设计模式在通讯接口设计中的应用①

严　华，张欲蓉

(中国电子科技集团公司 第五十一研究所，上海 201802)

摘　要：为了应对系统中通讯接口的日益多样化，基于设计模式提出了统一接口模型和数据到达通知模型，使数据源层的通讯接口解耦于领域层的功能模块，实现了接口的动态配置和同构冗余，并以一个电子对抗领域的典型干扰系统为例，说明了两个模型的设计思路、使用方法和关键代码。

关键词：设计模式；接口设计；数据源层；Bridge 模式；Observer 模式

Application of Design Pattern to Communication Interface Design

YAN Hua, ZHANG Yu-Rong

(51st Research Institute of CETC, Shanghai 201802,China)

Abstract: With all kinds of communication interface in system, this paper constructs two models named Unified Intertace and Data Arrived Inform based on design pattern. These models make the interface of the data domain independence of the functional domain. The interface can be used in dynamic configuration and redundance. Taking the jamming system in ECM as an example, explain the thought, and the way of usage, and key code of the two models.

Key words: design pattern; interface design; data domain; bridge pattern; observer pattern

目前在各种系统的设计中，普遍采用分层和分类的思想，以降低系统实现的复杂性[1]，而由 GOF 提出的 23 种设计模式[2]就是对这种思想的总结。模式是从专家的经验中提炼出来的，它使设计者可以复用别人的成功设计[3]，从而使系统的设计更加灵活、优雅和简洁，并且提供更好的可复用性。

随着系统功能的日益复杂，系统中各种通讯接口将日益多样化，如何屏蔽不同接口的差异性，用一种统一的方法访问各种接口，从而使系统的功能设计能够解耦于系统所使用的具体通讯接口，同时做到接口的动态配置和同构冗余，并且在不同的系统间复用接口设计，变得日益重要。基于设计模式，本文提出了统一接口模型和数据到达通知模型，就是用于解决多接口差异性的整体设计方案。

1　问题分析

图 1 是电子对抗领域典型干扰系统的通讯接口示意图，从系统控制软件的角度看，图中涉及 4 类，共 7 个通讯接口，包括 CAN 网络、TCP/IP 网络、RS232 串口和 IO 端口。

从各个通讯接口实现的功能上看，都是用于字节流的收发，但是它们都有自己特有的配置参数和数据收发参数。例如，TCP/IP 网络需要 IP 地址和端口配置；CAN 网络有波特率和 ID 号的配置，按固定长度的报文收发数据；RS232 串口有波特率和串口号的配置；IO 端口数据的读写需要有相应的 IO 地址，收发的数据长度一般为 4 字节。

从以上描述可以看出，各个通讯接口既具有共性：字节流的收发；又具有个性：不同的配置参数和数据收发参数。那么我们如何才能屏蔽不同通讯接口的“个性”， 而只体现它们的“共性”，从而能够用一致的方法访问各通讯接口，这就是本文所要解决的问题。

2　系统设计

依据图 1 中的系统组成及各分机间的接口关系，整个系统可以设计为三层架构[1]，具体见图 2。设计

① 收稿时间:2011-08-23;收到修改稿时间:2011-09-23

中遵循了单一职责(SRP)[4]和开放封闭原则(OCP)[5]，保证系统的扩展性需求。其中数据源层由不同的接口功能模块组成，封装了系统中各种通讯接口的功能，并作为整个领域层的数据来源。领域层主要包含各分机的功能模块，表现层则提供人机操作界面，这两层在本文中不作具体描述。

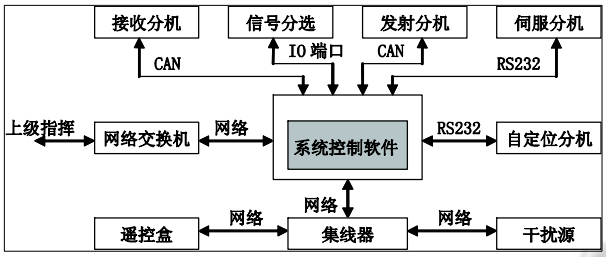

图 1 通讯接口示意图

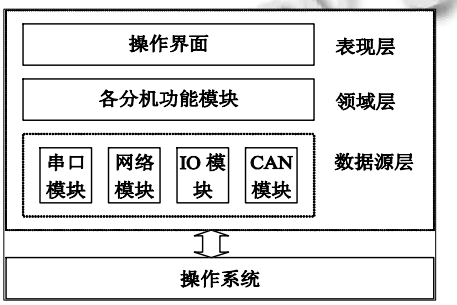

图 2 系统架构

3 数据源层的设计

3.1 数据源层的类图

依据系统的三层架构，对数据源层进行设计，图 3 是完成设计后的类图，其中包含了统一接口模型和数据到达通知模型的主要设计思路。统一接口模型主要解决不同接口如何使用一致的方式发送数据的问题，而数据到达通知模型则解决不同接口读取数据后如何通知领域层功能模块的问题。

类图中的 CPort 是所有数据源层模块的基类，为领域层各功能模块提供统一的接口。而 CSys 则是所有领域层功能模块的基类，为数据源层的数据到达后如何通知领域层提供了统一的接口。CHeadParam 和 CPortParam 则是统一接口模型的配套类，主要用于提供一致的接口。由于各通讯接口的功能模块具有结构上的相似性，所以只以 CAN 网络通讯接口进行说明，具体的实现代码以 C++[6]为例。CAN 网络是一种面向工业底层控制的通讯网络，需要设置波特率和单双滤波等配置参数，各节点的数据收发以 ID 为识别标志[7]。

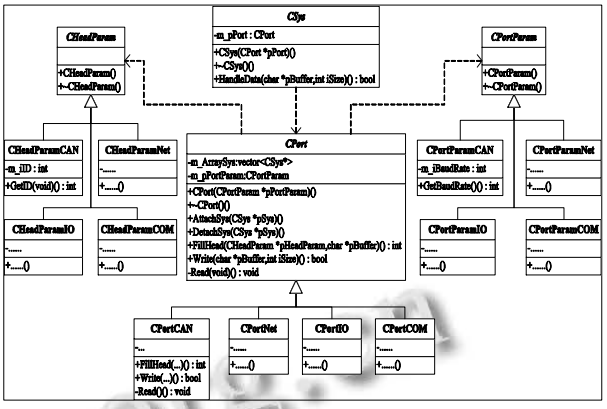

图 3 数据源层的类图

3.2 统一接口模型

统一接口模型用于数据源层的统一数据发送，模型的实现使用了 Bridge 模式[2]，将接口的抽象部分和它的实现部分分离，从而使数据源层的变化不影响整体的系统设计，类的结构关系见图 4。

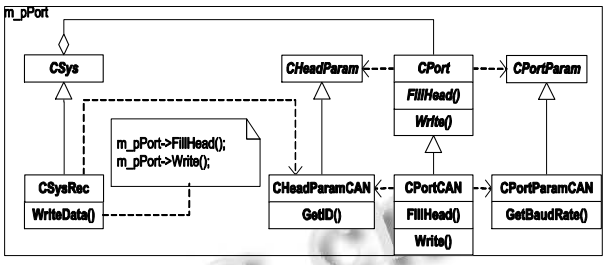

图 4 统一接口模型类图

在图 4 的类结构图中，CPortParam 和 CHeadParam 为客户 CPort 定义了配置参数抽象类和发送参数抽象类，统一了 CPort 相应接口的形参。而 CPort 为客户 CSys 提供了通讯接口抽象类，统一了数据源层对外提供的接口，体现了接口的“共性”。CPortParam 和 CHeadParam 派生的子类，则封装了不同接口的具体配置参数和数据发送参数。CPort 派生的子类，则封装了不同接口的具体底层驱动读写接口。这些子类体现了接口的“个性”。正是在 CPortParam、CHeadParam 和 CPort 的配合下，对 CSys 提供了一致的接口，才最终实现了统一接口模型，从而做到了领域层和数据源层的数据发送解耦。

CPortParam 是一个抽象类，从 CPortParam 派生的各个类则封装了具体的配置参数，从而使 CPort 的构

造函数能够定义成一致的形参。CPortParamCAN 定义了 CAN 接口的相关配置参数，代码只以波特率参数为例：

```
//CAN 接口的配置参数
CPortParamCAN:public CPortParam{
public:
  //仅以波特率参数为例
  CPortParamCAN(int iBaudRate);
  int GetBaudRate();
private:
  int m_iBaudRate;
};
```

CHeadParam 也是一个抽象类，从 CHeadParam 派生的各个类则封装了发送所需的具体参数，从而使 CPort 类的接口 FillHead 的形参能够一致。派生类 CHeadParamCAN 封装了数据接收方的 ID 信息：

```
//封装了 CAN 接口的发送参数
CHeadParamCAN:public CHeadParam{
public:
  //仅以接收方的 ID 为例
  CHeadParam(int ID);
  int GetID(void);
private:
  int m_iID;
};
```

CPortCAN 派生于 CPort，定义了接口 FillHead、Write 和 Read，封装了具体的 CAN 网络接口配置和数据收发功能。其中 FillHead 是整个模型的关键一步，用于把 ID 填入发送缓存，从而借用 char 类型的发送缓存保存了不同通讯接口的发送参数。当调用 Write 发送数据时，可以取出 ID 填入底层驱动提供的数据发送接口，最终实现数据的发送功能。其中 FillHead 通过定义指向 CHeadParam 的基类指针，Write 通过定义填入了发送参数的 char 类型缓存，从而使所有 CPort 派生类的形参保持一致：

```
CPortCAN::CPortCAN(CPortParam *pPortParam){
  CPortParam *param;
  param=(CPortParamCAN*)pPortParam;
  //调用底层驱动接口，设置波特率
  SetBaudRate(param->GetBaudRate());
}
int CPortCAN::FillHead(CHeadParam
*pHeadParam,char *pBuffer){
  CHeadParamCAN *param;
  param=(CHeadParamCAN*)pHeadParam;
  int id=param->GetID();
  //打包 ID
  memcpy(pBuffer,&id,4);
  //返回发送参数长度
return 4; }
//形参 pBuffer 中含有 ID 和具体数据
bool CPortCAN::Write(char *pBuffer,int iSize){
  int id;
  //提取发送参数：ID
  memcpy(&id,pBuffer,4);
  //调用底层驱动接口，发送数据
  WriteCAN(id,pBuffer+4,iSize-4);
}
```

CSysRec 从 CSys 派生，封装了领域层中的接收单元，使用 CPortCAN 和接收单元交互：

```
//发送控制码
CSysRec::WriteData(int ID,int iData){
  int size;
  char buffer[8];
  CHeadParamCAN head(ID);
  //填入发送所需的参数
  size=m_pPort->FillHead(&head,buffer);
  //填入发送的具体数据
  memcpy(buffer+size,&iData,4);
  //计算数据总长度
  size+=4;
  //通过端口发送数　m_pPort->Write(buffer,
size);
}
```

至此，所有的类定义都已经完成，剩下的就是具体实例的生成：

```
//配置通讯接口
m_pPort=new CPortCAN(new CPortParamCAN
(9600));
//配置接收功能模块
m_pSysRec=new CSysRec(m_pPort);
//订阅数据到达通知，说明见下节
```

```
m_pPort->AttachSys(m_pSysRec);
//发送控制字
m_pSysRec->WriteData(1,0x01);
```

在配置接收模块时，可以方便的传入不同的通讯接口指针，实现接口的动态配置。当然也可以很方便的扩充 CSysRec 构造函数，用于配置两个通讯接口，从而实现接口的同构冗余设计。

3.3 数据到达通知模型

数据到达通知模型用于数据的读取，以及读取后通知订阅了该数据的领域层功能模块，模型的实现使用了 Observer 模式[2]，具体的类结构关系见图 5。当接口收到数据后，所有订阅了该接口数据的对象都会得到通知，而接口并不需要确切的知道接收对象的具体信息。

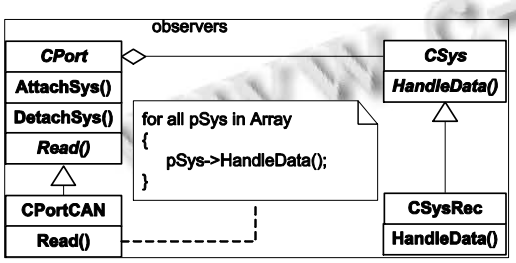

图 5 数据到达通知模型类图

CPort 中的接口 AttachSys 和 DetachSys 用于订阅和退订数据到达通知。CPortCAN 中的接口 Read 则实现了数据读取功能，一般作为独立的线程运行，当 Read 读到数据后，通过 CSys 的接口 HandleData，通知订阅了该数据的功能模块进行处理：

```
//订阅
void CPort::AttachSys(CSys *pSys) {
  m_ArraySys.push_back(pSys);
}
//取消订阅
void CPort::DetachSys(CSys *pSys) {
  vector<CSys*>::iterator it;
  it=find(m_ArraySys.begin(),m_ArraySys.end(),pSys);
  if (it!=m_ArraySys.end()) m_ArraySys.erase(it);
}
void CPortCAN::Read(void){
  int size;
  char buffer[100];
  vector<CSys*>::iterator it;
  //判断是否结束任务
  while (IsContinue())
  {
    //调用底层驱动接口，读取数据
    size=ReadCAN(buffer);
    for (it=m_ArraySys.begin();
      it!=m_ArraySys.end();it++)
    //通知订阅者
    (*it)->HandleData(buffer,size);
  }
}
```

通过以上接口的配合，实现了数据到达通知模型。对 CPort 而言，它只以字节流的形式读取数据，并不关心数据的具体格式，而订阅了该数据的领域层功能模块则负责具体数据的解析。CPort 无需知道实际使用数据的功能模块，以及到底有多少个功能模块订阅了该数据，而领域层的各功能模块也可以任意订阅感兴趣的接口数据，甚至可以订阅多个接口数据，这就实现了领域层和数据源层的数据读取解耦。

3.4 其它

虽然统一接口模型提供了一致的数据发送接口，但 CSysRec 中的接口 WriteData，明确使用了发送参数 CHeadParamCAN，从而影响了不同接口的可替换性。例如 CSysRec 的通讯接口由 CAN 网络改成串口，则 WriteData 代码也需要修改，把发送参数 CHeadParamCAN 修改为 CHeadParamCOM。这个问题很难从根本上解决，因为硬件设计的不同，必然会带来相应代码的修改，我们能做的只是如何使修改尽量限定在一定的范围内，或者尽量使修改保持一致，而这也正是设计模式所要解决的问题。

同时也可以对以上两个模型进行扩展，例如只要在 CSys 中增加设置 CPort 指针的接口，就可以轻松实现相同接口的动态配置和同构冗余，你甚至可以从 CPort 派生 CPortFile，直接把功能模块发送的数据写入文件，从而方便软件的调试和测试。在做到了数据源层和领域层的解耦后，这些功能都可以扩展运用、灵活实现。

在使用上面两个模型的同时，通常也会使用 Adapter 模式[2]或 Facade 模式[2]对底层驱动提供的接口

(下转第 108 页)

解决了 IC 卡必须定点充值的问题；在 1M 带宽的以太网络中，服务器向 8 台自动零售传媒系统远程传输 80M 高清视频共花费 1 小时，大大的减少了媒体更新时间；全新的远程充值功能在 5 分钟内即可完成；流畅运行 1080p 的.mkv 格式文件。

表 2 自动售货机性能比较

售货机	IC 卡充值时间（每天）	80M 视频更新时间（共 8 台）	远程充值	视频处理能力
自动零售传媒系统	24 /小时	1 小时	5 分钟	1080P
传统的高端自动售货机	8/小时（定点）	数天	无	低于 720p

系统在一个月中不间断运行，故障率低，销售正常，远程控制媒体传输和播放以及 IC 卡远程充值和自助充值均能实现，解决了传统自动售货机在媒体处理能力和 IC 卡自助服务充值方面存在的问题，支持自动售货机通讯协议 MDB 接口和以太网访问，方便用户使用和运营商远程管理，具有很好的市场前景。

参考文献

1 孔德强,蒋存波.基于 ARM 和_C_OS_的自动售货机系统软件设计;电脑编程技巧与维护, 2010,6.

2 陈小刚,吴志国,朱成健,李庆武.自动售货机智能多媒体控制单元的研究与设计;计算机系统应用, 2010 年.

3 Gu Hong. Qiao Shuang. Tian jiang. A wireless vending machine system based on GSM. Intelligent Control and Automation. WCICA . The Sixth World Congress on 2006.

4 Rusdiansyah A. Tsao DB. An integrated medel of the periodic delivery problems for vending-machine supply chains. Journal of Food Engineering. 2005, (3).

(上接第 175 页)

进行封装，使用 Factory 模式[2]创建一系列相关的对象，甚至你可以定义自己的抽象驱动层来达到接口模块的跨平台通用性，这些都是对以上两个模型的进一步扩充和完善。

当然这两个模型也不是能够随意使用的，通常在增加间接层获得灵活性和扩展性的同时，也会使设计变得更复杂并牺牲一定的性能。所以只有当你的系统使用了多种通讯接口，或者你的系统平台存在变更的可能，应用上面的两个模型才会体现价值。

4 结语

本文提出的两个模型是通讯接口设计的完整解决方案，已成功应用于多个项目的软件开发，在接口功能模块的复用，领域层和数据源层的解耦，以及提高代码质量上取的了很好的效果。当然如何更有效的优化两个模型，使之能适应更广范围的应用也是日后要努力的方向。

参考文献

1 Fowler M. 企业应用架构模型.北京:机械工业出版社,2010.

2 Erich G, Richard H, Ralph J, et al. Design Patterns:Elements of Reusable Object-Oriented Software.北京:机械工业出版社,2005.

3 Venners B. How to use design patterns. http://www.artima.com/lejava/articles/gammadp.html, 2005.

4 Robert C. Algile Software Development: Principles, Patterns, and Practices.邓辉,译.北京:清华大学出版社, 2003.

5 Alan S. Design Patterns Explained: A New Perspective on Object-Oriented Design.徐言声译.第 2 版.北京:人民邮电出版社, 2006.

6 Lippman SB, Lajoie J. C++Primer.北京:中国电力出版社, 2003.

7 Bosch. CAN Specification Version2.0. Bosch,1991. http://www.zlgmcu.com.