基于 UniNet 的对象概念研究*)

周国富[1,2]　余　鹏[2]　袁崇义[2]

(武汉大学计算机学院　武汉430072)[1]　(北京大学计算机系　北京100871)[2]

Object Concepts Research Based on UniNet

ZHOU Guo-Fu[1,2]　YU Peng[2]　YUAN Chong-Yi[2]

(Computer School of Wuhan University, Wuhan, 430072)[1]　(Department of Computer Science, Beijing University, Beijing, 100871)[2]

Abstract　This paper discusses concepts on object from UniNet view and shows that there exists the flow of control in object system except for exchanging of messages between objects. Meanwhile, the paper presents an independent mechanism of object communication separated from object that will result in a more general reuse of object. With help of the control flow and the data flow, UniNet can describe not only the static features, but also the dynamic features of object system, which naturally solve the inheritance anomaly and the flow of data and control. In addition, based on UniNet specification, the object system can be verified easily and create the program code automatically.

Keywords　Object, Petri net, UNITY, UniNet

1　研究背景

对象以其良好的模块化和重用性，已经在软件设计和开发过程中得到广泛的应用。面向对象的理论已成为计算机领域的一个研究热点。然而，由于对象技术最初是从程序设计的实践发展而来的，旨在为了重用，而这些重用又是基于顺序性计算机体系结构的，因此，对象理论的研究没有完全脱离顺序性计算机体系的影响，如：对象内部是顺序动作序列，隐含控制是顺序的假设，从而忽略了对象内部以及对象外部之间存在着其它的控制关系。总的来说，目前对象研究中存在着以下不足：1)强调对象系统中，对象之间只存在消息的传递，却忽略了对象之间事实上存在着的控制传递关系，如对象同步和并发；2)忽略了对象的动态特征有4个层次：系统行为、对象行为、对象中方法的交互行为以及方法的行为，而对象的继承异常等现象正是由于这个忽略所引起的[1,2]；另外，在系统分析和设计过程中所采用的对象技术，不仅方法多，而且千差万别[3]，从而对系统的理论验证造成一定的困难。

为解决这些不足，一些学者把适合描述并发的 Petri 网方法引入对象研究[4~10]。由于 Petri 网的动态性，不可避免地强调了对象的动态性。然而对象在其生命周期中，不仅有变化的一面，同时还有相对稳定的一面[4,11,12]。因此，Petri 网描述对象静态属性存在着不足，比如无法描述变量。同时，由 Petri 网形成的系统规格说明抽象难以理解[13]，与实际应用观点存在着很大差距，使得 Petri 网在对象技术研究中的优势没有完全体现出来。

UniNet 作为一种新的建模方法，结合 Petri 网[14]和 UNITY[15~17]各自的优势，提出独特的控制流与数据流并重的描述方法，不仅可以描述系统的动态特性，又可以刻画系统的静态特性。另外，UniNet 继承了类似于 PASCAL 的 UNITY 描述风格。使得系统的规格说明直观、易懂以及便于逻辑验证。

本文从 UniNet 的角度，非形式化地描述对象的一些基本概念；讨论了对象的特点，并用 UniNet 相关的概念对这些关系进行刻画。讨论了对象之间的关系。最后，通过有界缓冲区问题来阐述 UniNet 描述对象系统的独特之处。

2　对象的基本概念

以下讨论中所涉及到的对象术语都是引用于文[18]。UniNet 详细介绍见文[19,20]。

2.1　封装

封装实际上是一种逻辑抽象，它隐藏了对象内部结构以及对象行为的实施等。因此，封装使对象在逻辑上分为内部隐藏部分和外部可视部分。内部隐藏部分在对象组成元素之间是可见的，是内部视图。而对象外部则只能通过对象的可视部分，即外部视图来观察对象的特定部分，这种机制可以有效的阻止外部对对象的行为进行干涉。

在 UniNet 中，封装有两种类型：封装库所和封装变迁。封装库所就是把一个子网抽象成一个库所，对外提供的接口(外部视图)是库所的集合。如图1，其中库所 s_1，s_2是对象提供的外部接口，虚线表示子网中的其他部分(如果没有特别声明，以下讨论的图形中虚线部分也具有相同含义)，外界是不可见的。

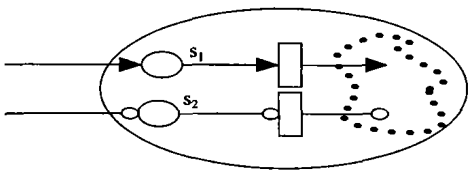


图1　库所 S 的封装

变迁的封装与库所的封装类似。只是封装后的变迁对外

*)国家自然科学基金资助项目：NON-INTERLEAVING 语义并行计算模型及语义规则，No. 69973003。周国富　博士研究生，主要研究兴趣为并行计算，Petri 网理论以及软件体系结构。余　鹏　硕士研究生，主要研究兴趣为分布算法，Petri 网理论以及生物信息。袁崇义　教授，博士生导师，主要研究兴趣为并行计算，计算模型及语义以及 Petri 网理论。

提供的接口(外部视图)是变迁的集合。封装的变迁可以有自己的内部子网。图2中,t_1,t_2,t_3,t_4是变迁 T 的外部视图,即它在对象外是可见的。

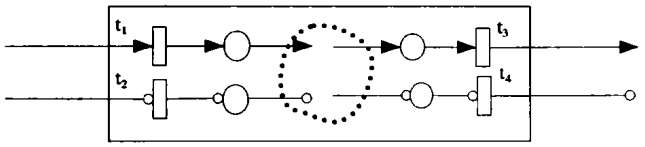


图2 变迁 T 的封装

2.2 类与实例化

类就是具有相同属性和服务的对象集合,它是对象的静态结构描述。类实际上是一个对象模板,定义了对象的属性和服务,以及任务所需要的参数及激活条件。对象是类的一个实例。对象具备所有类定义的关系。

UniNet 通过引用网[21]来描述类以及实例化。所谓引用网就是一个相对独立的子网,它描述了类的静态结构(图3),因此它是不可运行的。实例化的对象有一个在名字空间中唯一的名字,同时具有初始标识,是可以运行的,是活的系统中的一个组成部分。实例化时所产生的对象自动具有了类定义的结构和性质,包括属性、接口、内部元素之间以及对象之间的关系等。实例化的对象的初始标识可以不相同。

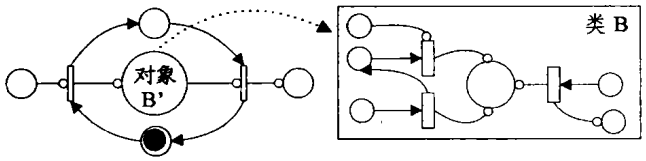
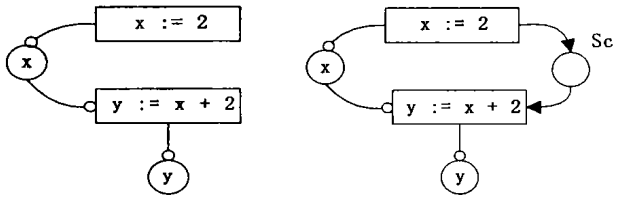


图3 类的引用

2.3 属性

对象包含标识对象特性的值。这些值不仅标识对象所处的状态,而且标识区别于其他的对象的控制,即方法的执行顺序或者并发。控制属性没有具体的值,但实际存在并起控制作用。图4描述了控制属性的作用。



(A) 没有控制属性　　(B) 引入控制属性 Sc

图4 控制属性作用

图4中,设 x 的初始值为0。(A)没有引入控制属性时,操作顺序 x:=2;y:=x+2和 y:=x+2;x:=2都是合法的,因此产生两种合法结果 x=2,y=4和 y=2,x=2。(B)引入控制属性 Sc 后,则操作顺序则是唯一,即 x:=2;y:=x+2,产生的结果也是唯一的,即 x=2,y=4。这种机制显然突出了对象内部数据流与控制流并重的思想。

2.4 方法

方法是对象的行为,它可以改变对象的属性。对象向其他对象提供方法,也可以激活其他对象所提供的方法。方法是对象定义的一部分。方法拥有自己的操作集,即方法是由一系列改变属性的变量变迁组成。图5描述了变量变迁的基本结构。根据封装原则,方法也可以有自己的内部属性及操作,即可以封装子网从而形成高一级抽象级别的变量变迁。这类抽象变迁也可以通过引用网[21]来描述。

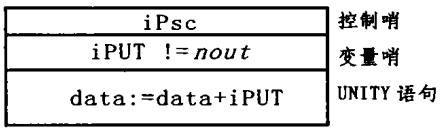


图5 变量变迁的结构

变量变迁的基本结构包括3部分:控制哨,变量哨和 UNITY 语句。其中控制哨控制变迁是否可以发生;变量哨控制该变量变迁的具体操作发生,只有变量哨为真时,该变迁的操作才可以继续;UNITY 语句具体描述变量变迁的操作。因此方法有3个级别的控制:控制流控制,变量哨控制以及 UNITY 哨控制[15]。引入哨机制,可以很好地解决文[1]中的继承异常[2]。

2.5 接口

对象有两类接口:对象向外提供的服务,以及向外请求的服务。对象的接口由库所集合来表示;而变迁(对象间交互)的接口由一系列的变迁组成,是变迁向外提供的视图。通过这个机制可以隐藏对象内部的信息。对象内部操作可以通过接口来引用。同样,变迁也实现了交互的隐藏,因此变迁的接口机制为交互的重用提供了可能。从某种角度来说,信息隐藏是通过接口技术来实现的。接口有控制流、读或写关系3种类型。

2.6 对象

对象是一个相对稳定的实体,起变化的只是对象内部的属性。UniNet 中,对象由封装的变量库所来描述。对象包括属性、操作、接口、初始标识、操作的定义和实现,由6-元组表示:

$O = (S, T; A; H, I, M_0)$;

其中:$S = S_p \cup S_v$--描述对象内部的属性集合,包括各类变量 S_v 和对象内部的控制属性 S_p;T--改变对象属性的动作集合;$A = R \cup W \cup F$--表示控制流(F)或属性变化的读(R)写(W)关系;H--继承的对象;I--外部视图(接口);M_0--对象的初始标识。

以下是一个无界缓冲区对象 BUFF 的 UniNet 描述。假设该缓冲区是由 general 对象继承而来。其中,if *exp* 是哨函数,控制着操作的发生;"~ "表示相反条件的选择[15]。nout 表示缺省值[19]。

```
object BUFF : general
    interface(I) iPUT, iPsc, iPOP, iGET, iGsc;
    attributes(S) data, iPUT, iPsc, iPOP, C, iGET, iGsc;
    initially(M0) data = nout;
    methods(T)
      PUT: ⟨iPsc:: data := data + iPUT⟩
      POP: ⟨iPOP::data := nout if data = nout
            ∽ data := data − data[1]⟩
    GET: ⟨iGsc::nout if data = nout
          ∽ iGET := data[1]⟩
    Net description A:
```

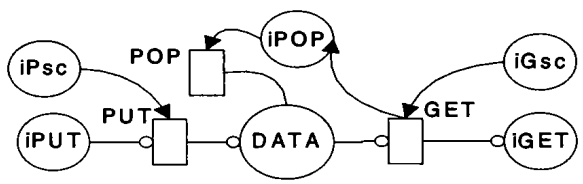


图6 BUFF 的 UniNet 网描述

图6中,带箭头的弧表示控制流;带小圈的弧表示读/写关

系，小圈的一边表示要写入的方向，另一边则表示读的方向；既不带箭头又不带小圈的弧表示同时具有读写关系。

2.7 消息

对象之间的消息交换通过变量变迁中的赋值语句实现。以下讨论生产者-消费者交互。设生产者生产一个元素并放到输出接口 iP；而缓冲区中有一个放置元素的操作 iPUT，则生产者与缓冲区之间的通讯可以通过变迁 PUT 实现，如图7所示，生产者生产一个元素并放入缓冲区。

同样消费者消费元素也有一个接口库所 iC，而缓冲区中有一个读取操作 iGET，则消费者与缓冲区之间的通讯可以通过变迁 GET 实现，如图8所示。

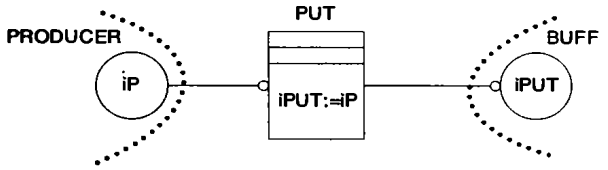


图7 生产者放入元素

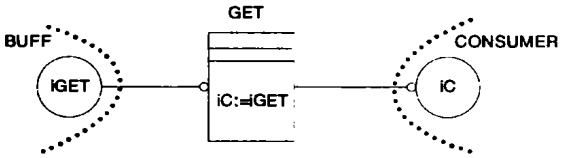


图8 消费者消费元素

消息的交换从对象内部独立出来，这种机制的好处是显然的：1)对象可以不关心提供服务的对象是谁，它仅仅是从自己的接口中放入或读取自己所需要的元素。2)对象的封装性更好，并且减少和特定对象的依赖，因为在实现消息通讯时，对象不必指明是与哪一个对象通讯。3)更加具有通用性。在实际系统实现过程中，只要对象之间的通讯协议通过变迁的变换（如图7中 PUT 和图8中 GET）进行协议的转换，就可以实现对象之间的通讯。

3 变换规则

以下讨论 UniNet 解决分析阶段存在的抽象级别变化，以及多态和重载现象。

3.1 抽象与细化

抽象是忽略掉不关心的细节以实现信息隐藏。从 UniNet 的角度来看，抽象分为：库所抽象、变迁抽象以及关系抽象。库所抽象和变迁抽象在前面已经讨论了，它们都是从一个子网抽象而形成一个库所或者变迁，同时对外提供统一的外部视图。在 UniNet 中，库所与变迁之间存在着流关系、读和写关系。对这些关系的抽象遵循以下规则：

1)当库所与变迁之间只有写关系时，抽象的关系就是写关系；

2)当库所与变迁之间只有读关系时，抽象的关系就是读写关系；

3)当库所与变迁之间既有写关系又有读关系时，抽象的关系就是读写关系；

4)当库所与变迁之间只有流出关系时，抽象的关系就是流出关系；

5)当库所与变迁之间只有流入关系时，抽象的关系就是流入关系；

6)当库所与变迁之间既有流出关系又有流入关系时，抽象的关系就是流出流入关系；

7)如果存在流关系和读/写关系，抽象的关系为两类关系：流关系和读/写关系。每类关系的抽象遵循以上抽象规则。此时，变迁必然是抽象的变迁，并且必定包含控制和变量子变迁。

细化是抽象的逆过程。其遵循的规则与抽象规则相逆。

3.2 多态与重载

多态就是系统运行中，对象或者方法可以被其他的对象或者方法替换的一种机制。因此，替代者与被替代者在性质上存在某种程度的共同性。多态包括方法的多态和对象的多态。

方法多态通过一个 selector 变迁实现。selector 根据方法（变迁）读外延来启动不同的服务。如：对象中有一个接口 P，在对象内部有3种不同的调用方式 P_1，P_2，P_3。selector 根据运行中不同的哨函数来匹配服务，相应的 selector 形式如图9所示。

int P(*integer*)；
int P(*string*)；
int P(*bool*)；

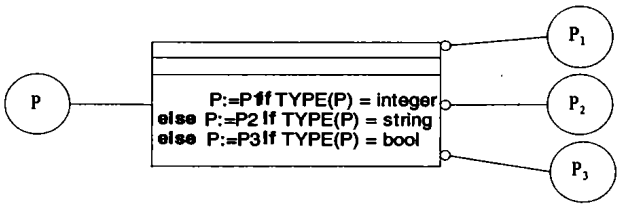


图9 方法的多态

其中 TYPE 为确定参数的数据类型。在运行中，当 P 的类型是 integer 时，则启动服务 P_1；是 string 时，则启动服务 P_2；是 bool 时，则启动服务 P_3。

由于子对象具有父对象所有的性质，因此在父对象出现的地方，子对象均可以取代父对象。如图10所示，父对象存在交互(o,T)，对象 o 中实际参与消息交换的库所是 s_1，s_2。由于子对象 o' 具有父对象 o 的所有属性，因此对象 o' 也具有库所 s_1，s_2。显然变迁 T 可以不加改变地与对象 o 或对象 o' 发生交互。

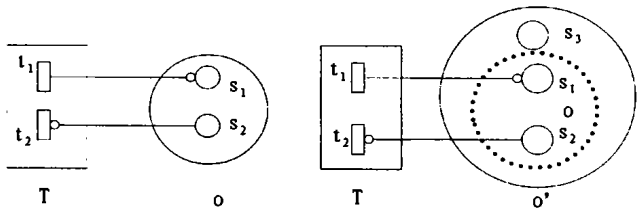


图10 对象多态

对象多态替换的原则可以推广开来，只要满足替换库所中包含有与被替换库所相同性质的接口，并具有相同的约束条件，多态替换就可以合法地发生。

重载是在相应的变迁中增加哨来实现。子对象的方法是继承父对象，其名字、通讯协议仍然保持不变，变化的部分实际上就是方法的实施。UniNet 对重载的描述与方法多态类似，重载的方法虽然名字相同，但是它们处于不同的名字空间。图11表示了重载通过 selector 变迁实现。

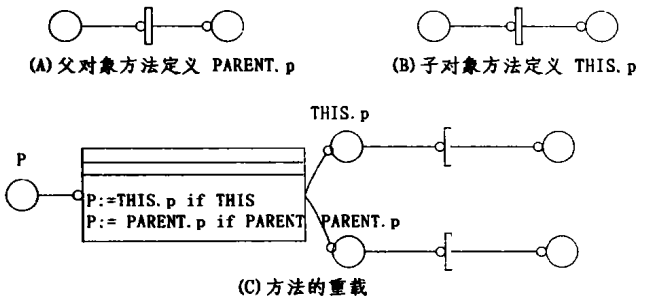


图11 方法的重载

其中，THIS 表示 P 激活子对象的方法；PARENT 表示 P 激活父对象的方法。

4 对象关系

对象系统中系统的活动是通过对象之间交互而实现的，下面只讨论对象之间的静态关系，关于动态关系，如顺序、并发等关系，在以后的文章中详细讨论。

4.1 调用关系

对象之间的调用可以通过变量变迁来描述。实际上可以抽象成相关变量的读写关系组合。图12是两个对象之间的一种简单调用。为了简洁，变迁的哨函数没有标明。

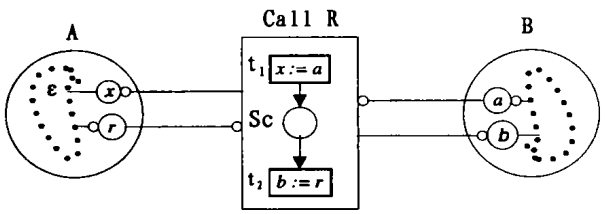


图12 调用关系

对象 B 调用对象 A 中的操作 ε，ε 在对象 A 中定义，对 B 是不可见的。ε 操作需要参数 x，变迁 Call R 中子变迁 t_1 就是从对象 B 读出参数 a，并赋予对象 A 中的接口库所 x。根据 UniNet 的触发规则，由于控制库所 Sc 的控制，t_2 只能在 t_1 之后发生。同时操作 ε 的最终结果就是 r。因此，Call R 中 t_2 读出 r 的值，并赋值给对象 B 中的变量 b，完成一次调用。

显然对象 A 和 B 交互是在变迁 Call R 中发生。B 不关心与谁发生交互，它关心的只是结果，即它发出一个消息，只要返回的消息是自己期待的，交互就正确实现。同样，对象 A 中的 ε，只要得到一个满足条件 x，就可以产生一个返回消息 r。A 不关心消息 x 是谁传送来的，以及消息 r 传送到哪里去。

每个读写弧可以是多个的调用合成。即可以读写结构化变量，甚至是对象。如图13所示，B 激活 A 中的 ε，而 ε 需要对象作为激活条件。

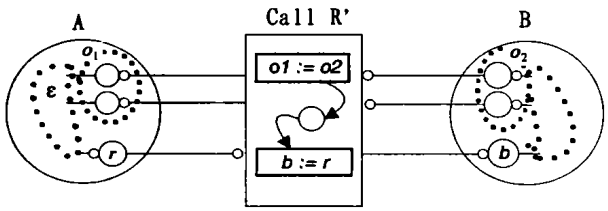


图13 对象的传递

4.2 组成关系

对象包括属性和方法，也可以由其他的对象组合而成。根据抽象规则，调用可以是简单调用，也可以是多个调用的复合。组合对象遵循以上讨论的调用规则。

对象内部的组成可以包含对象。内部对象的接口以及实现具有封装性和信息隐藏特性，并且提供外部视图，是一个相对独立的成分。如图14所示，o_1 只是 o 的内部组成部分，并且 o_1 只对 o 内部的其他属性提供接口。如果 o 的外部要调用 o_1 的方法，则 o 的外部只能通过 o 提供的接口来对 o_1 进行访问。对象 o 内部属性与对象 o_1 的关系遵循以上讨论的调用关系和组成关系。

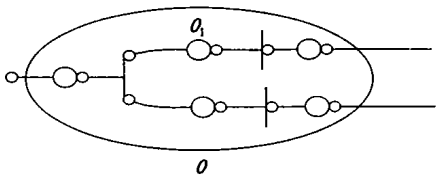


图14 对象的组成

4.3 继承关系

继承是属性和方法在对象之间的一种传递关系，即子对象与父对象共享结构和行为。子对象自动具有父对象的属性，方法以及响应消息。因此，父对象的接口隐含在子对象的接口中，同时子对象可以增加新接口。如图15，对象 o' 继承对象 o。

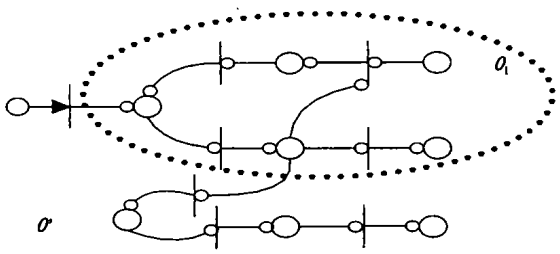
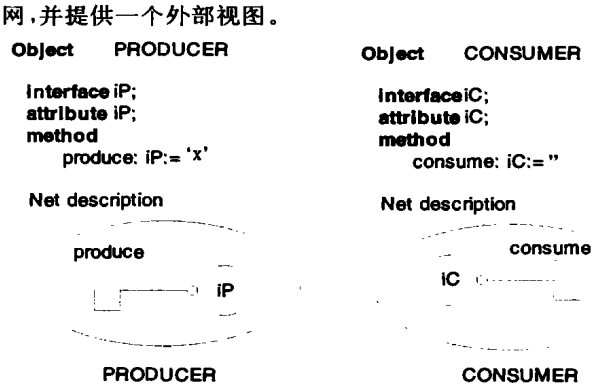


图15 对象的继承

继承在 UniNet 遵循“嵌入/分割”原则[22]。嵌入就是父对象的接口、属性方法以及约束一致性地嵌入到子对象中。父对象的方法可以被子对象重载，子对象也可以增加新的接口、属性、方法以及约束。继承是封装的一定程度的解体，即父对象的所有属性及方法，都是向子对象开放。同时，父对象的接口成为子对象的接口。

分割是嵌入的逆过程，是抽象的应用，即是把对象的属性声明、方法实施或者接口从对象中分离，从而形成一个独立子网，并提供一个外部视图。



Object BBUFF: BUFF

attribute N, *max*;
initially N=0, *max*=100;
method
PUT: <iPsc::DATA:=DATA+iPUT, N:=N+1 **if** N <max>
GET: <iGsc::iGET:=DATA[1], DATA:=DATA-DATA[1], N:=N-1 **if** N>0>
Net description

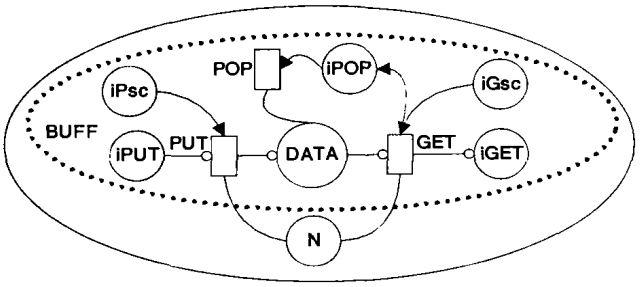


图16 对象的描述

5 有界缓冲区

在这节中通过生产者-消费者来阐述基于 UniNet 的对象概念的应用。系统中有3个对象分别是一个生产者、一个消费者和一个长度不大于 max 的缓冲区来保存生产者生产的元素。消费者每次可以从缓冲区消费掉一个元素。为讨论简洁起见，忽略哨函数，即系统中涉及到的变迁在一定的条件下都可以发生。

设生产者 PRODUCER 只有一个方法 produce－生产一个元素。消费者 CONSUMER 也只有一个方法 consume－消费一个元素。有界缓冲区 BBUFF 从无界缓冲区 BUFF(图4)继承而来，相应地有3个方法：一个是放置元素至缓冲区，一个则从缓冲区取出一个元素，同时从缓冲区中删掉取出来的元素。则对象的 UniNet 描述如图16所示。

生产者生产一个元素，与缓冲区交互，并通过缓冲区的方法放置该元素。同样当消费者需要消费元素时，与缓冲区交互取得一个元素消费。因此根据对象之间的交互，抽象掉不关心的部分，可以得到生产者－消费者关系图。图17是生产者－消费者系统的静态结构描述。

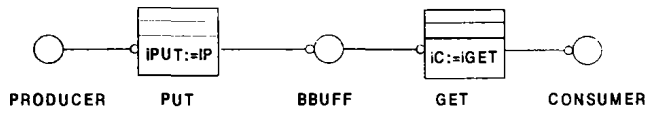


图17 生产者－消费者交互

从图17中，根据文[14,19]所讨论进程规则，容易得到系统一个进程片断(图18)。从图18中可以容易得出，PUT 和 GET 是并发的，即当缓冲区中存在多个元素而元素的个数没有超过 max 时，生产者和消费者是可以同时发生的。

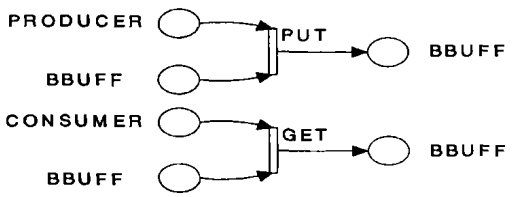


图18 进程片断

UniNet 分别独立描述对象和对象之间的交互，克服了传统对象关系图中的对象与交互分界模糊的不足。对象之间的交互独立抽象成一个消息交互(变迁)。实际上 UniNet 描述可以容易地与经典对象描述进行转换。不考虑变迁的影响，可以得到传统的对象关系图(图19)。

图19 对象关系图

同样，我们可以抽象掉对象，可以形成操作的同步序列，也就是系统要实现的动作序列。如图20所示，图中没有标明关系的方向，是因为操作之间是可以并发的。在实际的实现中，根据一定的计算机体系结构，可以由图20形成一个动作序列，而每个动作是由 UNITY 描述，从而可以形成 UNITY 代码序列，也就是程序。关于其它性质讨论和逻辑验证，由于文章篇幅所限，不再详细讨论。

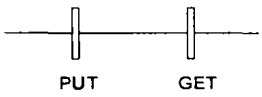


图20 操作序列

结束语 对象系统行为是多方面的。但是对象的动态特征总的来说，从高到低有4个层次：系统行为、对象行为、对象中方法的交互行为以及方法的行为[4]。当前对象技术对这4个层次描述所采取的方法不尽相同，甚至风格迥异，因此对系统的设计和验证没有一套完整有效的方法。

我们的工作是提供简单但强大的描述方法，来描述对象系统的形式语义，从而可以对系统进行验证。UniNet 结合了 Petri 网和 UNITY 的优势，不仅具有 Petri 网简洁直观的描述能力，而且吸取了 UNITY 的逻辑基础。本文从 UniNet 的观点出发对对象相关概念的讨论，使得 UniNet 应用于对象系统分析设计，有以下优点：系统分析和建模阶段都可以采取统一的方法描述对象系统的静态特征和动态特征，并且可以完整地描述对象系统4个层次的特征；提供基于逻辑的系统验证手段；解决了当前对象系统中对象交互必须在对象的设计阶段指定的不足；根据 UniNet 形成的系统规格说明可以容易地向特定体系结构的计算机映射[20]。

参 考 文 献

1 Matsuoka S, Yonezawa A. Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages. In: G. Agha, P. Wegner, A. Yonezawa, eds. Research directions in concurrent object-oriented programming, MIT Press, 1993. 107～150

2 Yu Jian, Wang Shengyuan, Yuan Chongyi. Solving Inheritance Anomaly with OMNets. Journal of Computer Science and Technology, 2001,17(1):101～105

3 张海藩. 软件工程导论. 清华大学出版社,1998

4 喻坚,孙泉,袁崇义. 并发面向对象系统的 petri 网多级建模. 计算机科学,2001,28(7. 增刊)

5 王生原,喻坚,袁崇义. 用加标的基本网系统设计群件编辑器. 计算机科学,2001,28(7. 增刊)

6 Genrich H J, Lautenbach K. System modelling with high-level petri nets. Theoret. Comp. Sci., 1981,13:109～136

(下转第38页)

0.45695777, 0.45257384, 0.44824200, 0.44417031, 0.44037421)。

限于篇幅,f_7($n=50$)新的最优解不列出了。f_6的最好结果虽优于已知的最优值,但三个约束的满足情况不如已知最优解,前者只达到0.0001,而后者达到0.0000001,考虑到这点,不能认为这里给出的结果优于已知最优值。

表2列出了 Michalewicz 的 GENOCOP II 求解 f_1,f_3~f_6 10次运行的统计结果和 GENOCOP III 求解 f_7的最好结果[2]。其中 f_5的最好结果对应的解有一些违约。比较表1和表2可见,除了 f_1外,CEA/HCIM 的求解质量优于 GENOCOP 系统。

表2 GENOCOP 优化 f_1,f_3~f_7的结果

函数	最好	中等	最差
f_1	−15.000	−15.000	−15.000
f_3	680.642	680.718	680.955
f_4	7377.976	8206.151	9652.901
f_5	18.917	24.418	44.302
f_6	0.054	0.064	0.557
f_7(20)	0.80351067		
f_7(50)	0.83319378		

表3 Deb 算法优化 f_1~f_7的结果

函数	最好	中等	最差	δ≤1%
f_1	−15.000	−15.000	−13.000	47
f_2	−30665.537	−30665.535	−29846.654	47
f_3	680.634460	680.641720	680.650879	50
f_4	7060.221	7220.026	10230.834	17
f_5	24.37248	24.40940	25.07530	41
f_6	0.053950	0.241289	0.507761	19
f_7(20)	0.7139	0.6623	0.6371	

表3是 Deb 在文[4]报告的求解 f_1~f_6的50次运行统计结果和在文[5]报告的求解 f_7($n=20$)的10次运行统计结果,其中 δ≤1%项表求解结果与已知最优值的相对误差小于等于1%的次数。比较表1和表3可见,CEA/HCIM 的求解质量也优于 Deb 的算法。Deb 在他的算法中采用了简单的约束处理技术,但是没有采取有效措施提高算法的搜索能力,优化的效果也不理想。

由于有关文献没有提供各函数计算次数,算法之间的求解效率无法进行比较。就 CEA/HCIM 自身而言,有些函数的求解效率较高,有些较低。如何更好地兼顾求解质量和求解效率仍需进一步的研究。

结论 本文从提高算法的搜索能力,简化对约束的处理入手,提出了一种新的约束优化演化算法。在算法中,采用混合杂交和间歇变异来提高算法的搜索能力,采用违约函数和个体的直接比较来处理约束,数值实验和比较显示了所提算法的有效性。本文的研究也表明,求解约束优化问题时,不应只限于处理好约束,提高算法的搜索能力对获得好的优化效果也是至关重要的。

参 考 文 献

1 Michalewicz Z, Schoenauer M. Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation 1996, 4(1): 1~32

2 [美] Z. 米凯利维茨著,周家驹,何险峰译. 演化程序——遗传算法和数据编码的结合. 北京:科学出版社,2000

3 Herrera F, Lozano M. Gradual Distributed Real-Coded Genetic Algorithms. IEEE Trans on Evolutionary Computation, 2000, 5 (1): 43~63

4 Deb K. An efficient constraint handling method for genetic algorithms. Computer methods in applied mechanics and engineering, 2000, 186: 311~338

5 Deb K, Agrawal S. A niched-penalty approach for constraint handling in genetic algorithms. In: Montana, D ,ed. Proceedings of the ICANNGA-99. Portoroz, Slovenia, 1999. 234~239

(上接第18页)

7 Yao Weili, He Xudong, Mapping Petri Nets to concurrent programs in CC++. Information and Software Technology, 1997, 39: 485~495

8 Hong J-E, Bae D-H. Software modeling and analysis using a hierarchical object-oriented Petri net. Information Sciences, 2000, 130: 133~164

9 Peterson J L. Petri Net Theory and the Modeling of System. Prentice-Hall, April 1981

10 Goldsack S J, Kent S J H. Formal Methods and Object Technology. Springer-Verlag, London Limited, 1996

11 Janicki R, Koutny M. Semantics of inhibitor nets. Information and Computation, 1995, 123: 1~16

12 Gerogiannis V C, Kameas A D, Pintelas P E. Comparative study and categorization of high-level pertri nets. The Journal of System and Software, 1998, 43: 133~160

13 He Xudong. PZ nets --- a formal method integrating Petri nets with Z. Information and Software Technology, 2000, 43: 1~18

14 袁崇义. Petri Net 网原理. 北京:电子工业出版社,1998

15 Chandy K M, Misra J. Parallel Program Design: A Foundation. Addison-Wesley, 1989

16 Queinnec. Tailoring UNITY to Distributed Program Design. In: Intl. Workshop on Formal Methods for Parallel Programming: Theory and Applications, April 1998. 820~832

17 Yuan Chongyi, Qu Wanling. UNITY and its Missing Structure. In: Proc. of 3rd publishing company, 1998. Workshop on Advanced Parallel Processing Technologies, Changsha, China, Publising House of Electronics Industry, 1999. 172~176

18 Booch G. Object-oriented Design with Application. Menlo Park, CA, Benjamin Cummings, 1993

19 付弘宇. 从 Petri net, UNITY 到程序语义模型:UniNet:[硕士论文]. 北京大学,2001

20 Zhou Guofu, Yuan Chongyi. Mapping PUNITY to UniNet. To be appeared in Journal of computer science and technology

21 Annette L, Matthias L, Daniel M, Ivana T. Modelling Intra- and Inter-Object Control using Reference Nets. In: Jürgen Ebert and Ulrich Frank, eds. Modellierung 2000, St. Goar, Band 15 von Koblenzer Schriften zur Informatik, Fölbach Verlag, 2000. 89~102

22 Reisig W. Petri nets in software engineering. In advances in Petri nets 1986. Number 225, 1993. 77~87