# 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 1‑1所示。



图 1‑1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表 1‑1及图 1‑2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 1‑1 系统总体操作状态转换表



图 1‑2 系统总体操作状态转换图

### 工厂模式的使用

# 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。具体的序列过程如图 2‑1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 2‑2所示。



图 2‑1 图元创建过程序列图



图 2‑2 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2.1.3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 2‑3所示。



图 2‑3 移动图元序列图

### 流图元Stream伸缩序列图

在2.1.2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 2‑4所示。



图 2‑4 流图元Stream伸缩序列图

### 图元改名序列图

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 2‑5所示。



图 2‑5 图元改名序列图

### 删除图元序列图

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 2‑6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 2‑7所示。



图 2‑6 删除图元序列图



图 2‑7 EditTool基本操作的活动图表示

### 创建/打开子图操作序列图

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 2‑8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 2‑9所示。



图 2‑8 创建/打开子图操作序列图



图 2‑9 创建/打开子图的活动图表示

### 退出子图返回上层图序列图

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 2‑10所示。



图 2‑10 退出子图编辑返回上层图的序列图

1. [↑](#footnote-ref-1)