CIMBASE中分布式C预编译程序的实现研究*

周立柱 王小京

(清华大学计算机科学与技术系,北京 100084)

摘要 CIMBASE是为计算机集成制造系统而设计的一个异构分布式数据库管理系统. CIMBASE的重要用户接口之一是它的C预编译程序. 它允许用户在C里访问远程数据库,是用户开发计算机集成制造系统的重要编程接口. 在这篇论文里,我们先介绍它的主要功能,而后集中讨论有关它的实现技术. 特别是必要的通信命令设计,主程序、代理程序的工作模式、结构及其生成,以及与之相关的某些算法.

关键词 面向对象数据库*,分布式数据库,预编译程序*.

计算机集成制造系统(Computer Integrated Manufacturing System, 简称CIMS)是一个非常复杂的计算机系统. 它具备下述特点:1. CIMS是一个异构的计算机环境. 就DBMS(数据库管理系统)而言,可能包含各种不同类型. 2. 它的各个子系统分布在不同场地并通过它们之间的数据共享来协调全局的行动,因此分布式数据管理对于CIMS至关重要. 3. 各场地高度自治.

考虑到CIMS的上述特点,我们设计并正在实现一个支持CIMS的分布式数据库管理系统CIMBASE. CIMBASE是一个以各场地的异构DBMS为基础的完整的数据库管理系统,它采用面向对象的语义关联数据模型OSAM*及查询语言OSDL[1,2],为用户提供交互式OSDL,图形OSDL, C预编译程序等界面,在全局实现数据共享. CIMBASE的作用,总体结构及计算机环境如图1所示. 图1中的分布式处理器用于处理分布查询,而翻译器则负责将OSDL命令转换成本地DMBS可执行的查询语句.

针对CIMS的特点,CIMBASE采用下述分布式数据处理的策略:

1. 各场地的OSAM*数据库具备高度的自治性. 本场地的数据库定义及操作不受其它场地上数据库系统的影响.

2. 为了使用远程场地上的OSAM*数据库,用户必须通过显式的OSDL命令联结远程的OSAM*数据库.

3. 一个事务所处理的数据库仅限于一个场地.

* 本文1991-11-10收到,1992-06-20定稿

作者周立柱,47岁,副教授,主要研究领域为计算机软件,数据库. 王小京,女,34岁,讲师,主要研究领域为数据库.

本文通讯联系人:周立柱,北京 100084,清华大学计算机科学与技术系

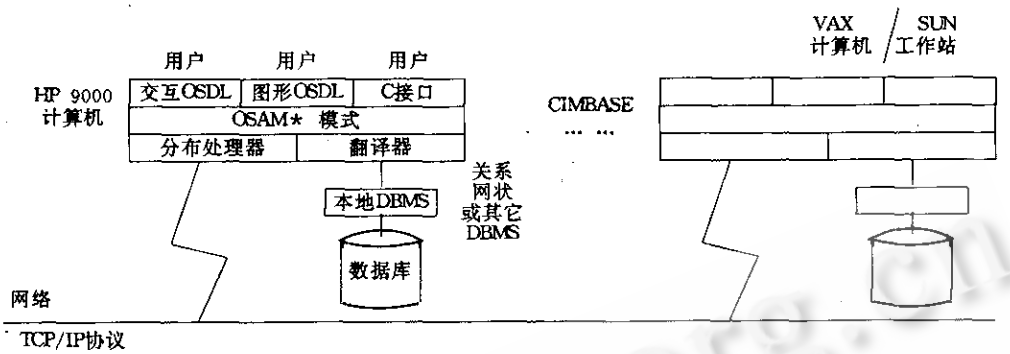

图1 CIMBASE总体结构

基于上述指导原则,下面我们来讨论具有分布处理功能的宿主语言接口,及它的预编译程序的实现技术.首先我们介绍集中式的C宿主语言接口及其预编译程序,而后通过实例说明分布式C宿主语言接口的功能,接着再讨论分布式预编译程序的有关实现技术.

1 集中式C预编译程序

集中式C预译程序C_OSDL允许用户在C里嵌入对本地OSAM*数据库进行操作的OSDL语句.为了能处理由OSDL查询语句返回的结果,C_OSDL增加了指针层次定义及利用指针进行导航式查找的FETCH语句,参考文献[3,4]详细地讨论了C_OSDL的设计与实现.

OSAM*数据模型及OSDL查询语言在参考文献[1,2]里有详细介绍.我们用图2为例说明OSAM*的基本结构.图中的方框代表实体对象类,圆代表域类对象类.两个对象类之间的箭头及字母G、A、I分别代表语义关联中的概括关联、聚合关联与交互关联.OSDL所提供的数据操纵语句包括INSERT(插入)、DELETE(删除)、UPDATE(更新)与RETRIEVE(检索).

嵌入C里的所有OSDL语句都用字符串前缀"OSDL"加以标识.含有OSDL的C程序一般都要包括数据库定义,宿主变量定义,指针层次定义,数据库通信区定义以及必要的数据操纵语句.附录1给出了一个针对图2的数据库进行插入与检索操作的程序实例.该程序先在DEVICE实体类里插入一个对象,而后显示出每一辆小车(即TYPE="cart"的DEVICE)及其所运载的所有物件在仓库STORAGE里的位置(DEVICE的超类CONTAINER与STORAGE间的A关联表达了这种关系).

集中式C_OSDL预编译程序的结构及工作流程如图3所示.

图中的"词法分析器"对含有OSDL语句的源程序进行扫描并记录下嵌入的OSDL语句的位置."语法分析器"根据数据字典里的信息对识别出的OSDL语句进行分析并产生出一系列内部表格.C_OSDL的中心部分是一个"翻译器",由它负责将OSDL语句的内部表示翻译成底层的数据库操作,例如,关系的SQL或网状的导航语言.这些操作以函数的形式保存,最后由"结果安装器"对原来的OSDL语句替换从而得到最终的C源程序.图3的C_OSDL已在ORACLE的基础上实现,目前正在运行.

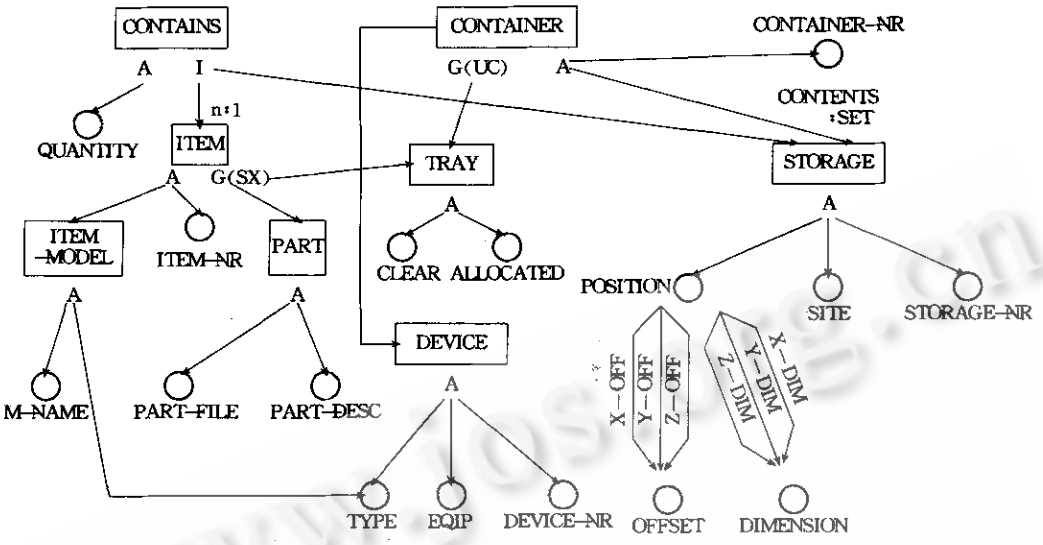

图2　语义图表示的数据库模式

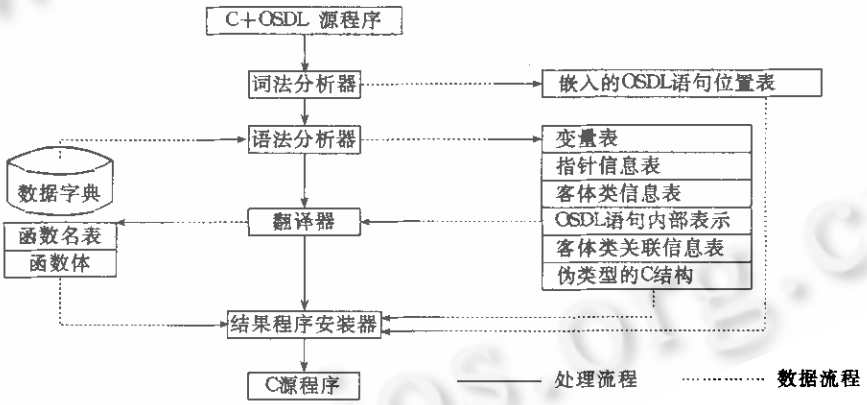

图3　C—OSDL预编译程序结构及工作流程

2　分布式 C 预编译程序及预编译过程

对集中式 C_OSDL 略加扩充,我们便可得到分布式 C 预编译程序——分布式 C_OSDL. 扩充的具体方法如下:

1. 原有的 DEFINEDB 命令除了要求提供用户口令及数据库名外(附录 1 的第 6 行中 gp1 为口令,cambase 为数据库名),增加一个可选的参数用来指定数据库所在场地.

2. CONNECTDB 语句将根据 DEFINEDB 指定的场地联结远程的 OSAM * 数据库.

因此,分布式 C_OSDL 的使用十分简单. 例如,若附录 1 中的数据库 cambase 位于场地 @yournode,则其它场地的用户只需将附录 1 的第 6 行改成

DEFINEDB gp1/cambase/@yournode'

就可以使用该程序在本场地完成对该数据库的插入及检索操作.

分布式 C 预编译程序可以在集中式 C 预编译程序的基础上实现.它的预编译过程描述如下:

1.分布式 C_OSDL 对含有 OSDL 的 C 源程序扫描,识别出该程序要访问的远程 OSAM * 数据库以及与该数据库操作有关的宿主变量定义,指针层次定义与 OSDL 数据操纵命令.

2.为识别出的远程数据库所在场地生成一个称之为代理(Agent)的 C 程序.代理程序含有在上阶段识别出的在远程数据库上使用的宿主变量定义、指针层次定义以及 OSDL 数据操纵语句.此外,代理程序还含有网络通信命令.这些通信命令将在运行时完成与主场地程序(见下步关于主场地程序的的描述)之间的命令交换与数据交换.

3.为步骤(1)中的嵌有 OSDL 语句的 C 源程序生成一个称之为主场地程序(Master)的 C 源程序.它是通过将 OSDL 数据操纵语句替换成网络通信命令而产生的.这些网络通信命令与代理程序中网络通信命令相配合完成必要的命令交换与数据交换.

4.远程场地上的集中式 C_OSDL 对代理程序进行预编译,生成可执行目标码.

5.用户所在场地对主场地程序用集中式 C_OSDL 进行预编译,生成可执行的目标码.

经过这样的编译后,用户便可执行主场地程序.在执行过程中,主场地程序会激活位于远程场地的代理程序,两者进行一系列命令交换与数据交换,完成分布式数据处理功能.

3 支持分布数据处理的通信命令及操作

为了实现主场地程序与代理程序之间的命令与数据交换,我们需要一组在程序运行时进行这种交换的通信命令.下面是这组通信命令的描述.

1. activate_agent(site,agentName)

这一命令主要由主场地程序去激活在远程场地的代理程序.参数中的 site 是远程场地的网络结点名,agentName 是代理程序名.

2. send_to(site,buffer)

该命令是主场地程序或代理程序向对方发送数据及命令的主要通信手段.参数 buffer 是存放要发送的命令,与命令有关的宿主变量及它们的值的缓冲区,它具备下述形式:

"command#;var1:val1;var2:val2;…;varn:valn"

其中的 command# 是主场地程序用于标识 C 中嵌入的 OSDL 语句的编号.var1,var2,…,varn 是与执行该语句有关的变量名,val1,val2,…,valn 是这些变量的值.

send_to 中的参数 site 指出 buffer 要传送到的场地名.

3. receive_from(site,buffer)

该命令是主场地程序或代理程序用来接收对方命令及数据的主要接口.参数 site 指明从哪一场地接收,而 buffer 则存放接收到的结果.buffer 的结构与 send_to 中的 buffer 参数完全一致.

4.“组装”与“拆卸”操作

在执行 send_to(site,buffer)命令之前,主场地程序与代理程序必须首先把要发送的

OSDL 语句的编号以及与之有关的变量名、变量值组织到 buffer 之中,而后才能调用 send_to 命令.这种在 buffer 中按照约定格式放入上述信息的过程称之为"组装"(packing)操作.

在执行 receive_from(site,buffer)之后,主场地程序或代理程序在 buffer 之中得到了对方发来的与某一 OSDL 语句有关的宿主变量及其值的信息.但这些信息在 buffer 中以字符串形式存放,它们并未和自身程序中的宿主变量发生任何联系.为了获取自身程序中宿主变量正确的值,主场地程序或代理程序必须执行"拆卸"操作(unpacking).由于预编译程序在工作过程中保存宿主变量名,宿主变量类型以及有关的 OSDL 语句的信息,因此"组装"与"拆卸"的代码生成并不困难.

5. 查询结果通信区的信息交换

在对 OSAM * 数据库执行一条 OSDL 语句所规定的操作时,系统通过一个 C 的结构 osdlca 向用户提供其执行结果状态.这些结果状态均表示成 osdlca 的结构成份.附录中的程序含有使用 osdlca 的有关语句.

当通过 CONNECTDB 语句连接远程场地上的 OSAM * 数据库时,由于此后的 OSDL 语句都是针对远程 OSAM * 数据库的操作,因此代理程序必须即时地把它执行的每一条 OSDL 语句所产生的 osdlca 状态信息返回给主场地程序,这一要求可以通过下述办法来实现.

当代理程序每执行完一条 OSDL 语句并进行"组装"操作时,将它的 osdlca 中的各成份及对应的值以"var:val"的形式附加到 send_to 的命令参数 buffer 之中.这样,当执行 send_to(site,buffer)命令时,osdlca 的状态也随之传送给主场地程序.而后主场地程序通过 receive_from(site,buffer)命令从代理程序中得到结果.

4　主场地程序与代理程序的程序结构

假定 P 为嵌有 OSDL 语句的 C 程序.P 所在的场地为 M.P 要使用位于场地 A 的 OSAM * 数据库.由 P 所生成的主场地程序与代理程序运行时的工作方式可描述如下:

1. 主场地程序首先运行,在执行到 CONNECTDB 语句时通过 activate(A,Agent)激活在 A 场地的代理程序 Agent.

2. Agent 打开 P 想操作的数据库并等待主场地程序的下一条 OSDL 语句.

3. 主场地程序继续运行,对于每一条嵌入的 OSDL 语句按下述原则处理,并且用 receive_from命令等待处理结果:

. 产生 OSDL 语句的标识编号

. 执行"组装"操作

. 将组装结果用 send_to 命令发送至代理程序 Agent

4. Agent 通过 receive_from 命令收到主场地程序发来的命令及有关数据,执行"拆卸"操作,并根据语句标识编号执行对应的 OSDL 语句,对其执行结果(包括 osdlca)进行"组装",而后再通过 send_to 命令送回给场地 M 的主场地程序.

5. 主场地程序通过 receive_from 命令获取 Agent 送来的结果并进行"拆卸".

6. 重复步骤 3 至 5 直至主场地程序执行关闭 A 的数据库的 DISCONNECTDB 为止.

7. Agent 程序结束运行.

8. 主场地程序继续执行 P 的其它任务直至 P 的结束.

采用上述工作方式的主场地程序与原来含有 OSDL 语句的源程序 P 大体相同. 它可以在 P 的基础上通过下列变换构成:

1. 去掉对远程数据库的 DEFINEDB 语句及指针层次定义语句.

2. 将 CONNECTDB 语句替换成 activate_agent 命令.

3. 将有关事务处理,insert,delete,update,retrieve,以及围绕指针而使用的 open,close,fetch 等语句替换成下述 C 的代码段:

为执行上述语句的"组装"操作;

send_to 通信命令;

receive_from 通信命令;

对 receive_from 中的结果的"拆卸"操作.

上述工作方式中的代理程序 Agent 主要完成主场地程序要求执行的一系列 OSDL 语句,它的结构如下:

```
Agent()
  {
    OSDL DEFINEDB;          /* 定义源程序 P 指定的数据库 */
    OSDL 宿主变量定义;      /* 由源程序 P 中拷贝而来 */
    OSDL CONNECTDB;
    OSDL 指针层次定义;      /* 由源程序 P 中拷贝而来 */
    for (;;)
      {
      receive_from (M,buffer);        /* 等待主场地传送命令 */
      对 buffer 进行"拆卸"代码,从中得到 OSDL 语句的标识编号 command#;
      switch (command#)
        {
          CASE command_1:
              对应 command_1的 OSDL 语句;
              对于此语句执行结果与 osdlca 的"组装"代码;
          .
          .
          .
          CASE command_k:
              对应 command_k 的 OSDL 语句;
              对于此语句执行结果与 osdlca 的"组装"代码;
        }send-to (M,buffer);      /* 向主场地发送执行结果 */
      }
  }
```

附录1—3详细地给出了一个含有分布查询 OSDL 语句的 C 程序实例及其对应的主场地程序与代理程序实例. 产生主场地程序与代理程序中的"组装"代码(PACKING code)与"拆卸"代码(UNPACKING code)由分布式 C 预译程序生成.

附录1　嵌有 OSDL 查询语句的 C 程序实例

```
/*************************************/
/*   OSDL. SMP1: An example of C-OSDL program.   */
/*************************************/

/*  First indicate the Database to be used  */
    OSDL DEFINEDB 'gp1/cambase';
/*  Define VARs to be used in OSDL statements.  */
    OSDL define section begin
        char dev-type[10];
        int dev-nr,str-nr;
    OSDL define section end;
/*  Define communication area with the database.  */
/*  The osdlca is a C structure                   */
  osdl include osdlca;

main()
  {
    strcpy(dev-type, "cart");
    /*  Open database first.  */
    OSDL connectdb;
    /*  Define a cursor hierarchy and its query.  */
    OSDL DECLARE RESULT c0 /*  c0 is the root.  */
        FROM RETRIEVE device-nr, storage-nr
            CONTEXT DEVICE[type=:dev-type] * STORAGE
            VIEWPOINT DEVICE;
    OSDL DECLARE CURSOR c1 /*  c1 is a child of c0.  */
        FOR STORAGE
        WITHIN c0;

    /*  Insert an instance in DEVICE class  */
    OSDL INSERT DEVICE
        < DEVICE-NR=202, EQIP='V-MTool, TYPE='cart'
        >;

/*  Open the root cursor to get ready for operation.  */
OSDL OPEN c0;
for (;osdlca.osdlcode !=4;)
    /*  Repeatetaly get a DEVICE instance until no  */
    /*  more is available.                          */
    {
        /*  Use c0 to get DEVICE instance.  */
        OSDL FETCH c0
            ATTRIBUTE device-nr
            INTO :dev-nr;
        if (osdlca.osdlcode<0) /*  an error occurs  */
            goto errexit;

        for (;osdlca.osdlcode !=4;)
            /*  Repeatetaly get a STORAGE associated  */
            /*  with DEVICE instance until no more is  */
            {                    /*  available.  */
            OSDL FETCH c1
                ATTRIBUTE STORAGE-NR;
                INTO :str-nr;
            if (osdlca.osdlcode<0) goto errexit;
            /*  Display the required information.  */
            printf("%d %d \n",dev-nr,str-no);
            }
    }
OSDL COMMIT
OSDL CLOSE c0;
OSDL DISCONNECTDB;
exit;
errexit;
OSDL CLOSE c0;
    printf("An OSDL error occurs!");
}
```

附录2　由附录1的 C 程序生成的 Master 程序

```
/*************************************/
/*  The Master C program generated from osdl_smp1 by  */
/*  C-OSDL through replacing OSDL statements. The  */
/*  replaced OSDL statements are masked by comments.  */
/*************************************/

/*****  OSDL DEFINEDB 'gp1/cambase/@yournode';  */

/*  Define VARs to be used in OSDL statements.  */
  OSDL define section begin
      char dev-type[10];
      int dev-nr,str-nr;
  OSDL define section end;
/*  Define communication area with the database,  */
/*  the C structure osdlca.                       */
  osdl include osdlca;

main()
  {
  char *buffer[];    /*  used to trnsmit command & data  */

  strcpy(dev-type, "cart");
  /*  Open database first.  */

  /*****  OSDL connectdb;  */

  activate-agent('@yournode',MyAgent);

  /*  Insert an instance in DEVICE class  */
  /*****  OSDL INSERT DEVICE
      < DEVICE-NR=202, EQIP='V-MTool, TYPE='cart'
      >;  */

  PACKING code; /*  for INSERT DEVICE with
                command #='INSERT1'  */
  send-to(@yournode,buffer);
  receive-from('@yournode',buffer);
  UNPACKING code; /*  for INSERT DEVICE  */

  /*  Open the root cursor to get ready for operation.  */
  /*****  OSDL OPEN c0;  */

  PACKING code; /*  for open c0 with command #
                ='OPENc0'  */
  send-to('@yournode',command-buffer);
  receive-from('@yournode',result-buffer);
  UNPACKING code; /*  for OPEN c0  */

  for (;osdlca.code !=4;)
      /*  Repeatetaly get a DEVICE instance until no  */
      /*  more is available.                          */
      {
```

```
            /*   Use c0 to get DEVICE instance.     */
            /* * * * * OSDL FETCH c0
                ATTRIBUTE device-nr
                INTO ,dev-nr;/*

            PACKING code; /*   for FETCH c0 with
                        command# = 'FETCHc0'    */
            send-to('@yournode',buffer);
            receive-from('@yournode',buffer);
            UNPACKING code; /*   for FETCH c0    */

            if (osdlca.code<0)
                        /*   an error occurs     */
              goto errexit;

            for (;osdlca.code != 4;)
                /*   Repeatetaly get a STORAGE associated    */
                /*   with DEVICE instance until no more is   */
                {           /*   available.     */
                  /* * * * * OSDL FETCH c1
                        ATTRIBUTE STORAGE-NR;
                        INTO ,str-nr; */

                PACKING code; /*   for FETCH c1 with
                        command# = 'FETCHc1'    */
                send-to('@yournode',buffer);
                receive-from('@yournode',buffer);
                UNPACKING code; /*   for FETCH c1    */

                if (osdlca.code<0)
                    goto errexit;

                /*   Display the required information.     */
                printf("%d %d \n",dev-nr,str-no);
                }
            }
        /* * * * * OSDL COMMIT; */
        PACKING code; /*   for COMMIT with command#
                = 'COMMIT'   */
        send-to('@yournode',buffer);
        receive-from('@yournode',buffer);
        UNPACKING code; /*   for COMMIT    */

        /* * * * * OSDL CLOSE c0;     */
        PACKING code; /*   for CLOSE c0 with command#
                = 'CLOSEC0'    */
        send-to('@yournode',buffer);
        receive-from('@yournode',buffer);
        UNPACKING code; /*   for CLOSE c0    */

        /* * * * * OSDL DISCONNECTDB; */
        PACKING code; /*   for disconnectdb with command#
                = 'DISCONNECTDB'    */
        send-to('@yournode',buffer);
        receive-from('@yournode',buffer);
        UNPACKING code; /*   for disconnectdb    */

        exit;
        errexit:
            /* * * * * OSDL CLOSE c0; */
            PACKING code; /*   for CLOSE c0 with
                        command# = 'CLOSEC0'    */
            send-to('@yournode',buffer);
            receive-from('@yournode',buffer);
            UNPACKING code; /*   for CLOSE c0    */
            printf("An OSDL error occurs!");
    }
```

附录3 由附录1的C程序生成的Agent程序

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
/*   The Agent C program generated from osdl.smp1 by    */
/*   C-OSDL.                                            */
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

OSDL DEFINEDB 'gp1/cambase';

/*   Define VARs to be used in OSDL statements.     */
  OSDL define section begin
      char dev-type[10];
      int dev-nr,str-nr;
  OSDL define section end;
/*   Define communication area with the database,    */
/*   the C structure osdlca.                        */
  osdl include osdlca;

Agent()
  {
    char *buffer[];   /*   used to transmit command & data   */

    OSDL connectdb;

    /*   Define a cursor hierarchy and its query.     */
    OSDL DECLARE RESULT c0
        FROM RETRIEVE device-nr, storage-nr
        CONTEXT DEVICE[type=,dev-type]*STORAGE
        VIEWPOINT DEVICE;
    OSDL DECLARE CURSOR c1
        FOR STORAGE
        WITHIN c0;

    for(;;)
    {
        receive-from('@Mnode',buffer);
        UNPACKING code;

        switch(command) /*   command is from UNPACKING    */
        {
          case "INSERT1":
              /*   Insert an instance in DEVICE class    */
              OSDL INSERT DEVICE
                  < DEVICE-NR=202, EQIP='V-MTool,
                      TYPE='cart'>;
              PACKING code; /*   for INSERT1;    */
          case "OPENc0":
              /*   Open the root cursor to get ready
                          for operation.    */
              OSDL OPEN c0;
              PACKING code; /*   for OPENc0;    */
          case "FETCHc0":
              /*   Use c0 to get DEVICE instance.     */
              OSDL FETCH c0
              ATTRIBUTE device-nr
              INTO ,dev-nr;
              PACKING code; /*   for FETCH c0    */
          case "FETCHc1":
              OSDL FETCH c1
                  ATTRIBUTE STORAGE-NR;
```

```
59              INTO :str—nr;
60          PACKING code; /*   for FETCH c1   */
61      case "COMMIT":
62          OSDL COMMIT;
63          PACKING code; /*   for COMMIT   */
64      case "CLOSEc0":
65          OSDL CLOSE c0;
66          PACKING code; /*   for CLOSE c0   */
67      case "DISCONNECTDB":
68          OSDL DISCONNECTDB;
69              PACKING code; /*   for DISCONNECT DB   */
70          } /*   end of CASE   */
71
72          send—to("@Mnode",buffer); /*   send back the result
73                  of the execution of a command   */
74          if command == "DISCONNECTDB"
75              break; /*   the time to get out the loop   */
76      } /*   end of loop   */
77      /*   AGENT's work is over!   */
78  } /*   end of Agent   */
```

参考文献

1 Su S *et al*. An object—oriented semantic association model (OSAM＊). In: Kurama S *et al* eds. A. I. in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, American Institute of Industrial Engineering, 1988.

2 张麟,王国仁,李贵等.CIMSERC_DDBMS 全局数据模型 OSAM＊.第一届中国计算机集成制造系统学术会议,北京,1990.

3 周立柱,王小京,衣丰超等.面向对象的语义关联数据模型查询语言在 C 语言中的嵌入.软件学报,1992,3(1):60—64.

4 周立柱,王健斐.面向对象数据库的 C 宿主语言接口实现技术.软件学报,1994,5(1):1—9.

THE IMPLEMENTATION STUDY ON A DISTRIBUTED C PRE_COMPILER FOR CIMBASE

Zhou Lizhu and Wang Xiaojing

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

Abstract　CIMBASE is a distributed heterogeneous database management system designed for Computer Manufacturing Systems. One of CIMBASE's important user interfaces is a C pre_compiler that enables a user to access remote databases in C programs. This paper first describes the main functions of this C pre_compiler and then discusses its implementation issues, especially on the design of supporting communication commands, Master/Agent program structure, their working mode, and some related algorithms as well.

Key words　Object_oriented database＊, distributed data base, pre_compiler＊.