能是源类型的图元），查找出的路径会高亮在图中。

#### 跨子图查找路径

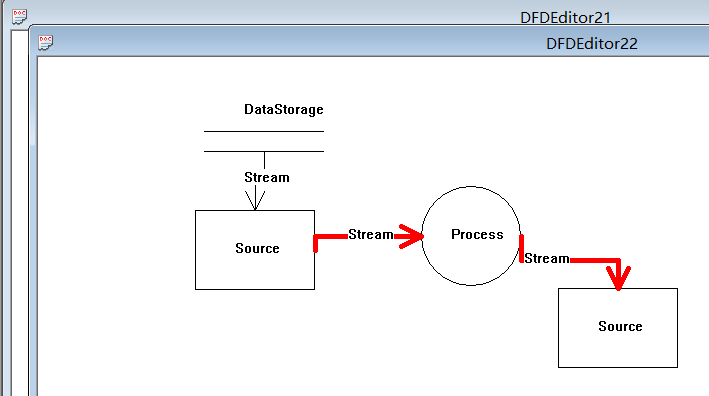


图 1‑7 子图显示查找路径

查找子图路径在查找主图路径的基础上的，只有主图中经过的有子图的加工类型的图元才能跨子图查找并显示，如图 1‑7所示，鼠标左键双击图 1‑6中有子图的加工图元，则打开该图元的子图并显示出查找路径。

### 退出查找路径

在图形编辑区域点击鼠标右键则退出查找路径，返回图形编辑功能。

# 系统概述

## 解决方案及设计思想

### 实现方式概述

本系统基于Microsoft提供的MFC框架开发，并采用多文档(MDI)结构实现。

多文档(MDI)结构是一种可实现具有多子窗口(ChildFrame)的系统的结构。在本系统中，多子窗口可以用来与“多层”的数据流图相对应，这为系统最终实现“一个窗口对应于一层的数据流图”的效果提供了基本的可能性和保障。所以，采用多文档(MDI)结构开发本系统，可以满足数据流图本身的“多层”需求。

其次，MFC框架提供的文档视图模型可以很好地满足“可视化”这一需求。在MFC中，文档类Document结构负责存储窗口对应图形的数据，包括图元的坐标、类型、名称等；视图类View负责根据文档类中的数据，绘制出相应的图形及其中包含的图元。且在多文档(MDI)结构中，每一个子窗口均持有一个文档成员和一个视图成员，及子窗口、文档、视图之间是一一对应的关系。这就可以确定，在本系统在开发过程中，使用MFC框架开发可以非常有针对性地实现“一个窗口对应于一层的数据流图”这一关键需求。

为了支持以上实现方式，本系统采用集成开发环境Microsoft Visual Studio 2013开发， C++语言编写，并在小组合作中使用软件Github进行版本控制。

### 类设计思想概述

一个好的多层数据流图可视化编辑系统应支持多张数据流图的创建、编辑、及管理等操作，而不是仅支持一张数据流图的以上操作。所以，本系统不仅需要将每张数据流图抽象为一个图形类Diagram，并且需要一个图形管理类（器）以管理这些图形，即DiagramEditor。

再次，由于一张数据流图由多个图元组成，每种图元有共同的特点，如图元坐标、名称等，也同时都具有各自不同的特点，如形状、操作等，互不相同。这就使得系统在开发时必须考虑到这一点，并为每种图元设计自身对应的类，且这些类均继承自一个共同的基类——图元类Element，以便于区分它们的相同点及不同点，且能够统一管理。具体的类间关系设计及接口设计将在“静态模型”部分说明。

除此之外，考虑到多种图元整体数量较多的相同操作及不同操作，为减轻图形管理类（器）DiagramEditor响应用户操作的压力，分离图元对象的使用和创建，提高抽象层次，本系统在开发过程中采用了工厂(Factory)模式，创建了控制图元操作的工具类Tool。并且，针对不同图元的不同创建过程、编辑过程、查找过程，工具类Tool作为基类派生出了多个派生类Tool，如RectangleTool、StreamTool、EditTool及LookupTool等。这些派生类分别控制于数据流图图元源(Source)的创建操作、数据流图图元流(Stream)的创建操作、图元的编辑操作及数据流路径的查询操作等。具体的类间关系设计及接口设计将在“静态模型”部分说明。

## 系统框架概述

### 窗口主框架MainFrame与图形管理类（器）DiagramEditor

在MFC中，一个主窗口(MainFrame)可包含多个MDI子窗口(ChildFrame)。一个MDI子窗口对应于一个文档类(CDocument)及一个视图类(CView)。也就是说，一个子窗口对应一个图形，一个视图对象对应一个图形，一个文档对象对应一个图形。作为图形管理类（器），由于DiagramEditor需要支持对多个图形的管理操作，所以它必须能够像一个主窗口管理多个MDI子窗口一样管理多个图形。由此我们想到，DiagramEditor与MainFrame的层次及特点都是类似的。所以，在本系统中，我们将整个系统层次最高的图形管理类（器）DiagramEditor的对象，作为主窗口MainFrame的一个成员。通过MainFrame与DiagramEditor的一对一关联，我们将MFC框架与本系统联系起来。

### 文档视图与图形间的关联方式

在本系统中，为了使主窗口下的各MDI子窗口在每次刷新（包括用户手动点击标题栏刷新、最小化最大化刷新、拖动子窗口时各子窗口的刷新等）的时候能够正确地绘制出其自身对应的图形，我们在每个子窗口各自对应的文档类CDocument中存入了其自身对应的图形指针d(Diagram \*)。在每次子窗口刷新时，每个子窗口各自的视图类CView将通过GetDocument()方法，得到相应文档类CDocument中存储的对应图形，根据此图形进行绘图工作。一个概括性的绘图过程状态转换图如图 2‑1所示。



图 2‑1 绘图过程状态转换图

### 系统总体操作状态转换

在本系统中，对图形、图元的编辑操作会经历7个主要状态：初始状态、待编辑图元状态、选择图元类型状态、创建图元状态、选中图元状态、选择查找路径状态及显示查找路径状态。这7个主要状态覆盖了本系统提供的各种操作。状态间的转换操作是通过用户的鼠标操作完成的，例如，在待图元状态下，用户点选图元即进入选中图元状态等。详细的状态转换方式如**错误!未找到引用源。**所示。

|  |  |  |
| --- | --- | --- |
| **起始状态** | **状态转换操作** | **目标状态** |
| 初始状态 | 未选择查找路径 | 待编辑图元状态 |
| 初始状态 | 选择查找路径 | 选择查找路径状态 |
| 待编辑图元状态 | 选择图元类型 | 选择图元类型状态 |
| 待编辑图元状态 | 点选图元 | 选中图元状态 |
| 选择图元类型状态 | 输入位置信息 | 创建图元状态 |
| 创建图元状态 | 空 | 选中图元状态 |
| 选中图元状态 | 拖动/改名/选中流编辑点 | 选中图元状态 |
| 选中图元状态 | 选择图元类型 | 选择图元类型状态 |
| 选中图元状态 | 点击空白位置/删除图元 | 初始状态 |
| 选择查找路径状态 | 选中终点图元 | 选中图元状态 |
| 选中图元状态 | 查找路径 | 显示查找路径状态 |
| 显示查找路径状态 | 选中终点图元 | 选中图元状态 |
| 显示查找路径状态 | 点击空白/其他图元位置或点击右键 | 初始状态[[1]](#footnote-1) |

表 2‑1 系统总体操作状态转换表



图 2‑2 系统总体操作状态转换图

# 案例分析

在案例分析中，将系统功能划分为三个大的案例，新建图形、编辑图形、查找路径，下面将对每个案例通过案例图、案例描述、案例文档进行阐述。

## 新建图形

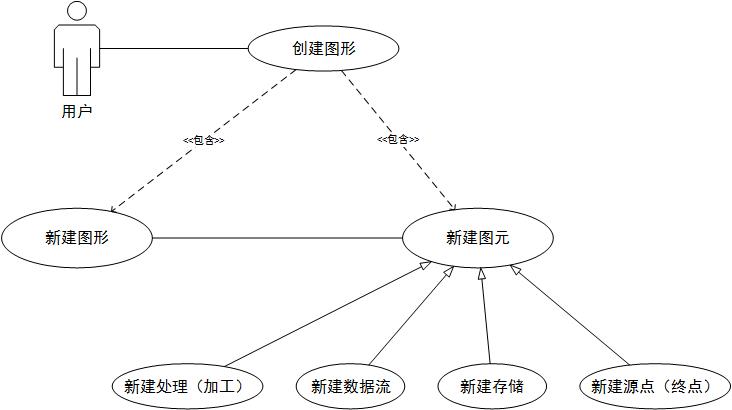


图 3‑1 新建图形案例图

### 案例描述

|  |  |
| --- | --- |
| 新建图形 | |
| 用户 | 编辑系统 |
| 1. 选“新建” 2.选“新建图元-数据流” 3.选“新建图元-处理（加工）” 4.选“新建图元-存储   5.选“新建图元-源点” | 1.创建空图 2.创建数据流直线 3.创建处理（加工）椭圆 4.创建存储双线  5.创建存储矩形 |

|  |  |
| --- | --- |
| 新建图元 | |
| 用户 | 编辑系统 |
| 1. 用户左键单击 2. 用户抬起鼠标左键 | 1. 记录图元位置并显示图元 2. 高亮当前图元 |

### 案例文档

|  |
| --- |
| 用例名称：新建图形 |
| 参与者：多层数据流图可视化编辑系统用户 |
| 简要说明：用户可在系统中创建新的图形，并在图形中添加各种新的图元 |
| 前置条件：用户打开可视化编辑系统 |
| 后置条件：进入等待下一个用户操作状态 |
| 基本事件流:   1. 用户点击新建图形，系统弹出空白的新窗口（一个窗口对应一个图形） 2. 用户点击新建处理（加工）（或其他图元），在空白窗口中鼠标单击选定一个位置，系统在该指定位置显示该图元 |
| 其他事件流：无 |
| 异常事件流：无 |
| 注释：无 |

## 编辑图形

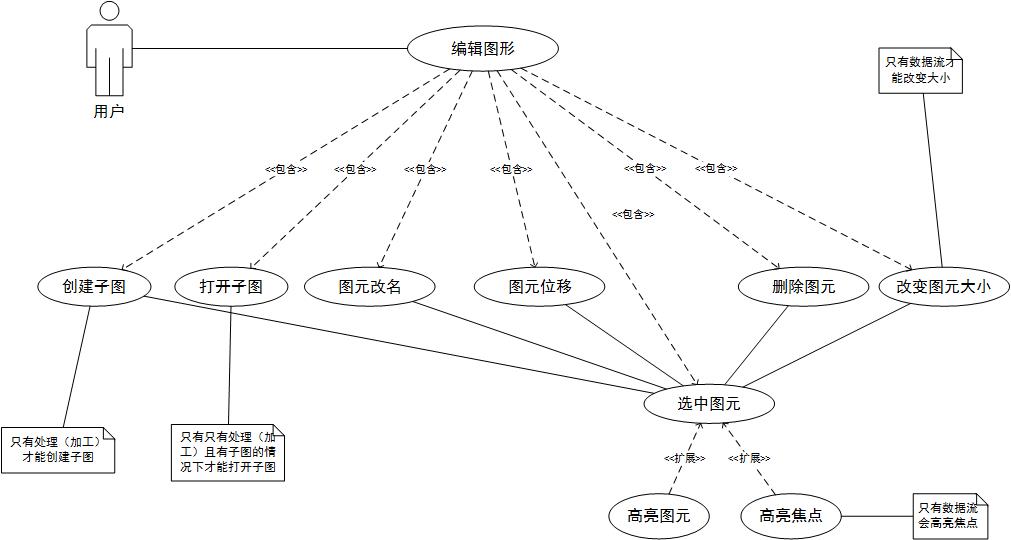


图 3‑2 编辑图形案例图

### 案例描述

|  |  |
| --- | --- |
| 编辑图形 | |
| 用户 | 编辑系统 |
| 1.选择图形  2.选择图元  3.移动图元  4.图元中单击右键  5.输入文本  6.按键盘ESC键 | 1.显示图形  2.高亮图元  3.修改图元位置  4.弹出对对话框  5.编辑文本  6.删除图元 |

|  |  |
| --- | --- |
| 编辑数据流 | |
| 用户 | 编辑系统 |
| 1.选择图形  2.选择数据流  3.移动数据流  4.图元中单击右键  5.输入文本  6点击数据流起点或终点  7.拖动数据流端点  8.按键盘ESC键 | 1.显示图形  2.高亮图元  3.修改图元位置  4.弹出对对话框  5.编辑文本  6.高亮数据流起点或终点  7.更改数据流长短和箭头指向  8.删除数据流 |

|  |  |
| --- | --- |
| 编辑处理（加工） | |
| 用户 | 编辑系统 |
| 1.选择图形  2.选择处理加工  3.移动处理加工  4.图元中单击右键  5.输入文本  6双击图元  7.按键盘ESC键 | 1.显示图形  2.高亮处理加工  3.修改图元位置  4.弹出对对话框  5.编辑文本  6.打开或新建子图  7.删除数据流 |

### 案例文档

|  |
| --- |
| 用例名称：编辑图形 |
| 参与者：多层数据流图可视化编辑系统用户 |
| 简要说明：用户可以对指定的图形进行编辑，对图形中的图元进行位移的改变、大小的改变、名称的改变，也可以对指定图元进行删除，也可以进入子图进行编辑。 |
| 前置条件：用户已选择某一个需要操作的窗口（即） |
| 后置条件：进入等待下一个用户操作状态 |
| 基本事件流:   1. 用户选择需要编辑的数据流图（即鼠标点击到某一窗口，当前活动窗口中的图形即为需要编辑的数据流图） 2. 用户单机选中需要编辑的图元（数据流、处理（加工）、存储、源点（终点）） 3. 系统高亮当前选中的图元 4. 用户拖动鼠标移动当前图元的位置 5. 用户右击图元，弹出更改名称对话框 6. 用户输入图元新名称，单击确定 7. 系统在图元旁边显示图元新名称 8. 用户敲击键盘ESC键，当前选中图元被删除 |
| 其他事件流:  用户选中的图元若为数据流   1. ，用户单击“数据流”的起点和终点 2. 系统高亮数据流的起点和终点 3. 用户拖动鼠标可以改变数据流图元的大小和长短   用户选中的图元若为处理（加工）   1. 用户双击“处理（加工）”图元的内部 2. 若该“处理（加工）”有子图则打开子图并显示，若无子图则新建空子图并显示 |
| 异常事件流：无 |
| 注释：无 |

## 查找路径

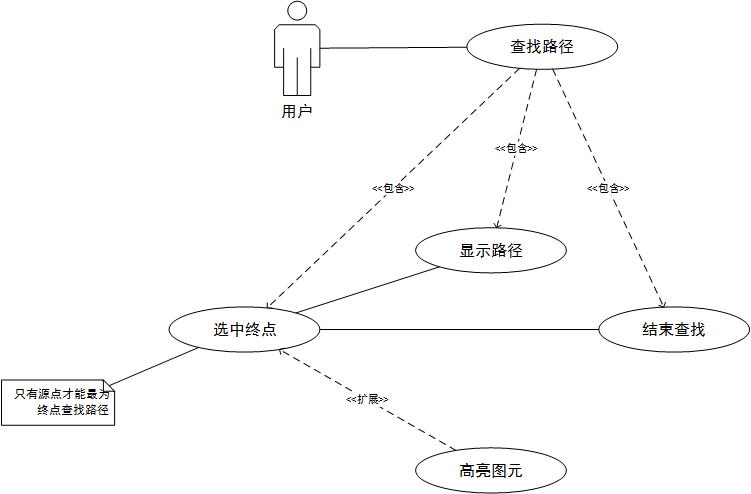


图 3‑3 查找路径案例图

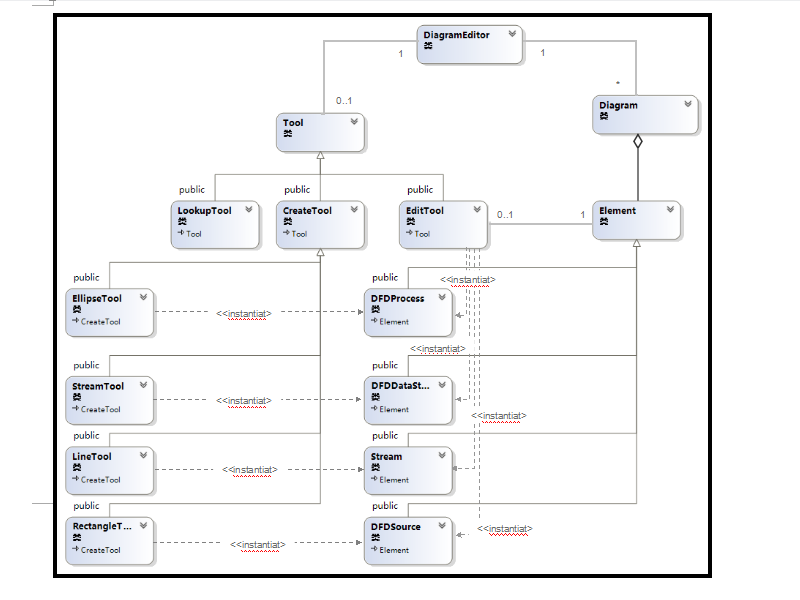
### 案例描述

|  |  |
| --- | --- |
| 查找路径 | |
| 用户 | 编辑系统 |
| 1.选择图形  2.左键单击选择源点  3.右键单击结束查找 | 1.显示图形  2.高亮到达该源点的路径  3.恢复高亮路径 |

### 案例文档

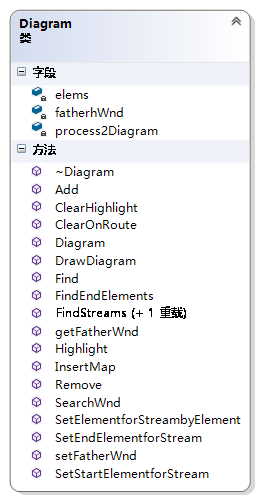
|  |
| --- |
| 用例名称：查找指定源的路径 |
| 参与者：多层数据流图可视化编辑系统用户 |
| 简要说明：用户指定某一终点，系统高亮显示来自所有起点到达该终点的路径 |
| 前置条件：鼠标指定的当前图形中，已经有各种以数据流相连接的图元 |
| 后置条件：进入等待下一个用户操作状态 |
| 基本事件流:   1. 用户点击查找路径菜单 2. 用户在指定终点内部单击鼠标左键 3. 系统高亮到达该终点的全部路径 4. 用户右击鼠标，结束路径查找 |
| 其他事件流:  1.在未结束查找时，可用鼠标左击其他图元内部，更改指定的终点 |
| 异常事件流：无 |
| 注释：无 |

# 静态模型



本系统中涉及的主要类及类之间的关系如上图所示，各类具体的说明将在下面的小节中详细说明

## 图形类（Diagram）



说明：

多层数据流图可视化编辑系统应支持多张数据流图的创建、编辑、及管理等操作，而不是仅支持一张数据流图的以上操作。所以，本系统将每张数据流图抽象为一个图形类Diagram

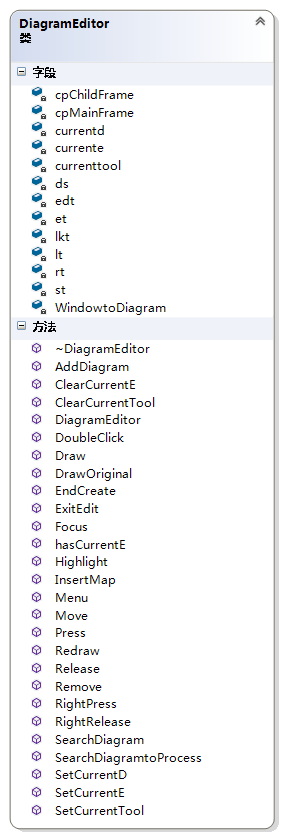
主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| fatherhWnd; | HWND | 父窗口句柄 |
| elems; | vector<Element\*> | 保存图形中的图元 |
| process2Diagram; | map<Element\*, HWND> | 保存过程图元和图形的映射 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Add | Element\* e | void | 添加图元 |
| Find | CPoint pos  Element \*&e | bool | 根据坐标查找图元 |
| Remove | Element \*e | void | 从图形中删除图元e |
| SetStartElementforStream | Stream \*se, CPoint pos | void | 设置起始图元 |
| DrawDiagram | vector<CPoint>&poss  vector<int>&types  vector<CString>&strs  CPoint startmidend[][3] | void | 画出图形中的全部图元 |
| InsertMap | Element \*e, HWND hWnd | void | 向Map中插入<Process,HWND>对 |
| FindStreams | Element\* currente  queue<Stream\*>& elemq | int | 根据结束图元查找所有以该图元为终点的路径 |
| Highlight | CDC \*pDC | void | 高亮路径 |
| ClearHighlight |  | void | 清除高亮 |
| FindEndElements | vector<Element\*>& endElements | void | 查找所有终点图元（只能是右边无连接的Source） |
| ClearOnRoute | queue<DFDProcess\*>& fathers | void | 清除当前图形中的Process标志位并返回 |

## 图形管理器类（DiagramEditor）



说明：图形管理类（器）以管理系统各种的所有图形，具体类的说明如下

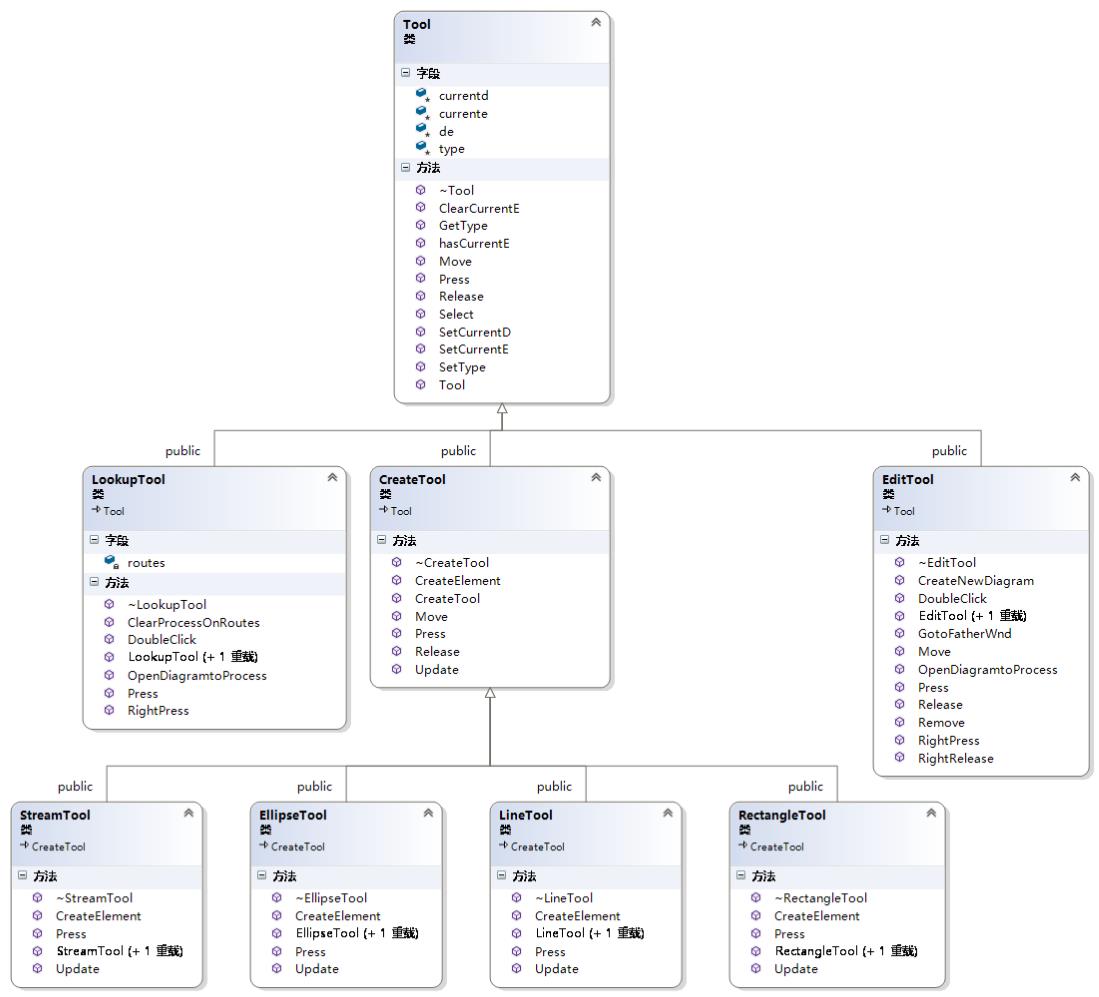
主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| cpMainFrame | CMainFrame \* | 指向父窗口 |
| cpChildFrame | CChildFrame \* | 指向子窗口 |
| currente | Element \* | 指向当前图元 |
| currentd | Diagram \* | 指向当前图形 |
| currenttool | Tool \* | 指向当前工具 |
| ds | vector<Diagram\*> | 保存所有图形 |
| WindowtoDiagram | map<HWND, Diagram\*> | 窗口句柄对应图形的映射 |
| r | RectangleTool \* | 矩形工具 |
| et | EllipseTool \* | 椭圆工具 |
| lt | LineTool \* | 线型工具类指针 |
| s | StreamTool \* | 连接线工具类指针 |
| edt | EditTool \* | 编辑图元工具类指针 |
| lkt | LookupTool \* | 查找路径工具类指针 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| hasCurrentE | void | bool | 返回是否有选中的图元 |
| InsertMap | HWND hWnd, Diagram \*d | void | 创建窗口句柄与图形映射 |
| Menu | int item | void | 菜单选项处理（创建何种类型图元） |
| SetCurrentE | Element \*e | void | 设置当前图元 |
| SetCurrentD | Diagram \*d | void | 设置当前图形 |
| SetCurrentTool | Tool \*t | void | 设置当前工具 |
| AddDiagram | void | void | 新建图形 |
| SearchDiagram | HWND hWnd, Diagram \*&d | void | 根据句柄查找对应图形并返回 |
| Press | CPoint pos | void | 鼠标左键按下 |
| DoubleClick | CPoint pos | void | 鼠标左键双击 |
| Release | CPoint pos | void | 鼠标左键释放 |
| RightPress | CPoint pos | void | 鼠标右键按下 |
| RightRelease | CPoint pos | void | 鼠标右键释放 |
| Remove |  | void | 删除图元操作 |
| Draw | Element \*doce  Diagram \*docd bool original  CPoint pos  CDC \*pDC | void | 画图元 |
| Redraw | bool original | void | 刷新重画 |
| Highlight(); | CDC \*pDC | void | 高亮图元 |
| Focus(); | CDC \*pDC | void | 高亮控制点 |
| EndCreate() | Element \*e | void | 结束图形创建 |
| SearchDiagramtoProcess | Diagram \*&d, DFDProcess \*p | HWND | 返回加工图元对应的窗口句柄 |

## 工具类



说明：

多种图元整体数量较多的相同操作及不同操作，为减轻图形管理类（器）DiagramEditor响应用户操作的压力，分离图元对象的使用和创建，提高抽象层次，创建了控制图元操作的工具类Tool。并且，针对不同图元的不同创建过程、编辑过程、查找过程，工具类Tool作为基类派生出了多个派生类Tool，如RectangleTool、StreamTool、EditTool及LookupTool等。这些派生类分别控制于数据流图图元源(Source)的创建操作、数据流图图元流(Stream)的创建操作、图元的编辑操作及数据流路径的查询操作等。具体的类的说明如下所示

## Tool

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| currentd | Diagram \* | 指向当前图形指针 |
| currente | Element \* | 指向当前图元指针 |
| de | DiagramEditor \* | 图形编辑器指针 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| hasCurrentE | Void | bool | 检查当前图元指针是否为空，若不为空，则已指向当前图元，返回真 |
| SetCurrentE | Element \*e | void | 设置当前图元指针为指定图元 |
| SetCurrentD | Diagram \*d | void | 设置当前图形指针为指定图形 |
| ClearCurrentE | Void | void | 清除当前图元指针为空 |
| Press | CPoint pos,  HWND hWnd | void | 鼠标按下时进行的操作，在子类中重载此方法 |
| Move | CPoint pos, CPoint oldpos | void | 鼠标移动时进行的操作，在子类重载次方法 |
| Release | CPoint pos | void | 鼠标释放时进行的操作，在子类中重载次方法 |
| Select | Diagram \*&d, HWND hWnd | void | 获取当前窗口中的当前图形 |
| SetType | ToolType type | void | 设置工具类型  CREATTOOL = 1,  EDITTOOL = 2,  LOOKUPTOOL = 3 |
| GetType | Void | ToolType | 返回工具类型 |

## CreateTool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos  HWND hWnd | void | 鼠标按下时进行的操作，由于不同图元的具体实现会有差异，在子类中重载此方法 |
| Move | CPoint pos CPoint oldpos | void | 鼠标移动时进行的操作，不断更新鼠标坐标 |
| Release | CPoint pos | void | 鼠标释放时创建相应图元，并设置此图元为当前图元，将其加入到当前图形中 |
| CreateElement | CPoint pos Element \*&e | void | 创建新图元，子类中具体实现有差异，将会重载次方法 |
| Update | CPoint pos | Void | 子类中实现此方法，刷新 |

## RectangleTool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos  HWND hWnd | void | 鼠标按下时获取当前窗口的图形刷新 |
| CreateElement | CPoint pos Element \*&e | void | 设置活动窗口中的图形为当前图形，在鼠标点击创建新的矩形数据源图元，名称为Source |
| Update | CPoint pos | Void | 刷新 |

## LineTool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos  HWND hWnd | void | 鼠标按下时获取当前窗口的图形刷新 |
| CreateElement | CPoint pos Element \*&e | void | 设置活动窗口中的图形为当前图形，在鼠标点击创建新的双线形数据存储图元，名称为storage |
| Update | CPoint pos | Void | 刷新 |

## EllipseTool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos  HWND hWnd | void | 鼠标按下时获取当前窗口的图形刷新 |
| CreateElement | CPoint pos Element \*&e | void | 设置活动窗口中的图形为当前图形，在鼠标点击创建新的椭圆形处理（加工）图元，名称为process |
| Update | CPoint pos | Void | 刷新 |

## StreamTool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos  HWND hWnd | void | 鼠标按下时获取当前窗口的图形刷新 |
| CreateElement | CPoint pos Element \*&e | void | 设置活动窗口中的图形为当前图形，在鼠标点击创建新的单线形的数据流图元，名称为stream |
| Update | CPoint pos | Void | 刷新 |

## EditTool

主要操作

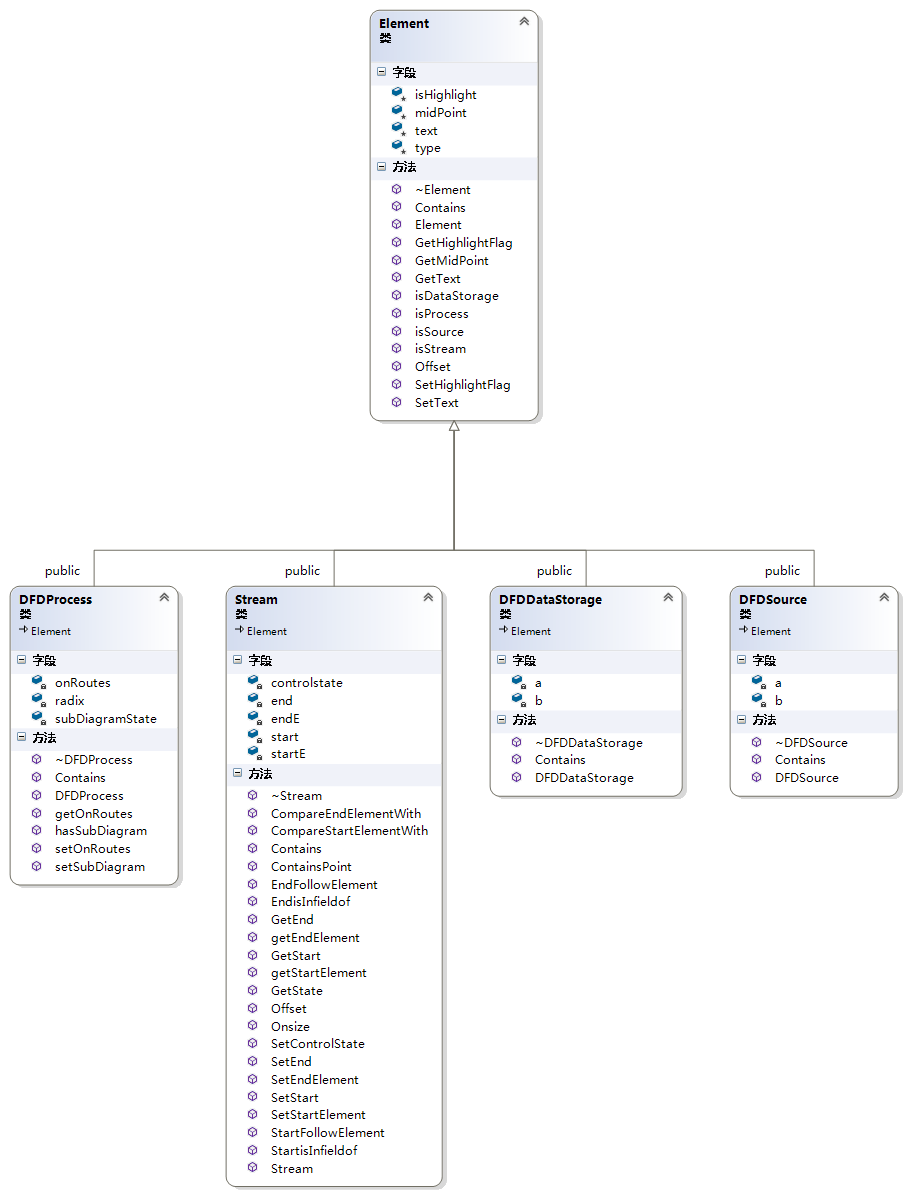
|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos, HWND hWnd | void | 单击鼠标左键，判断鼠标点在某一个图元中 |
| RightPress | CPoint pos, HWND hWnd | void | 单击鼠标右键进行图元改名操作 |
| DoubleClick | CPoint pos, HWND hWnd | void | 双击鼠标进入process的子图编辑 |
| Move | CPoint pos, CPoint oldpos | void | 鼠标移动更改图元位置 |
| Release | CPoint pos | void | 鼠标释放结束图元编辑 |
| RightRelease | CPoint pos | void | 右键释放弹出图元改名窗口 |
| GotoFatherWnd | Void | void | 打开父窗口 |
| Remove | Void | void | 删除图元 |
| CreateNewDiagram | Void | void | 创建新图形 |
| OpenDiagramtoProcess | HWND hWnd, Diagram \*d | void | 打开proscess的子图 |

## Lookuptool

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Press | CPoint pos, HWND hWnd | void | 点击鼠标左键选中终点 |
| RightPress | void | void | 结束查找路径 |
| DoubleClick | CPoint pos, HWND hWnd | void | 触发打开process子图操作 |
| OpenDiagramtoProcess | HWND hWnd, Diagram \*d | void | 打开process子图 |
| ClearProcessOnRoutes | queue<DFDProcess\*>& fathers | void | 清除打开process队列 |

## 图元类



说明：

一张数据流图由多个图元组成，每种图元有共同的特点，如图元坐标、名称等，也同时都具有各自不同的特点，如形状、操作等，互不相同。每种图元设计自身对应的类，且这些类均继承自一个共同的基类——图元类Element，以便于区分它们的相同点及不同点，且能够统一管理。Element类及不同图元的具体类说明如下所示

## Element

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| isHighlight; | bool | 是否高亮标志 |
| type | int | 图元类别标志 |
| midPoint | CPoint | 图形中间点 |
| text | CString | 图元名称 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| GetMidPoint | Void | CPoint | 获取图元中点 |
| GetText | Void | CString | 获取图元文本 |
| SetText | CString s | void | 设置图元文本 |
| SetHighlightFlag | bool isHighlight | void | 设置高亮标志 |
| GetHighlightFlag |  | bool | 获取高亮标志 |
| Contains | CPoint pos | bool | 判定传入点是否在图元内部，不同图元具体实现不同 |
| isProcess | Void | bool | 判断是否是加工（有子图） |
| isStream | Void | bool | 判断是否是流（连接线） |
| isSource | Void | bool | 判断是否是源（查找子图的终点） |
| isDataStorage | Void | bool | 判断是否是存储 |
| Offset | CPoint pos, CPoint oldpos | Void | 移动图元位置，不同图元实现不同 |

## DFDDataStorage

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| a | int | 图形中间点到左边距离 |
| b | int | 图形中间点到上边直线距离 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Contains | CPoint pos | bool | 判定传入点是否在数据存储图元内部 |

## DFDProcess

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| radix | int | 半径 |
| subDiagramState | bool | 是否有子图标志 |
| onRoutes | bool | 是否经过查找标志 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Contains | CPoint pos | bool | 判定传入点是否在数据加工图元内部 |
| hasSubDiagram |  | bool | 返回是否有子图 |
| setSubDiagram | bool hassub | void | 设置子图标志位 |
| setOnRoutes | bool has | void | 设置查找标志位 |
| getOnRoutes |  | bool | 返回查找标志位 |

## DFDSource

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| a | int | 图形中间点到左边距离 |
| b | int | 图形中间点到上边直线距离 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Contains | CPoint pos | bool | 判定传入点是否在数据源的内部 |

## Stream

主要属性

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 说明 |
| startE | Element \* | 起始点连接的图元 |
| endE | Element \* | 终止点连接的图元 |
| start | CPoint | 数据流起始点坐标 |
| end; | CPoint | 数据流终点坐标 |
| controlstate | int | 控制焦点状态标志：  0未选中控制点，1左控制点，2右控制点 |

主要操作

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数 | 返回值 | 说明 |
| Contains | CPoint pos | bool | 判定传入点是否在数据流的范围内部 |
| GetState | Void | int | 返回控制焦点状态标志值 |
| GetStart | Void | CPoint | 返回起始点坐标 |
| GetEnd | Void | CPoint | 返回终点坐标 |
| SetStart | CPoint pos | void | 设置起始点坐标 |
| SetEnd | CPoint pos | void | 设置终点坐标 |
| ContainsPoint | CPoint pos | void | 根据鼠标点击的控制点设置焦点状态标志位的值 |
| Onsize | CPoint pos | void | 改变数据流大小 |
| Offset | CPoint pos, CPoint oldpos | void | 移动数据流位置 |
| StartisInfieldof | Element \*e, CPoint pos | bool | 判断起始点是否在图元中 |
| EndisInfieldof | Element \*e, CPoint pos | bool | 判断终点是否在图元中 |
| SetStartElement | Element \*e | void | 设置起始点连接的图元 |
| SetEndElement | Element \*e | void | 设置起始点连接的图元 |
| SetControlState | int state | void | 设置控制点标志位值 |
| getStartElement | Element\* | void | 返回开始点连接的图元 |
| getEndElement | Element\* | void | 返回结束点连接的图元 |

# 动态模型

在本系统的开发过程中，我们不仅需要建立类设计的静态模型，同时还需要建立一定数量的动态模型（序列图、状态图、活动图等）以细化本系统内部的执行过程及对象间的交互过程。该动态模型在建立过程中可以对静态模型起到查漏补缺、修正的作用，是静态模型的一个很好的补充。

本系统中向用户提供的操作有：创建图元操作、编辑图元操作（拖动图元、为图元改名、删除图元、进入“加工”图元的子图、退出子图编辑返回父层图编辑）、查找路径操作。其中，较为复杂的操作有：创建图元操作、拖动图元操作、进入“加工”图元的子图操作以及退出子图编辑返回父层图编辑的操作。在本章节，我们将为系统支持的所有操作提供序列图表示以细化其执行过程及对象间的交互过程，在此基础上，还将为以上四种较为复杂的操作提供活动图表示以做进一步说明。

## 各操作的序列图模型

序列图是一种按照时间顺序，跟踪对象之间事件发生、传递过程的动态模型图，其主要特点是自上而下进行，必须从参与者开始，且需要保持事件的连续性。此处我们给出本系统提供的全部操作对应的序列图，并逐一作说明。

### 图元创建过程序列图

本图以创建图元源为例，其他图元创建操作基本相同。

用户通过点击菜单项“创建源（终点）”创建该图元。其对应的菜单相应函数为Menu(1)，点击后系统认为即将开始图元源的创建操作，故将后续操作分配给工厂模式工具类RectangleTool来执行。该工具类在接到用户鼠标按下的消息后首先向图形管理类（器）DiagramEditor询问用户选择的是哪个图形。得到该图形后将其置为当前正在操作的图形，并重绘窗口以提起绘出图元源的外形，方便用户建立图元。值得一提的时，虽然此时需要先绘出图元，但实际上该图元还未建立。

考虑到一些用户习惯于在建立图元时拖动图元，故本系统还加入了在创建图元时的鼠标拖动消息相应操作。该消息响应操作即是在当前鼠标位置重绘源图元的外形。

当用户发出鼠标释放消息的时候，RectangleTool即认为用户已确定图元创建位置，故其新建一个图元源的对象，并判断该对象是否与某些已有的流图元相接，若相接则将新建的图元置为流图元的开始或终止图元，随后将新建图元加入到当前正在操作的图形对象中，将新建图元置为当前正在操作的图元。重绘窗口以高亮新建图元。具体的序列过程如**错误!未找到引用源。**所示。

为了更清晰地说明图元创建过程，以创建图元源(Source)为例，我们给出了图元创建过程的活动图表示，如图 5‑2所示。



图 5‑1图元创建过程序列图



图 5‑2 创建图元过程的活动图表示

### 移动图元序列图

本系统默认用户的操作为编辑操作，即系统在默认状态下是处于工具栏EditTool的操作范围中。此时，EditTool若接收到鼠标左键按下消息，即做出响应，向图形管理类（器）DiagramEditor询问用户选择的是哪个图形，将该图形置为当前正在操作的图形。而后向该图形询问用户选择的是哪个图元，将该图元置为当前正在操作的图元，并重绘以高亮该图元。

此时若用户发出鼠标拖动消息，EditTool接收到后将首先判断当前正在操作的图元是不是流图元(Stream)，若是则继续判断用户是否点击到流图元的控制点上，若是则做伸缩stream操作（伸缩stream将在5.1.3中介绍），若不是则做移动图元操作。若正在操作的图元不是流图元，也做移动图元操作。

移动图元操作主要根据用户拖动过程中鼠标的位移量，修改当前正在操作的图元的坐标，并重绘该图元，若是流图元还将标识出流控制点，直到用户发出鼠标释放消息时停止重绘图形。具体序列过程如图 5‑3所示。



图 5‑3 移动图元序列图

### 流图元Stream伸缩过程

在5.1.2中曾介绍，本系统在接收到鼠标拖动消息时，若判断用户点击到了流控制点上，则做伸缩Stream操作。伸缩Stream操作即根据用户发出鼠标拖动消息的位置，修改流控制点的坐标，并重绘该流图元及流控制点。具体序列过程如图 5‑4所示。



图 5‑4 流图元Stream伸缩序列图

### 图元改名过程

图元改名仍属于工厂模式下工具类EditTool的操作范围，EditTool在获取到用户鼠标右键按下的消息后做出响应。首先向图形管理类（器）DiagramEditor询问用户选择的是哪个图形，将该图形置为当前正在操作的图形。而后向该图形询问用户选择的是哪个图元，将该图元置为当前正在操作的图元，并重绘以高亮该图元。需要说明的是，与鼠标左键不同，EditTool不响应用户拖动鼠标右键的消息，仅响应鼠标右键按下和释放的消息。

EditTool在获取到用户鼠标右键按下的消息后做出响应，即显示改名对话框提示用户输入新名词并获得，若用户输入不为空则将当前正在操作的图元的名称改为用户输入的内容，并重绘且高亮该图元。若用户输入为空则不做任何操作。具体序列过程如图 5‑5所示。



图 5‑5 图元改名序列图

### 删除图元过程

与其他序列图中EdiTool的鼠标左键按下响应操作一样，EditTool在接收到鼠标左键按下消息后经过一系列操作，将获得用户点击到的图元，并置为当前正在操作的图元。此时，若EditTool接收到用户按下键盘Del键的消息，则将做以下操作：首先判断当前正在操作的图元是否好似流图元，若不是，则在当前正在操作的图形中寻找以该图元作为起点或终点图元的流图元，并将他们的对应位置置为空，若是，则跳过该步，继续移除操作，即从当前正在操作的图形中移除该图元，并回收该图元的空间。清空当前正在操作的图元，并重绘窗口。具体序列过程如图 5‑6所示。

为了更加清晰地表示EditTool的基本操作流程，我们将图元的拖动、改名及删除操作组合在一起给出了他们的活动图表示，如图 5‑7所示。



图 5‑6 删除图元序列图



图 5‑7 EditTool基本操作的活动图表示

### 创建/打开子图操作过程

EditTool接收到用户鼠标左键双击的消息后，经过一系列操作后，EditTool将获得用户点击到的图元，并置为当前正在操作的图元。如果该图元时加工图元(Process)，则EditTool会首先询问其是否有子图，如果没有则为其创建一个子图，执行创建子图操作；若有子图则打开该图元对应的子图。

创建子图操作主要包括以下几个方面：首先EditTool将当前图形保存起来，后调用新建图形方法，创建一个新图形，并将该新图形置为当前正在操作的图形。然后EditTool为用户打开一个新窗口且激活，然后将新图形与新窗口映射起来，并加入图形管理类（器）DiagramEditor的映射表中以供后续使用。并在旧图形中加入当前正在操作的图元与新窗口的映射关系。此时用户看到的位于屏幕Z轴顶层的是新建的窗口，即新建的子图。

打开子图操作主要包括以下几个方面：EditTool在当前正在操作的图形中找到该图元对应的子图所在窗口的句柄，并向图形管理类（器）DiagramEditor获取与该窗口相映射的图形，将该图形置为当前正在操作的图形。依据子图所在窗口句柄，打开该窗口且激活，并重绘该窗口。具体序列过程如图 5‑8所示。

由于创建/打开子图操作是本系统的重要部分，故为了进一步说明该操作，我们提供了该操作的活动图表示，如图 5‑9所示。



图 5‑8 创建/打开子图操作序列图



图 5‑9 创建/打开子图的活动图表示

### 退出子图返回上层图过程

EditTool若接收到用户按下键盘ESC键的消息，则会执行返回上层数据流图的操作。EditTool在执行过程中，首先向当前正在操作图形获取其父层图所在窗口的句柄，再依据句柄，向图形管理类（器）获取与该句柄相映射的图形，获取该图形后将其置为当前正在操作的图形。根据句柄打开窗口并激活，在新打开的窗口上重绘获得的图形。具体序列过程如图 5‑10所示。



图 5‑10 退出子图编辑返回上层图的序列图

### 主图查找路径过程



图 5‑11 主图查找路径序列图

如图 5‑11所示，首先通过点选菜单栏的查找路径项将当前的工具类设置为LookupTool，之后捕获用户单击鼠标左键的操作，通过鼠标左键单击的位置和单击的窗口句柄获取用户是否单击到图元上，若图元类型为源（终点），首先清空所有上次执行查找路径所留下的高亮标记，之后通过广度优先搜索的方式，查找所有以找到的源作为终点的流，并获取流的开始点连接的图元作为下一次搜索的起点，并且当下一次起点为加工图元时，标记本次查找路过该图元，以便跨子图查找的时候确定是否经过该图元，以此类推，找到本层中所有与终点（源）连通的流，设置流的高亮属性为1，之后进行该图的重绘，在重绘时若高亮属性为1则进行高亮，主图查找路径完成。

### 跨子图查找路径过程



图 5‑12 跨子图查找路径序列图

如图 5‑12所示，该图表现了在跨子图查找路径时的过程，用户进行双击操作，点选有子图并且在查找路径上的加工类型图元，因为MFC在执行双击操作时会捕获四个消息，LButtonDown->LButtonUp->LButtonDClk->LButtonRelease，所以这个操作会先调用Press的过程，所以在Press时已经确定了当前选中的图元，所以在这个过程直接判断点选的图元是否是加工图元，如果是，判断该图元是否在查找的路径上，如果在则打开子图并在该图中查找所有为终点的源类型图元，之后执行5.1.8所述的查找子图路径过程，即可查找并高亮出所有子图中的路径。

### 退出查找路径过程

如图 5‑13所示，通过在图形编辑界面单击鼠标右键退出图形查找，用户点击鼠标右键后，首先清除所有流上的高亮标记，之后递归的清除所有在路径上的加工类型图元的OnRoute标记，包括主图的OnRoute标记和其经过的子图上的OnRoute标记，之后将当前使用的工具类设置成为EditTool。



图 5‑13 退出查找路径序列图

为了更加清楚的看到整个查找路径操作的过程，我们还提供了查找路径的活动图，如图 5‑14所示，该图描述了在查找路径时捕获到用户使用的三种鼠标操作（左键单击/左键双击/右键单击）后系统所执行的操作。



图 5‑14 查找路径过程活动图

# 详细设计

## DiagramEditor

class DiagramEditor

{

private:

CMainFrame \*cpMainFrame;

CChildFrame \*cpChildFrame;

Element \*currente; // 指向当前图元

Diagram \*currentd; // 指向当前图形

Tool \*currenttool;

vector<Diagram\*> ds; // 保存所有图形

map<HWND, Diagram\*> WindowtoDiagram; // 窗口句柄对应图形的映射

RectangleTool \*rt; // 矩形工具类

EllipseTool \*et; // 椭圆工具类

LineTool \*lt; // 线型工具类

StreamTool \*st; // 连接线工具类

EditTool \*edt; // 编辑图元工具类

LookupTool \*lkt; // 查找路径工具类

public:

DiagramEditor();

~DiagramEditor();

bool hasCurrentE(); // 当前有选中的图元

void InsertMap(HWND hWnd, Diagram \*d); // 创建句柄与图形映射

void Menu(int item); // 菜单选项（创建何种类型图元）

void SetCurrentE(Element \*e); // 设置当前图元

void SetCurrentD(Diagram \*d); // 设置当前图形

void SetCurrentTool(Tool \*t); //

void ClearCurrentE();

void ClearCurrentTool();

void AddDiagram(); // 新建图形

void SearchDiagram(HWND hWnd, Diagram \*&d); // 根据句柄查找对应图形并返回

void Press(CPoint pos); // 鼠标左键按下

void DoubleClick(CPoint pos); // 鼠标左键双击

void Release(CPoint pos); // 鼠标左键释放

void RightPress(CPoint pos); // 鼠标右键按下

void RightRelease(CPoint pos); // 鼠标右键释放

void ExitEdit();

void Remove(); // 删除图元操作

void Draw(Element \*doce, Diagram \*docd, bool original, CPoint pos, CDC \*pDC);

void Redraw(bool original);

void DrawOriginal(CDC \*pDC,CPoint pos);

void Highlight(CDC \*pDC);

void Move(CPoint pos, CPoint oldpos);

void Focus(CDC \*pDC);

void EndCreate(Element \*e);

HWND SearchDiagramtoProcess(Diagram \*&d, DFDProcess \*p);

};

## Diagram

class Diagram

{

private:

HWND fatherhWnd;

vector<Element\*> elems; // 保存图形中的图元

map<Element\*, HWND> process2Diagram; //

public:

Diagram();

~Diagram();

void Add(Element\* e); // 添加图元

bool Find(CPoint pos, Element \*&e); // 根据坐标查找图元

HWND getFatherWnd();

void setFatherWnd(HWND hWnd);

HWND SearchWnd(DFDProcess \*e); // 查找图元（Process）对应的窗口句柄

void SetElementforStreambyElement(Element \*e, CPoint pos); // e是除stream类型其他类型时使用

void SetStartElementforStream(Stream \*se, CPoint pos); // e是stream类型时使用

void SetEndElementforStream(Stream \*se, CPoint pos);

void Remove(Element \*e); // 从图形中删除图元e

void DrawDiagram(vector<CPoint>&poss, vector<int>&types, vector<CString>&strs,CPoint startmidend[][3]);

void InsertMap(Element \*e, HWND hWnd); // 向Map中插入<Process,HWND>对

int FindStreams(Element\* currente, queue<Stream\*>& elemq); // 根据结束图元查找所有以该图元为终点的路径

void FindStreams(vector<Element\*>& elemq); // 查找路径-找到该图中的连接线

void Highlight(CDC \*pDC); // 高亮路径

void ClearHighlight(); // 清除高亮

void FindEndElements(vector<Element\*>& endElements); // 查找所有终点图元（只能是右边无连接的Source）

void ClearOnRoute(queue<DFDProcess\*>& fathers); // 清除当前图形中的Process标志位并返回

};

## Element

class Element

{

protected:

bool isHighlight; // 是否高亮标志

int type;//图元类别标志

CPoint midPoint;//图形中间点

CString text;//图元名称

public:

Element();

~Element();

CPoint GetMidPoint(); // 获取图元中点

CString GetText(); // 获取图元文本

void SetText(CString s); // 设置图元文本

void SetHighlightFlag(bool isHighlight);

bool GetHighlightFlag();

virtual bool Contains(CPoint pos){ return true; } // 判定传入点是否在图元内部

bool isProcess(); // 是否是加工（有子图）

bool isStream(); // 是否是流（连接线）

bool isSource(); // 是否是源（查找子图的终点）

bool isDataStorage(); // 是否是存储

virtual void Offset(CPoint pos, CPoint oldpos); // 位移（移动图元）

};

### DFDSource

class DFDSource : public Element

{

private:

int a;

int b;

public:

DFDSource(int type, CPoint midPoint, CString text);

~DFDSource();

bool Contains(CPoint pos);

};

### DFDProcess

#include "Element.h"

class DFDProcess : public Element

{

private:

int radix; // 半径

bool subDiagramState; // 是否有子图标志

bool onRoutes; // 是否经过查找routes

public:

DFDProcess(int type, CPoint midPoint, CString text);

~DFDProcess();

bool hasSubDiagram(); // 返回是否有子图

void setSubDiagram();

void setOnRoutes(bool has);

bool getOnRoutes();

bool Contains(CPoint pos);

};

### DFDDataStorage

class DFDDataStorage : public Element

{

private:

int a;//中间点到左边距离

int b;//中间点到上边直线距离

public:

DFDDataStorage(int type, CPoint midPoint, CString text);

~DFDDataStorage();

bool Contains(CPoint pos);

};

### DFDStream

class Stream : public Element

{

private:

Element \*startE; // 起始点连接的图元

Element \*endE; // 终止点连接的图元

CPoint start;

CPoint end;

int controlstate; //0未选中控制点，1左控制点，2右控制点

public:

Stream(int type, CPoint midPoint, CString text);

~Stream();

int GetState();

CPoint GetStart();

CPoint GetEnd();

void SetStart(CPoint pos);

void SetEnd(CPoint pos);

void ContainsPoint(CPoint pos);

void Onsize(CPoint pos);

void Offset(CPoint pos, CPoint oldpos);

void StartFollowElement(Element \*e);

void EndFollowElement(Element \*e);

bool Contains(CPoint pos); // 移动的时候需要找mid点，end点和start点的最小和最大界，然后判断是否包含

bool CompareStartElementWith(Element \*e);

bool CompareEndElementWith(Element \*e);

bool StartisInfieldof(Element \*e, CPoint pos); // 起始点在图元e中

bool EndisInfieldof(Element \*e, CPoint pos); // 终止点在图元e中

void SetStartElement(Element \*e); // 设置起始点连接的图元

void SetEndElement(Element \*e); // 设置终止点连接的图元

void SetControlState(int state); //

Element\* getStartElement(); // 返回开始点连接的图元

Element\* getEndElement(); // 返回结束点连接的图元

};

## Tool

enum ToolType

{

CREATTOOL = 1,

EDITTOOL = 2,

LOOKUPTOOL = 3

};

class Tool

{

protected:

Diagram \*currentd;

Element \*currente;

DiagramEditor \*de;

ToolType type;

public:

Tool();

~Tool();

bool hasCurrentE();

void SetCurrentE(Element \*e);

void SetCurrentD(Diagram \*d);

void ClearCurrentE();

//void getDiagramEditor(DiagramEditor \*d);

virtual void Press(CPoint pos, HWND hWnd){}

virtual void Move(CPoint pos, CPoint oldpos){}

//virtual void Move(CPoint pos, Element \*currente){}

virtual void Release(CPoint pos){}

//virtual void Release(CPoint pos, Element \*currente){}

//virtual void RightPress(CPoint pos, HWND hWnd){}

//virtual void RightRelease(CPoint pos){}

//virtual void DoubleClick(CPoint pos, HWND hWnd){}

void Select(Diagram \*&d, HWND hWnd);

void SetType(ToolType type);

ToolType GetType();

};

## CreateTool

class CreateTool : public Tool

{

public:

CreateTool();

~CreateTool();

virtual void Press(CPoint pos, HWND hWnd){}

void Move(CPoint pos, CPoint oldpos);

void Release(CPoint pos);

virtual void CreateElement(CPoint pos, Element \*&e);

virtual void Update(CPoint pos);

};

### RactangleTool

class RectangleTool : public CreateTool

{

public:

RectangleTool();

RectangleTool(DiagramEditor \*d);

~RectangleTool();

void Press(CPoint pos, HWND hWnd);

void CreateElement(CPoint pos, Element \*&e);

void Update(CPoint pos);

};

### EllipseTool

class EllipseTool : public CreateTool

{

public:

EllipseTool();

EllipseTool(DiagramEditor \*d);

~EllipseTool();

void Press(CPoint pos, HWND hWnd);

void CreateElement(CPoint pos, Element \*&e);

void Update(CPoint pos);

};

### LineTool

class LineTool : public CreateTool

{

public:

LineTool();

LineTool(DiagramEditor \*d);

~LineTool();

void Press(CPoint pos, HWND hWnd);

void CreateElement(CPoint pos, Element \*&e);

void Update(CPoint pos);

};

### StreamTool

class StreamTool : public CreateTool

{

public:

StreamTool();

StreamTool(DiagramEditor \*d);

~StreamTool();

void Press(CPoint pos, HWND hWnd);

void CreateElement(CPoint pos, Element \*&e);

void Update(CPoint pos);

};

## EditTool

class EditTool : public Tool

{

public:

EditTool();

EditTool(DiagramEditor \*d);

~EditTool();

void Press(CPoint pos, HWND hWnd); // 单击

void RightPress(CPoint pos, HWND hWnd); // 右击->修改名字

void DoubleClick(CPoint pos, HWND hWnd); // 双击->创建子图

void Move(CPoint pos, CPoint oldpos); // 移动图元

void Release(CPoint pos); // 鼠标左键释放

void RightRelease(CPoint pos); // 鼠标右键释放

void GotoFatherWnd(); // 打开父窗口（对应ESC）

void Remove(); // 删除图元（对应DELETE）

void CreateNewDiagram(); // 创建新图形

void OpenDiagramtoProcess(HWND hWnd, Diagram \*d); // 打开Process的子图

};

## LookupTool

class LookupTool : public Tool

{

private:

vector<Element\*> routes; //保存所有Stream

public:

LookupTool();

LookupTool(DiagramEditor \*d);

~LookupTool();

void Press(CPoint pos, HWND hWnd);

void RightPress();

void DoubleClick(CPoint pos, HWND hWnd);

void OpenDiagramtoProcess(HWND hWnd, Diagram \*d); // 打开Process的子图

void ClearProcessOnRoutes(queue<DFDProcess\*>& fathers);

};

1. [↑](#footnote-ref-1)