基于 CWM 的元数据管理策略

谢培基,余金山
(华侨大学 计算机科学与技术学院,福建 泉州 362021)

摘 要:对三种经典的元数据管理策略进行了分析和比较,总结了基于 CWM 的元数据管理策略的优势,对该策略给出了改进的元数据管理体系结构,解决了各软件产品或工具间的元数据的便捷交换问题,做到元数据存储、管理和交换的协调统一,重点讨论了其核心部分(元仓库)的设计与实现。

关键词:CWM;元数据管理;元仓库;对象关系映射
中图分类号:TP3 文献标识码:A 文章编号:1674-7720(2011)23-0012-04

The metadata management strategy based on CWM

Xie Peiji, Yu Jinshan
(College of Computer Science and Technology, Huaqiao University, Quanzhou 362021, China)

Abstract: In this paper, analyze and compare three meta-data management strategies, and sum up the advantage of the metadata management strategy based on CWM. Based on this strategy, give an improved metadata management architecture to solve the problem of the metadata exchange which in the various software products or tools, and the metadata can be stored, managed and exchanged harmoniously. Finally, this paper discusses the core that how to design the meta store.

Key words: CWM; metadata management; meta store; the object-relational mapping

数据仓库技术是在数据库基础上发展而来的新一代信息管理技术,主要用于支持企业信息集成、数据挖掘、企业决策支持等的应用。在数据仓库建设过程中,由于各工具厂商采用不同的元数据标准和不同的元数据管理策略,使得依靠这些工具进行数据集成、数据共享显得十分困难。为了统一数据仓库开发商元数据管理策略,实现元数据的交流和数据的集成,2001 年 OMG 组织在其已制定的规范 UML、MOF、XMI 的基础上提出公共仓库元模型(CWM)。CWM 是 OMG 制定的一个互操作标准,为数据仓库和业务分析领域中使用的元数据定义了一种通用语言和交换机制[1]。CWM 不仅提供了极受欢迎的描述数据仓库与业务分析元数据的公共元模型,而且还提供了基于 XML 的交换工具。CWM 本质上是一种交换技术,其目的是促进多个厂商的不同软件工具间的元数据交换活动。

本文基于 CWM 的元数据管理策略,介绍了元数据管理的三种主要策略,并对此三种策略进行比较,分析各策略的功能与复杂度,总结基于 CWM 的元数据管理策略的优势,进而给出基于 CWM 元数据管理策略改进的元数据管理体系结构,最后讨论了元仓库的设计。

1 元数据的不同管理策略及其比较

元数据管理策略为元数据集成、共享和重用制定目标和需求。成功进行元数据集成的关键之一是建立一个有效的元数据管理策略。从元数据的发展历史[2]来看,元数据管理策略包括三种:搭建元数据桥、搭建元数据中央存储库、构建元数据仓库。

(1)搭建元数据桥[3]:是一种能够将一个产品的元数据转换成另一个产品所要求的格式的软件。使用元数据桥实现不同工具间的元数据集成是一种点到点的元数据体系结构。在这种结构中,每一对被集成的工具之间都需要一个独立、双向的桥,对于 n 个产品要实现元数据交换必须建立 $n\times(n-1)$ 个元数据桥。这种方式集成元数据大幅增加了开发和维护费用,而且通常将一种格式的元数据转换为另一种格式时,都会有一定的信息损失。

(2)搭建元数据中央存储库[4]:是具有特定目的的数据库,它存储、控制并能操作环境中其他所有相关软件产品的元数据。软件产品从中央存储库中检索、使用元数据,每个产品必须实现它自己的存储库访问层(另一

种形式的桥)。通过使用元数据中央存储库可以在一定程度上解决定义全局可用且被广泛理解的元数据的需要但并没有完全消除问题，它仍然需要使用到元数据桥，使得成本无法降低很多，也没有消除受制于特定的厂商的问题。

由于点到点的体系结构和中央存储库的集成体系结构并没有形成一个统一的元数据标准，所以其方案成本都相对昂贵。

(3)构建元数据仓库：即是基于 CWM 的元数据管理策略。元数据仓库的功能包括对元数据源的 ETL、元数据的 Warehouse 以及终端用户的各种分析、挖掘、报告工具。通过构建基于 CWM 的元数据仓库，使得各软件产品元数据有一个一致的标准(CWM 标准)，各软件工具之间只需要建立一个与元数据仓库连接的“桥”(即 CWM 适配器)就能实现元数据的交换或共享。元数据仓库与元数据存储库都要求建立通用的元数据标准，但二者相比有所不同：①元数据存储库的刷新周期相对于元数据仓库来说要慢，元数据仓库的元数据是准实时的，而元数据存储库的元数据通常的刷新周期在 1 天以上；②元数据仓库所集成的元数据不断变化，其保存元数据的更新情况。而元数据存储库则是将所有不同时期的元数据都存储下来。

通过对上述三种元数据管理策略的分析，得出如图 1 所示的三种策略的对比图。

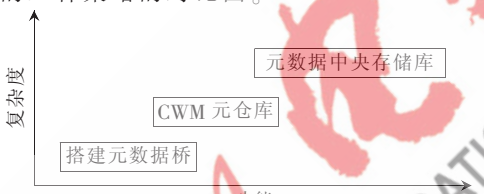

图 1 元数据管理策略对比

基于 CWM 的元数据管理策略使得元数据的交换有了一个统一的具体通用标准，解决了用元数据桥式的元数据交换带来的元数据的杂乱无章、不可理解性，以及为了读懂接收的元数据，对交换的元数据进行从一种格式到另一种格式的转换所带来的一定数量的元数据的丢失，破坏了元数据的完整性、准确性。

比较三种元数据的功能复杂度，CWM 元数据管理策略中使用的 CWM 元仓库既满足了性能的要求，又能够提供很廉价的存放元数据的场所；而对于元数据中央存储库，虽然性能上偏优，但元数据在获得库所提供的复杂性能的同时也会受到它的限制，例如库本身的复杂性和资源需求，导致人员培训费用的增加等。

2 基于 CWM 的元数据管理体系结构

通过对三种元数据管理策略的分析比较可以发现，搭建基于 CWM 的元数据仓库的管理策略存在较多的优点，它能够以较少的代价提供丰富的数据管理功能，即持续存储、允许并发、对数据仓库环境中复杂元数据提供事物访问，而相对于一个完善的基于元数据中央存储库的元数据管理系统来说，基于 CWM 的元数据管理系统在功能上还存在一些不足，如版本化、生命周期的管理等。针对基于 CWM 的元数据管理策略不足，本文给出了一种如图 2 所示改进的元数据管理体系结构。

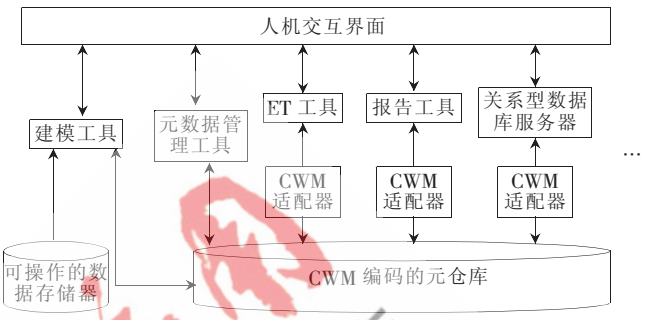

图 2 基于 CWM 的元数据管理体系结构

该系统结构主要由元仓库、可操作的数据存储器、建模工具、元数据管理工具及其他一些数据库应用工具或软件产品等模块组成。

(1)元仓库：是整个架构的核心，是一个全局共享元数据的以 CWM 编码的关系型数据库，主要用于存储管理各以 CWM 为元数据标准的工具产生的元数据和以 CWM 规范建模和操作过程中所产生的元数据以及从各类型数据库提取的元数据。

(2)建模工具：可供用户对可操作的数据存储器中的某一主题建立元模型实例，直接收集元数据，然后存入到元仓库中，作为其他工具所需要元数据的数据源。

(3)元数据管理工具：可供对元仓库中的元数据进行增删改查操作，方便用户对元仓库的更新和维护。但元仓库中的元数据一般的用户很难读懂，很难将一堆杂乱的元数据联系起来理解其实际意义，应当将其与建模工具相结合，用图形化的形式显示这些元数据的元数据模型。

(4) 与元仓库连接的各软件工具、应用程序都支持 CWM 标准，它们都实现了相同的用于元数据交换的公共接口，这也意味着任何支持 CWM 的软件组件都能够很容易地理解其他支持 CWM 的组件的元数据实例，并且能通过标准的 CWM 元数据接口集来访问元数据。在这些支持 CWM 的工具与元仓库的元数据交流中，同样需要一个元数据“桥”，这个桥可以用一个 CWM 适配器的轻量级的桥代替。

(5)CWM 适配器是一个软件，它可以理解 CWM 元模型以及拥有该适配器的软件产品内部的元模型和专有的元数据接口。适配器的公共端实现 CWM 的标准接口，另一端则连接到产品的特定接口。针对某一软件产品或工具写适应它的适配器只需要编写一次，所以它相对于其他的元数据桥节省了很多开销。

3 元仓库的设计

在元数据管理体系结构中，CWM 编码的元仓库是

整个架构的核心。因为 CWM 本身是一个面向对象的元模型，而各软件产品或工具的元数据大部分是建立在关系数据库之上，且目前面向对象的数据库管理系统不普及[5]，因而无法使用面向对象的 DBMS 来创建一个元仓库，因此，需要将 CWM 的面向对象元模型映射到一个关系数据库中，建立基于关系数据库的元仓库。在将 CWM 面向对象的概念映射到关系表上同时又要保持元数据本身的逻辑结构不变，需要解决 CWM 中数据类型、类、继承和关联等在关系数据库中的映射问题。在 CWM 中大量使用了 UML 的继承特性，CWM 的 21 个元模型包[6]中就存在着大量的泛化层次结构，因此，选择如何实现泛化层次在元仓库的开发是最关键的。为此，下面首先分析了继承模式的几种映射策略，选择了一种实现方便、节约存储空间的方法来实现元仓库的建立，然后讨论对象关系映射中数据类型、关联的映射问题。

3.1 继承映射

从类到表的映射并不是简单的一一对应关系，因为一个子类可以继承父类的一些特性和操作，而继承是 UML 和面向对象系统的一个主要特征，它把类组织成逆向的树形结构（即泛化层次），因此必须考虑泛化层次的对象的映射问题。以图 3 所示的 CWM 关系型包中的 Relational::Trigger 的泛化层次为例，给出几种对象关系映射策略。

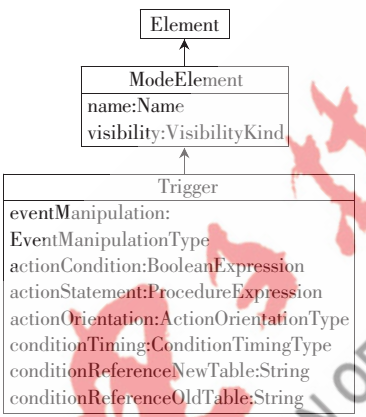

图 3 Relational::Trigger 的泛化层次

(1)整个泛化层次结构映射成一张表

将一个完整的泛化层次结构映射成一张表，层次结构中所有类的属性都存放在这个表中，每个实例对应表中的一行。为了表明一个对象实例是属于哪个类的，必须添加一个属性“对象类别”OID，这种映射对应的表如表 1 所示。这种映射策略的优点是结构相对简单，只有一个单一的表，其缺点是存储空间浪费大、利用率低、类层次间的耦合非常高，当泛化层次结构中的某一个类要增加属性时，必须将一个新属性增加到表中，这将影响到泛化层次中的所有类。

表 1 整个泛化层次映射成一张表

Relational_trigger						
oid	name	visibility	eventManipulation	actionCondition	actionStatement	…

(2) 每个类映射成一张包含该类继承属性的表

使用这种映射策略，是将每个类映射成一张表，在表中既包含该类的属性又包括其父类的属性，如表 2 所示。这种映射策略的优点是对单个类的查询操作比较简单，但是同样会造成存储空间的浪费，且类层次间的耦合性较高，当父类的结构改变时，子类的结构也要跟着改变。此外在实现上不方便，因为 ModelElement 有两个子类（Trigger 类和 A 类），Trigger 有一个 ID，A 也有一个 ID，它们都是 ModelElement 的子类。如果这两个表的主键相同，父类是无法识别这两个子类的，所以这两个 ID 一定不能相同，因此也就不能使用 ID 自动生成策略。

表 2 每个类映射成一张表
（包含该类的属性又包括其父类的属性）

Element					
OID					

ModelElement		
id	name	eventManipulation

Trigger					
OID	name	visibility	eventManipulation	actionCondition	…

(3) 每个类映射成一张只包含该类特有属性的表对各个表进行连接

使用这种策略子类对应的表 ID 应该参考父类对应的表的 ID，即进行主键连接，对应的表结构如表 3 所示。这种策略的优点是存储空间比较小，类间的耦合性比较低，查询的时候不用找很多的表，缺点是要进行表连接，每当增加一个子类还应添加一个表。

表 3 每个类映射成一张表进行连接

Element	
ID	
124	

ModelElement		
id	name	eventManipulation
124		

Trigger				
ID	eventManipulation	actionCondition	actionStatement	…
124				

由上分析可以得出：(1)第一种策略是将整个泛化层次映射成一个表结构。由于 CWM 元模型包中的泛化层次树形结构很深，所以会使得元组很庞大，浪费大量的存储空间，且每次读取时需要读入很多不必要的属性，因此不适合用这种策略建立元仓库；(2) 第二种策略同样也会造成存储空间的浪费，在实现上也不方便，亦不适合采用；(3)第三种策略是将每个类映射成一个表，每个表只包含该类含有的属性，这种方法可节省存储空间，虽然查询操作涉及多个表的连接操作，连接操作将多个连接的表组成一个表较耗时，但在计算机速度不断提高且不要求实时性的情况下适宜选用这种策略。所以本文使用第三种策略来实现 CWM 元模型包中各泛化层次结构的映射问题，建立元仓库。

3.2 数据类型映射

每个 CWM 属性可以是以下三类中的一种：简单数据类型、枚举数据类型和基于类的数据类型。这些 CWM 的数据类型集必须映射到相应的 T-SQL 语言支持的数据类型上，如果没有直接对应的 T-SQL 数据类型，元仓

库必须创建数据结构使其能模仿所要求的类型。如简单的数据类型映射规则是：只要将 CWM 简单数据类型映射到 CORBA IDL 类型码，而后映射到 T-SQL 数据类型即可。

枚举数据类型映射规则是将每个 CWM 枚举数据类型用一个创建的表表示。枚举数据类型表中的行只有一列，名为 _EnumLiteral，它包含枚举文字值。当基于枚举数据类型的列值变化时，新的取值按照该表确定以保持列数据的完整性。

基于类的数据类型映射规则，则需要创建一个基于类的类型的实例，然后将该实例的键存放在一个向表示拥有该类属性的类的表增加的基础列中。

3.3 关联映射模式

关联是 UML 中表示类的实例间的关系的一种方法，有一对一、一对多、多对多三种关联。

对于一对一关联映射规则，只要在如图 4 所示的两个关联类各对应的表中增加一列。

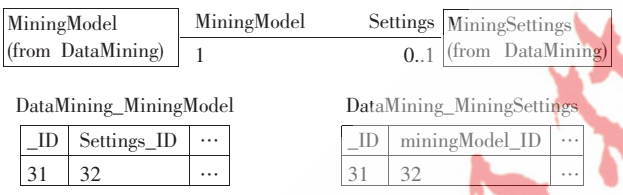

图 4 1:1 关联映射

一对多的关联映射规则，仅仅需要改变关联的多端的固定表，在该固定表中增加一列，以存放一端的一个实例的标识值。如果关联的多端是有序的，则需要在多端的固定表中增加一列，以存储记录多端实例次序的顺序值，如图 5 所示。

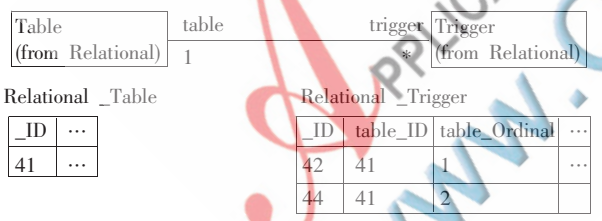

图 5 1:N 关联映射

多对多关联映射规则需要将关联单独映射成一张关系表，如图 6 所示。

本文分析了三种典型的元数据管理策略，并进行了详细的比较，提出了基于 CWM 管理策略的元数据管理

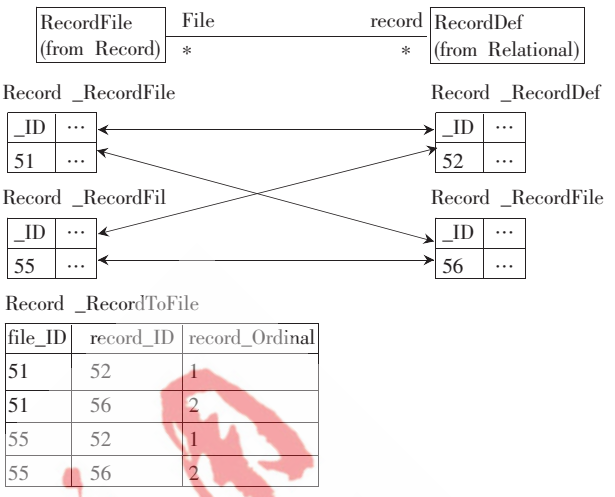

图 6 M:N 关联映射

体系结构，解决了多个厂商的产品之间的元数据交换使用繁琐而复杂的软件工具来实现交换的问题。CWM 为数据仓库和商业智能环境下的元数据交换制定了一个标准，元仓库是对其的映射实现，用元仓库来存放管理元数据，做到元数据存储、管理和交换的协调统一。

参考文献

[1] 吴晓渊，宁洪.基于 CWM 的企业数据集成研究[C].中国计算机大会，2005.

[2] VADUVA A, DITTRICH K R. Metadata management for data warehousing: between vision and reality[C]. 2001 Int'l Database Engineering &Amp; Application Symp,2001.

[3] 杨华甫，邓守城，高张.CWM 中基于元模式的数据集成研究与实现[J].现代计算机，2008(8).

[4] 聂茹，张虹.数据仓库元数据管理模式的分析与比较[J].计算机应用研究，2005，22(2).

[5] 马思红.浅谈面向对象数据库的技术和发展[J].安徽农业科学，2007，35(24).

[6] INMON W H. Building the data warehouse [M]. John Wiley& Sons Inc. 2005.

(收稿日期：2011-08-01)

作者简介：

谢培基，男，1986 年生，硕士研究生，主要研究方向：数据集成。

余金山，男，1952 年生，教授，硕士生导师，主要研究方向：软件工程。