重新输入。

下面我们看一个简单的例子，如图3，一个简单的有限状态机，其中状态{s_0,s_1,s_2,s_3,s_4}用结点表示，弧表示状态间转换，弧上的字母表示输入。

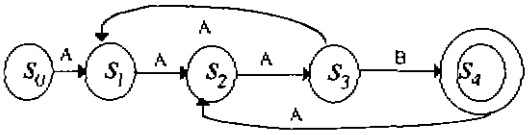

图3

在状态 s_i 时，输入字母 I 使状态转换至 s_k，当且仅当 s_i、s_k 之间有以 I 为标记的弧，否则进入错误状态。终止状态由双层圆圈表示。

众所周知，每一个有限状态自动机定义了一种正规语言，反之，任一正规语言也必可由一有限状态自动机表示。图3表示了正规语言 aaa[aaa|baa]*b。

我们 JAVA 语言实现了该自动机。自动机中每一状态{s_0,s_1,s_2,s_3,s_4}由一个带隐含属性的 FORM 表示。图4输出的 FORM 表示状态 s_1：

```
static String FORM1="<h1>This is state 1</h1>\n
    <SELECT NAME=\"input\"><OPTION>A\n</SE-
LECT>\n
    <INPUT TYPE=\"submit\"VALUE=|"Step\"SIZE
=50}";
System. out. println("<FORM METHOD=\"POST\"AC-
TION=\""+WEB_exe_filename+"\")\n");
System. out. println("<input type=\"hidden\"
    name=\"WEB_state\"
    value=1)\n");
System. out. println("<input type=\"hidden\"name=\"
name\"value=\""+name+"\")\n");
System. out. println("<input type=\"hidden\"
    name=\"input\"value=\""+input+"\")\n");
System. out. println(FORM1);
```

图4

其中 WEB_exe_filename 为 CGI 程序名，从而是一个递归 CGI 程序，当该 FORM 被提交时，当前的状态及用户的输入作为参数传递给 CGI 程序，从而确定了下一状态，一旦下一状态被确定，就产生了代表下一状态的 FORM，并返回给浏览器。若不能确定下一状态，则进入错误状态，用户可重新进入初始状态。若下一状态为终止状态，用户可退出或重新进入初始状态。下面给出了上例的算法描述：

步骤1：读取环境变量，确定 CGI 参数传递方式为 POST 方式。

步骤2：由标准输入中读入 CGI 参数，形成名字/值对方式。

步骤3：确定用户当前的状态及输入，根据状态转换表得到下一状态。

步骤4：输出新状态对应的 FORM。

图3中的有限自动机只有5个状态，很容易实现。通常情况下，状态转换表会复杂得多。因此，我们需要用简单的方法来生成 CGI 程序。

由以上分析，我们不难看出，该方法实现的关键是状态自动机的状态转换描述表和代表不同状态的 FORM。因此我们可以开发一种 CGI 生成工具，通过输入不同的状态转换描述表和代表不同状态的 FORM，自动生成递归 CGI 程序，在 Web 上实现该自动机。

参 考 文 献

[1] Louis Perrochon, W3" Middleware": Notions and Concepts, Workshop on Web Access to Legacy Data, Boston, MA, Dec. 1995

[2] L. Shklar et al., Putting Legacy Data on the Web: A Repository Definition Language, WWW'95, Darmstadt, Germany, Computer Networks and ISDN Systems, 27(6), Elsevier Science, 1995

[3] J. Gosling and H. McGilton, The Java Language Environment: A White Paper, Sun Microsystems, Mountain View, CA, May 1995

（上接第109页）

参 考 文 献

[1] B. Cambell and J. M. Goodman, HAM: A General Purpose Hypertext Abstract Machine, CACM, 31(7) 1988

[2] H. A. Schutt, and N. A. Streitz, Hyperbase: A Hypermedia Engine Based on a Relational Database Management System. Proc. of the European Conf. on Hypertext(ECHT'90), Versaille, France, Nov. 1990

[3] J. J. Legget and J. L. Schnase, Viewing Dexter With Open Eyes, CACM, 37(2)1994

[4] K. Gronbak et al, Cooperative Hypermedia Systems: A dexter-based Architecture, Same to[3]

[5] 李光亚、周学海、赵振西，超媒体系统的开放性探析，计算机科学，24(4)1997

[6] http.\\www.cs.wisc.edu\shore, An Overview of Shore, Aug. 9, 1997

[7] 李光亚，周学海、龚育昌、赵振西，一种基于语义网络的开放式超媒体系统结构，计算机科学，25(3)1998

[8] 李光亚、吴海英、文栋辉，开放式超媒体系统链接协议 OHP2.0，中国科技大学计算机系信息处理实验室技术报告，1997.11

计算机科学1998Vol.25 №.5

支持开放性的超媒体引擎

A Hypermedia Engine Which Supports Openness

文栋辉 李光亚 赵振西

(中国科学技术大学计算机系 合肥230027)

摘 要 This article analyzes the design method of hypermedia engine and presents an open hypermedia engine. The engine supports the specification of application semantics as application class and supports complex operations maintaining constraints, thereby simplifies the modeling of hypermedia applications.

关键词 Hypermedia engine, Open hypermedia application, Object-Oriented database, Semantics constraint

1. 引言

超媒体引擎是支持超媒体应用开发的抽象机，它提供了访问和操作超媒体结构的接口，利用这些功能可以容易地表达各种基于具体应用的超媒体系统数据模型；超媒体引擎还要提供复杂的操作来维护应用的语义。目前大多数开放式超媒体系统模型(OHS)[5]，如 Microcosm、DHM、SP3和 HyperDisco 等，主张将超媒体结构和内容分开存储，第三方应用在自己的存储系统中编辑文档，可以通过开放式连接协议(OHP)[5,8]将编辑内容集成到超媒体系统，开放式的超媒体引擎要提供灵活的机制来支持这种开放性。

研究者们已经开发出一些比较成功的超媒体引擎模型，但是这些模型，如 HAM[1]、HyperBase (GMD-IPSI)[2]，提供了一种相当固定的超媒体数据模型，通过增加属性来扩展语义，超媒体应用的开发者必须自己弥补应用数据模型和超媒体引擎数据模型的不匹配。另外一些模型，如 HB3[3]和 Devise[4]，过分强调通用性，只是简单地提供了存储功能，大部分工作都得由超媒体应用来完成，而且这些模型都不是针对开放式超媒体应用的，也没有考虑到超文档的分布存储。本文认为一个开放式超媒体引擎应该满足以下的前提：

(1)可剪裁的数据模型。能够抽象而准确地表达各种具体应用的超媒体对象。

(2)灵活的语义表达，可以方便地描述超媒体应用复杂丰富的语义。

(3)开放性的充分体现。充分支持开放式的链接协议，能方便地集成第三方应用，也可以将外部的各种存储系统透明地管理起来，如面向对象数据库系统、关系数据库和文件系统。

(4)强大的数据库功能。支持持久对象的多用户访问、并发控制、恢复和版本控制等。

本文提出了一个建立在面向对象数据库 Shore 之上的开放式超媒体引擎模型，它支持丰富的语义操作，包括各种应用类的定义和语义约束的复杂操作，提供灵活的机制支持具体超媒体应用的剪裁；它支持开放式超媒体系统，具体表现上是支持开放式超媒体链接协议，通过引擎可以透明地访问第三方应用的数据源。

2. Shore 面向对象数据库

Shore(Scalable Heterogeneous Object Repository)[6]是 Wisconsin 大学开发的一个面向对象数据库，它具有许多优秀独特的特征：

◇ 功能强大的对象定义语言 SLD，所有的 Shore 对象都是接口类型(Interface Type)的实例，

文栋辉 硕士生，主要研究兴趣为开放式超媒体系统、面向对象数据库。李光亚 博士生，主要研究领域为超媒体系统和多媒体数据库。赵振西 教授，博士生导师，主要研究领域为数据库系统、面向对象程序设计和多媒体系统。

它有方法、属性和关系等。用 SDL 定义好对象类型以后，可以直接送入数据库。

◇ 面向对象数据库的特征。如并发控制、事务管理、恢复、锁机制等等，

◇ 文件系统的特征。Shore 可以通过 Unix 的文件系统调用来操作 Shore 中的对象，这样，基于文件系统的第三方应用的数据就比较容易集成到开放式超媒体系统中。

3. 开放式超媒体引擎的设计

我们在 Shore 数据库上开发了一个开放式超媒体引擎。它不仅仅能灵活表达复杂丰富的超媒体语义，而且还直接支持开放式超媒体链接协议(OHP)，从而能方便地将第三方应用集成到具体的超媒体应用系统。

对象分类。为了能让应用程序能实时地定义具体语义的超媒体对象，我们将 Shore 中的对象类型分为两类：元对象类(Meta_Class)和实例对象类(Inst_Class)。Meta_Class 的实例称之为元对象，而 Inst_Class 的实例称之为实例对象。元对象用来描述实例对象的属性格式和语义约束规则，一个元对象可以对应多个实例对象，相同种类的实例对象只能对应一个元对象，它们之间是描述与被描述的关系，如图 1。其中，Meta_ Class，Inst_ Class，Meta_ Link，Link_Class 是用 Shore 的 SDL 定义的类，T_IsBaseOn 是应用程序定义的元对象，表示一种新的链类型，IsBaseOn1，IsBaseOn1 是实际的链(应用对象)，它们的属性和语义约束由 T_IsBaseOn 表示。

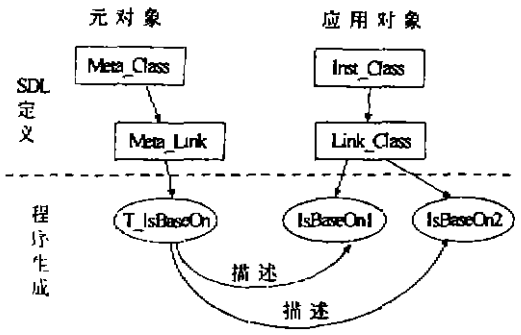

图1

元对象。依照开放式超媒体协议(OHP)，引擎提供了8种元对象类型，并且对于每种元对象类型，内置了一些常用的元对象，现说明如下：

- Meta_Anchor。负责创建所有的锚元对象。内置了三个元对象，其中文本锚用来表示文本文件的锚，位图锚用来表示 BMP 文件的锚、Word 锚用来表示 Word 文件的锚。
- Meta_Link。负责创建所有的链元对象。内置了两个元对象：有向链、无向链。
- Meta_Node。负责创建所有的结点元对象。内置了六个元对象：原子结点表示单一内容的结点、文本结点、位图结点和 Word 文档结点都是它的子结点；组合结点是多个结点组成的结点；而虚拟结点是通过自定义数据源得到内容的结点，例如通过 SQL 查询或存储过程得到结点的内容。
- Meta_Document。负责创建所有的文档元对象。内置了一个元对象、它描述了超媒体文档的位置、显示方式等等。
- Meta_Constraint。负责创建所有的语义约束元对象。内置了两个元对象：链约束表示链的两端结点的类型约束；组合约束表示组合结点的各个子结点的类型约束。
- Meta_Pspec。负责创建所有的展示元对象。内置了一个元对象、用来表示展示超媒体对象的环境，如应用程序的位置、显示环境等等。
- Meta_Service。负责创建所有的服务元对象。内置了一个元对象，提供一般的数据库和超媒体功能服务。
- Meta_Perspective。负责创建所有的视图元对象。内置了一个元对象，用来描述视图定义。

对于元对象，引擎还提供了类似于 C++ 的继承机制，如果元对象之间有父子关系，子元对象可以继承父元对象的属性格式和语义约束，见图2。链1和链2从有向链继承而来，具有有向链的特征，分别捆绑了链约束[A，B]和[A，C]；链3从链1和链2继承而来，因此，它隐含地捆绑了链约束[A，B]+[A，C]；通过继承机制，提高了元对象的语义描述能力。

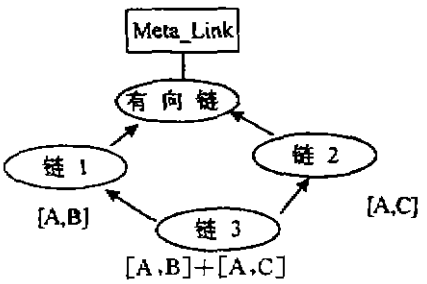

图2

对于语义约束，引擎提供了灵活的定义和捆绑机制，例如，应用程序可以继承链约束，自定义一种

语义约束 Constraintl，并且可以动态地和有向链 Link1进行捆绑，或者释放捆绑：

```
Meta_Constraint * Constraintl;
Constraintl=new Meta_Constraint(DirectedLink);
Constraintl->AddAttribute(a1,a2,a3);
Link1. BindConstraint(Constraintl);
Link1. UnBindConstraint(Constraint1);
```

这种机制充分体现了引擎的可剪裁性。

实例对象。当设计者对引擎进行剪裁，或者用户编辑超文档的时候，引擎会创建各种实例对象，它们和元对象有着被描述的关系。引擎提供了以下的实例对象类，它们和元对象类一一对应：

- Anchor_Class。负责创建所有的锚实例对象。
- Link_Class。负责创建所有的链实例对象。
- Node_Class。负责创建所有的结点实例对象。
- Document_Class，负责创建所有的文档实例对象。
- Constraint_Class，负责创建所有的语义约束实例对象。
- Pspec_Class，负责创建所有的展示实例对象。
- Service_Class。负责创建所有的服务实例对象。
- Perspective_Class。负责创建所有的视图实例对象。

当引擎创建实例对象时，从元对象中，它可以理解 Viewer 发过来的数据，从而在 Shore 数据库中建立真正的持久对象。如，Link_Class * lpAl = new Link_Class(T_IsBaseOn)；T_IsBaseOn 是设计者定义的链元对象，描述了实际的链 lpAl 的属性格式和语义约束规则。

实例对象组成的有机网络在元对象上的映射视图，称之为元视图。它表达了元对象的属性和它们之间的关系。通过对元视图的操作，可以增删元节点，修改锚的属性，定义链，捆绑新的语义约束，并且这些操作也反映在实例视图上。引进元视图的概念，使得对引擎进行可视化剪裁成为可能。

（下转第93页）

（上接第116页）

```
      S=0
      DO 130 K=1,10
      DO 140 M=1,10
      S=S+1.0D+00
      CALL F_AUTOPAR_LOOP_SPLIT (8.2, F_AU-
     TOPAR_LOOP_START, & F_AUTOPAR_LOOP_
     END)
      DO 150 J = F_AUTOPAR_LOOP_START, F_AU-
     TOPAR_LOOP_END
      DO 160 I=1,10
      A(I,J,K,M)=1010.0+j
      B(J,I,K,M)=20.0+j
160 CONTINUE
150 CONTINUE
140 CONTINUE
130 CONTINUE
      CALL f_autopar_set_range_d(3,1,10)
      CALL f_autopar_set_range_ud(2,0,0)
      CALL f_autopar_set_range_d(1,1,10)
      CALL f_autopar_set_range_d(0,1,10)
      CALL f_autopar_layout(a)
      CALL f_autopar_collect(a)
      CALL f_autopar_set_range_ud(3,0,0)
      CALL f_autopar_set_range_d(2,1,10)
      CALL f_autopar_set_range_d(1,1,10)
      CALL f_autopar_set_range_d(0,1,10)
      CALL f_autopar_layout(b)
      CALL f_autopar_collect(b)
      CALL F_AUTOPAR_OVER
      END
```

很明显，该程序并行的是150循环。在这里，读者应该特别注意的是那些小写字母表示的 AUTOPAR 函数调用语句，它们本应该在150 CONTINUE 和140 CONTINUE 之间生成，但由于数组 A，B 都没在循环130和140中被引用，因此经过通讯优化分析，它们被放到了130循环之外。

结论 我们简要地介绍了曙光2000上的并行库和并行识别工具 AUTOPAR。在经过对大量 BENCHMARK 程序(NASA，PERFECT 等)的实际测试后，我们发现尽管 AUTOPAR 能够并行大部分 DO 循环，但实际的运行效率并不高，主要瓶颈在于数据收集函数 COLLECT(通信)。因此，该系统今后还应该在并行库优化和通信优化上下工夫。虽然如此，该系统对一些通讯少的应用，还是能得到很好的并行效果。如对三维迭前深度偏移(克希霍夫方法) mig. f 就能使加速比大体呈线性。

参 考 文 献

[1] 吉敏阳，并行重构系统后端[硕士学位论文]，1994

[2] 张兆庆、乔如良，利用超级编译技术优化串行程序，软件学报，1995年增刊

[3] 吉晓梅等，含数组引用的过程间数据流分析，软件学报，1995年增刊

[4] 高念书等，实用数据依赖分析方法，计算机学报，1995年4月

[5] 张兆庆、乔如良，PORT：并行优化重构工具集，计算机学报，1994年12月

[6] 刘任，MPP 机上代码生成技术，硕士论文，1995

[7] ASPAR—An Automatic Parallelizer for C Programs，1991