事件驱动体系结构在 GIS中的设计实现

王 园,吉国力,苏 秦
(厦门大学自动化系,福建 厦门 361005)

摘要: 随着地理信息系统的发展和应用,其规模和复杂度日益增加,难以并行开发、方便测试发布、良好扩展等问题日益突出. 本文依据实际系统开发,提出了基于事件驱动风格的体系结构,针对该体系结构创建了虚拟窗口的概念,并给出了系统具体实现的方法. 通过该体系结构,可以降低类似系统的开发难度,提高软件开发速度、方便软件测试以及发布工作.

关键词: 地理信息系统;体系结构;事件驱动
中图分类号: TP 311. 13 **文献标识码**: A **文章编号**: 0438-0479 (2005) 04-0472-03

地理信息系统 (Geographic Information Systems,简称 GIS)是一种兼容、存储、管理、分析、显示与应用地理信息的计算机系统,是分析和处理海量地理数据的通用技术[1]. 随着 GIS的发展和应用的不断深入, GIS的体系设计也显得越来越重要,拥有了一个良好的体系结构, GIS软件能够到易于修改、易于重用,具有良好的可扩展性.

1 事件驱动 GIS软件体系设计

一般来说 GIS软件框架的功能可以分为两类: 1) 基本地图浏览功能,包括地图显示、缩放、量测、打印、地图配置信息等功能;这部分功能在通用 GIS是必备的; 2) 扩展功能,例如地图编辑功能、地图分析功能等专业上的应用;这些扩展的功能往往要划分为多个模块,多个模块共同开发,这样就给 GIS软件的开发、测试、合成带来了困难,因此有必要创建一种有利于软件开发、测试以及发布的体系结构. 笔者在实际开发过程中将事件驱动风格的体系结构运用到了 GIS软件中.

与传统的过程调用方式不同,事件驱动的思想是组件不直接调用一个过程,而是触发或广播一个或多个事件. 其它组件在系统中注册一个和事件相关联的过程. 当事件发生时,系统会自动调用所有和这个或这些事件关联的过程. 这种调用方式叫"隐式调用"[2].

计算机系统的中断调用是典型的事件驱动. "中断"就是事件,"中断服务程序"就是组件的过程. 组件通过修改中断向量表注册中断服务程序,当中断产生,系统自动暂停当前的工作,转入执行中断服务程序.

从结构上来说,事件驱动风格中的构件其实是一个包含了若干过程和事件的模块[3]. 过程可以按照一股的方法调用,但是构件可以把它的一些过程在系统的事件中进行注册. 当这些事件被触发时,会导致这些过程被执行. 因此,一个事件驱动系统中的连接件包括传统的过程调用,以及事件触发的过程调用. 这种风格中,事件触发时并不知道哪个构件会受到这个事件的影响,甚至可能根本没有任何构件受到影响. 因此,无法确定构件的执行顺序,也无法确定执行什么. 所以,大多数事件驱动系统也会使用显示调用 (一般的过程调用)作为交互的一种补充形式.

事件驱动对软件重用提供了强有力的支持. 任何组件只需要简单地对系统中的事件注册,就可以引入到这个系统中来;构件接口相对而言比较容易定义,避免了日后接口改变带来的困难;同时构件之间的耦合被降到了最低,构件的升级演化,并不会影响系统中的其他构件,容错性也增强了,一个构件出错,一般不会影响到其他构件[4].

2 事件驱动风格的 GIS软件结构的设计与实现

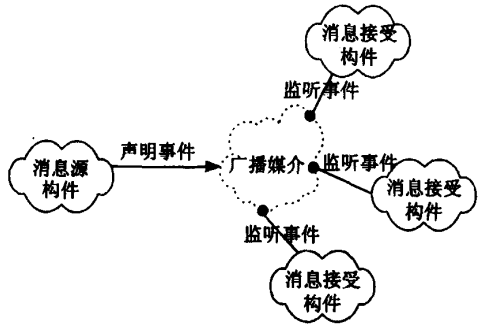

图 1 事件驱动过程

Fig 1 Event driven procedure

收稿日期: 2004-06-23
作者简介:王园 (1980 -),女,硕士研究生.

事件驱动风格的 GIS软件结构设计如图 2所示.

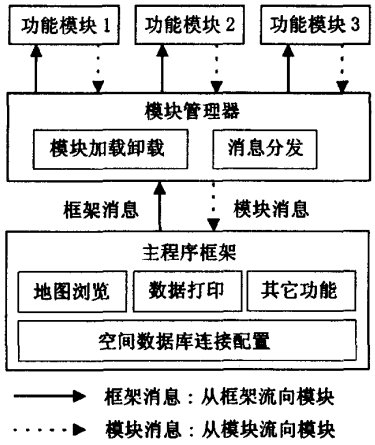

图 2　事件驱动风格的 GIS结构

Fig 2　Event driven GIS architecture

系统其中包括如下部分:

(1) 主程序框架

主程序框架是建立 GIS系统整体应用的基础,包括与空间数据库的连接、配置加载、地图的显示、缩放、浏览、打印等基本功能.

在实际系统开发中,采用的是 MFC中的视图/文档 (Document/View)风格的结构,符合了将数据的操作和数据的显示、数据模型区分开来的 MVC模式.

(2) 功能模块

功能模块指的是 GIS系统的扩展功能划分出来的,具有同类操作性质的功能体. 一般说来,可以将 GIS系统的各个模块封装成一个个的动态链接库,用于实现运行时加载,同时也便于系统的分工合作,方便开发调试.

主程序应用模块中的功能并非通过直接的过程或函数调用,而是接受来自模块管理器的消息,对此做出响应. 有时功能模块也需要向框架发出消息,来实现一定的功能.

在系统中,为了让功能模块不仅能接收主程序框架的鼠标键盘等标准消息,还可接收自定义消息,系统采用了创建一个虚拟窗口的方法,定义一个窗口类 CModuleWnd(自 CWnd派生)为模块窗口的基类,并且定义模块的基本信息.

```
CModuleWnd: : public CWnd    //模块窗口基类定义
{
  CMainView * m_pMainView;  //保持对主视图的引用
  CMainDoc * m_pDoc;    //保持对文档的引用
    …  //可根据系统需要添加相应的定义
}
  typedef struct   //定义模块结构
  {
  char Name[NAME_MAX_LEN];    //模块名称
  HINSTANCE hDll;    //模块动态链接库句柄
  CModuleWnd * pWnd;    //模块窗口
  …  //其它模块信息
} Module;
```

(3) 模块管理器

模块管理器主要的功能有两点: 1)模块的动态加载与卸载, 2)主程序框架与模块之间消息的传递.

(a) 模块的动态加载与卸载

通常模块加载到系统中的时候,需要添加工具条,或者是修改菜单,在模块移出系统的时候,就要删除自己的工具条,将已经添加的菜单删除. 因此在模块中还需要处理对主框架中工具条或菜单的修改. 为此我们在每个功能模块的动态链接库里定义了模块窗口输出,以及模块窗口删除函数,并在模块的输出和删除过程中,调用对主程序的工具条或菜单的修改. 动态链接库中定义如下输出、删除函数:

```
CModuleWnd * ModuleLoad ()    //定义输出基类
{
  g_pWnd = new CConcreteWnd;   //创建具体的模块窗口
  return g_pWnd;
}
  void ModuleUnload()    //删除
{
  delete g_pWnd;
  g_pWnd = 0;
}
```

在模块管理器中定义一个模块列表 ModueList,保存模块信息,并如下载入删除模块:

```
typedef CModuleWnd * (* fLoadModuleWnd) (void);
  //定义输出模块窗口函数指针
CModuleManager: : ModuleLoad (char * ModuleName)
  //根据模块文件名加载模块
{
  HINSTANCE hDll;
  hDll = LoadLibrary(ModuleName);
    //获得动态链接库句柄
  CModuleWnd * wnd;    //定义模块窗口指针
  (LoadModuleWnd * ) loadWnd = NULL;
    //函数指针,输出模块窗口
  LoadWnd = GetProcAddress (hDll, "ModuleLoad");
    //获得输出函数指针
  wnd = LoadWnd();    //获得输出的模块窗口
  Module aModule;
  aModule. hDll = hDll;
  aModule. pWnd = wnd;
wnd- > Initialize(); //模块初始化,可进行菜单工具条修改
```

```
    ModuleList pushback(aModule);    //保存
}
```

模块删除方法:

```
typedef long (* fUnLoadModuleWnd) (void);
    //定义删除模块函数指针
void CModuleManager::ModuleUnload(Module aModule)
{
    fUnLoadModuleWnd UnLoadModuleWnd = NULL;
    //获得动态链接库句柄
  UnLoadPlugWnd = (fUnLoadModuleWnd)::GetProcAddress
(aModule.Address,"ModuleUnload");    //获得删除函数
    aModule.pWnd->UnIntitialize(); //可以恢复菜单工具条
    (*UnLoadModuleWnd)();    //删除窗口
    ::FreeLibrary(aModule.hDll);    //卸载动态链接库
}
```

(b) 消息分发

在系统中,消息的分发有两种情况,一种是由框架向模块发送的消息,是主要的消息;还有一部分是模块向框架发送的消息,一般情况下较少用到.

在先前描述的事件驱动系统中,消息的分发采用了广播的方式,即所有的构件都能接收到消息,考虑到效率的问题,系统在消息的分发上,采用了过滤的手段,让模块在加载的时候,能够注册自己所感兴趣的消息,这样消息在发送上更有目的性,提高了系统的效率.下面代码是转发框架的窗口消息示例:

```
LRESULT CModuleManager::(UINT message, WPARAM wParam,
  LPARAM lParam)    //模块管理器转发窗口消息
{
  if (message == WM_PAINT||(message >= WM_MOUSEFIRST&&message <=
    WM_MOUSELAST)||(message >= WM_KEYFIRST&&message <=WM_KEYLAST))
      //对消息进行过滤,可加入自定义消息,也可包括按钮、菜单消息 WM_COMMAND
  {
  list<Module>::iterator iter=ModuleList.begin();
  for(; iter!=ModuleList.end(); iter++)    //检索各个模块
  {
    …    //判断模块是否对该消息感兴趣,若感兴趣则分发消息,否则跳过
    (*iter).pWnd->WindowProc(message, wParam, lParam);
  //模块窗口处理消息
  }}}
```

由于事件驱动系统本身的弱点,在数据交互这一方面,单独采用消息的机制,并不是非常完善,因此,在系统中还是可以适当地采用过程调用的方法,在这里可以采用全局对象的方法,或者可以采用类似于剪贴板技术,在系统专门开辟一块用于数据交换的区域,定义好模块数据交换格式,在模块和主程序之间进行交互,增强了系统在这一方面的处理能力.

3 总 结

本文讨论的基于事件驱动的软件体系结构在有着复杂界面操作的 GIS软件是很有意义的,它可以降低解决问题的难度,将系统划分为多个难度较低的模块,功能模块可以同时开发,相互之间耦合程度较低,便于软件的开发、调试、以及产品的发布.需要指出的是,本文提出的框架在数据交互上,还是不够灵活,希望能够提出新的方法来对这个问题进行改进.

参考文献:

[1] 邬伦,刘瑜,张晶,等.地理信息系统——原理、方法和应用[M].北京:测绘出版社,1996.

[2] 万建成,卢雷.软件体系结构的原理、组成与应用[M].北京:科学出版社,2002.

[3] 冯冲,江贺,冯静芳.软件体系结构理论与实践[M].北京:人民邮电出版社,2004.

[4] 杨安.软件体系结构及基于软件体系结构的系统开发[D].安徽:安徽大学,2003.

Design and Implementation in GIS of Event-driven Architecture

WANG Yuan, JI Guo-li, SU Qin

(Department of Automation, Xiamen University, Xiamen 361005, China)

Abstract: With development and application of Geographic Information System (GIS), its scale and complexity increase quickly and the problems of developing concurrently, test and deploy, extendibility become to be more protruded. According to the development of real system, the present paper brings forward event-driven soft architecture. Aiming at this architecture, the concept of virtual windows was created and the concrete implementation method was displayed. This method can reduces the difficulty of soft development and increases development speed, make soft testing and deploy conveniently.

Key words: GIS; soft architecture; event-driven