

Rhapsody 实时软件框架适配器设计和实现

张寅生¹, 庄丽葵², 王彪¹, 曹云峰²

(1. 南京航空航天大学 自动化学院, 江苏 南京 210016;

2. 南京航空航天大学 高新技术研究院, 江苏 南京 210016)

摘要:使用 UML 进行基于框架的实时开发是当今软件发展的一个热点。为了使开发的应用软件支持多平台, Rhapsody 实时软件框架将嵌入式操作系统的概念抽象出来, 采用了基于抽象操作系统的层次结构。而 Rhapsody 实时框架中所使用的抽象操作系统和适配器是抽象工厂设计模式在操作系统领域内的一次经典的应用。具体地, 通过针对具体的嵌入式 Vxworks 操作系统给出其 Rhapsody 实时软件框架适配器实现, 并结合一个简单的例子来说明, 抽象操作系统和适配器给 Rhapsody 实时软件框架带来了模块化、可移植性、可复用性等多方优点。

关键词:实时嵌入式软件开发; Rhapsody; 抽象操作系统; 适配器

中图分类号: TP31

文献标识码: A

文章编号: 1673-629X(2011)04-0033-04

Design and Implementation of Adapter in Real-Time Software Framework of Rhapsody

ZHANG Yin-sheng¹, ZHUANG Li-kui², WANG Biao¹, CAO Yun-feng²

(1. College of Automation, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China;

2. High-tech Research Institute, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

Abstract: The framework-based approach to real-time development with UML is one important aspect of computer software progress. In order to make the development of application software platform-independent, the real-time software framework of Rhapsody uses a layered structure based on the abstraction of the operating system and the concept of embedded operating system is abstracted. The concept of the abstraction of the operating system and the adapter is a classic application of the abstract factory design pattern within the field of operating systems. Then give the implementation of the adapter for the concrete embedded OS Vxworks. Finally, the advantages of real-time framework of Rhapsody with the abstraction of the operating system and adapter, such as modular, portability and reusability, are introduced by a simple example.

Key words: real-time embedded software development; Rhapsody; OSAL; adapter

0 引言

随着嵌入式软件产品推出时间的越来越短, 嵌入式软件的开发人员日益感到软件复杂度越来越大。由于内在地支持了对系统的抽象、分层和复用, 面向对象技术能够很好地控制系统的复杂性, 因此其在嵌入式领域得到越来越广泛的应用^[1]。而框架最大限度地体现了面向对象的重要思想。框架可以通过代码重用来减少重复工作, 并且可以提供一套为嵌入式和实时应用专门选择和优化的设计模板。

但是, 要想将面向对象的方法应用到嵌入式实时

系统的上层软件的设计中去还面临着一些挑战, 其中很主要的一点, 就是底层实时操作系统没有能够很好地向上层应用提供支撑, 所以即使上层软件采用了面向对象的方法, 但整个系统代码的模块化、可移植性、可复用性也难有提高。

在过去的几十年中, 有不少计算机工作者和专家的研究是针对实时嵌入式系统或者软件框架的。其中, 美国 I-Logix 公司的产品 Rhapsody 是业界惟一的完全基于 UML 的实时应用软件开发环境。统一建模语言 UML 做为标准的图形建模语言, 是使得从分析到编码整个开发过程更容易的软件自动化工具的建模语言基础^[2,3]。对于实时嵌入式系统也是如此, 因为通常可以利用 UML 的状态图和活动图来描述实时嵌入式系统动态行为, 而使用它们可以很自然地实现代码自动生成、测试、分析和验证过程^[4]。

Rhapsody 的实时框架结构如图 1 所示, 包含三个

收稿日期: 2010-08-07; 修回日期: 2010-11-18

基金项目: 总装武器装备预研基金重点资助项目; 江苏省“333”人才基金资助项目(P0994GX)

作者简介: 张寅生(1986-), 男, 江苏淮安人, 硕士研究生, 研究方向为飞行控制与 UML 建模; 王彪, 副教授, 研究方向为飞行控制与仿真。

层次,共五个主要部分,对象执行框架里是 UML 模型执行的基本结构,包括状态机的执行,管理线程等。对象间关系模式管理模型对象之间的关系,包括一对一关系、一对多关系和多对多关系。动画调试框架提供基于模型的调试,比如基于状态图的断点设置、查看信息和捕捉对象间消息映射等等^[5]。特别地,其抽象操作系统分为两层:一层为抽象操作系统接口层,它是向上层应用提供一致接口的一层,这一层对所有的具体操作系统来说都是一样的;另一层为操作系统适配器层,它与特定的操作系统相关,不同的操作系统就有不同的适配器,适配器是在具体底层操作系统上对上层抽象操作系统接口的实现。这样,抽象操作系统通过适配器封装具体的底层操作系统来屏蔽掉各类底层操作系统之间的差异,从而实现向上层应用提供统一的、面向对象的接口,使上层应用更容易支持多平台^[6,7]。

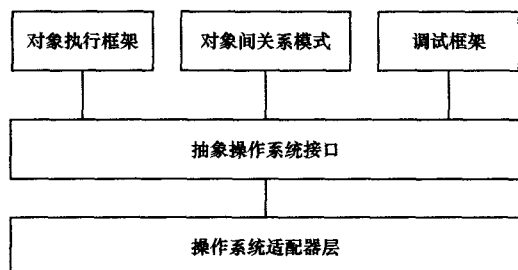


图 1 Rhapsody 实时框架结构

文中首先阐述了 Rhapsody 实时框架中抽象操作系统和适配器相关设计原理,然后针对具体的嵌入式 Vxworks 操作系统给出了其 Rhapsody 实时框架适配器实现。

1 Rhapsody 实时框架中抽象操作系统和适配器分析

Rhapsody 实时框架中的抽象操作系统和适配器是抽象工厂设计模式在操作系统领域内的一次经典的应用。

1.1 抽象工厂设计模式

抽象工厂模式是 GOF(四人组(Gang of four),他们四人合著了《设计模式》一书)提出的 23 种设计模式的一种,属于创建型模式。抽象工厂设计模式的概念:“提供一个创建一系列相关或相互依赖对象的接口,而无需指定它们具体的类”。可以这样理解这句话:一方面,要求软件应该带有一定的灵活性,从而可以适应外部需求可能发生的变化;另一方面,灵活性往往会带来软件内部的复杂性,因此,必须把这种复杂性封装起来,并为外界提供一组简单而又稳定的接口。这样带来两个方面的好处:一方面,充分实现了系统的可插入性和可扩展性,因为软件系统中实体模块的通信表现为接口或者抽象类的通信,变化的部分可以通

过继承接口或实现抽象类进行了封装;另一方面,系统高层部分会表现出很好的稳定性和可维护性,因为底层需求的变动不会波及到其它子系统或模块。

抽象工厂设计模式的结构见图 2。

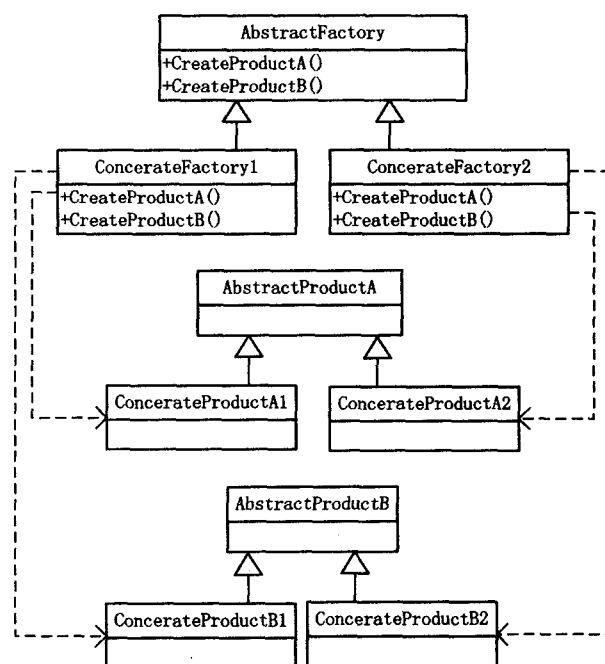


图 2 抽象工厂设计模式结构(UML 示意图)

由图 2 可知,在抽象工厂模式中主要有四个成员:

(1) 抽象工厂 AbstractFactory。它是抽象工厂模式的核心。对于用户,它提供一个统一稳定的接口,用户完全不需要知道系统内部具体是如何实现的;对于继承并实现它的子类,它提供了一个统一的标准。

(2) 抽象产品 AbstractProductA, AbstractProductB。它的主要作用就是向具体产品 ConcreteProductA 和 ConcreteProductB 提供统一的接口。

(3) 具体工厂 ConcreteFactory1 和 ConcreteFactory2。它的主要作用就是生产具体产品。用户直接调用具体工厂来创建具体产品的实例,含有选择产品对象的逻辑。

(4) 具体产品 ConcreteProductA1, ConcreteProductA2, ConcreteProductB1, ConcreteProductB2。它的主要功能和职责是继承抽象工厂并实现自己的功能。整个抽象工厂模式所创建的所有产品对象都是某一个具体产品类的实例,这个实例便是用户的最终需求。

1.2 Rhapsody 实时框架中的抽象工厂模式应用

Rhapsody 实时框架中抽象操作系统有唯一的抽象工厂 OSFactory, OSFactory 负责生产抽象产品,比如 OSThread(抽象线程类)、OSMessageQueue(抽象消息队列类)等。同时,对于每一个底层操作系统都有一个对应的具体工厂 XXXFactory,来负责生产属于该底层操作系统的诸如 XXXThread、XXXMutex 等具体产品。

就是具体工厂用于生产的具体模具;另外,具体产品类又实现了对底层操作系统的 API 函数接口的封装,所以,又统称这些具体产品类为适配器类。抽象工厂类是具体工厂类的父类,抽象产品类是适配器类的父类。另外,通过一个全局函数 TheFactory(),来实现将抽象工厂转变为具体工厂。

2 Rhapsody 实时框架针对具体 Vxworks 操作系统适配器实现

为了具体说明 Rhapsody 实时框架针对具体操作系统如何进行封装适配,以 Vxworks 操作系统为例来说明其抽象操作系统的适配器实现。首先,假设抽象工厂 OSFactory 仅生产一个抽象产品 OSMutex,它是用于保护临界区的。抽象工厂 OSFactory 实际上是一个抽象基类,它是供每一个特定操作系统的具体工厂继承的,比如相应于 Vxworks 操作系统的具体工厂 VxworksOSFactory,就是从抽象工厂 OSFactory 继承而来,这样,OSFactory 便隐藏了 Vxworks 操作系统如何实现系统调用的具体细节^[8,9]。定义抽象工厂 OSFactory 如下:

```
class OSFactory {
public:
    //纯虚函数,创建用于保护临界区的互斥信号量,该抽象操作用于创建抽象产品。
```

```
    virtual OSMutex * createOSMutex() = 0;
};
```

上层应用需要的所有操作系统服务在抽象工厂 OSFactory 中都有定义,与每一特定操作系统对应的特定工厂都必须从该抽象工厂 OSFactory 继承而来,并实现在 OSFactory 中定义每个抽象操作。另外,应用是如何与 OSFactory 相联系的呢?

对于每一个特定操作系统的特定工厂,都定义了一个静态全局指针 XXXOSF 来表示这个特定操作系统的特定工厂。比如对于 Vxworks 操作系统的特定工厂 VxworksOSFactory,就定义一个静态全局指针 VxworksOSF,来表示 VxworksOSFactory。采取了这种方式,就可以保证对于每个特定操作系统只有一个特定工厂的实例存在。具体地,定义一个全局函数 theOSFactory,如下外部声明:

```
extern OSFactory * theOSFactory();
```

该函数用来初始化静态全局指针 XXOSF。比如对 Vxworks 操作系统,有如下初始化:

```
static VxworksOSFactory * VxworksOSF = NULL;
OSFactory * theOSFactory()
{
    if (NULL == VxworksOSF)
        VxworksOSF = new VxworksOSFactory;
```

```
    return VxworksOSF;
```

```
}
```

这样初始化之后,便可以确保只有一个特定 VxworksOSFactory 的实例存在。这样,便可以通过使用 theOSFactory() 返回来的指针来调用在相应工厂中定义的任何方法,如创建一个操作系统实体。比如,使用下面的语句便可以创建一个互斥信号量:

```
theOSFactory->createOSMutex();
```

定义互斥访问抽象类 OSMutex 如下:

```
class OSMutex {
public:
    virtual void lock() = 0; //上锁信号量。
    virtual void free() = 0; //解锁信号量。
    virtual ~ OSMutex() {} //析构函数;
};
```

特定工厂 VxworksOSFactory 类继承了抽象工厂 OSFactory。它在具体操作系统 Vxworks 环境下具体实现在抽象工厂 OSFactory 定义的所有抽象操作,定义 VxworksOSFactory 如下:

```
class VxworksOSFactory: public OSFactory {
    //OSFactory 隐藏了特定操作系统实现同步和任务管理等的实现机制;
public:
    virtual OSMutex * createOSMutex()
    //实现互斥信号量的创建;
    { return (OSMutex *) new VxworksMutex(); }
    //在具体操作系统 Vxworks 环境下创建具体产品。
};
```

这里 VxworksMutex 就是一个具体的适配器,它继承 OSMutex 抽象类,用来在具体操作系统 Vxworks 环境下实现对临界区的互斥操作,即用互斥信号量来实现对临界区的互斥操作,适配器 VxworksMutex 封装了底层 Vxworks 操作系统提供的系统调用。

定义 VxworksMutex 如下:

```
class VxworksMutex: public OSMutex {
    pthread_mutex_t HdMutex; //指向表示互斥信号量的对象;
public:
    VxworksMutex(); //构造函数,创建互斥信号量对象;
    ~ VxworksMutex(); //析构函数
    void lock();
    void free();
    void * getHandle() { return (void *) &HdMutex; }
};
```

具体实现如下:

(1) 构造函数:

```
VxworksMutex:: VxworksMutex()
{
    pthread_mutexattr_t attri;
    attri._mutexkind = PTHREAD_MUTEX_RECURSIVE_NP;
```

```
pthread_mutex_init(&HdMutex,&attri);
```

(2)析构函数:

```
VxworksMutex::~VxworksMutex()
```

```
pthread_mutex_destroy(&HdMutex);
```

(3)lock():

```
void VxworksMutex::lock()
```

```
pthread_mutex_lock(&HdMutex);
```

(4)free():

```
void VxworksMutex::free()
```

```
pthread_mutex_unlock(&HdMutex);
```

3 Rhapsody 实时框架适配器应用实例及其优点

举一个简单的例子,说明 Rhapsody 实时框架适配器的具体应用。假设 Rhapsody 应用需要进行临界区互斥操作。利用上面设计的抽象操作系统和相应适配器,用下面的代码便可实现临界区的互斥操作:

```
OSMutex * mymtx;  
mymtx->theOSFactory()->CreateOSMutex();  
mymtx->lock();  
临界保护区  
mymtx->free();
```

从前面的代码中可看出,使用抽象操作系统和适配器的 Rhapsody 实时框架有以下优点^[10~12]:

(1)可以看出,由于使用了抽象操作系统和适配器,上述代码是跟平台无关的,因此,所有的 Rhapsody 应用对任何的平台来说都是独立一致的,是可以完全移植的;

(2)抽象操作系统很好地支持以抽象为中心的、可重用的设计。这样,当 Rhapsody 开发环境需要支持一种新的 OS 时,只需编写相应适配器,用相应的 OS 所提供的各种 API 实现在抽象操作系统里所定义的每一个抽象操作,而不用去修改应用的其他任何部分就可以支持新的 OS 了。这样,使 Rhapsody 的嵌入式实时开发环境便更具有开放性,有利于使它很好地支持新的 OS;

(3)加入了抽象操作系统和适配器后,应用可以大规模重用代码。抽象操作系统把 Rhapsody 实时软件框架中与平台相关部分隔离开来,这样需要支持一

个新的操作系统时,只需修改抽象操作系统有关部分就可以了,与平台无关部分就不用进行任何修改,从而很明显地提高了代码的重用性;

(4)伸缩性是抽象操作系统另一个特点。不同的上应用可能对系统调用的需求也不同,这时,可以根据应用的具体需求,构造出与之相应的抽象操作系统来。一句话,应用需要怎样的系统调用,就可以构造怎样的抽象类来。比如说,有的应用不需要文件系统,那么就可以构造出不包含与文件系统相关的系统调用的抽象操作系统来,这样便使得抽象操作系统具有很大的伸缩性。

4 结束语

文中,阐述了 Rhapsody 实时框架中抽象操作系统和适配器相关设计原理,然后针对具体的嵌入式 Vxworks 操作系统给出了其 Rhapsody 实时框架适配器实现。从以上论述可知,使用抽象操作系统和适配器的 Rhapsody 实时框架可以很方便地实现嵌入式软件代码的模块化,并大大提高应用的可移植性和可复用性。

参考文献:

- [1] Booch G. Object-oriented Analysis and Design with Applications[D]. USA: Benjamin/Cummings, 1999.
- [2] Douglass B P. Real-time UML[D]. USA: Addison Wesley, 1998.
- [3] 陈燕. 基于 UML 的嵌入式系统系统级设计方法研究[D]. 上海: 复旦大学, 2005.
- [4] 张莉,葛科,王云,等. UML 软件开发过程和支持环境研究[J]. 北京航空航天大学学报, 1998(4): 407-410.
- [5] 左光,高晓峰. 基于 UML 的实时框架的分析与应用[J]. 半导体技术, 2002, 27(4): 42-47.
- [6] Jiang Bo, Chen Ying, Hu Tao. OS abstraction layer for the reuse of USB stack and device drivers[J]. Journal of Beijing University of Technology, 2003(2): 241-246.
- [7] 鲍国超,柳长安,芦东昕. 面向对象的操作系统抽象层和运行支撑层[J]. 计算机工程, 2003, 29(22): 59-61.
- [8] 尚海忠,朱培彦,王霞,等. 操作系统抽象层——一种支持跨平台的新技术[J]. 计算机工程, 2002, 29(2): 109-111.
- [9] 何先波. 嵌入式软件开发环境中操作系统抽象层的设计与实现[J]. 成都大学学报, 2004, 23(4): 18-22.
- [10] 侯志光. 基于 OXF 框架构建微机保护嵌入式实时多任务软件平台[D]. 天津: 天津大学, 2007.
- [11] 刘紫嫣. 基于 UML 的水下机器人运动控制系统研究[D]. 哈尔滨: 哈尔滨工程大学, 2007.
- [12] 文武红. Rhapsody 在柴油机电子控制上的应用研究[D]. 太原: 中北大学, 2005.