Ada-I:支持保护对象继承的Ada95扩充语言*

陆嘉 温冬婵 王鼎兴

(清华大学计算机科学与技术系 北京 100084)

E-mail: wdc@mail.Tsinghua.edu.cn

摘要 面向对象并发程序设计语言能够帮助程序员利用面向对象技术编写并发程序,从而获得面向对象技术给软件开发带来的种种好处.然而,由于继承异常现象的存在,影响了并发程序设计语言引入继承特性.Ada95语言是支持并发程序设计的面向对象语言之一,但并不支持保护对象的继承.Ada-I语言在Ada95语言的基础上进行扩充,从而支持保护对象的继承,并且避免了继承异常现象的出现.

关键词 Ada95,保护对象,继承,继承异常,面向对象并发程序设计语言.

中图法分类号 TP312

目前,面向对象技术在计算机科学的各个领域上有非常广泛的应用,出现了许多面向对象并发程序设计语言,如ABCL/1,Presto,ACT++等等.但由于继承异常(inheritance anomaly)现象,一些面向对象并发程序设计语言不能很好地支持代码复用,破坏了对象的继承性.针对该问题,许多学者提出了解决方案[1~4].

Ada95语言是少数几个能支持并发程序设计且广泛应用的高级语言之一.它定义了任务(task)类型,保护(protected)类型,以及一系列支持并发程序设计的语法语义.在Ada95中,并发程序以任务对象和保护对象为单位,通过任务对象和保护对象的并发处理,协同完成某一特定的计算和处理任务.我们将Ada95中的任务对象和保护对象统一称为并发对象.

虽然Ada95已经在顺序程序设计方面适应了面向对象程序设计的要求,支持继承、多态、封装等典型面向对象特性,在并发程序设计方面,Ada95中的并发对象也具有了一些面向对象特性,如封装性,但是,Ada95并不支持并发对象的继承,因此,Ada95还不能满足面向对象并发程序设计的要求.

因为Ada95中任务对象的语言描述框架与面向对象技术中的对象描述框架差异较大,如任务对象之间可以通过数据共享交互,面向对象技术中的对象只通过消息传递交互,所以任务对象不易抽象出面向对象技术中方法的概念.而Ada95中的保护对象的语言描述框架与面向对象技术中的对象类似,比较容易引入继承的特性.因此,在本文中我们主要讨论保护对象的继承问题.

在本文中,我们基于Ada95语言提出了一个支持保护对象继承的扩充语言Ada-I,并在该语言中消除了继承异常现象.

1 继承异常现象

继承是面向对象程序设计的关键概念之一,是在顺序面向对象程序设计中解决代码复用问题的常用方法.当前,虽然许多面向对象技术可以很方便地引入到并发程序设计中,进行面向对象并发程序设计,但是继承却不能按照顺序面向对象程序设计中的使用方法简单地引入并发程序设计,因为这样会导致继承异常现象的出现,也就是说,子类不能进行增量式声明,也即只声明父类中不具有的变量、方法等,而其他的变量、方法从其继承的

* 本文研究得到国防科工委国防科技预研项目基金(No.15.5.2)资助.作者陆嘉,1971年生,博士,讲师,主要研究领域为并发面向对象语言,并行/分布处理技术.温冬婵,女,1946年生,副教授,主要研究领域为并发面向对象语言,并行编译技术.王鼎兴,1937年生,教授,博士生导师,主要研究领域为并行/分布处理技术.

本文通讯联系人:温冬婵,北京 100084,清华大学计算机科学与技术系

本文1998-12-21收到原稿,1999-04-19收到修改稿

© 中国科学院软件研究所 http://www.jos.org.cn

父类中进行代码复用,无需再作声明.当出现继承异常现象时,子类不仅要声明父类中不具备的变量、方法,而且不得不对父类中定义的某些方法,甚至所有方法在子类中重新定义.

在面向对象并发系统中,由于可以同时有多个消息访问一个并发对象,对象之间的消息传递具有不确定性,所以,为了保证并发对象的完整性和并发对象中数据的一致性,每个并发对象对接收到的消息通常按照运行条件检查、运行相应方法以及运行后相应的状态转换这3个步骤来处理.当并发对象的当前状态不满足所接收消息的运行条件时,并发对象则不处理该消息,直到并发对象的状态满足该消息条件为止.满足某个接收消息运行条件的并发对象的所有状态称为该消息的可纳状态(acceptable states).所有消息的可纳状态的总和称为该并发对象的可纳状态集.

Matsuoka与Yonezawa在文献[1]中分析了并发对象继承异常现象出现的原因,认为并发对象继承异常问题的出现是因为子类对象在继承父类对象方法时增加的新方法改变了父类对象的可纳状态集的情况,又因为与处理可纳状态相关的运行条件检查和状态转换是和对象的方法紧密相关的,从而导致需要重新改写父类对象的方法,以满足新的运行条件检查和状态转换的要求.文献[1]将可纳状态的改变分成以下3类:(1) 可纳状态集的重新划分;(2) 可纳状态集的历史相关;(3) 可纳状态集的扩充与修改.

下面,我们以类C++的面向对象并发程序设计语言对继承异常现象出现的原因作一简单介绍.

1.1 可纳状态集的重新划分

由于每个并发对象的运行可以考虑为一系列的状态转换,而每一个状态是由并发对象的所有变量当前的取值构成的.我们可以根据并发对象每一个方法的可接收条件,也就是相对应消息的可纳状态,形成可纳状态集的一个划分.

以有界队列Buffer为例,有界队列有两个基本方法,get()从队列中取一个元素,put(x)将元素x放入队列.显然,当队列元素个数为0,即队列空时,get()不能运行,当队列满时,put(x)不能运行.于是,我们可以得到有界队列可纳状态集的一个划分:empty,partial,full,相应为队列空、队列半满、队列满.

我们将有界队列Buffer作为基类,由其派生一个新的子类SubBuffer.在该子类中,我们增加一个新的方法get2(),即从队列中取出两个元素.显然,当队列中元素个数为1时,get2()不能运行,而get(),put(x)依然能够运行.我们可以看出子类SubBuffer的可纳状态集与父类Buffer相比多了一个get2()的可纳状态one,因此,子类SubBuffer对父类的可纳状态集进行了新的划分:empty,one,partial,full,如图1所示.

```
class Buffer {
    int in=0, out=0;
    behavior {
    empty ={put(x)};
        //在 empty 状态下可运行 put(x)
    partial ={put(x),get};
        //在 partial 状态下可运行 put(x),get
    full={get};
        //在 full 状态下可运行 get
    }
    method put(x) {
        code for put;
        if ((in-out)==MAX) become full;
            else become partial;
        }
    method get {
        code for get
        if ((in-out)==0) become empty;
            else become partial;
    }
}

class SubBuffer: Buffer {
    behavior {
    //empty 可以从父类继承
      partial={put(x),get,get2};
      full={get,get2};
      one={put(x),get};
    }
    method get2 {
        code for get2;
        if ((in-out)==0) become empty;
        else if ((in-out)==1) become one;
            else become parial;
    }
    method put(x) {
        code for put;
        if ((in-out)==MAX) become full;
        else if ((in-out)==1) become one;
            else become partial;
    }
    method get {
        code for get;
        if ((in-out)==0) become empty;
        else if ((in-out)==1) become one;
            else become parial;
    }
}
```

Fig.1 Repartition of acceptable states set

图1 可纳状态集的重新划分

© 中国科学院软件研究所 http://www.jos.org.cn

我们可以看到,子类 SubBuffer 重写了父类中已有的方法,而不能直接继承.这就出现了继承异常现象.

1.2 可纳状态集的历史相关

可纳状态集的历史相关指子类派生的新方法与父类方法运行的次序相关.例如,我们以有界队列 Buffer 为基类派生一个新类 NewBuffer.这个新类中增加了一个新的方法 gget(),它的运行条件是在它运行之前有连续两个 put(x)方法运行.

如图2所示,为了描述 gget()的可纳状态,我们需要重新描述父类中的 get(),put(x)等方法,从而导致出现继承异常现象.

```
class NewBuffer:Buffer {
    int put_his=0;
    behavior {
        //empty 可以从父类继承
        partial={put(x),get};
        full={get};
        put2_full={get,gget};
        put2_partial={put(x),get,gget};
    }
    method gget {
        code for gget;
        put_his=0;
        if ((in-out)==0) become empty;
        else become parial;
    }
    method put(x) {
        code for put;=put_his ++
        if (put_his>=2)
            if ((in-out)--MAX) become put2_full;
            else become put2_partial;
        else
            if ((in-out)==MAX) become full;
            else become partial;
    }
    method get {
        code for get;
        put_his=0;
        if ((in-out)==0) become empty;
        else become parial;
    }
}
```

Fig. 2 History relation of acceptable states set

图2 可纳状态集的历史相关

1.3 可纳状态集的扩充与修改

对于子类 SubBuffer 而言,它并没有扩充或修改父类的可纳状态集,而只是改变了可纳状态集的划分.如果我们需要从基类 Buffer 派生出一个新的子类 LockBuffer,这个子类引入一个新的条件:是否加锁.如果已经加锁的话,方法 get(),put(x)不能运行;如果没有加锁,则能运行.这样,方法 get()和 put(x)的可运行条件就增加了,也就是说,相应的可纳状态集扩充了.

因为如果我们采用和图1、图2相同的语言描述方法,将不会出现继承异常现象.为了说明问题,我们采用新的语言描述方法,如图3所示.在这种描述方法下,父类的方法需要重新定义,出现了继承异常现象.

```
class Buffer {
    int in=0, out=0;
    method put(x) when
        ((in-out)<MAX) {
        code for put;
    }
    method get when ((in-out)>0) {
        code for get;
    }
}

class LockBuffer: Buffer {
    int locked=0;
    method lock {
        locked = 1;
    }
    method unlock {
        locked=0;
    }
    method put(x) when
        (((in-out)<MAX) &&!locked) {
        code for put;
    }
    method get when
        (((in-out)>0) &&!locked) {
        code for get;
    }
}
```

Fig. 3 Modification of acceptable states set

图3 可纳状态集的修改与扩充

从这个例子我们可以看到,不同的语言描述方法,出现继承异常现象的原因也是不同的.

© 中国科学院软件研究所 http://www.jos.org.cn

2 保护对象

保护对象是 Ada95语言新增加的语言成分,用来描述专门为其他对象服务的并发对象.保护对象的结构与程序包和任务的结构相似,但其中的说明和体不同.保护对象的说明部分给出访问权限,而体则给出实现的细节.与程序包和任务不一样的是,保护对象的体不能说明任何数据,而只能说明子程序和实体.数据必须放在私有部分说明.

保护对象可以由保护类型声明.保护类型与保护对象之间的关系与面向对象语言中的类与对象实例相同.保护对象的操作分为保护子程序和保护入口,而保护子程序又分为过程和函数.过程、函数和入口分别由关键字 procedure,function 和 entry 说明.过程可以用任意方式访问私有数据,函数则只能读私有数据,而入口可以任意访问私有数据,但它带有入口限制条件,只有在满足限制条件的情况下,保护对象才能运行相应的入口体.

3 我们的扩充语言方案

Ada95在顺序程序设计方面支持面向对象程序设计,支持继承、多态、封装等典型的面向对象特性.而在并发程序设计方面,Ada95不能全面支持面向对象程序设计.任务对象、保护对象等支持封装,但不支持继承.由于保护对象的语言描述特性与典型的面向对象程序设计语言描述特性相似,例如,数据的私有、多种操作的独立描述.因此,我们在 Ada95语言的基础上,针对保护对象增加支持继承的特性,设计了 Ada-I 语言,并且在这种语言描述方法的基础上,消除了继承异常现象.

3.1 保护类型声明

Ada-I 中的保护类型声明与 Ada95中的类似,由类型说明和类型体构成.在类型说明中声明保护类型的可调用操作和内部数据.在类型体中声明各个操作的具体实现.

保护类型可以派生子类型.为了使 Ada-I 语言简单易懂,我们支持单继承,而不支持多继承.子类型从父类型中继承父类型的变量和操作.子类型只定义不同于父类型的那些变量和操作,一方面,子类型可以增加父类型中没有的变量和操作,另一方面,子类型还可以重新定义父类型的变量和操作.

子类型声明在类型说明中采用关键字 inherit 来说明该子类型是从某个父类型派生出来的.同时,在类型说明中声明父类型中没有的变量和操作,以及需要重新定义的操作.

子类型的类型说明中所提及的操作的具体实现在类型体中加以声明.

子类型声明的语言描述如下:

```
protected type〈子类型标识符〉inherit〈父类型标识符〉is ——类型说明
   〈父类型中未定义的操作〉
   〈子类型重新定义的父类型已有操作〉
   ...
private
   〈父类型中未定义的数据〉
   ...
end〈子类型标识符〉
```

3.2 保护类型操作描述

由于 Ada95中原来的保护对象操作的描述方法与图3中的描述法相类似,因此会导致继承异常现象的发生.为了消除继承异常现象,我们将改变保护对象原来的描述方法.

因为保护对象中的过程操作和入口操作具有一定的相似性,两者的差别就在于过程操作没有运行条件检查,而入口操作需要在运行前进行运行条件检查,所以,我们用不进行运行条件检查的入口操作代替过程操作.于是,在 Ada-I 中保护对象就有函数和入口这两类操作.

同时,我们将保护对象原来的入口操作分成两个部分:运行条件检查和操作运行动作.在 Ada-I 中分别用关

© 中国科学院软件研究所 http://www.jos.org.cn

键字 condition,action 加以说明.我们可以将 condition 和 action 理解成保护类型入口操作的两种属性.属性 condition 是一个条件表达式,而属性 action 是一系列动作.

Ada-I 中保护对象入口操作的基本描述结构如下:

```
entry〈标识符〉(参数表) is
begin
    condition
    begin
        〈入口条件表达式〉
    end condition;
    action
    begin
        〈操作运行动作〉
    end action;
end;
```

当 Ada-I 中的保护对象入口被调用时,保护对象首先检查入口条件表达式,如果条件表达式结果为真,继续执行 action 中描述的运行动作;否则,该入口调用排队等待.其中的 condition 可以缺省.Condition 缺省表示入口条件永为真,也就是不作入口条件检查.用缺省 condition 描述的入口操作相当于 Ada95中的过程操作.

Ada-I 中保护对象的函数操作与 Ada95中的一样,没有运行条件检查,并且多个函数操作之间可以并发运行.

以有界队列为例,其描述如图4所示.

```
protected type Buffer is
  entry put(X: in Item);
  entry get(X: out Item);
private
  A: Item_Array;
  I,J: Index:=0;
  Count: Integer range 0..MAX:=0;
end Buffer;

protected body Buffer is
  entry put(X: in Item) is
  begin
    condition
    begin
      Count<MAX
    end condition;
    action
    begin
      A(I):=X;
      I:=I+1;
      Count:=Count+1;
    end action;
  end put;
  entry get(X: out Item) is
  begin
    condition
    begin
      Count>0
    end condition;
    action
    begin
      X:=A(J);
      J:=J+1;
      Count:=Count-1;
    end action;
  end get;
end Buffer;
```

Fig.4 Bounded buffer

图4 有界队列

3.3 操作代码复用

为了实现操作代码的复用,我们可以在子类型操作中直接引用父类型的操作.于是,我们引入描述父类型操作和子类型操作关系的关键字 super,用以表示父类型.

Super 关键字可以用在某一入口操作的 condition 的声明部分,也可以用在 action 的声明部分.它表示子类型的入口操作在 condition 和 action 中分别继承相应的父类型中入口操作的相应部分.

在 condition 部分使用,其语言描述如下:

© 中国科学院软件研究所 http://www.jos.org.cn

super.〈父类型操作标识符〉'condition

在action部分使用,其语言描述如下:

super.〈父类型操作标识符〉'action

在condition部分使用表示继承父类型中某入口操作的condition条件表达式,因此,该表达式可以与其他条件表达式进行一定的布尔运算,即可以进行与、或、非运算,分别用and,or,not表示.

而在action部分使用表示继承父类型中某入口操作的action动作.

在子类型中,如果我们重新定义一个父类型入口操作,在描述过程中需要将condition和action两部分都重新说明.如果缺省condition或action中的一部分,则表示相应部分从父类型中继承.

我们以有界队列Buffer和派生子类型LockBuffer为例.该子类型引入一个条件是否加锁和两个操作:加锁lock和开锁unlock.当队列被锁时,get与put都不能操作,直到队列被开锁.如图5所示,我们在子类型LockBuffer的condition中采用super.get和super.put来分别表示父类型Buffer的get和put的入口条件表达式,重写了get和put的入口条件,用缺省action表达了对父类型的动作代码复用.

```
protected type LockBuffer inherit Buffer is
  entry put(X: in Item);
  entry get(X: out Item);
  entry lock;
  entry unlock;
private
  locked: Boolean:=False;
end LockBuffer;

protected body LockBuffer is
  entry lock is
  begin
     action
     begin
        locked:=True;
     end action
  end lock
  entry unlock is
  begin
     action
     begin
        locked:=False;
     end action
  end unlock
  entry put(X: in Item) is
  begin
     condition
     begin
        super.put(X)' condition and (not
        locked)
     end condition;
  end put;
  entry get(X: out Item) is
  begin
     condition
     begin
        super.get(X)' condition and (not
        locked)
     end condition;
  end get;
end LockBuffer;
```

Fig. 5 Locked bounded buffer

图5 加锁有界队列

我们可以看到,在Ada-I中,子类型可以最大限度地继承父类型中的操作,而不需要重新定义父类型的操作.

3.4 消除继承异常现象

我们在第1节分析了出现继承异常现象的3个原因.Ada-I可以有效地消除由这些原因所引起的继承异常现象,从而提高代码复用的效率.

因为在Ada-I中,保护对象操作的代码被分为两部分:condition和action,而这两部分可以分别进行代码复用,所以由于运行前检查条件变化引起的继承异常现象都可以通过重新定义操作中的condition部分得以解决,无需重新定义操作的动作部分.

可纳状态集的重新划分和可纳状态集的扩充与修改都是由于父类方法在子类中的运行条件发生变化而引起的,因此,如图5所示,我们在Ada-I中可以消除可纳状态集的扩充与修改所引起的继承异常现象.如图6所示,我们可以消除可纳状态集的重新划分所引起的继承异常现象.对于可纳状态集的历史相关引起的继承异常现象,如图6所示,我们可以将子类型中操作的action部分分成两个步骤.第1步,通过super引用父类型操作,第2步,对引入的新变量进行处理.这样,我们可以复用父类型中操作的condition部分代码和action部分代码,而不

© 中国科学院软件研究所 http://www.jos.org.cn

必再将这些代码在子类型中重写一遍.

```
protected type SubBuffer inherit Buffer is
  entry get2(X:out Item; Y: out Item);
end LockBuffer;

protected body SubBuffer is
  entry get2(X:out Item; Y: out Item) is
  begin
    condition
    begin
      Count>1
    end condition
    action
    begin
      super.get(X)' action;
      super.get(Y)' action;
    end action
  end get2;
end SubBuffer;
```

Fig. 6　Bounded buffer for get2

图6　增加get2的有界队列

```
protected type NewBuffer inherit Buffer is
    entry put(X: in Item);
    entry get(X: out Item);
    entry gget(X: out Item);
private
    put_his: Integer:=0;
end NewBuffer;

protected body NewBuffer is
begin
    entry get(X: out Item) is
    begin
        action
        begin
            super.get(X)' action;
            put_his:=0;
        end action;
    end get;
    entry put (X: in Item) is
    begin
        action
        begin
            super.put(X)' action;
            put_his:=put_his+1;
        end action;
    end put;
    entry gget(X:out Item) is
    begin
        condition
        begin
            super.get(X)' condition and put_his>=2
        end condition
        action
        begin
            super.get(X)' action;
        end action;
    end gget;
end NewBuffer;
```

Fig. 7　New bounded buffer

图7　新有界队列

从上面的简单分析和例子我们可以看出,在Ada-I中的保护类型(对象)继承可以消除继承异常现象,或者说可以最大限度地进行代码复用.

4　小　结

面向对象并发程序设计语言能够帮助程序员利用面向对象技术编写并发程序,从而获得面向对象技术给软件开发带来的种种好处.然而,由于继承异常现象的存在,影响了并发程序设计语言引入继承特性.

Ada95语言是支持并发程序设计的面向对象语言之一,但并不支持保护对象的继承.我们在Ada95语言的基础上设计了Ada I语言.它支持保护对象的继承,并且避免了继承异常现象的出现.

Ada-I语言将保护类型的入口操作分为运行条件描述和运行动作两部分.这两部分作为入口操作的两种属性,可以在子类型中加以引用.这为程序员提供了灵活、易用的描述手段以进行面向对象并发程序设计.

参考文献

1　Matsuoka S, Yonezawa A. Analysis of inheritance anomaly in object-oriented concurrent languages. In: Agha G, Wegner P, Yonezawa A eds. Research Directions in Concurrent Object-Oriented Programming. Cambridge, MA: MIT Press, 1993. 107～150

2　Neusius C. Synchronizing actions. In: America P ed. Proceedings of the ECOOP'91. Berlin: Springer-Verlag, 1991. 118～132

© 中国科学院软件研究所 http://www.jos.org.cn

3 Meseguer José. Solving the inheritance anomaly in concurrent object-oriented programming. In: Nierstrasz O ed. Proceedings of the ECOOP'93. Berlin: Springer-Verlag, 1993. 220~246
4 Frolund S. Inheritance of synchronization constraints in concurrent object-oriented programming languages. In: Madsen O ed. Proceedings of the ECOOP'92. Berlin: Springer-Verlag, 1992. 185~196

Ada-I: An Extension Language Supporting Inheritance of Protected Objects Based on Ada95

LU Jia WEN Dong-chan WANG Ding-xing

(*Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract Object-Oriented concurrent languages can help programmers to develop concurrent programs and to gain benefits by using object oriented technology, such as encapsulation and inheritance. But inheritance anomaly influences designing object-oriented concurrent languages. Ada95 is one of the object-oriented languages that support concurrent program designing, but it does not support the inheritance of protected objects. Ada-I language based on Ada95 supports the inheritance of protected objects, and can avoid the inheritance anomaly.

Key words Ada95, protected object, inheritance, inheritance anomaly, obj © 中国科学院软件研究所 http://www.jos.org.cn