机科学家自发组织起来，准备对二十年来代数语义方面的研究成果进行总结，集各家之长，设计出一个统一的基于代数方法的程序规约语言，以便为欧洲甚至国际计算机软件工业界提供一个用户乐于使用的程序规约描述语言和一套能保证正确性的开发方法。这种语言定名为COFY。在访问中，我也了解到，在并发程序设计和并行计算软件方面，在实时系统描述技术以及函数式程序设计语言方面，欧洲都正在酝酿或已推行了统一化标准和实现技术。这是一个值得注意的动向。

三、程序设计方法的实用化

按照欧洲的传统，特别是在大学中开展的计算机科学和技术研究，通常都不是非得与生产相结合不可，因此就更谈不上校办企业问题了。80年代初，欧共体实行ESPRIT计划，一是强调欧共体国家之间的合作，二是强调科研与生产相结合。该计划规定每个ESPRIT项目至少有三个欧共体国家参加，至少三分之一课题组来自欧洲工业界。这个计划推行十几年来，的确取得了成效。与我接触的计算机科学家中，有三分之一已签定了与计算机公司和其他高技术公司的合作开发或产品研究合同，他们的任务大多是为欧洲传统工业强项开发应用软件。如：与欧洲航天航空界合作开发火箭、卫星、飞机的控制模拟软件，与汽车公司开发汽车导航系统和自动设计模拟软件，与石油公司开发应用软件，以及为各种精密机械和武器系统的计算机开发控制软件等。我的初步印象是：除理论计算机科学家外，多数人都对承担高技术公司的合作项目抱有热情，并希望他们的理论和方法能与实际结合，并产生经济效益。有的朋友甚至找到我，讨论承接中国项目的可能性。这与我十七年前在欧洲学习时经历相比，实在是一个巨大的进步。

以上是我这次短期访问的一些粗浅印象，写出来供国内同行参考。

致谢 这次访问是在国家自然科学基金委和国家科委基础高技术司的支持下进行的，在此对他们深表谢意。

（上接第50页）

参考文献

[1]Dunren Che and Zhongfan Mai, The Overall Design of MIDS/BUAA: A Multimedia Intelligent Database System, In proceedings of ICYSC'93 (Third intl. Conf. for Young Computer Scientists), July, 1993, Beijing

[2]车敦仁，面向对象的智能数据库：研究、设计与实现，博士学位论文，北京航空航天大学，1994

[3]Dunren Che and Lizhu Zhou, The IntDBS Next Generation Database System. In Proc. of ICII'96, April, 1996, Beijing

[4]Kamran Parsaye et al., Intelligent Database. John Wiley & Sons, Inc. 1989

[5]Olivia R., Liu Sheng, et al., Object-Oriented Modeling and Design of Coupled Knowledge-base/Database Systems. In Proc. of IEEE Data Engineering, 1992

[6]D. McCarthy, U. Dayal, et al., The Architecture of an Active Data Base Management System. In Proc. 1989 ACM SIGMOD Conf. June 1989

[7]E. Anwar, L. Maugis, et al., A New Perspective on Rule Support for Object-Oriented Databases, In Proc. 1993 ACM SIGMOD Conf. 1993

[8]N. H. Ghana and H. V. Jagadish, Ode as an Active Database: Constraints and Triggers. In Proc. of VLDB, Barcelona, Sep. 1991

[9]O. Diaz, N. Paton, et al., Rule Management in Object-Oriented Databases: A Unified Approach, Same to[8]

[10]Dunren Che, The Implementation Techniques of an Compiler-based PROLOG Database, Master's thesis (in Chinese), Changsha Institute of Technology, 1988

[11]Stefano Ceri, A Declarative Approach to Active Databases, Same to[5]

[12]Michael Stonebraker, Greg Kemnitz, The Postgres Next Generation Database Management System, CACM, 34(10), 1991

[13]Won Kim, et al., Architecture of the ORION Next-Generation Database System. IEEE Trans. on Knowledge and Data Egineering, 2(1), 1990

[14]S. Chakravarthy, et al., ECA Rule Integration into an OODBMS: Architecture and Implementation. In Proc. of IEEE Data Engineering, 1995

智能数据库系统中对主动性规则的支持*)

车敦仁 周立柱
(清华大学计算机系 北京100084)

摘 要 目前,尽管"智能数据库"这一术语还未得到很好的定义,许多实现问题仍未解决,但"智能数据库"这一概念无疑是很诱人的。本文拟探讨智能数据库原型系统 IntDBS 的(主动)规则机制,特别介绍有关规则的关联机制(association mechanism)及相关实现技术。我们为 IntDBS 提出的规则关联策略成功地克服了事件监测器(event dectector)通常难于避免的时间上的低效性。

关键词 规则支持,主动数据库,OODB,智能数据库。

1. 引言

或许仍有不少人认为"智能"一词是计算机领域的一个时髦而缺少正式定义的术语,那么"智能数据库(IDB)"就更是如此。尽管这样,IDB 概念背后所隐含的思想[4]却很鼓舞人!在本文,我们不拟就"智能数据库"概念进行可能毫无结论性的讨论。我们只想阐明我们自己关于 IDB 的观点:一个数据库要想成为智能数据库,至少应同时具备演绎能力和主动能力,即把演绎数据库和主动数据库的基本特征集成在一个系统之中,特别是基于 OO 模型来集成一这就是我们在 IntDBS 研究项目中所采用的观点。

最初引入到数据库系统的规则是简单的(演绎)推理规则,这些规则可用于定义内涵数据库,其中的数据是从显式存储的数据(即外延数据库)中计算出来的。这是研制演绎数据库的初衷。而后,经过对规则扩展新的语义,如产生式规则,进一步发展导致了所谓的主动数据库概念的产生。产生式规则能够使(外延)数据库在一定的条件下自动地改变,从而表现出所谓的主动性。

我们基于前一个原型系统 MIDS/BUAA[1][2],目前正在开发一个新的研究原型——IntDBS[3],作为对 MIDS/BUAA 的设计进行改进和进一步探讨在 OO 环境下主动和演绎能力的合理集成技术的一个框架。受篇幅所限,下面集中讨论 IntDBS 对主动规则(类产生式规则)的支持。

相关工作

尽管已有不少的研究课题在尝试把 AI 技术,尤其是主动和演绎推理机制引入到数据库环境中,但直接与我们相关的并不多。就笔者所知,若从文献[4]那样高的观点看,在目前的产品及原型系统中,极少有能称得上是智能数据库系统。IDB 技术并非完全是从零开始的,象 Postgres、Ode、ADAM、Sentinel、HiPAC 等系统,尽管照文[4]的观点来说尚不是充分的 IDB,但它们都为 IDB 技术作出了贡献,与 IntDBS 的设计与实现技术关系较大的系统有:HiPAC、Ode、ADAM 和 Sentinel。

HiPAC 是早期的一个主动数据库系统。它的主要贡献是其 ECA(event-condition-action)规则模型被很多后来的主动的 OODB 所效仿。在 HiPAC 的 ECA 模型中,主动规则是第一类对象,并且支持两个特殊的属性:E_C_coupling 和 C_A_coupling,二者皆有三种可能的取值:Immediate(立即)、deferred(延迟)和 separate(分离)。

Ode 是通过约束和触发器形式来实现其主动行为的。约束和触发器均由条件和动作两部分组成,并且是在类级定义的。Ode 同时引进了"软"、"硬"约束的概念。Ode 对条件的计值总是作为当前事务的一部分在当前事务中完成;E_C_coupling 永不取"分离"值,C_A_coupling 永不取"延迟"值。

ADAM 和 Sentinel 是另外两个有影响的主动式 OODB,二者把规则和事件都当作第一类对象来对待。为支持主动性,二者均对类的结构扩充了专门的属性用以登录与该类的对象关联的所有规则;这一特殊属性在 ADAM 中被命名为 class_rules。在 Sentinel 中命名为 consumers。ADAM 仅支持对象内(intra-object)事件和类级规则;Sentinel 既支持对象内事件和类级规则,又支持对象间事件和实例级

*)本文得到国家自然科学基金项目资助。

规则。在 Sentinel 中，实例级规则与对象的关联是通过预约机制完成的。Sentinel 的预约机制是把相关的规则直接记录到每个主动对象的 consumers 属性上。

在以上四个相关系统中，Sentinel 与我们的 IntDBS 相似之处更多一些，特别是其规则预约思想。但 IntDBS 的规则关联机制用了不同的关联策略：直接与每个主动对象关联的是规则的事件属性（对象）而不是规则对象本身；而事件（对象）是可为多个规则（对象）所共享的。此外，IntDBS 还支持系统级规则，Sentinel 则不支持。

目前，还没有一个系统尝试过在 OODB 环境下同时集成主动和演绎两种机制，IntDBS 的通用对象模型—univObject[2]也是其较有特色的一个方面。

IntDBS 的主要贡献

同相关的研究项目相比，我们的主要贡献包括：

·IntDBS 在一个统一的 OO 环境下，同时支持三类规则，即系统级、类级和实例级规则。

·充分利用 OO 范型的一般机制实现了事件（模式）对象在规则之间的共享。

·与 Sentinel 的预约机制不同，我们的关联机制向主动对象直接预约的是事件，而不是规则本身。由于一个事件能够象一个连接枢纽一样，把一组主动对象同一集主动对象联系起来，这样一个事件对象就能被一集规则对象所共享，即只需在主动对象的 evnt_objs 属性（见图3）中维护一个公共的事件模式，因而我们的关联机制有较好的空间利用率。

2. 背景

本节简介 IntDBS 及其对象模型 univObject。

2.1 IntDBS 引论

IntDBS（即 MIDS/BUAA[1,2]的改进版）是为了研究最新的数据库技术而开发的一个研究原型，我们的主要目的是探讨如何在统一的 OO 框架下把主动和演绎功能集成在一个数据库系统中。该原型含有八个主要的功能模块：1）一个高层信息工程开发环境（HLTools）；2）一个命令处理及语言解释器模块；3）一个查询（优化）处理器（QI）；4）一个模式管理器（SM）；5）一个对象管理子系统（OMS）；OMS 是整个系统的核心部分，它又进一步包括五个子模块：消息接收器（MH）、对象管理器（OM）、规则管理器（RM）、事务管理子系统（TMS）、和通信管理器（CM）；6）一个存储管理子系统（SMS）；7）一个多媒体信息管理模块（MIM）；8）一个推理机模块（IE）。其中 IE 是一个概念性模块，它是各个推理单元的一个逻辑组合。

2.2 univObject 对象模型概述

IntDBS 系统的 univObject 对象模型是在 C++ 对象模型的基础上扩充而来的。扩充的内容主要有：类似于 Minsky 的框架模型中的（多）侧面、产生式系统中的规则、和仿真系统中的事件等。univObject 模型中对象的一般结构（扩充的 C++ 类结构）可示意于图1。

```
Class class_name[WithVers],[ParentList]
{/* 选项 WithVers 指明是否要为该类的实例提供版本支持 */
  Public:|Protected:|[Private:]
  /* 以下为属性定义部分 */
  type_or_class AttrName;
  /* C++风格的属性的定义 */
  type_or_class Attr Name
  /* univObject 扩充的、可带侧面的属性的定义 */
     with
      ([:AttrCons AttrCons_Spec.:]
       /* constraint on attribute */
       [:Inverse AttrName:]
       /* indicates reverse reference attr. */
       [:Exclusive Boolean:]
       /* defaulted as FALSE */
       [:Dependent Boolean:]
       /* defaulted as FALSE */
      )
        ...
  IndKeys:{AttrList};
  /* declaration of indexing keys */
  Public:|Protected:|[Private:]
  /* C++风格的方法定义: */
  Meth_Sig;/* 普通 C++方法的声明 */
  Public:/* 能够产生事件的方法的声明: */
  [Event before|after|befote&after]Meth_Sig;
  /* event interface declaration */
      ...
  Soft|[hard]Constraint:
            /* 约束声明部，缺省为硬约束 */
       [cons_name:]{Constraint_Expression returning
        Boolean value}
      ...
}
```

图1 univObject 模型的对象结构

univObject 对象模型对 C++ 对象具体扩充了如下内容：用选项 WithVers 可以声明是否要支持对象版本；允许每个属性（最多）有四个侧面（由 AttrCons 引出对该属性的约束、由 Inverse 引出该属性的逆引用属性、由 Exclusive 和 Dependent 分别指明引用的类型是否是"排它的"和"依赖的"，具有与 ORION 类似的语义）；由键字 IndKeys 引出用于对类的全体实例进行索引的属性，方法名前的"Event before|after|before&after"用以声明该方法的执行是否可能产生事件；选项 hard 和 soft 引出对象的约束（与 Ode 做法某些相似）。软约束只要求在事务提交是验证可满足性，而硬约束则要求即时进行验证（即采用类似于典型的 ECA 规则中 E-C 或 C-A 耦

合属性取值为"immediate"时的语义)。在univObject对象中,仅与单个属性发生关系的约束可以直接定义于该属性的约束侧面上(这时称为属性约束),更一般的对象约束(包括"硬"约束和"软"约束,通常是涉及到多个属性甚至另外的对象的约束)通过hard和soft选项引出。

3. IntDBS中的主动规则

近年来,有两种形式的规则:主动规则和演绎规则,在数据库领域获得了广泛研究。IntDBS原型系统能够成功地支持这两种规则,本文主要探讨IntDBS对主动规则的支持。

在ECA模型中,如ADAM[9],HiPAC[6],Sentinel[7][14]等系统,主动规则被当作第一类对象。这种做法有两个好处:(1)系统能够一致地对待规则对象和其它对象;(2)能够相对容易地对知识库(规则库)进行一体化的管理和维护,因为OO范型的典型机制,如继承、动态创建、修改和删除对象等能够一样地适用于规则对象[3,4]。

3.1 规则对象的设计考虑

从概念上讲,每个ECA规则由三个主要部分组成:事件、条件和动作。当一个事件的发生被探知以后,便对条件计值,如果条件为真,则进一步调度动作的执行。在主动数据库中,规则作为触发器首先要受到一定事件的触发,然后才能在条件满足时接受点火。在IntDBS中,只有主动对象,即被显式声明为active类(见图3)的实例,才能产生事件。在IntDBS中我们把ECA规则称为主动规则是因为它是类产生式规则,同时也为了和演绎规则有所区别,但ECA规则本身未必都是主动对象(即能引发事件)。

1)主动对象 除了能够象普通C++对象那样接受操作外,它们还能够产生事件,从而去触发其它对象(相应地称为被触发对象)。主动对象是潜在的事件发生器,被触发的对象则是事件的消费者(这里主要指ECA规则,也包括事件对象)。在IntDBS中,事件对象不仅是联系主动对象和相关的ECA规则的纽带,同时还要把主动对象中产生的事件信号传播到相关的事件消费者。

在OO环境下,对方法的激活就是事件(的发生)。事件需要被监测并传播到相关的对象。为此,主动数据库一般都采用事件监测器完成此任务。但事件监测器通常是非常费时的,特别是当把任何一个对象的每一次方法激活都当作潜在的原始事件时。一种比较可行的变通办法是对C++对象类的原有定义扩充事件接口声明,如Sentinel系统[7],并显式地声明主动对象的类为active类(或其它主动类)的导出类。这意味着主动类的定义现在是传统的C++类定义再加上事件接口声明。事件接口声明应该指明该主动类中定义的哪些方法是潜在的事件发生器。在事件接口中声明的每个方法将缺省地产生两类事件,"方法执行前"事件和"方法执行后"事件。

2)事件 正象文[7]所指出的那样,在OO环境下目前有三种方法用于事件的描述:1)在类的定义中声明事件表达式(如Ode[8]);(2)把事件当作规则的属性(如Baus[7];3)事件作为第一类对象(如ADAM[9])。

在IntDBS中我们采用的是第三种方法,因为使用第三种方法有这样一些好处:1)事件能够象其它对象一样被一致地对待;2)象普通对象一样,事件的性质也能够容易地模拟为属性;3)能够方便地引进新的事件类型;4)能够表达跨越多个不同的类(的对象)的事件(即复合事件)。

IntDBS中的事件分为两种类型:原始事件和复合事件。原始事件由对象的方法激活所引发,复合事件是由多个原始事件复合而成的(它们甚至可源自不同的主动类)。

为了更好地理解IntDBS中的事件处理,有必要详细阐述下列与事件概念有关的一组术语:

·事件接口—指主动对象的一个方法子集,在对象的类定义中它们已被声明能够产生事件。

·事件发(产)生器—指一个主动对象(能够引发事件),有时也指列举在某主动类的事件接口中的一个方法。

·事件(的)发生—指列举在主动类的方法接口中一个方法的执行"之前"或"之后"。

·事件信号—标明某主动对象中有一个事件产生的一条消息。

·事件属性—ECA规则对象的三个主要属性之一,即Event属性(见图5)。

·事件消费者—通过它们的事件属性(对象)与主动对象关联的ECA规则,或复合事件。

·事件对象—除了作为ECA规则对象的一个重要成分被Event属性所引用,事件对象作为一个一般的对象,还起着连接事件发生器和事件消费者(ECA规则及复合事件)的作用。

·事件模式—事件对象是可以被共享的,单个事件对象可同时被多个ECA规则共享,即同时被多个ECA规则的Event属性引用,这些ECA规则于是

被说成具有同一个事件模式。

3)ECA 规则　ECA 是目前主动数据库广泛采用的一种规则模型，它含有三个主要的属性成分[6]。

·Event：数据库本身的状态转移或由应用定义的外部事件，二者都是通过对象的方法激活而产生的。

·Condition：一般情况下是一组数据库查询，当一个查询之结果为空(集)时，则逻辑上认为它是不可满足的(即"假")，否则认为是可满足的(即"真")。在 IntDBS 中，ECA 规则的条件部分当然也可包含由 IntDBS 谓词组成的谓词表达式。

·Action，即一段用户定义的程序。

在 IntDBS 的上述 ECA 规则的三个主要成分中，只有事件部分被建模成第一类对象，并由 ECA 规则的 Event 属性直接引用；ECA 规则的条件和动作部分则被表示为成员函数。这种设计选择的理由是：一个事件的发生通常要触发一集(不止一个)规则，因而完全有理由认为：规则通常需要共享其事件成分。而实现事件模式共享的一种非常有效的办法是把事件当作第一类对象来建模，这时的事件其实就是事件模式对象，这样也比较容易获得对复合事件的支持(事件复合即对象复合)。

当一集规则受到一个事件的触发并且它们的条件都被满足时，IntDBS 采用了类似于 HiPAC[6]的做法，即允许多条规则并发执行；因而 IntDBS 不需要进行专门的规则冲突排解。被触发的多个规则的实际执行次序完全由系统的事务管理器来决定，因为每个规则的动作部分的执行都要根据该规则的耦合要求被嵌入到某一个相应的事务中。

对象通常彼此很不相同，特别当从不同的抽象级去看时，如从整个系统的角度、从(对象的)类的角度、从单个实例的角度，不同的对象通常要求不同的主动规则与之关联。IntDBS 首先提出并尝试同时支持三种不同范畴的规则，即系统级规则、类级规则和实例级规则。

4)主动对象和规则的关联　在 IntDBS 中，系统级规则自动地与全系统范围的每一个主动对象相关联，类级规则与类的全体实例相关联，实例级规则只与具体的实例对象相关联。IntDBS 实现对象与规则关联的机制与 Sentinel[2]采用的预约思想有一定的相似，但在具体的实现途径和效果上则又很不相同。IntDBS 采用的方法对 Sentinel 的方法进行了很大的改进：通过专门为主动类扩展的成员函数(即 associate( )和 disassociate( )等)把相关的 ECA 规则的事件模式登录在每个主动对象的 evnt_objs 属性中，而不是象 Sentinel 那样直接登录 ECA 规则本身。

所有的系统级规则通过主动对象的构造函数在它们创建之时自动地与每个主动对象相关联。所有的系统级规则共享同一个事件名"ANY"。实际上，ANY 是全体系统级规则构成的集合的名字，ANY 代表的是全体系统级规则。

每个主动类的类级规则通过该类的构造函数自动地与该类的每个实例关联；而实例级规则与相关的主动对象的关联则必须通过显式地激活主动对象的 associate 方法来建立。

与 Sentinel 相比，IntDBS 关联机制的一个明显的优点是：事件监测的性能得到了很大的提高，因为事件模式对象作为事件生产者和事件消费者之间的中介直接将二者联系起来，成为事实上的事件监测器和事件处理器，当事件发生时，能即时地把事件信号传播到所有相关的消费者，即所有相关的 ECA 规则及复合事件。复合事件会进一步将之传播到相关的消费者。在这种情况下，需要进行的事件监测处理仅仅是匹配事件信号所携带的方法名和事件模式中所描述的方法名，这是一个非常简短的过程。再若 ECA 规则通过它们的事件成分直接关联到主动对象事件接口中的每个具体的方法，此时事件监测过程实际上不需做任何事情，因而能获得最大的事件监测性能。

IntDBS 关联机制的第二个优点是提高了空间利用率。由于规则能够共享事件，事件则不能共享规则，故可以认为在一般情况下，规则比事件的数目多很多，因而 IntDBS 在每个主动对象中登录的是事件(模式)而非规则，这样便可节省存储空间。

IntDBS 关联机制的另一个好处是逻辑概念清晰。通过事件对象在主动对象和规则之间直接建立起联系，负责对主动对象中发生的事件进行"监测"并把事件信号传播到事件消费者。

4. 实现

开发 IntDBS 原型的目的是探索和实验一些最新和最先进的数据库技术。IntDBS 目前正在利用 C++ 在 Sun 工作站平台上进行开发。规则(即 ECA)、事件和谓词都被模拟成第一类对象。为此，专门为 C++ 的类层次结构增加了相应的系统类：ECA、Event、Predicate、Variable 等。其中类 Variable 的引入主要是为了表达变量型的谓词项(term)并支

持项的一致化。与此同时，还要求对所有其它类增加一个特殊的类型声明方法，即在子根 DBClass（见图2）下增加方法 TypeTag( )，这是为了便于在 OO 环境下支持谓词项的一致化（鉴于篇幅有限，不加详述）。IntDBS 类层次结构的 DBClass 分支[2]示意于图2，为支持规则而扩充的系统类基本都隶属在该分支上。

在下面，我们拟给出实现规则机制若干主要类的细节，最后讨论 IntDBS 规则关联机制的实现。

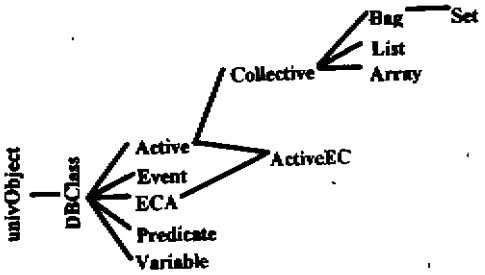

图2 IntDBS 系统类库的 DBClass 分支

4.1 Active 类

主动类是这样的类，它们的实例要求成为主动对象，即当其事件接口中声明过的方法在激活时便能够产生事件。

```
Class Active
  {
   Event * evnt_objs; /* a list of event objects which are
        referenced by associated rules as the values of
        their Event attributes */
  Public:
   Associate(ECA * Obj);
   Disassociate(ECA * Obj);
   Report(int Oid,char * evnt-name,int argc…);
   End_report(char * evnt-name);
  }
```

图3 主动类基类—Active 类

在 IntDBS 中，所有的主动类都必须直接或间接地派生于一个特殊的基类—Active。Active 类是系统预定义的主动类的基类，其主要成分的规格说明示于图3。为了实现主动能力，除了在每个主动类的定义中增加事件接口声明外，主动基类 Active 中预定义的数据成员 evnt_objs，和函数成员 Associate，Disassociate，Report 和 End-report 也将自动被每一个主动类所继承。

·evnt_objs：该数据成员用以保存一集事件模式对象，并由此把每个主动对象和与它们相关的规则联系起来。主动类的构造函数会自动地为 evnt_objs 属性赋上初值"ANY（即系统级规则）加上每个主动类的类级规则"。

·Associate(ECA * obj)：该方法用于把一条 ECA 规则关联到该主动对象，即把一条 ECA 规则的事件对象注册到该主动对象的 evnt_objs 属性中。

·Disassociate(ECA * obj)：该方法切断一条 ECA 规则与该主动对象的关联，它是方法 Associate(ECA * obj)的逆操作方法。

·Report(…)：该方法向相关规则通报本对象内发生的任何事件，通报过程实际上完成的是事件的监测过程：当一个主动对象中有一个在事件接口中声明过的方法被激活时，Report 方法就把相应的事件信号依次发送到在属性 evnt_objs 中注册过的每个事件消费者，发送过程是由激活每个事件对象的 Signal 方法（Signal 方法定义在事件基类 Event 中，详见图4）来完成的；之后，收到事件信号的事件对象用自己的事件模式与事件信号携带的信息进行匹配，如果匹配不成功则事件信号被忽略，否则事件对象便去点火共享该事件对象的所有 ECA 规则，同时把事件信号进一步传播到包含该事件对象的复合事件中。

·End_report(char * evnt-name)：该方法使事件对象复位到"非发生"状态，即把 raised 属性置为"0"；End_report 方法在当前事务结束之前会自动被调用。

4.2 Event 类

IntDBS 系统同时支持原始事件和复合事件，每个 ECA 规则的事件描述部分将被自动地翻译成第一类对象。这些事件对象然后用于建立主动对象和相关规则之间的联系。IntDBS 为事件模式对象提供了一个预定义的事件基类—Event（见图4）。一些更为具体的事件类可直接或间接地从 Event 类派生。原始事件（的发生）是直接由对主动对象的方法激活而引起的，并且具有三种具体的形式：1）方法执行之前；2）方法执行之后；3）方法执行之前和之后。这三种具体形式均需要在事件接口描述中予以显式声明：before，after，before&after。

```
Class Event
{
char * evnt_name;
char * time_tag; /* "bef","aft"or "b&a"for before,af-
                                ter,and before&after */
int oid; /* stores the OID of an active object that gener-
                                ates this event(occurrence) */
int raised; /* indicates whether the event is raised */
int actual_args; /* list of actual parameters of the event
                                generator method */
```

```
Event * comp_evnts;/ * a list of associated composite
                                        events * /
ECA * rules;/ * a list of rules which have this event
                              pattern as event attribute * /
public:
Signal(int oid,char * evnt_name,int arge···);
Propagate(int oid,this);/ * propagates this event to the
          associated composite events by invoking
          their signal method * /
}
```

图4 事件基类 Event

IntDBS 的复合事件是由原始事件的复合构造而成的，具体的复合形式也有三种[7]：析取、合取、和序列。复合操作是通过 Event 类的三个预定义子类来实现的。

4.3 ECA 规则类

一个普通的 ECA 规则含有三个主要的成分：事件、条件和动作。在 IntDBS 中，ECA 规则被建模成系统预定义的 ECA 规则类的实例。ECA 类的规格定义示意于图5。主要的属性有 rule-name（规则名），event（事件模式），condition（条件），action（动作），coup mode（耦合方式），和 enabled（是否活跃）。其中，event 在实现上是指向一个事件模式对象的指针；condition 和 action 分别是指向条件函数和动作函数的两个成员函数指针；coup_mode 则指明条件计值和动作执行之间的时态耦合关系。coup_mode 有四种可能的取值："I"，"D"，"IS"，"DS"，指明对规则动作部分的调度要分别按"即时"、"延迟"、"即时分离"、和"延迟分离"的时态语义要求来进行。这里的"分离"指要求另外创建一个单独的事务来执行规则的动作，IntDBS 目前还不支持"分离"语义要求。

```
Class ECA
{
  char * rule_name;
  Event * event;
  FuncP condition,action;/ * FuncP is the type of
                                  Function Pointer * /
  char * comp_mode;/ * coupling mode of Condition and
                                          Aetion * /
  int enabled;
public:
  virtual int Enable();
  virtual int Disable();
  virtual Change(Event * evnt_id);
  virtual Schedule();
      / * if coup_mode=="I" then Fire() * /
      / * else register the rule into the to_be_fired_
      rules list of the current transaction * /
virtual Fire();/ * if the condition is eva  luated to
                  TRUE then execute the action * /
}
```

图5 ECA 规则基类—ECA 类

在 IntDBS 中，规则动作的"延迟"执行要求是通过当前事务的 to-be-fired_rules 登记表来实现的。IntDBS 的事务管理子系统自动为每个事务构造一个 to-be-fired rules 表，并由 schedule() 方法（即由系统为每个事务对象定义的一个成员函数）把于该事务期间触发的要求延迟调度执行的规则记录在这个表中。每个事务在结束之前将自动调度执行 to_be_fired_rules 表中记录的所有规则的动作部分。

我们赞同文[8]中提出的这样的观点：ECA 规则的"事件"和"条件"间的"D"和"S"耦合与"条件"和"动作"间相应的耦合语义是冗余的，因而我们在 IntDBS 的设计上，默认"事件-条件"耦合采取"即时"语义。

4.4 规则的关联

规则的关联问题涉及到事件和规则、主动对象和规则、对象和事件、事件和（复合）事件等的关联。在 IntDBS 中，规则、事件、复合事件和主动对象都是第一类对象，它们之间的关联关系（象普通对象间的关联一样）可方便地通过统一的对象引用机制来建立。

IntDBS 的所有系统级规则共享相同的规则名 ANY，ANY 规则集与每个主动对象的关联由系统在对象创建时自动建立。

类级规则是类范围上的一个共享规则集，而且要被子类所继承。类级规则和该类的对象实例之间的耦合也是在对象被创建时由系统自动建立的。换言之，每个主动对象在创建之初自动地从其类继承 evnt_objs 属性的初值（即该类的类级规则集，亦含系统级规则）。

实例级规则和实例对象间的关联需要显式地由程序员或用户去建立，即显式地去激活主动对象的 Associate 方法，并以一个实例级规则作为该方法唯一的实参。

总结 在本文，我们首先对 IntDBS 研究原型和它的对象模型 univObject 进行了背景性概述，然后阐述了 IntDBS 的主动规则机制的一般设计（策略）考虑，最后讨论了 IntDBS 在统一的 OO 环境下的主动规则机制的实现问题。下一步工作是：

·实现对规则的"条件-动作"耦合的"分离"（事务）语义支持；·完善和改进对演绎规则支持的实现；·重新对 IntDBS 系统的 P++ 语言进行设计，特别加强对主动和演绎两种规则机制的描述支持；·对 IntDBS 某些重要设计的（策略）选择进行分析评价。

（下转第21页）