维普资讯 http://www.cqvip.com

2003年6月
第33卷第3期

西北大学学报(自然科学版)
Journal of Northwest University(Natural Science Edition)

Jun. 2003
Vol. 33 No. 3

连续事件的语义数据模型

孙 蕾[1],周明全[1],J. H. ter Bekke[2]

(1.西北大学 计算机科学系,陕西 西安 710069;2.荷兰代尔夫特理工大学 计算机科学系,荷兰 代尔夫特)

摘要:语义抽象概念 aggregation 和 generalization 不仅对不包含时间概念的复杂情况的数据模型的建立很有用,而且还可以成功地用于连续事件的建模。建立了两种情况并存的语义数据模型:ordering(次序)和 sequencing(先后顺序),并且可以在任何程序环境下实现。

关 键 词:数据模型;语义数据库;集 合;概 括

中图分类号:TP311.132.3 **文献标识码**:A **文章编号**:1000-274X(2003)03-0277-04

1 简 介

语义数据模型中的重要概念:

Type:真实的世界是用一些相关对象的 type 及基本的 type-attribute 关系来描述的,在图中用一个长方形来表示[1]。

Aggregation:是一定数量 type 的集合,Aggregation 本身可以是一个新的 type,在 aggregation 中的 type 又叫作这个新 type 的一个属性[2]。例如:

type 运输=车辆,目的地,交付_时间,货物,

type 车辆=厂商,型号,价格,燃料,制造_年。

其中,属性还可以有一个前缀,用下划线来表示这种关系。例如:与 type"年"相关的"制造_年"。

Generalization:把不同 type 的相同属性组合成一个新的 type,即 generalization。用图1来表示。

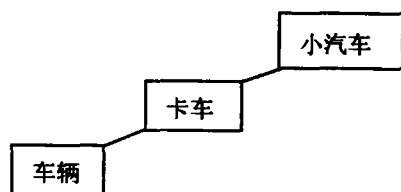

图1 概括层级结构图

Fig. 1 Generalization hierarchy

图1也是 specializations 不相交的体现,即小汽车和卡车组成了一个块,但并不是所有种类的车辆都必须被具体化。这种结构保证了 generalization 的属性的惟一性。此外,在一个块中的这些属性的值也是惟一的,如:v1 和 v3 不可能出现在 truck 的表中。

以上所介绍的 aggregation 和 generalization 的定义已经证明了这些概念的层级的特点,下面着重阐述这个特点在 ordering 和 sequencing 中的应用。

2 次 序

图2所表示的 aggregation 的层级关系图说明了一个自然次序(ordering):每一个职员有一个经理,所以一个经理必须在一个职员被任命为经理之前就已经存在。

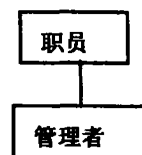

图2 简单次序

Fig. 2 Simple ordering

这个自然顺序可以限制来保证。比如,职员的工资少于经理的工资:

assert employee its salary constrain (true) = salary<manager its salary。

一般情况下并不是只有两层关系而是有几层 aggregation 关系,用圆形曲线来表示这种关系。

type 职员=姓名,住址,工资,管理_职员。

又如,假设一个软件有不同的版本,那么就可以

收稿日期:2002-09-27

作者简介:孙 蕾(1968-),女,河南洛阳人,西北大学博士生,从事数据库和图像处理研究。

维普资讯 http://www.cqvip.com

用图3的圆形层级结构来表示这种关系。下面的type的定义说明了每一个版本有一个旧的版本。

type 版本=描述,完成_时间,前面_版本。

这个结构允许软件版本的连续性:每一个版本都有一个旧的版本,还可以有几个新的版本。其限制如下:

assert version its time constraint (true) = previous _ version=NULL or completion _ date > previous _ version its completion _ date。

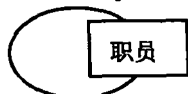

图3 递归次序

Fig. 3 Recursive ordering

3 先后顺序

Specialization可以应用在连续事件中。假设有关软件项目的时间必须登记。那么,一个项目可以处于以下阶段中的一个阶段:开始阶段、进行阶段、完成阶段。

第一种方法是创建一个type,包含所有的(目前和将来)要登记的属性。此type的定义如下:

type 项目=题目,雇员,开始_时间,完成_时间,报告_题目,评价,评审人,阶段。

但是,这种方法有以下缺点:① 在特定的阶段一些属性无意义[1]。例如:在项目的进行阶段评价的意义是什么;② 一些属性在不同的阶段有不同的意思,比如,开始_时间在开始阶段是一个计划的日期,而在完成阶段是一个具体和现实的日期;③ 一些属性同时具有一个意义,例如,报告_题目和评价在项目完成时都要登记。但是,在完成之前它们都没有意义。

所以,一个比较好的方法就是应用specialization。根据一个项目的不同阶段,将其划分为(开始)项目,正在进行中的项目和已经完成的项目。后两项可以被认为是一个项目的两个不相交的specialization,如图4所示。

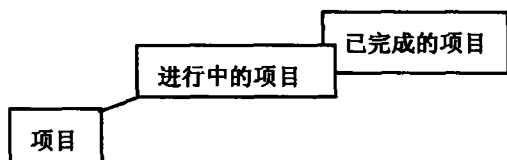

图4 初始的先后顺序

Fig. 4 Preliminary sequencing

type 项目=题目,计划开始_时间。

type 进行中的项目=[项目],开始_时间,计划完成_时间,完成_时间,雇员。

type 已完成的项目=[项目],开始_时间,完成_时间,雇员,评价,评审人,报告_题目。

但是,这种方法仍然有可能使得赋予属性的值没有意义。在进行中的项目和完成的项目都具有项目的特性,并且一个项目不能同时即是在进行中的项目,又是完成的项目。然而,当一个项目结束了它的进行阶段,它就只能处于完成阶段了。但是,进行中的项目和完成的项目在图4中是相互不相交的specializations。因此,这种方法有一定的缺点:① 特定的属性必须从在进行的项目转移到完成的项目,比如,开始_时间和雇员;② 数据的转移将丢失历史数据。

为了克服以上缺点,我们运用图5的方法,即完成的项目是进行中的项目的specialization,这样一来,进行中的项目的历史数据就不再丢失了。

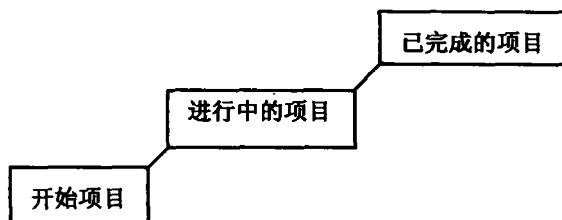

图5 最后的顺序

Fig. 5 Final sequencing

type 开始项目=题目,计划开始_时间,阶段。

type 进行中的项目=[过去_开始项目],开始_时间,计划完成_时间,雇员。

type 已完成项目=[过去_进行中的项目],完成_时间,评价,评审人,报告_题目。

以上的方法有如下优点:① 在任何时候都可以确保项目有完整的历史;② 没有无意义的属性,每一个属性都在恰当的时候被赋值;③ 事件的次序完全由图5的结构来决定;④ 数据的转移没有必要了,在后一个阶段可以得到前一个阶段的数据。

因此,在处理一定数量的连续事件时,这个数据模型很受青睐。即一个完成的项目过去是一个进行中的项目,一个进行中的项目过去是一个开始的项目。如果项目阶段的数目不定时,见图6所示。

图6 递归顺序

Fig. 6 Recursive sequencing

维普资讯 http://www.cqvip.com

type 项目阶段=[前面的_项目阶段],题目,雇员,报告,开始_时间,完成_时间。

4 版本管理

先看一个例子:设计阶段是一个软件项目的开始阶段,在这一阶段研究新产品的功能性和实用性,并完成一个有关软件模块的报告。每一个软件模块都有特定的功能,所有这些模块组成了软件产品的结构(configuration)。如果这一阶段通过,下一个阶段可以开始。在这一阶段,开发了几个版本的软件,并不断的改进,最后当有一个版本满足了所有的要求,就把它作为新产品发布。这个过程不断的进行。因此,可以归纳出项目的主要阶段:① 设计阶段,研究软件的功能和实用性,这一阶段可以分成开始设计阶段和完成设计阶段;② 实现阶段,完成产品的开发,这一阶段是一个连续的过程;③ 维护阶段,负责产品的维护,这一阶段也可以分成开始维护阶段和完成维护阶段。

这几个阶段可以用图5的方法来建模。

在实现阶段的产品的几个版本可以定义如下(参照前面阐述的 ordering 的应用):

type 版本=描述,实现阶段,状态,前面_版本。

其中,状态说明版本处于研制或发布状态,前面_状态说明此版本与其前一版本的关系。然而,这个例子的情况就复杂了。前面所阐述的处理 ordering 的方法只允许一个版本有几个后继的新的版本,并不允许只有一个后继的新版本(图7)。很明显,前面所描述的圆形结构适用于 y-module 和 z-module. 的情况,然而对于合并(即 x-module 的情况,x2,x3 和 x4 合并成为 x5)就不适用了。因此,考虑有图8的两种方法。

方法1:type 关系=前面_版本,其次_版本。

这种方法很明显忽视了次序(ordering)。这种 type 的定义即允许有意义的合并,也允许无意义的合并(如,x5 和 x1 的合并)。

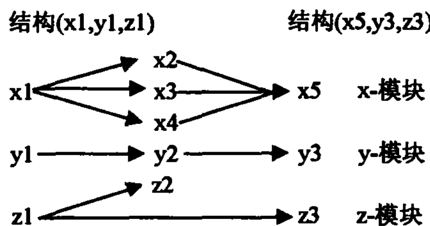

图7 版本的历史

Fig. 7 Version histories

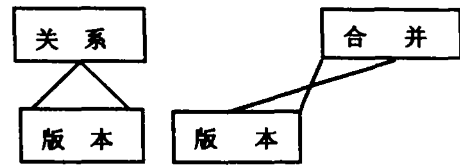

图8 合并的两种方法

Fig. 8 Alternatives of merging

方法2:type 合并=[前面_版本],合并_版本。

下面的限制保证了合并的正确性,每个版本在一次合并中最多只能出现一次。

assert merging its uniqueness (true) =previous _ version≠merged _ version its previous _ version。

那么,剩下的问题与产品的外形结构(configuration)有关,见图9。

方法1:type 出现=版本,结构。

体现了外形结构和版本之间的关系是 n 对 m 的关系。一个版本可以发生在几个结构中,而一个结构可以包含几个版本。而实际情况是:

在一个结构中不是任何3个版本的组合都是允许的。如图7中的 x1,x2 和 y3 的组合是不允许的。

在一个结构中并没有确切表示出每一个版本,如图7,有而且只有3个版本。

方法2:type 结构=x _版本,y _版本,z _版本。此结构和图7是个完美的结合,很容易添加限制。例如,所有的版本都处于工作阶段(如对用户都是可见的),可用下面的语句加以限制:

assert configuration its x _ working (true)=x _version its state="working"

仍然可以用圆形结构来表示现在的结构和前面的_结构的关系,最后可以归纳出:① 产品的结构,是产品的模型,对所有用户都是可见的;② 发布,将最终的产品移交给维护人员,产品结构中的所有版本就不能使用了。

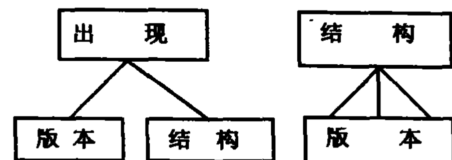

图9 结构的可选择的方法

Fig. 9 Configuration alternatives

type 设计阶段=项目_名称,开始_时间,研究_

维普资讯 http://www.cqvip.com

小组,报告,评价。

type 实现阶段=[过去_设计阶段],开发_小组,开始_时间。

type 维护阶段=[过去_实现阶段],维护_小组,开始_时间。

type 版本=描述,实现阶段,状态,前面_版本。

type 合并=[前面的_版本],合并的_版本。

type 结构=x _版本,y _版本,z _版本,前面_结构。

type 发布=[结构],维护阶段,发布_时间。

在图 10 中,每一个版本都和实现阶段有一个关系,而版本都有一个次序关系的关系(从属性前面的一版本可以看出),因此下面的限制是很有必要的:

assert version its correct realization (true) = previous _ version = NULL or realization stage = previous _ version its realization stage。

在图 10 中,发布有两条路径指向实现阶段,显然必须都指向同一个实现阶段,因此必须给出下列限制:

assert release its x _ allowed (true) = configuration its x _ version its realization stage= maintenance stage its realization stage。

下面的限制保证了所有的版本被冻结:

assert release its x _ frozen (true)= configuration its x _ version its stage ="released"。

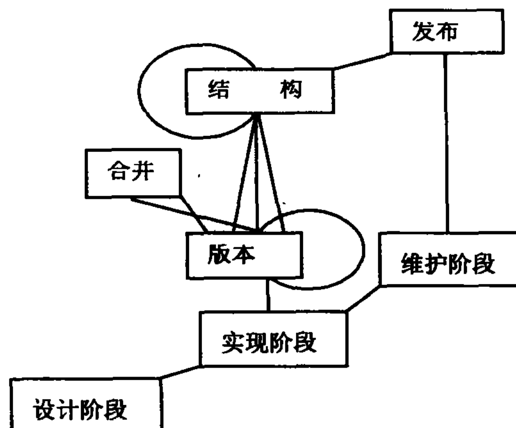

图 10 项目管理

Fig. 10 Project management

参考文献:

[1] ter BEKKE J H. Sementic Data Modeling[M]. Hemel Hempstead:Prentice Hall,1992.

[2] BERTINO E,MARTINO L. Object-oriends database management systems:Cencept and issues [J]. IEEE Computer,1991,(4):33-47.

(编 辑 曾大刚)

Semantic modeling of successive event

SUN Lei[1],ZHOU Ming-quan[1],ter BEKKE J H[2]

(1. Department of Computer Science,Northwest University,Xi'an 710069 China;2. Department of Computer Science,Delft University of Technology,Delf,the Netherland)

Abstract:The semantic abstractions aggregation and generalization are not only extremely useful for modeling complex situations containing time-independent events but also very successful for modeling of successive events. The semantic data models are set up considering two situations: ordering and sequencing, which do not require special constructs and can therefore be implemented in any programming environment.

Key words:data modeling; semantic database; aggregation; generalization