CAD CAM 动态模式处理

计算机科学 1994 Vol. 21№. 3

31-34

一种动态模式处理策略

TP391.72

张新访 王同洋 杨志新

(华中理工大学 CAD 中心 武汉 430074)

A

摘 要 本文论述了一种基于转换法的动态模式处理策略：动态模式操作应遵循的不变性原则；模式动态变化的分类及其操作相应的语义。并讨论了保持行为一致性、并发控制、访问权限等对动态模式处理的影响。

关键词 数据库管理系统，动态模式

一、引言

适合于机械 CAD/CAM 的工程数据库管理系统 O_2S(Object Oriented Semantic System)是我们在 Sun 工作站上研制的面向对象数据库管理系统，它基于面向对象语义数据模型 EO_2SD(Engineering Object Oriented Semantic Data-model)，本文阐述 O_2S 的动态模式处理模型。

EO_2SD 用对象及其分类来描述 CAD/CAM 中的各类实体、现象和概念；用对象依赖、聚合、概括、引用四种语义关系来描述设计对象间的语义关系；用可建和不可建版本对象类来分别描述动态数据和静态数据。可建版本对象类的所有实例构成设计对象的版本导出历史。

二、动态模式问题与处理策略

工程应用领域的数据在结构和行为方面都表现出动态性，是区别于商业应用领域的主要特性之一。不像商业应用主要是静态环境，工程应用既有静态环境，又有动态环境。动态环境要求数据库有支持动态环境建模的功能。因为设计问题有其固有特性：渐近性和反复性。设计人员对设计对象的认识并不是事先能定义好的，随着设计者对设计对象的逐步了解，将逐步往设计对象添加信息，用后来的决策修改前面的设计决策也是常事。

和动态环境相关的就是动态模式问题，模式的修改对应设计环境的变更。所谓支持动态模式，也就是当对旧模式加以修改产生新的模式时，系统应根据新旧模式之间的结构和行为差别，自动使旧模式下的对象能适应新模式环境，保持结构和行为上的前后相互一致。

现有的面向对象系统具有少量的几种对象定义的修改功能，主要原因是这些系统为程序系统，使用的是先束定(这是和 Late-binding 相对应的一个概念)的原则，对类定义的修改将极大地使程序系统复杂化[1]。传统的数据库系统绝大部分要求在没有具体数据的情况下修改模式，不属动态模式的范畴，动态模式指的是模式描述与数据同时发生动态变化。

处理动态模式有二种方法：一是过滤法(screening approach)，二是转换法。过滤法也叫迟后修改法，其基本思想是：当模式发生变更时，并不立即修改外存中的内容以适应新的模式环境，而是记载环境变更的历史，在某一环境和前面的历史之间建立一种过滤机制，在旧的实例被新环境下的应用使用时，旧的实例对象得到过滤和更正，以和新环境相匹配，达到正确使用。采用这种方法的关键问题是记载环境历史和过滤机制的选定。和过滤法不同，转换法在模式修改时立即对旧模式下的对象进行修改以适应新的环境。这两种方法的差别在于前者保留环境发展的历史，而后者仅保留了环境发展结果。

这二种处理方法代表了一种权衡：把时间花在使用时和把时间花在修改时的权衡。对于前一种方法，使用某一对象实例时系统所耗费的时间包含过滤所需的时间；对后者，不需耗费额外的时间，因为在修改时已完成了转换。

在动态模式研究方面，最为深入的可能算 Skarra 和 Zdonik 的研究[1]。他们提出的方法是过滤筛选，用类版本来记载类的发展历史，用一种对用户透明的处理器(hander)来处理不同类版本下的实例对象的使用问题。对每一类版本，处理器作为版本中的一部分，亦即作为类定义的一部分，不同的处理器适合于

张新访 博士后，主要研究兴趣：面向对象数据库等。

不同的情况。Skarra 把处理器分成前、后处理器，不仅处理结构方面的问题，还处理诸如继承某行为方面的一致性问题。Banerjee 和 Jason 等人也对动态模式作了研究，其研究成果分别用于面向对象数据库管理系统 ORION 和 Gemstone 中[3,4]，所采用的方法是转换法。这二个系统共同的特点是：都是在面向对象的编程语言上发展起来的面向对象系统，他们的研究和语言环境有很大关系，是将模型和动态模式相结合的研究。

O_2S 的动态模式处理采取的是转换法。之所以不采用过滤法是因为：第一，在设计过程中，模式的变更和版本都用来描述对象的动态性，大多数设计者都是在一定模式下进行变型设计（如变动某一参数的设计），设计者往往把更多的注意力放在具体的设计参数的变化上，如对直齿圆柱齿轮的设计，其设计参数就是为模数 m、齿数 z、和齿轮内径等确定模式的参数。即使是全新设计，设计者将对某一设计对象描述成层次的，每一次设计基本上局限于设计层次的叶节点上，尽管有回溯设计过程，这种回溯也主要是针对一定模式下的某一参数的修改。因此，没有必要让系统花费很大的力气去保留类的历史。第二，每一设计对象类下的对象实例一般很少，通常仅为 1～2 个，转换所花费的代价并不是 Skarra 所描绘的那样高；再则，对设计数据使用频繁，多次使用过滤的时间消耗远远比一次转换消耗的多。第三，系统要求用户过多地参与过滤处理（如前述的用户透明的处理器），会使用户感到复杂而无所适从，如果版本数据多，所需的处理器的数目也将急剧增加。

当然，对方法和应用程序而言，系统是没有办法自动修改以适应新的环境，过滤法避免了这个问题。然而我们认为，尽管方法和应用程序应该适应新环境而存在，但在无法自动适应的情况下使用人工干预的办法（如重新编译或使用数据库系统提供的功能）是必需且可行的，Skarra 的方法实际上是把实现自动化的负担放在了数据库维护人员身上。

三、不变性规则

所谓不变性规则是指模式在变化前后都应保持的一些特性，这些特性来源于 EO_2SD 数据模型本身的要求。

3.1 表达不变性 对象的表达由其所属类决定，其存贮格式、尺寸、属性和对属性的约束都由对象类决定。因此，在转换过程中，应保证经转换后的实例在以上各方面和新的环境保持一致。

3.2 类格不变性 对 EO_2SD 类格是一有根的有向非循环图 DAG 或有向循环图 DCG。DAG/DCG 图有唯一的一个根节点 OBJECT。因此，在类格图中，用户定义的类节点不能是孤立的，它至少有一个扇入。

3.3 全继承 某一对象类必须从其超类中继承所有的属性和方法，应该没有选择地继承，除非在继承过程中发生了某种冲突（如名冲突），在发生冲突时，按 EO_2SD 规定的方式解决冲突。

3.4 严格子集 类和其超类之间严格的子集关系，对继承属性只能施加相等的或更为严格的约束，如对象类 T 的域 D1 是从其超类 ST 中的 D2（用不同名以示区别）继承而来，则 D1 的值域是 D2 的值域的真子集或相等。

3.5 语义关系一致 在对象转换过程中，应维持其原有的语义关系，如对象唯一依赖关系、共享依赖关系、引用依赖关系等。引用对象、引用指针和依赖对象三者以整体方式存在，原则上不允许出现引用指针悬空状态，也不允许出现依赖对象不被引用的状态，除非这二种不完整状态得到用户的许可。

四、模式动态变化分类

O_2S 允许用户动态改动类格中的节点、边以及类与类之间的语义关系，对节点的修改又包括两个方面：对节点本身的修改和节点的增删。下面是模式动态修改的分类。

4.1 对象类的修改

对一个对象类的修改，包含以下 5 种：

（1）加一个新属性；（2）删除一个属性；（3）改变属性名；（4）修改属性的值域；（5）修改属性的继承性。

4.2 对类格的修改

对 DAG/DCG 边的改变，包括以下 4 种操作：

（1）使类 S 成为类 C 的超类；（2）从 C 的超类表中删除一类 S，使 S 不再成为 C 的超类；（3）使一类 C 成为类 S 的子类；（4）删除某一超类 S 的子类 C。

4.3 对类格节点的修改

对 DAG/DCG 节点的修改，包括以下 3 种操作：

（1）加一新类；（2）删除一类；（3）改变类名。

4.4 对象类之间的语义关系的修改

这类操作包括 6 种操作：

（1）撤消对象类之间的依赖语义关系；（2）对聚合关系增加一成分类；（3）删除一聚合关系成分类；（4）撤消引用关系；（5）增加对象类之间的引用关系；

(6)增加对象类之间的依赖关系。

五、模式对象变化的语义

不变性原则表明了模式在静止状态时应保证的特性，它们对确定每一模式动态变化操作的语义有指导作用。下面依次阐述 O_2S 每一动态模式操作的语义，也就是应采取的方法和步骤。在具体叙述之前，设定 O_2S 系统允许属性值为空，且用 NULL 表示。在属性不为逻辑类的情况下，空值表示某种不完整状态，如引用指针的空值表示悬空状态，是不一致状态，数据库系统提供处理空值的函数。

如果属性 P 加入到类 C 中作为新属性，C 类的所有实例增加一属性 P，已有实例对象的 P 属性值置空为 NULL；转换 C 类的实例对象时，如果对象的标识改变了，则应将新的对象标识 OID 填写到引用指针处，并把需要进行修改传播检查的属性的对象之改动时标置为当前时间，C 类的所有子类继承 P 属性。如果发生冲突，按 EO_2SD 数据模型的名冲突解决原则处理，如果冲突无法解决，则不允许用户添加新属性 P，否则对每一个子类执行以上操作，直到所有的子类传播完毕。将 C 类的实例转换后，可能引起引用对象的确认修改时标小于引用修改时标，形成引用不一致，用户可以通过检查引用一致性的工具检测到这种引用不一致性，以做进一步的修改。

从类 C 中删除一个属性 P，会对 C 类的实例对象进行压缩，以保证在存贮格式上删除属性 P，若经压缩后实例对象标识 OID 改变了，则应确定引用对象，将引用对象的引用指针置成新的对象标识。当该对象的引用对象的引用属性需要进行修改传播检查，则应把对象的改动时标 CN 置成当前时间；C 类的所有子类执行以上操作，直到所有子类完成转换。

改变属性名要传播到各子类中；修改属性的值域产生和删除某一属性相类似的操作；修改属性的指针要反映到修改类及其子类中，如果修改传播性导致了存贮结构的变化，则会产生同上面一样的保证引用完整性的修改操作。

对使类 S 成为类 C 的超类的动态模式修改，在类格中，应从 C 到 S 加一条边，C 及 C 的子类继承 S 的全部属性和约束方法，C 及子类的实例对象中新继承的属性值为 NULL。反复调用添加属性的模式动态改变操作完成属性的继承。

从 C 的超类表中删除一类 S，在类格中删除 C 到 S 的一条边，若 S 为 C 的唯一超类，则 S 的直接超类成为 C 的直接超类，应保证 C 不会成为孤立的节点。C 从 S 的直接超类中继承的属性未变，失去的仅仅是 S 中定义的那部分属性，因此对 C 及 C 的子类反复施加删除一属性的操作。

使类 C 成 S 的子类，首先必须保证 S 的属性值是 C 及 C 的子类的属性集的子集，且 C 和 S 的相同属性的值域是 S 中相同属性的值域的子集，否则不允许这类操作，对 C 进行抽象操作，保证实例存在依赖关系和严格子集关系。

若删除类 C，可使用删除超类的操作删除 C 到所有子集的边；删除 C 的所有子类；删除 C 的所有实例，对 C 类有引用关系的引用指针全部置空；删除和 C 相关的语义关系；最后删除 C 的定义。

撤消对象类之间的依赖语义关系，首先把实现语义关系的系统定义属性删除，并使存贮格式与类定义一致。撤消引用关系的操作与上相同。增加对象类之间的引用关系、增加对象类之间的依赖语义关系是前面两种操作的逆操作。

对聚合关系增加/删除一成分类会导致存贮格式的变更，其语义动作和对一般类增加一属性的操作相同，只是要注意，由于对复杂对象采用的可能是集结存贮方式，这种动态模式操作可能会对尺寸很大的存贮区进行重新集结，频繁的操作必将耗费很多时间，因此在设计存贮格式时应适当地考虑到集结的修改与修改频度。

以上各类模式动态变更操作应具有原子性，且不论在任何一层发生了不允许的情况，则终止整个修改。另外，对对象引用问题，修改时可能导致修改的传播，应允许出现这种传播，从用户友好的角度来看，系统碰到这种修改操作传播应给用户提示信息，以便征得用户的同意。

六、影响动态模式处理的其它因素

前面主要阐述了保持结构一致性，没涉及行为的修改。对象的行为应和对象的结构保持一致，控制对象行为的约束依赖于结构，如果约束和结构不能保持一致，在执行结束时就不能得到正确的结果。例如，某一对象类的某一属性已被删除，若和该属性相关的约束和操作函数依旧没有进行适当修改，当激活该结束或操作函数时，则不会产生正确的结果。

EO_2SD 数据模型中控制对象行为的手段有约束规则，广义操作函数与操作函数。当对象类结构改变以后，系统自动激活与修改类相关的约束和操作函数，对它们进行分析检查，判定其中是否存在对删除掉的属性或以其它方式变动的属性的调用。如果存

在非法引用，则给出警告信息，对约束规则和操作函数的具体修改是由用户完成的。比较难处理的问题是广义操作函数，它是用一般高级语言编写的，若对它进行检查，势必要增加检查高级语言语法的功能，为简便起见，对广义操作函数系统仅提示而不作任何检查。

任何一动态模式操作应具有原子性，在并发修改模式时应和其它数据库操作一样对数据进行封锁，封锁的方式和管理与其它封锁管理一致。例如对某一类删除一属性，由于需要进行实例对象存贮空间压缩，可对该类加 x 封锁。对由于实例对象标识 OID 的改动而引起的需修改的引用对象所属类也施加 x 封锁，在全部修改完成后才可以释放这些封锁。如果在修改的任何一处出现不允许的情况，则应中断修改，利用日志文件使系统恢复到修改前的状态。因此，对每一动态模式操作，O_2S 系统为其产生一事务。

访问权限问题也和动态模式相关，在 O_2S 中，类与类之间存在着语义关系，由于对某一类的模式进行修改时会导致对另一类的修改。如果用户仅对起源对象类拥有修改权限而对另一对象类没有修改权限，迫使从下面三种办法中选择一种，一是放弃全部修改，二是仅修改源类，三是扩大用户权限。采用第二种方法会导致不一致的信息存贮在数据库中，第三种方法可能导致数据的失密。因此 O_2S 系统采用第一种方案，先拒绝操作，当用户获得足够的权限时方允许进行这类操作。

七、结束语

动态模式反映了设计对象在结构和分解方面的动态性，它来源于设计任务的分解性、设计过程的试错性以及自顶向下的设计方法等设计特征，是工程数据库管理系统应支持的重要功能之一。本文提出的动态模式处理策略已在 O_2S 系统中得到应用。

参 考 文 献

[1] *Peter Wegner, Dimensions of Object-Based Language Design, Proceedings of OOPSLA' 87, 1987*

[2] Andrea H. Skarra, Stanley B. Zdonik, Type Evolution in an Object-Oriented Database, In Research Directions in Object-Oriented Programming, B. Shriver & P. Wegner (editors), MIT Press, 1987

[3] Jay Banerjee, et al., Data Model Issues for Object-Oriented Applications, ACM Trans. on Office Information Systems, January, 1987

[4] D. Jason Penney, Jacob Stein, Class Modification in the Gemstone Object-Oriented DBMS, Proceedings of OOPSLA' 87, 1987

（上接第 8 页）

义的候选理论，它必须接受更深的数学和更深的实验检验。

我相信，在这个普遍的科学方法上，即使不是在其实验标准上，计算机科学与物理学差别不大。同许多计算机科学家一样，我希望有一种关于人为和自然现象的广义信息科学能与现有的丰富的物理科学相匹配。如果我所描述的基本思想能在该方向上前进了即使一小步，我将感到很欣慰。

鸣谢

我认为这个荣誉在很大程度上应归功于为此做出努力的许多人，我感谢他们，尤其是与我一直并肩战斗的同事 Rod Burstall 和 Gordon Plotkin，从他们那里我学到了许多东西，得到了大力支持。David Park 在一个要点上对我影响很大并一直鼓励我，Tony Hoare，Carl-Adam Petri 和 Dana Scott 的工作给我以启迪。（参考文献 38 篇略）

[李舟军　刘海燕　译自 CACM Jan, 1993, pp78—89，陈火旺　校]