第7卷第5期 软 件 学 报 Vol. 7, No. 5
1996年5月 JOURNAL OF SOFTWARE May 1996

RKB/PL:C++的持久性扩充*

田 忠 刘 畅 陈 莹 钱乐秋

(复旦大学计算机科学系 上海 200433)

摘要 需求工程知识库/PL——RKB/PL(requirement－engineering knowledge base/PL)是保持C++原有风格对C++进行的持久性扩充.为支持对象的持久性,RKB/PL在C++对象类的基础上扩充了以约束声明加强对象状态的用户监控;引入簇来表达对象类的集合含义;引入集合、簇、簇闭包的遍历机制来支持对象查询.为支持这些语言机制,RKB/PL具有一个由一组build－in对象类层次、类型信息库及接口函数、系统状态表以及系统服务函数等构成的运行时系统.本文讨论了RKB/PL中这些机制的表示、相应的运行时系统的组成以及它们的主要实现技术.RKB/PL已成功地用于实现"软件需求获取助手FRA"系统的需求工程知识库子系统.

关键词 持久性,面向对象,簇,约束,查询.

C++是一种面向对象通用语言[1],它本身并不支持持久性.对C++进行持久性扩充有2种方法:①不改动语言,而引入专用类库来支持持久性;②引入少量新语言机制,扩充C++数据模型.前者工作量小,实现直接、方便,但不易处理多对象类联合/嵌套查询,后者应努力使新语言机制与源语言在风格上保持一致.目前许多OODB语言均采用在现有语言中引入数据库机制的办法.持久性是指一程序所创建的对象的生命期超过了该程序的运行期,它是对象(而不是对象类)的性质.对象类定义并不确定其实例的存取区域的性质,而是由程序员根据应用的实际需要来决定.持久性程序设计中,对象应能方便地在主存与持久存储区域间进行交换.需求工程知识库RKB(requirement－engineering knowledge base)是"软件需求获取助手FRA"的存储子系统.智能助手FRA能够辅助分析员交互地获取和修改软件需求,并随时生成用户需求文档.FRA将需求获取过程看成是用户问题及领域的知识获取过程,即将一组散乱不确切的用户需求陈述开发成精确一致的形式需求表示.RKB负责管理FRA辅助分析员进行需求获取所需的4类知识:一般知识、需求知识、推理知识和领域知识.FRA采用统一的面向对象模型来描述和表示这些知识,而RKB也采用面向对象模型来统一地组织、存取这些知识,这些知识为FRA的其它子系统(如非单调对象推理器、释义器、文档生成器等)所共享.

---

* 本文研究得到国家863高科技项目基金资助.作者**田忠**,1968年生,博士,主要研究领域为基于知识软件工程,面向对象需求分析,知识表示与处理,数据库设计与应用.**刘畅**,1970年生,助教,主要研究领域为面象对象需求分析,用户界面设计.**陈莹**,女,1971年生,硕士,主要研究领域为知识表示与处理,时态信息的表示与推理.**钱乐秋**,1942年生,教授,主要研究领域为软件工程.

本文通讯联系人:田忠,上海200433,复旦大学计算机科学系

本文1995－02－23收到修改稿

本文讨论RKB实现语言RKB/PL的设计与实现技术.RKB/PL是在C++的基础上扩充对象类的存储、查询、约束等机制.[2]这种扩充主要在2个方面进行:①对C++语言机制的扩充,增加对持久性、约束、查询的支持;②对运行时系统的扩充,提供一组用于维护持久对象状态的系统表和函数.

1　RKB/PL的数据模型及其语言支持

RKB/PL基本上沿用了C++的面向对象数据模型,但在C++对象类的基础上,引入了约束成员声明.约束成员使得RKB/PL在通常的类型系统之上,为维护对象状态提供了更灵活的支持.当对象上的某一操作违背了约束条件,该操作的结果就应被取消.应用状态(程序中的全体对象、变量和相关持久对象的状态总和)的清理工作则由RKB/PL运行时系统和程序员分担:持久存储区域的恢复依赖于底层数据库(目前的实现采用了Sybase DBMS)的事务管理机制,而主存中的清理则主要由C++的原有对象构造机制和新引入的约束机制来完成.

1.1　对象类声明的扩充

RKB/PL的对象类声明是在C++原有的对象类声明中增加一项"约束声明":

〈约束声明〉::='constraint' ':' {〈约束成员〉';'}

〈约束成员〉::=〈约束条件〉['@'〈约束动作〉].

其中的〈约束条件〉为普通的布尔表达式,〈约束动作〉是当〈约束条件〉不成立时,用以清理现场的函数.图1给出了2个RKB/PL对象类及相应的操作.

```
#include "rkb_pl.h"
class Person
{public:
  String name, profession;
  char sex;
  Company workFor;
  Person * spouse;
  Set(Person * ) children;
  Person(String nm,char s);
  Person(String nm,char s, String prof);
  int marriedWith(Person& p);
  int changeSex();
 constraint:
  (sex == 'f'||sex == 'F'||\
   sex == 'm'||sex == 'M') @ throw BadSex();
  this != spouse     @ throw BadSpouse();
  !children.has_member(this);
};
class Female: public Person
{public:
  Female(String name);
  Female(String name,String prof);
 constraint:
  (sex == 'f'||sex == 'F') @ throw BadSex();
   // override previous constraint
   // definition in Class Person.
};

int main()
{Person p("John", 'm',"Singer");
 Female f("Mary","Dancer");
 p.sex = 'f';     // Be careful.
 // direct access of data member.
 f.changeSex();// constraint violated
 // exception BadSex raised!!
 CID P_CID, F_CID;
 F_CID=create Female;
 // cluster Female created,
 // but cluster Person not yet
 P_CID=create Person;
 pinsert(&f);
 // persistent object created
 destroy F_CID;
 // cluster Female together with
 // all its persistent obj destroyed
}
```

图1 RKB/PL的对象类Person和Female

约束声明可以出现多次,并且不受访问控制机制的制约,其作用域是从constraint到下一个private/protected/public为止.同构造函数和析构函数一样,约束成员不能被程序员直

接访问,而是由 RKB/PL 的运行时系统使用.约束条件的检查是在每次访问该对象的成员函数、友员(包括友员函数及友员类的成员函数)返回之前进行,但 RKB/PL 不能禁止对该对象数据成员的直接访问,这时对象状态只能由程序员自己负责.

约束可以继承,但导出类中,新的约束声明将覆盖其基类中原有的约束声明.当然约束动作中不一定立即处理现场,也可通过发信号(Signal)或激发异常(Throw Exception)来唤醒其它的意外处理程序.由于持久对象往往要脱离创建它的程序并延续到其它的应用中,因此约束声明提供的显式说明意外处理机制是十分重要的.

1.2 簇的创建与撤消

RKB/PL 支持一般的面向对象程序设计,对象类的实例并不自动具有持久性,因此对象类声明并不自动地在持久存储区域上产生结构映象.要产生这样的映象,以便该类的实例能在持久存储区域上分配空间,还需要对该类创建相应的模式(我们称之为簇).

关于对象类有2方面含义:一是关于特定数据及其上操作的性质声明,相当于类型概念;二是指称该对象类的全体实例,相当于集合概念.目前流行的面向对象程序设计语言(如 C++[1],Smalltalk[3]等)的类都只具有前者的含义,而后者在支持持久性时十分重要.[4]为与 C++保持一致,我们不扩充其类的含义而引入簇(Cluster)来刻画一组具有相同类型的持久对象:任何持久对象都必定属于某一特定簇.创建簇和撤消簇的句法如下:

〈创建簇〉::= 'create' 〈对象类名〉

〈撤消簇〉::= 'destroy' 〈对象类名〉 | 'destroy' 〈CID 号〉

1.2.1 簇的创建

如果创建成功,则返回该簇标识,其类型为 CID 且全局唯一,此后我们可以用此标识或它对应的对象类名来指称该簇;创建失败(如对象类重名、持久空间耗尽等)则返回0. create 操作负责在底层 Sybase 上创建相应的模式来描述该对象类:它的各个域是该对象类实例的数据成员,而该对象类全体实例共享的部分(包括数据成员、函数成员、约束成员等)则存放在另一由 RKB/PL 维护的 ClassSchema 中.

簇创建过程是递归的:RKB/PL 采用在对象类层次上的向下扩展模式的存储策略以减少查询中的连接次数.首先依次将其每一基类扩展到该类的声明中;其次为每一数据成员的基类型创建簇(基类型是指去除了 * /&/[]等类型修饰符后的类型,基本类型如 int,char,char * 等可以直接存储而不必另创建簇),最后为自身创建簇,并置主动创建标识.这时:

①如成员是数组类型,则展开该数组,簇中域名为数组名加下标;

②如成员的基类型是基本类型(如 int,char,char *),则簇中对应域类型为此基类型;

③如成员的基类型是复杂类型(如 class,union,struct,若其簇已创建则隐式创建标识加1;否则先创建此簇),则簇中此域的类型为 OID(全局唯一的对象标识字类型).

RKB/PL 不支持函数指针、静态成员、volatile 成员作为簇的域,因为这3种类型的成员都跟程序运行有关,而持久性则要求对象的性质超出程序的运行期.图1中当执行了 create Person 后就有了如图2所示的簇定义.

Person

域名	oid	name	sex	profession	workFor	spouse	children
类型	OID	varchar	char(1)	varchar	OID	OID	OID

Company...Children...Set(Person *)...

图2 对象类 Person 对应的簇(cluster)的定义

1.2.2　簇的撤消

撤消簇不仅要撤消其定义,而且要回收该簇的持久对象所占的空间.成功地撤消一簇,要求它存在(已被创建),且其隐式创建标识为0(否则说明该簇在被其它簇所使用,因而不能撤消).这时首先撤消它自己,然后撤消其成员之基类型对应的簇(隐式撤消).如隐式撤消失败(主动创建标识=1或隐式创建标识>1),则使对应的隐式创建标识减1.

1.3　持久对象的管理

与堆中分配的对象一样,持久对象的管理也存在空间分配和回收的问题.同时考虑到为使持久对象与内存对象状态一致而进行同步刷新的开销太大,还需要有对持久对象进行状态刷新的操作及一组进行内外存交换的函数.这些操作的函数原型如图3,其应用如图1、图5所示.

```
//持久对象的创建、刷新、销毁
template<T> OID pinsert(T * obj, CID cid);
int prefresh(OID oid);
template<T> int prefresh(T * obj);
int pdelete(OID oid);
template<T> int pdelete(T * obj);
//内存指针与持久对象标识号(OID)的转换
template<T> OID getOID(T * obj);
template<T> void getOPTR(OID oid, T *& obj);
//持久存储区域与内存的交换
template<T> void fetchObject(OID oid,T *& obj);
template<T> void fetchCluster(CID cid,Set<T *>& cls);
template<T> void fetchClosure(CID cid,Set<T *>& cls);
template<T> int detachObject(OID oid);
int detachCluster(CID cid);
int detachClosure(CID cid);
```

图3 RKB/PL 中管理持久对象的基本函数

持久对象在持久空间的创建、销毁、刷新不影响它所对应的内存对象的状态.持久对象保存了该内存对象生命期中一个相对稳定的状态,这一状态在内存中的变化并不立即反映到持久空间上,而需直接或间接地使用持久对象刷新操作.持久对象的空间分配,先要根据ClassSchema 中对其簇所对应的对象类的记录,对内存对象进行转换,然后再将其状态记录到对应的簇中:

①判断本对象是否已有对应的持久空间:如有则分配失败,返回已有的对象标识号;否则给本对象分配一个新的对象标识号.如本对象为顶层对象,则置其所属簇的主动创建标识.

②根据簇定义,逐个域地填写内容:如该域为简单类型,则直接填入对应数据成员的值(如其类型有类型修饰符,则该值为其最终引用或指向的值);如该域为 OID 类型,则填入对应的对象标识号(如该对象尚无对应的持久对象,则为它分配持久空间).此过程中,对象创建标识的使用与簇创建标识的使用类似.

持久对象刷新及内外存交换过程与此类似,持久空间回收过程则正好与此相反.

2　RKB/PL 的数据查询

与 ODE/O++[5]类似,RKB/PL 引入簇、集合以及对它们进行遍历的机制,从而提供

对持久对象进行查询的有效手段.

2.1 集合与簇的遍历

foreach 语句给出如何有条件地遍历一给定集合或簇(句法如图4),并要求对象指称变量至少在查询条件中出现1次,否则容易造成程序不中止.

```
〈集合遍历语句〉::='foreach' '(' 〈set_list〉 ';' 〈condition〉')' 〈LOOP_BODY〉
〈簇遍历语句〉::='foreach' '(' 〈domain_list〉 ';' 〈condition〉')'〈LOOP_BODY〉
〈簇闭包遍历语句〉::='forall' '(' 〈domain_list〉 ';' 〈condition〉')' 〈LOOP_BODY〉
〈set_list〉::=〈set_decl〉 | 〈set—decl〉 ',' 〈set_list〉
〈set_decl〉::=〈集合对象〉〈对象指称变量〉
〈domain_list〉::=〈domain_decl〉 | 〈domain_decl〉 ',' 〈domain_list〉
〈domain_decl〉::=〈簇名〉〈对象指称变量〉
```

图4 RKB/PL 持久对象查询句法

RKB/PL 的这一实现中,集合遍历转化为相应的 SetIterator 控制结构,实际上在主存中进行;簇遍历语句最终将转化为宿主数据库的 SQL 语句. 这时依照 foreach 语句构造相应的 SELECT/FROM/WHERE 子句,并将 LOOP_BODY 作用到所选出的对象上. WHERE 子句的构造较为复杂:如果条件中使用的成员的基类型是简单类型,则可直接使用;如果条件中使用了成员的成员,则需要构造嵌套查询.

例:RKB/PL 对象类及簇如图1所示,找到所有在上海工作的人

```
foreach (Person p;p. workFor. loc==“Shanghai”)
  cout〈〈p. name〈〈'\n';
```

据此构造出如图5的 Sybase 查询过程.

```
dbcmd(dbproc,“SELECT Person. oid”);
dbcmd(dbproc,“FROM Person”);
dbcmd(dbproc,“WHERE Person. workFor IN \
  (SELECT Company. oid \
   FROM Company \
   WHERE Company. loc =\"Shanghai\"))”);
     dbsqlexec(dbproc);
     Person *p;
     while((return_code=dbresults())
           !=NO_MORE_RESULTS)
     {if (return_code==SUCCEED)
      {dbbind(dbproc,1,INTBIND,(DBINT)0, \(BYTES*)&RKB_temp_oid);
      while (dbnextrow(dbproc)!=NO_MORE_ROWS)
        {fetchObject(RKB_temp_oid,p);
         // LOOP_BODY
         cout〈〈(p—)name〈〈'\n';
         detachObject(RKB_temp_oid);
        }
       }
      }
```

图5 簇元素遍历的内部实现

2.2 簇闭包(cluster closure)的遍历

我们称一对象类的导出类所对应的簇为该类对应的簇的导出簇,所谓簇闭包是指该簇及其全体导出簇组成的集合. RKB/PL 中继承关系保持“is—a”性质,即作用于某类对象的操作均可作用于该类的所有导出类的对象上. 同样,相同的查询操作也可以作用在该类对应的簇及该类所有导出类对应的簇的元素上.

forall 语句用于遍历簇闭包(句法如图4),同样要求对象指称变量在条件句中至少出现

1次,并且该语句最终也要转换成宿主数据库的SQL语句.其翻译工作与foreach语句的翻译类似,只是在FROM子句中要先在RKB/PL系统表ClassHierachy中查出指定类的全体导出类,并加入到FROM子句中.

假定对象类定义如图1所示,试比较:

①找出所有与自己孩子同行的母亲:

```
foreach(Female f;)
  foreach(f.children c;f.profession==c.profession)
    cout<<(f.name<<'\n';
```

②找出所有与自己孩子同行的人:

```
forall(Person p;)
  foreach(p.children c;p.profession==c.profession)
    cout<<(p.name<<'\n';
```

它们最终转换结果的主要差别是①中FROM子句为"FROM Female",而②中FROM子句为"FROM Person,Female".

3 RKB/PL的实现

RKB/PL是在支持C++模板和意外处理[1]的GNU CC 2.3.1编译器的基础上进行扩充的.前2节中,我们在说明对C++扩充的同时也大致说明了相应的实现技术.本节我们主要讨论RKB/PL运行时系统和RKB/PL程序的编译、执行流程.

3.1 RKB/PL的运行时系统

RKB/PL具有由Build-in类层次(包括集合、B-树、字典等以及相应的遍历器对象类)、类型信息库接口、系统服务函数以及一些系统状态表组成的较为复杂的运行时系统.Build-in类层次基于模板类(Template Class)技术来实现.RKB/PL运行时系统还要为RKB/PL程序访问底层存储子系统提供一致的接口,提供内外存交换机制以及辅助RKB/PL程序运行的系统表格.前两者均以函数的形式提供给程序员,如图3的get.../fetch.../detach...系列函数,此外还有其它一些辅助转换函数和系统表格查询函数.RKB/PL的系统表格主要有:

①ClassSchema

详细记录RKB/PL程序中每一对象类的定义,包括每一数据成员的类型、名称;每一函数成员的返回类型、参数、名称及实现;每一约束成员的约束条件和约束动作.这在将一内存对象分解以便转化成持久对象和在将一持久对象读入内存以便构造内存对象时十分重要.

②ClassHierachy

针对RKB/PL的每一数据库,记录其中的所有簇的对应类的导出关系,主要用于簇闭包语句翻译及程序员关于类层次的查询.ClassSchema和ClassHierachy一起记录了对象类的完整信息,为其它RKB/PL程序利用现有对象类提供了方便.

③ClusterDefinition

用以记录簇与对象类之间的对应关系、各个簇的定义与引用情况以及簇标识的分配情况,这在创建和撤消簇时都十分有用.

④其它系统表

RKB/PL 中还有其它的一些用以标记持久对象的类型、簇、引用情况、当前位置等的辅助表，如系统持久对象分配表、活跃对象映射表等.

3.2 RKB/PL 程序的编译/执行流程

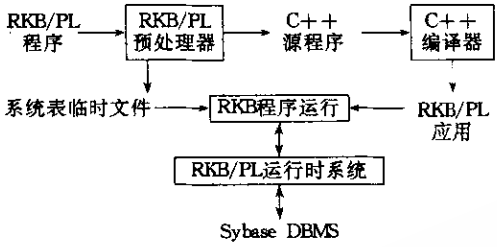

图6 RKB/PL程序的编译流程

图6给出了 RKB/PL 程序的编译－执行流程. RKB/PL 程序经 RKB/PL 预处理器产生 C++源程序和用以填写各种系统表的临时数据文件，C++源程序经 GNU CC 编译后生成可运行的 RKB/PL 应用，而临时数据文件在该应用首次运行后被自动删除. RKB/PL 运行时系统处于 RKB/PL 应用与底层存储系统之间，对 RKB/PL 程序起了屏蔽底层存储实现的作用，提高了 RKB/PL程序的可移植性.

4 小 结

通过对 C++进行持久性方面的扩充，我们得到了一个兼顾通用程序设计和持久性程序设计的集成式语言 RKB/PL. 我们在 C++中引入了簇、约束、查询等语言机制以使 RKB/PL 能支持持久性程序设计，并设计了相应的语言表示和运行时支持. 这些变化都遵循 C++原有风格，不改变其原有语言机制. 我们已成功地使用 RKB/PL 开发了 FRA 系统的需求工程知识库，具有较好的使用方便性和执行效率.

参考文献

1 Stroustrup B. The C++ programming language. 2nd ed., Addison－Wesley, 1991.

2 Richardson J E, Carey M J. Persistence in the E programming language, issues and implementation. Software －Practice and Experience, 1989, 19(12): 1115～1150.

3 Goldberg A, Robson D. Smalltalk－80: the language and its implementation. Addison－Wesley, 1984.

4 Beech G. Groundwork for an object database model. In: Research Directions in Object－Oriented Programming, MIT Press, 1987. 317～354.

5 Agrawal R, Gehani N H. ODE(object database environments): the language and the data model. In: Object －Oriented Database with Application to CASE, Networks, VLSI CAD, Prentice－Hall, 1991. 365～386.

RKB/PL: THE PERSISTENCE EXTENSION OF C++

Tian Zhong Liu Chang Chen Ying Qian Leqiu

(*Department of Computer Science Fudan University Shanghai* 200433)

Abstract RKB/PL (requirement－engineering knowledge base/PL) is a persistence extension of the C++ programming language, while adhering to the style of C++. In

order to support persistence in RKB/PL, constraint declaration is introduced into the C++ class declaration to facilitate user's monitoring of object state; the notion of cluster is introduced to capture the "set—of—objects" conception of class; set iterator, cluster iterator, cluster closure iterator are also introduced to support object query. RKB/PL run—time system facilitating these new language mechanisms is composed of a set of build—in class hierarchies, type information base interface functions, system service functions and also a set of system state tables. RKB/PL has been used to implement the RKB (requirement—engineering knowledge base) of the "software requirements assistant FRA" system.

Key words Persistence, object—orientation, cluster, constraint, query.