数据库 规则管理 OODB 一致性 (8)

39-44

计算机科学Vol.20№.1 1993

OODB中的规则管理：一种一致性方法

Oscar Diaz, Norman Paton 和 Peter Gray

TP311.13

Diaz,O Norm.,OD 冯铃

摘要

数据库管理系统中已提出用规则支持主动行为，但规则与其他对象是分别对待的。本文提出一种一致性方法，规则可以像其他对象一样加以描述和处理，无需任何附加机制，并且规则可以与其他对象相联系，按层次组织；利用视之为对象的规则和类这两者间的关系，可以为规则提供基于类的索引，减少对可用规则的搜索。该方法已在ADAM（Prolog-OODB）中实现。

一、引言

主动数据库系统能够自动响应系统内、外所产生的事件，这种响应使用了ECA规则"事件-条件-动作"描述，它包含有一个触发规则的事件，描述给定状态的条件以及当条件满足时执行的动作，因此，系统既知道干什么，又知道何时干。

〔BBBC90〕中，规则被认为是未来数据库中的一个主要特点，而OODB设计者往往忽略了规则重要性。我们的研究旨在深入到面向对象环境下的规则内部，重点探讨提供一种一致性方法。

所谓一致性方法，是指规则像系统中其它对象一样，用属性和方法加以描述，通过方法来表达并实现规则操作，使OO视图中的所有优点（如：封装性、模块性、重用性等）带到规则管理中，并且所有为对象引入的功能（如事务机制、锁机制等）都能自动地运用到规则上。

二、相关的工作

RDBMS通过主动行为来增强完整性约束，定义视图，传送修改要求和计算导出属性，在〔BBBC90〕中，规则可看成是一个统一的视图，可提供更广范围的DB设施，但RDBMS中的规则均是用一个特殊层实现的，需用额外的机制和结构来支持规则管理。

〔Kotz88, Dayal89, Hudso89, Chakrav89, Bauz90〕中描述了支持规则的几种OO系统，〔Bauz90〕列举了支持规则的如下几种机制：

1．基于方法的机制：将预编译过的规则插入到代码中任何一个可能激活该条规则的地方，或者将激活规则的命令插入到任何将用到的地方；

2．基于对象的机制：增强对对象的描述，当有消息发出时，可以指示出它将涉及到的规则，本文将采用该方法；

3．外部机制：定义额外的结构，使得当有事件发生时，能够支持检测功能。

上述第一种方法有如下的缺陷：

1．规则限制在方法内部，很难查询规则的属性，如：条件、行为或是否被激活动作；

2．对规则的任何一个属性的修改意味着所有支持该规则的方法都要发生变化；

3．既然规则可相互作用，那么给方法中的规则编码就需程序员对方法中所有规则有一个全面了解；

4．OO思想鼓励将一个给定对象的所有信息集中存放，而规则定义分散在不同的地方，在这一点上与之相违背；

5．方法代码中包含两部分：操作自身

如何实现以及如何增加规则，这严重破坏了方法的加载。方法加载性是OO系统中十分有用的机制，它可以根据特殊要求，实现特殊的操作，此时问题不仅在于加载操作，而且还要加载那些用规则描述的嵌入概念（如：完整性约束）。

〔BBBC90〕指出："规则只可通过DBMS来加强，不受任何函数（方法）或集合的约束"。后两种方法克服了上述缺点，是通过DBMS提供的机制实现的。

〔Bauz90〕中为O_2提供了规则管理机制。规则就是将事件作为属性的对象，并定义额外结构存储规则表。当特定事件发生时，检测这些规则，但事件自身不能看成对象，因而具有特殊需求的事件。复合事件很难引入，影响了系统的扩展性。另外还必需使用额外提供的规则继承机制，而不能用基于对象层次的机制。

〔Dayal88〕中将规则和事件看成是具有自身属性和方法的两类不同实体，采用一种合理方法，对规则实行支持，并且注重事务管理和优化技术，但是，有些OO视图的特点还未被考虑，如类的主要作用，方法作为类定义的一部分等等。

三、规则管理概述

在OODB环境下对规则管理者的最基本要求就是要能识别重要的实体及相互间的作用关系。简单地说，规则管理者的作用就是用规则，对系统发生的事件做出快速反应。这一过程可分成三部分。

1．规则：描述何时以及如何对事件做出反应；

2．事件：当需要系统做出某种反应时，事件就是该信号的一个指示器，并非所有系统将事件当作对象对待，例如，事件可以作为简单属性值，但破坏了系统的扩展性。因为系统要能处理来自不同源的事件或需特殊对待的复合事件；

3．事件生成程序：事件可由DBMS自身产生，或由其它外部系统，如时钟或应用程序产生。

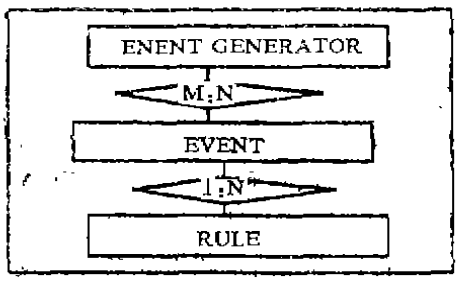

图1 规则管理的E-R图

图1是一实体-关系图，其实体间的联系可描述为：

1．事件生成程序产生一个事件，并通过消息signal，将事件发给事件管理者；

2．事件管理者检查是否有规则可被发出的信号激活，若有，则它将消息fire发给适当的规则；

3．规则接收到消息fire后，检查规则条件。若满足，则执行规则的动作。

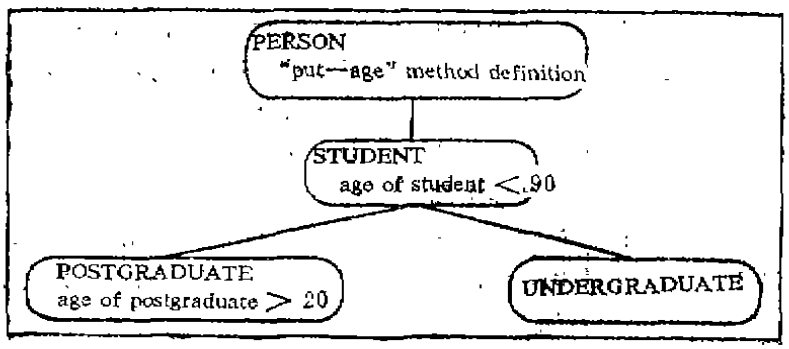

图2 person层次

四、面向对象环境下的事件

RDBMS中，事件可用操作和操作时间（before，after）描述，例如：（insert，before）表示事件在insert操作开始之前就产生了。在OO系统中，操作（即方法）不是孤立的，是类定义的一部分。相同的方法名可在不同类上用不同的方法实现，加载或者方法的处理可通过层次中的低层类实现。图2定义了一条完整性规则："学生的年龄不能超过90岁"。

对于类STUDENT，在消息put-age开始执行之前，如当student的age修改前，该条规则将被触发。既然OO系统允许方法的继承性，方法put-age可以在类PERSON上定义，子类STUDENT继承此方法，因此，不仅插入一个person的age，而且当此person是student时，均涉及该条规则。

由于方法是根据类获得其意义的，故方法自身不能指定其引用范围，此类又称为主动类，已有几种支持主动类的方案：

1、主动类可以嵌入到一条规则的条件部分。如将（put-age，before）作为一个事件，规则的条件部分检查消息的接收者是否是类STUDENT的一个实例，该方法除了使涉及规则的环境难以理解外，还未利用主动类（如索引机制）的优点。

2、事件定义可以扩大到包含主动类属性。如将（student，put-age，before）作为一个事件，但消息接收者可能是某些子类的实例（如postgraduate），因此主动类不是直接的处理类，此时有两种选择方案：一是检查消息接收者是否为主动类的一个实例（即student）。由于此方法对每条发送的消息和每个事件均要进行检查，故代价昂贵；二是自动产生所有可能的继承性事件。例如，若POSTGRADUATE和UNDERGRADUATE均是STUDENT的子类，事件（postgraduate，put-age，before）和（undergraduate，put-age，before）将会产生。值得注意的是，有些产生的事件可能已经被定义过了，如完整性规则限制postgraduate的age＞20，这种情况下不生成一个新的事件，而是用younger-than-ninety规则的标识符扩展该事件激活的规则集合。另一方面当有一个新的子类引入时，必须产生相应的事件。例如，若引入phd-student作为POSTGRADUATE的子类，则phd student继承了postgraduate student的所有事件，该处理过程很麻烦，维护起来代价很大；

3．规则定义扩展到包含主动类属性。既然规则是对象，可用规则和类间的双向关系实现额外的主动类属性。与主动类相对的就是在一个类中定义一个"类-规则"属性，例如：younger-than-ninety规则将student作为主动类的属性值，STUDENT类将该条规则标识符作为"类-规则"属性值，这样做有两个优点：

·用类对规则进行索引。当一条消息发送给某类的任何一个实例时，证实的规则集就是此类的"类-规则"属性值，这样，对将用到的规则的搜索大大减少了；

·规则的继承性转换到类层次上，而无需定义任何附加机制。正如上面讨论的，影响一个给定实例的规则不仅与它的直接类相联系，而且与它的上层类也有联系。例如，若消息put-age发送给类POSTGRADUATE，运用的规则（如表示age属性完整性约束）既与POSTGRADUATE相关，又与STUDENT和PERSON相关，要处理这种情况，则必须扩展类的定义，即增加一个属性actived-by，它包含了"规则-类"类型的对象，一个给定类的actived-by属性值等价于其"类-规则"属性值的规则总和，这些规则与该类相关，由actived-by属性获得的规则与上层类相关。

五、ADAM中一致的规则管理

5.1 ADAM的概要

ADAM中规定元类是一个类，它的所有实例均是类，元类不仅允许使用数据模型的设施，存取和访问类，而且允许使用特化

和继承，为类的产生规定缺省的行为，因此一致性和扩展性大大提高。本文旨在为ADAM扩展一个规则管理者，ADAM的元类在[Paton90]中有详细的讨论。

ADAM中对象可以是元类、类或实例。当系统被编译时，称为meta-class的元类已经存在了，所有后续类通过发送消息给元类。如meta-class得以产生，这些消息就定义了new，put-slot和put-method等等方法。

无论是元类、类还是实例，都是发送消息new给类产生新对象的，而新对象是此类的一个实例，例如，产生一个新类person，它是元类entity-metaclass的一个实例，其调用如图3所示。

```
new([person,[
    attribute(att_tuple(cname,global,single,total,string,[]),
    attribute(att_tuple(sex,global,single,total,string,[]),
    attribute(att_tuple(born_in,global,single,optional,string,[])
])) => entity_metaclass.
```

图3 ADAM中类person的定义

"new"是一个表序列，第一个元素是对象名，第二个元素是对象的属性表。一个属性有一个名字，用五个方面来描述：可见性、基数、状态、类型和约束（本例中是空表）。属性值的检索、删除与更新方法由系统自动产生，属性通常由这些方法处理。

当用"new"产生一个实例，而不是一个类时，传给new表中的第一个元素与系统产生的对象唯一标识符相统一。例如：4@person（4@person是内部标识符，实际上，使用一个变量Baby，用get-by-cname([odile], Baby) ⇒person具体说明，即用name为odile的person对象标识符说明Baby），在变量OLD中产生类person的一个实例，则可发送下列的消息：

new ([OLD, [cname ([odile]), sex ([female]), born-in([usurbil])]])⇒person

5.2 事件对象

在[Bauz90]中，事件作为规则的属性，其自身没有属性和方法。尽管该方法是可行的，但它破坏了系统的扩展性，因为系统要求能够处理来自不同地方或者需要特殊对待的事件。

事件定义涉及到结构描述（属性）和行为描述（方法），一个事件指出了什么时刻规则须执行，该时刻由激活规则（事件的active-method属性）的消息以及消息（事件的when属性）的状态描述。

不像前面的方法，OO环境下我们产生的事件定义更丰富、更复杂，即：

1．事件不仅限于更新操作，它可以是系统定义的任何消息（如：显示、产生一个新类、移动）。

2．扩大描述消息状态的可能值，前面仅考虑了两个值：before和after。在OO环境下，操作由方法具体化，除了before和after外，值的范围可以扩展到考虑那些找不到方法的状态，或者其它一些选项，这些选项反映了由基本Prolog计值策略（如回溯方法）支持的方法行使中的状态。这些宽谱状态企图反映OO系统中方法的主要作用和消息发送过程中状态的多样性。

在我们的方法中，所有有关方法的信息，不管是输入参数，还是输出参数，均由系统来传送，通过系统定义的谓词current-arguments，规则管理者使得这些信息对于规则的条件部分和行为部分是可见的，下一节中有一个例子，作为对象，事件与其它对象相联系（如一个给定事件激发的规则），事件也可按层次安排。图4是一分层结构，事

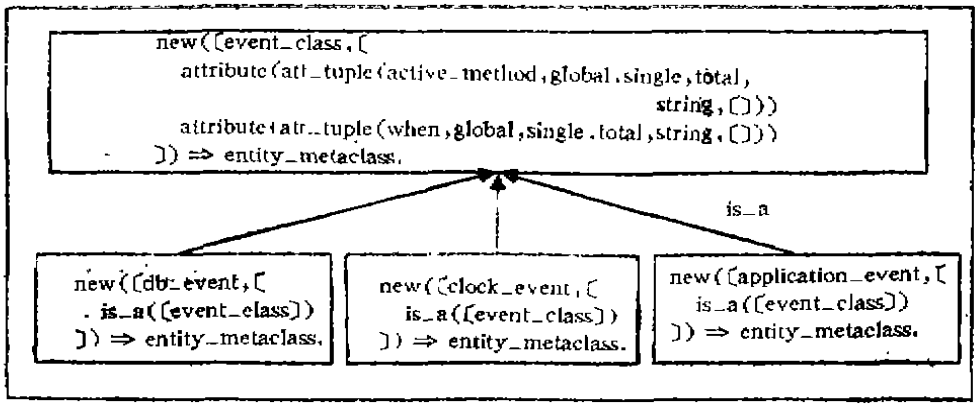

图4 ADAM的事件层次定义

件分为DB事件、clock事件和application事件，由事件源决定。若需要，则可包含新的属性或指定存在的方法。

事件由事件产生程序操作和发出信号，与产生、修改或删除一样，采用同一种风格，例如，发送如下消息可产生一个事件：

```
new([OLD, [
    active-method([put-age])
    when([before])
]])=⇒db-event
```

在方法put-age执行之前，产生该事件。

类db-event,clock-event和application-event有某些共同属性和方法，在event-class的更高层上抽象出来，而且产生一个新事件时发生的过程与产生其它对象一样（如person的实例），方法的删除和修改与删除或修改其它对象也没有区别，因此，系统提供的行为可继承（如entity-metaclass）。

5.3 规则对象

规则的结构主要由触发规则的事件、检测的条件和条件满足时执行的动作三部分描述，条件是查询集合，它检测适于执行动作的数据库状态，动作是一组具有不同功能的操作集，如：增强完整性约束，用户调整、方法传递等等，条件和动作定义涉及到规则应用的当前对象和激活规则的方法的当前状态。

上一节已经讨论了，执行的范围由active-class主动类和事件属性描述，事件属性值是事件实例的对象标识符，例如：younger-than-ninety规则定义如下：

```
new([OID,[
 event([3@db_event]),
 active_class([student]),
 is_it_enabled([true]),
 disabled_for([1@student,23@student]),
 condition([(
   current_arguments([StudentAge]),
   StudentAge > 90
 )]),
 action([(
   current_object(TheStudent),
   current_arguments([StudentAge]),
   get_cname(StudentName) => TheStudent,
   writeln(['The student ',StudentName,
            'with age ',StudentAge,
            'exceeds the expected age']),
   fail
 )])
]) => integrity_rule.
```

若3@db-event是上节中事件的对象标识符，则该条规则在执行put-age方法之前被触发，并检查age是否大于90岁，如果条件不满足，则规则不采用，方法接着执行，否则执行规则的动作。这时显示完信息后，以失败结束，put-age方法不执行，current-object和current-argument谓词仅涉及到规则应用的当前实例和激活规则的方法的当前信息。这里增加了两个属性说明规则自身的

状态，如该规则是否可实现，属性is-it-enabled描述了主动类级别上的状态，disabled-for属性描述了类的实例级上的状态，上例的规则对于所有的student是可实现的(is-it-enabled值为True)，除了对象标识符为1@student和23@student的实例。因此，当is-it-enabled属性为false，或当前对象标识符出现在disabled-for属性中时，此条规则不会被激活。

规则可与系统中其它对象相联系，按层次组织。主动类就是类与规则间关系的反映，它可用来提高系统的速度，与主动类概念相对的"类-规则"属性可用于按类进行索引。按层次组织规则将继承性的所有优点带到规则中来。图5是一个规则层次，有user-defined-rule，integrity-rule和propagation-rule子类，user-defined规则由用户定义，integrity-rule和propagation-rule是系统产生的规则，即系统分别根据完整性约束的描述说明和关系的操作语义产生的规则。

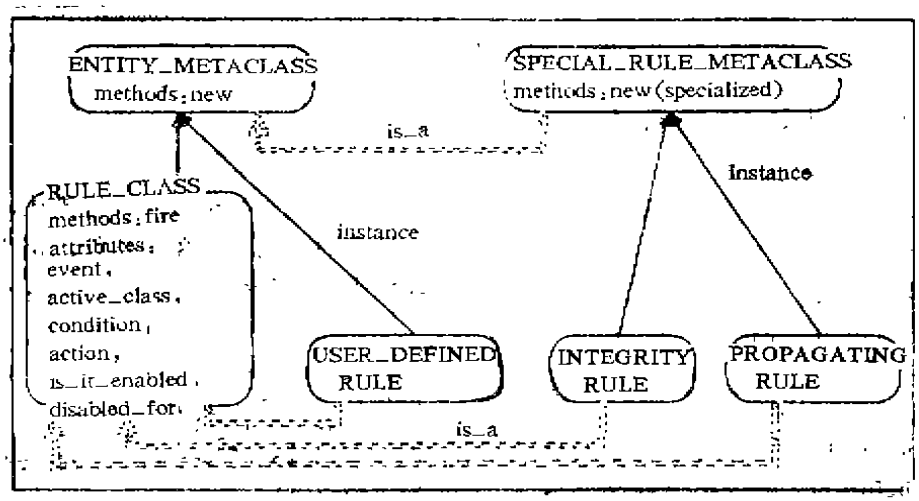

图5 规则层次

规则管理使用的机制就是系统提供的处理其它类对象的机制，但是，当integrity-rule和propagation-rule产生时，有特殊的需求，因此，方法new必须对这些类进行特化，由于ADAM中的元类机制，我们可以定义special-rule-metaclass，能比较容易而清楚地支持这一特化。所有的规则类用同样的方法处理，但是当有消息new发送给integrity-rule或propagation-rule类时，要用到new的特殊定义。（参考文献略）

〔冯铃摘译自"Proceedings of the 17th Intl.Conf.on VLDB"，冯玉才校〕