计算机科学 1996 Vol.23 No.3

# 面向对象数据模型的研究

田增平 曲云尧 汪 卫 施伯乐
(复旦大学计算机系 上海 200433)

**摘 要** 由于面向对象技术缺乏坚实的理论基础，因而在早期的各种面向对象数据模型中对对象概念的表达和使用都有差异。本文研究了以往提出的多种面向对象数据模型，讨论了对象、类型、继承和方法等基本面向对象概念在这些模型中的表达和使用，说明了面向对象数据模型研究中尚待解决的问题，为进一步研究面向对象数据模型的形式化理论奠定基础。

**关键词** 面向对象，数据模型，形式化理论。

数据模型是数据库系统描述客观世界实体及实体与实体之间关系的基础，从第一个数据库的诞生到今天，对数据模型的研究工作一直没有间断。近年来，面向对象数据模型的研究已成为数据库领域的一个热点。有关标准化组织也在积极从事着这方面的标准化工作。在 93 年，ANSI X3H2 和 ISO DBL 委员会以及 ODMG 分别提出 SQL3[35,36] 和 ODMG93[37] 作为面向对象数据库的建议标准，SQL3 是 SQL-92 的超集，它对 SQL-92 扩充了有关面向对象的概念，而 ODMG-93 则是完全基于面向对象概念的标准，它与 SQL-92 不完全兼容，而且它的面向对象模型的定义与 SQL3 的模型定义也有差别。

由于面向对象技术缺乏坚实的理论基础，因而它与早期的函数数据模型、扩充关系数据模型和逻辑数据模型等相结合后，就产生了具有不同特点的面向对象数据模型，然而，弄清楚面向对象的基本概念在各种模型中的意义是研究面向对象数据模型形式化理论的基础。本文按照对象/函数模型、关系-对象模型、面向对象的逻辑模型和纯面向对象概念的数据模型，对十余种面向对象数据模型进行了研究与比较，分别讨论了面向对象的基本概念，如对象、类型、继承和方法在这些模型中的表达和使用，并讨论了面向对象数据模型研究中尚待解决的问题。

## 1 对象/函数模型

函数风格的面向对象数据模型是基于早期的函数数据模型 Daplex[7] 提出来的。这类模型主要有 PDM[6]，FUGUE[5] 和 IRIS[1,2,4] 等，其主要特点是所采用的概念自然且能力较强。在这些模型中通常只有三个基本概念，即对象、类型和函数。其中函数和类型也视为对象，故而这种模型是高阶的。模型中的任何操作和运算均可用函数施用来表示。类型之间的关系可用多元逻辑函数表示。利用导出函数(derived function)可定义视图等。

1)对象 在对象/函数模型中，对象分为文字对象和代理对象(抽象对象)，任何客观对象均可表示成两者之一，其中没有值的概念。由于对象与其特征之间的联系是通过以其对象为参数的函数表达的，因而具有嵌套结构的复杂对象就表示成沿着对象组成层次连续施用特征函数而得到的对象的集合。在此情况下，维持对象的引用完整性变成了对特征函数的完全性要求(即维持对象的特征函数为全函数)。由于对象标识唯一确定了对象，而且这些模型中没有"值"的概念，因而对象相等是指对象标识的相等。另外，由于函数也是对象，其标识符具有唯一性，因此在允许多类继承时，不会出现各种冲突现象。对象之间的关系可以用二维表(或称为多元逻辑函数)表示。

2)类型 类型是对具有相同特征的对象的抽象。虽然在这些模型中类型都是由一组函数(特征函数)定义的，但是对不同的模型，类型的语义略有差别，在 PDM 和 IRIS 模型中，一个类型的定义不但说明了能作用于该类对象的特征函数，而且类型名称本身可用来指代该类型对象的全体，即该类型的外延。而在 FUGUE 模型中，类型的定义则说明了能施用于该类对象的函数、对象的创建函数及此类型的外延函数，一个类型的外延是此类型(类型本身是对象)的一个特征，类型名称并不指代类型的外延。这些模型中对类型之间的关系的表达形式也有差别。PDM 和 IRIS 将类型间的 is-a 关系作为一种固有的

关系，而 FUGUE 则不然，它提供了表示类型之间各种关系的方法，is-a 关系仅是其中的一种。这样，FUGUE 模型也能表示类型之间的其它关系（如 version-of 等）。此外，FUGUE 模型还可以定义对类型外延操作的函数（在 Gemstone 和 ORION 中称为类方法）如，对某类对象的某种统计运算等。在类型的数据结构定义方面，IRIS 模型允许函数值域具有 tuple、set 和 list 等结构；FUGUE 模型则允许函数值域具有集合类型；*PDM* 模型则提供点集结构为空间数据建模。

*3*）方法　对象/函数模型中没有方法概念。对象的属性和行为均用函数描述。在 IRIS 模型中，根据函数的不同实现方法分为：存储函数（stored），导出函数（derived）和外部函数（foreign）；在 PDM 模型中，将函数分为存储函数和计算函数，后者相当于 IRIS 的导出函数和外部函数。FUGUE 模型将函数的说明（基调）与实现分离，函数的实现可用模型外部的某种程序设计语言如 C 或 Ada 等来完成。IRIS 和 PDM 模型还区分了有副作用的函数（函数调用会改变数据库状态）和无副作用函数（函数调用不会改变数据库状态）。此外，这些模型中还有高阶函数的概念，如 PDM 的 apply 算子、IRIS 中的 set_add 等、FUGUE 模型的函数类定义中的函数等，这些高阶函数主要用来实现一般函数的复合和更新操作。

4）继承　子类对父类的继承主要体现在两个方面：（1）父类定义的特征函数全部适用于子类的对象（特征继承）；（2）可以用父类对象作为参数的函数都可以用子类对象作为参数（可替换特性）。此外，这些模型都允许多类继承，而且允许一个对象同时属于多个类型或动态地改变类型。

## 2　关系/对象模型

面向对象数据库研究的一种途径是在关系系统的基础上，增加必要的语义表达设施及抽象机制以满足复杂应用的要求。在系统开发方面采用进化的方法；在模型研究方面形成所谓的关系/对象模型，这类模型有：POSTGRES[11,12-13,14] 模型、STARBURST[9] 模型、*UniSQL*[15] 模型及 *Scholl* 等人的关系/对象模型（ROM）[16]等。这些模型保留了关系模型的集合操作、视图定义及代数优化等优点，增加了面向对象的概念，扩充了表达复杂关系语义和数据抽象的机制，如：类型，类层次，规则等，使其不但具有支持复杂应用所需的语义表达能力，而且具有较强的数据操作能力和系统效率。这些模型是对关系模型的继承和扩充，受到了许多公司及数据库专家的重视。有人认为基于这类模型的面向对象数据库系统比较有效实用，很有可能成为未来面向对象数据库的主流。

在关系/对象模型中，关系模型的概念与面向对象模型的概念可建立起对应关系；如具有唯一标识的元组可与对象对应，关系对应于类，元组或关系定义中的有关约束和规则可对应于方法。有些关系/对象模型（如 ROM）是以嵌套关系或复杂对象模型为基础的扩充，有些则是以"平"关系为基础的（如 *STARBURST*）。

1）对象　关系中的元组通常对应着对象，然而由于这些模型依赖的基础不同，因而不同模型中的对象概念略有差别。ROM 模型以复杂对象模型为基础，因而它的对象概念亦包含了复杂对象。而在 STARBURST 模型中则将复杂对象分为非结构化复杂对象和结构化复杂对象。前者为通常的"长域"数据，后者为一种扩充范式（XNF，实质为平关系上的视图机制）的实例。在 POSTGRES、UniSQL 及 *ROM* 模型中，*复杂对象即为复杂类型的实例*。ROM 模型则具有函数模型的特点，其对象的所有特征均由函数来表示。

2）类型　类型的概念在这些模型中也稍有差别，有些模型还区分类（class）和型（type）的概念。UniSQL 模型没有型的概念，它使用 TABLE 来定义具有相同特征的对象的结构，相当于通常的类概念。而且，TABLE 的名称也用来指代其实例的集合，即类的定义既是内涵的也是外延的。ROM 模型则区分类和型的概念，型只是描述了具有某些特征的对象的结构，由一组函数组成，类则是具有某种型的对象的集合（未必为该型的外延）。一个类中的所有对象必须具有相同的型，具有某种型的对象可以组成多个类。STARBURST 模型的型概念与 UniSQL 的 TABLE 概念相似。POSTGRES 模型也区分类和型的概念，它的类由基本类，导出类和版本类组成；型则由基本型、基本型数组和复合型组成。它的型概念与程序设计语言的类型概念相似，基本上"是"型。在类的定义中，要说明其对象所具有的属性及其相应的型。另外，POSTGRES 模型还可以支持 SET 型，*具有此型的属性的值可以是任何类的一个或多个实例*。

3）方法　在关系/对象模型中，通常以过程、函数或规则指代方法，这是对传统关系模型的扩充。UniSQL 系统允许每个 TABLE 定义一组过程，这些

过程描述了 TABLE 的每个实例的行为。POSTGRES 模型则允许关系的某些列取过程为值(该模型中过程为一种数据类型)。另外,它还允许用户的定义中使用 C 语言函数及 POSTQUEL 函数等。然而,这些函数的定义不与特定的类相联系。POSTGRES 模型还提供了较强的规则定义机制,通过规则对某些操作进行响应,使得它具有主动数据模型的某些特征。此外,通过规则还可以进行视图管理、完整性检查并提供触发子和版本控制的有关功能等,STARBURST 模型也提供了函数和规则处理机制,但是其函数定义及加入系统的方式不如 POSTGRES 灵活方便。它的规则系统的功能与 POSTGRES 模型的规则系统的功能相似。

4)继承 在这些模型中,相应的类或型组成层次结构并允许多类继承。遇到继承冲突时,要求用户干涉解决(在 POSTGRES 中会拒绝创建产生继承冲突的子类)。

## 3 面向对象的逻辑模型

众所周知,一阶逻辑为关系数据库提供了一个完美的理论框架。利用一阶逻辑既可以定义关系数据库,也可以表达有关的约束和数据操作。长期以来,人们也试图为面向对象的数据库寻找这样的理论框架。然而有些学者认为[26]:逻辑途径本身是面向"值"的,与面向对象的某些特征,如对象标识符是不相容的。另一些学者则认为[15,14,17,18,19]:数据语言可以分成互相独立的两个方面,即编程风范和表示风范。面向对象方法是一种复杂数据结构的表示风范,而逻辑方法则是一种编程风范,两者在本质上并不矛盾。为了说明这点,他们对数据语言的编程风范和表示风范进行了深入的研究,并给出了图 1 的分类:

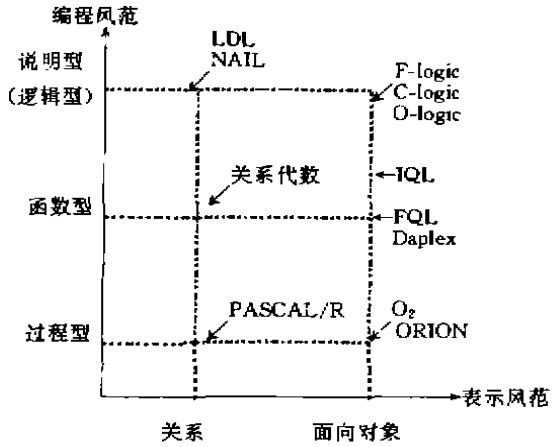


图 1

数据语言的编程风范与程序语言的相似,可分为过程型,函数型和说明型(逻辑型);表示风范分为关系和面向对象两种类型。这样,由编程风范和表示风范组成一个平面,各种数据语言都可以表示为该平面的点。面向对象的表示风范与逻辑型的编程风范相结合就产生了 F-logic[18,19]、C-logic[15]和 O-logic[16,17]等作为面向对象理论基础的逻辑系统。此外,借鉴程序自动生成理论的研究成果,利用代数与逻辑方法相结合研究面向对象数据库理论基础的方法也受到了重视[21,22,23],这方面的工作有 Maudelog 语言[20]。

F-logic 是基于 C-logic 和 O-logic 提出的框架和面向对象程序语言逻辑。按文[23]的观点,它是一种具有一阶语义、高阶语法的逻辑系统。它能表示对象标识、继承、方法及类层次等面向对象概念,而模式推理能力使其具有高阶特性。它的表达能力是 C-logic 和 O-logic 能力的超集。文[20]提出的基于重写逻辑的面向对象数据语言 Maudelog 语言虽然不是高阶的,但是它可以直接表达对象的动态特征及用户定义的数据类型,并且集数据查询、更新和编程于一个简单的说明性语言之中。下面分别介绍 F-logic 和 Maudelog 的基本思想。首先说明面向对象的概念在 F-logic 中的表示。

1)对象 在 F-logic 中,对象由 id 项表示,id 项相当于对象标识符。但是,这里的对象标识符不是唯一的。一个对象可能对应多个 id 项,例如,John 既可以表示成 John 也可以表示成 father(Bob)(若 John 是 Bob 的父亲)。此外,F-logic 也不区分对象和类的概念,例如:student 既可以是表示学生的集合(类概念),也可以是表示 person 类的一个对象实例(对象概念),具体所指要视其出现的环境而定。

2)类型 在 F-logic 中,类型定义是由基调项表示的,例如,基调项 student[name@⇒{string,english},papers @year⇒⇒{string,chinese}]定义了 student 类的对象实例或子类的结构。它说明了 student 类的对象或子类有两个方法,name 和 papers,其中 name 的取值是单个英文字符串(该值同时为 string 和 english 类的实例),papers 的取值可以是多个汉字字符串。

3)方法 在 F-logic 中,方法是由带参数的属性表示的,可以取单值或集合值。例如数据项 John [name @→"JOHN", papers @1993→→{"人","风"}]表示 John 有两个方法,其中 name 是无参数单值方法,papers 是以"年号"为参数的集合方法。此

数据项表示 John 的名字为"JOHN",他在 1993 年发表了"人"和"风"两篇文章。另外,由于对象的方法名、参数及其取值在对象的数据项中是显式表示的,因而 F-logic 支持方法名的重载。

4)继承 在 F-logic 中,类层次项表示一般面向对象模型中的 is-a 关系和 instance-of 关系,一个类层次项具体代表那种关系是由其环境决定的。例如,类层次项 John:student 表示 John 是 student 的对象实例或子类。具有层次关系的类型之间的继承关系是由 F-logic 的语义结构定义的。

上面仅讨论了对象、方法、类型和继承的概念在 F-logic 中的语法表示形式,这些概念之间的联系,如对象之间的类属关系、类型之间的继承关系等均由 F-logic 的语义结构定义。F-logic 为面向对象数据库系统提出了一套逻辑框架,它具有完备的证明理论。然而,由于它的高阶特性,特别是它必须包括等词归结作为其证明理论的一部分,因而缺乏实用性。

Maudelog 语言的基本思想是:它是一个基于重写逻辑的说明性数据语言。Maudelog 语言的数据库定义分为两部分,函数模块定义和对象模块定义。函数模块定义说明了用户定义的数据类型、类型层次及能够施加于类型实例的操作,类似于传统的 ADT 说明,对象模块定义则说明了对象类、类层次及能作用于该类对象上的重写规则。一个良定义的函数模块定义了一套有序类等式逻辑(order-sorted equational logic)的理论,具有初始有序类代数。一个良定义的对象模块是一套重写逻辑理论,具有初始模型,虽然两者基于不同的逻辑体系,但前者可以逻辑嵌入后者,故而,它们可以统一于重写逻辑框架中。

虽然重写逻辑也可作为面向对象数据库的理论基础,但仍有许多问题有待研究,如合一、消息传递和查询机制等,所以,还没有基于它的实用系统。

## 4 纯面向对象概念的数据模型

完全基于面向对象概念的数据模型是相对于前面讨论的几类模型而言的。这类模型主要基于面向对象的思想和面向对象程序设计语言(如 C++ 和 Smalltalk)研究面向对象概念在数据库中的表达和使用。因而,基于此类模型的数据库系统有些具有较好的形式化模型,如 $O_2$[33,34]、ORION[32]、GOM[28] 和 Vodak[35];有些则是 C++ 或 Smalltalk 的持久化系统,如 ObjectStore[25]、ONTOS/VBASE[28,29,30,31] 和 GemStone[26,27,28] 等。下面我们主要讨论 $O_2$、ORION、GOM 和 Vodak 等具有较好形式化表达的数据模型。

1)对象和值 对"值"和"对象"概念的不同处理是这些模型的差别之一。$O_2$ 和 GOM 模型用对象概念表示可供共享的实体,而值用来表示不可共享的实体。这样,实体之间的关系可用类组合层次(class composion hierarchy)的类-值类型或类-类方式来表示。与之不同,ORION 模型只有对象概念,在表示实体之间关系时,只能用类组合层次中的类-类方式表示。为了便于表示对象之间的某些关系,如 part-of 等,它根据对象是否可被共享及是否具有依赖关系,定义了四种类型的引用约束。Vodak 模型则是上面两者的折中。它通过类-值类型表示实体之间的非共享 part-of 关系,用类-类及依赖约束表示实体之间的依赖关系。虽然这些模型都能表示实体之间的大部分关系。但是 $O_2$ 和 GOM 模型采用概念的差别(值与对象的差别)以简单的形式表示它们;而 ORION 模型则采用简单的概念加上引用约束来表示它们;Vodak 模型则是上面两者的折中。

2)类型 在这些模型中,类型定义的语义也有差别。ORION 和 Vodak 模型的类型定义既是内涵的也是外延的。$O_2$ 和 GOM 模型的类及值类型的定义均是内涵的,它只说明了所定义的类或值类型的结构。对象和值的集合及类的外延均是由构造子 set 显式(或隐式)构造的。在定义数据结构的能力方面,ORION 模型能定义由 set 和 tuple 构造子嵌套使用的任意复杂数据结构;Vodak 除了具有 tuple 构造子的能力,还提供了相异集合构造子,其表达能力强于 set 构造子;$O_2$ 模型则提供了 set、tuple 和 list 等值类型构造子。GOM 模型也提供了与 $O_2$ 模型相同的一组构造子,但是,它的这组构造子不但可用于复杂值类型的构造,还可用于构造复杂对象类。

3)方法 ORION、GOM 和 GemStone 模型在类型定义中,不但允许定义适用于某类型所有对象的方法,而且还可以定义表达类型所有对象某种特征的方法(如,对这类对象进行某类统计的方法),称为类方法;$O_2$ 模型允许对单个对象定义其特有的方法,称为例外(exception),这与 POSTGRES 模型的过程属性的值相似;Vodak 模型支持一般的方法定义。这些方法定义机制扩充了模型的表达能力。

4)继承 $O_2$、ORION、GOM 和 Vodak 模型均支持多类继承。当发生继承冲突时,ORION 按照某种预先定义的继承顺序由子类继承父类的特征,以解决冲突;$O_2$ 模型要求用户干涉,通过重命名等方法来解决冲突;GOM 和 Vodak 模型则按照用户定义

的继承顺序来处理继承冲突。在继承语义方面，$O_2$ 和 ORION 模型支持集合包含语义（即子类对象集是父类对象集的子集），替换语义（即可使用父类对象的任何地方也可使用子类对象）及特征继承语义（即子类完全继承父类的所有特征）；GOM 模型除支持这三种语义外，还支持多替换语义（即允许在某些情况下，可用父类对象代替子类对象）。在 Vodak 模型的类层次中，类与类之间不是通常的 is-a 关系，而是某种强迫（coercion）关系。它支持五种强迫关系，其中包括 is-a 关系。在支持封装性方面，$O_2$、GOM 和 ObjectStore 将类的特征（属性和方法）分成私有和公有两部分，只有公有部分可被继承。其它几个模型则无此区别，子类继承父类的所有特征。

## 5 有关问题的讨论

前面讨论了对象、类型、继承和方法等概念在各种模型中的表达和使用。由此可见，各个系统设计者对这些概念的理解并不统一，终究哪些理解更合理些，尚无明确定论。下面，我们就有关问题做进一步讨论。

**1）对象与值** “值”与“对象”是紧密联系的两个概念。两者的主要差别体现在：对象具有唯一标识符，可以被共享和拷贝，具有时、空特性及外延与内涵的区别；而值不具有这些性质。同一系统中是否同时支持“值”和“对象”两个概念是面向对象数据库系统应当考虑的重要问题之一。然而，对此问题尚无明确标准。例如 ORION 和 GemStore 等系统只支持“对象”概念，而 $O_2$、GOM 和 Sturburst 等则同时支持两者。与此问题相关，存在至少两种相等性概念，即对象 oid 相等和对象值相等。ODMG-93 虽然没有明确提出值的概念，但是其中定义的不变对象具有值的性质，因而是一种折中方法。

**2）类型** 类型定义的内容及语义也是须讨论的问题。有些系统（如 $O_2$、GOM、ONTOS 和 IRIS 等）的类型定义只说明属于此类型的对象的特征，还有一些系统（如 ORION 和 GemStore 等）的类型还定义了类方法，类属性及属性的缺省值等。当然，类型定义能表达的语义信息越多越好，但这会使模型复杂化。同样，应当提供哪些构造子（通常的有 set、tuple）来支持类型的数据结构定义也值得讨论。各模型中关于类型定义的语义到底应是内涵的、还是外延的或是两者都是也不统一。各模型支持封装性的机制也不尽相同，有些模型（如 $O_2$、ONTOS、ObjectStore 等）提供类似 C++ 机制支持封装性，而有些模型则没有此类机制。对象之间的关系通常由对象属性及有关约束（如 INVERSE 等）表示，也有的模型建议用多元关系表示（如 FUGUES、IRIS 等），两种表示是否一致及各自的特点，也未做过研究。

**3）方法** 在已讨论过的各种模型中存在四类方法定义：①只适用于特殊对象的方法（如 $O_2$ 的 exception 及 POSTGRES 的过程属性值）；②适用于类型中所有对象的方法（即通常所指的方法）；③适用于类型外延的方法（如，ORION 及 GemStone 的类方法）；④适用于多个类外延或对象集合的方法（如 PDM 的广义方法）。一个模型应当在怎样程度上支持这些方法尚不清楚。

**4）继承** 继承的语义尚需研究。通常继承的语义包括：特征继承、集合包含、行为继承（可替换性）及 GOM 模型中提出的多替换概念。大部分模型同时支持前三种语义。但是在某些情况下，特征继承与集合包含会产生不一致（如，在允许并类型的系统中，按集合包含语义，$A \cup B$ 是 A 和 B 的父类，但是 A、B 未必继承 $A \cup B$ 所有的特征）。在不同的系统中，引入不同的规定来消除这种不一致性。

## 6 结束语

由于面向对象技术缺乏坚实的理论基础，而这个领域中的研究与开发又非常活跃，因而，使得面向对象的概念在对象数据模型中有多种表达和使用形式。这些差别甚至反映在有关的标准工作中，虽然我们将面向对象数据模型划分四类来讨论，但这并不是对以往面向对象数据模型的一个完全分类。由于面向对象有关概念理解的多样性，要给出其完全的分类是困难的，这里只是为讨论方便而给出的粗略划分。

本文的目的是展示和对比面向对象的概念在各个面向对象模型中的表达和使用的差异，为进一步研究面向对象数据模型的形式化理论奠定基础，随着面向对象数据库技术有关标准如 ODMG-93，SQL3 等的出现，其研究和开发工作逐步向这些标准靠拢，趋于统一化，因而，期望不久会出现一套面向对象数据模型的理论。（参考文献共 39 篇　略）