第33卷 第16期 计 算 机 工 程 2007年8月
Vol.33 No.16 Computer Engineering August 2007
·博士论文· 文章编号：1000—3428(2007)16—0001—03 文献标识码：A 中图分类号：TP391

信息集成系统中的模式融合问题研究

刘君强，彭智勇
（武汉大学软件工程国家重点实验室，武汉 430072）

摘 要：将模式融合分为模式映射、主键-外键融合和用户自定义的完整性融合3个部分。模式映射将关系模式或其他模式映射为对象代理模式类，主键-外键映射采用新的模式融合算法，充分考虑对象代理模型的特点。提出了一个新的操作 Merge 来解决用户自定义的模式融合。该操作克服了其他模型的用户自定义模式限制难于融合的问题。在 Smalltalk 环境中实现了基于对象代理模型的模式融合系统，并给出应用实例。

关键词：Smalltalk；对象代理模型；模式融合

Schema Merging in Information Integration System

LIU Jun-qiang, PENG Zhi-yong
(State Key Lab of Software Engineering, Wuhan University, Wuhan 430072)

【Abstract】 This paper develops a structure for schema merging that consists of schema mapping, key-foreignkey merging and user-defined integrity merging. Schema mapping transforms relational schema or other schema into deputy schema class. The paper suggests a new algorithm for key-foreignkey merging that makes use of deputy mechanism, proposes a new operator merge for user-defined integrity in which object deputy model makes schema transformation easier. And it implements the schema merging system based on object deputy model in Smalltalk environment, with some application examples.

【Key words】 Smalltalk; object deputy model; schema merging

随着社会各种学科的发展，信息集成系统也变得越来越复杂，特别是系统中的元数据操作问题。这些元数据的处理效率将影响到集成系统中的数据库模式，本体、工作流的定义、特别是限制的融合问题[1]。在实际系统中，解决元数据的操作问题是复杂的。主要有3个原因：

(1)各种不同的数据库(关系数据库、对象数据库等)会定义各自不同的模式。

(2)大多数的应用是面向特定领域和面向特定语言的。

(3)各个数据库的知识规则和限制是不同。

文献[2, 3]从理论方面阐述了模式融合，文献[1, 4]给出了一个通用的模式融合框架和实际应用。但这些系统模型的柔软性较差，使得异构信息源之间的融合比较困难。而且，面向对象的模型只能提供自上而下从父类到子类的继承，而缺乏自下而上以子类到父类的继承，面向对象模型的视图机制是很难实现的，而视图机制恰恰在多个局部模式集成后重构模式时扮演非常重要的角色。文献[5, 6]提出了混合的集成视图的解决方案，文献[7]实现了混合视图系统，并使用了文献[6]的算法很好地解决了视图选择问题。笔者的基于对象代理模型的信息集成系统在文献[7, 8]中的应用表明，它能够很好地解决数据级别的问题，而本文将给出基于对象代理模型的混合视图的方法，以解决模式的融合问题。

1 对象代理模型

对象代理模型[10]是针对数据库中管理复杂数据对象的要求，扩充传统面向对象数据模型形成的一种新的数据模型。对象代理模型是目前信息集成的一种较好的通用数据模型，对于解决上面提出的问题有其独特的优点。对象代理模型的主要特征如下：

(1)一个代理对象享有其自己的标识符，而且除了继承源对象的属性和方法外，还能拥有独自的属性和方法。

(2)一个对象的属性和方法可以被它的代理对象所继承，通过切换操作实现。在切换操作中能改变所继承的属性和方法的名字和类型。

(3)在对象和它的代理对象之间存在着双向连接，这样既能实现继承，又能实现更新迁移。

在对象代理模型中，一个对象可以有许多代理对象，分别代理原对象的不同方面。一个对象可能只有一个代理对象，许多对象也可能综合成一个代理对象，其定义如下：

定义 1(对象) 每个对象都有一个标识符，若干 *attribute* 和 *method*。具有相同 *attribute* 和 *method* 的对象的模式被定义成一个类，它由名字、外延和类型组成。类的外延是属于这个类的对象的集合，也叫作它的实例。类 C 可以表示成：

$C=<\{O\},\{Ta: a: c\},\{m:\{Tp:p\}\}>$

(1)$\{O\}$是 C 的外延，其中，O 是 C 的一个实例。

(2)$\{T_a : a : c\}$是C的*attribute*定义的集合，其中，a和T_a分别代表了一个*attribute*的名字和类型，c表示属性的限制。对象o的*attribute*的值以$o.a$来表示。每个*attribute*$\{T_a : a\}$有如下两个基本的方法：

$read(o.a) \Rightarrow \uparrow o.a$

基金项目：国家自然科学基金资助项目(60573095)
作者简介：刘君强(1978－)，男，博士研究生，主研方向：信息集成，数据挖掘；彭智勇，教授、博士生导师
收稿日期：2006-10-22 **E-mail：**liujunqiang@msn.com

$write(o,a,v) \Rightarrow o.a:=v$

$read\ (o,c) \Rightarrow \uparrow o.c$

$write(o,c,v) \Rightarrow o.c:=v$

其中，$\Rightarrow$，↑和:=分别代表激活、结果返回和赋值操作。

(3)对于一个 method 的引用，表示如下：$apply(o,m,\{p\})$。

定义 2(代理对象) 代理对象是基于对象和其他的代理对象来定义的。代理类从源对象的类(源类)中派生，通常 $Cs=<\{o^s\},\{T_a: a^s:c^s\}, \{m^s:\{T_p:p^s\}>$是一个源类，其代理类$C^d$定义如下：

$$C^d =< \{o^d \left| \begin{matrix} (o^d \to o^s)\vee(o^d \to ...\times o^s \times ...)\vee(o^d \to \{o^s\}) \\ sp(o^s)\vee jp(...\times o^s \times ...)\vee up(\{o^s\}) == true \end{matrix} \right.\}$$

$$\{T_{a^d} : a^d\}\cup\{T_{a_+^d} : a_+^d\},\{m^d : \{T_{p^d} : p^d\}\}\cup\{m_+^d : \{T_{p_+^d} : p_+^d\}\} >$$

(1) $\{o^d \left| \begin{matrix} (o^d \to o^s)\vee(o^d \to ...\times o^s \times ...)\vee(o^d \to \{o^s\}) \\ sp(o^s)\vee jp(...\times o^s \times ...)\vee up(\{o^s\}) == true \end{matrix} \right.\}$ 是 C^d 的外延。

其中，$o^d{}_i \to o^s{}_i \,|\, ...\times o^s{}_i \times ... \,|\, \{o^s{}_i\}$表示$o^d$是$o^s$，$...\times o^s \times ...$，$\{o^s\}$的代理对象(sp, jp和up 分别表示选择，连接和联合谓词)。

$\{o^d|(o^d\to o^s)\vee(o^d\to ...\times o^s\times ...)\vee(o^d\to\{o^s\})$

$msp(o.c^s)\vee mjp(...\times o.c^s\times ...)\vee mmp(o.c^s\vee ...)\vee up(\{o.c^s\})==$ true$\}$

其中，msp,mjp,mmp 和 up 分别表示模式操作的选择、连接、融合和联合谓词。

(2) $(Ta^d : a^d)\cup(Ta^d{}_+ : a^d{}_+)$是$C^d$ *attribute*定义的集合；

$\{T_a{}^d{:}c^d\}\cup\{T_{a+}{}^d{:}c_+{}^d\}$是$C^d$ *constraint*定义的集合。

1) $(Ta^d : a^d)$是从 C^s 的$(Ta^s : a^s)$中继承的*attribute*的集合，其中，切换操作被定义为

$read(o^d, a^d) \Rightarrow \uparrow fTa^s\to Ta^d\ (read(o^s, a^s))$

$write(o^d, a^d, v^d) \Rightarrow write(o^d, a^d, fTa^s\to Ta^d(v^d))$

$\{T_a{}^d{:}c^d\}$是从C^s的$\{T_a{}^s : c^s\}$中继承*constraint*的集合，切换操作定义为

$read(o^d, a^d) \Rightarrow \uparrow fTa^s\to Ta^d\ (read(o^s, a^s))$

$write(o^d, a^d, c^d) \Rightarrow write(o^d, a^d, fTa^s\to Ta^d(c^d))$

2) $(Ta^d{}_+ : a^d{}_+)$是C^d追加*attribute*的集合，其基本方法定义为

$read(o^d, a^d{}_+)\Rightarrow\uparrow o^d.\, a^d{}_+$

$write(o^d, a^d{}_+, v^d{}_+) \Rightarrow o^d.\, a^d{}_+ := v^d{}_+$

本文不对追加 *constriant* 集合进行读写操作，因为这会增加原来各个数据库中所没有的限制。

(3) $\{m^d : \{T_{e^d} : p^d\}\}\cup\{m_+^d : \{T_{e_+^d} : p_+^d\}\}$ 是C^d定义的*method*的集合。

$\{m^d : \{T_{p^d} : p^d\}\}$是$C^d$从$C^s$的$\{m^s : \{T_{p^s} : p^s\}\}$继承*method*的集合，切换操作定义为

$apply(o^d,m^d,\{p^d\})\Rightarrow\uparrow apply(o^s,m^s,fTa^s\to Ta^d(p^d)$

$\{m_+^d : \{T_{p_+^d} : p_+^d\}\}$是$C^d$追加的*method*的集合，其操作定义为

$apply(o^d,m_+{}^d,\{p_+{}^d\})$

对象代理模型提供了对象代理代数用于派生代理类，包括以下几种操作：Select，Project，Extend，Union 等。每个操作的结果是一个能被代理操作进行操作的代理类，这样就达到了与关系代数一样的灵活性。

文献[7]给出了数据仓库数据的提取算法，其思想是各个数据源的数据首先被物化到数据仓库中，然后通过各种操作符以虚拟视图的方式解决各种数据级别的冲突。本文也采用这种方式来实现模式的融合。

2 数据集成中的模式融合

2.1 模式映射

本文定义了映射规则来映射数据信息和模式信息。

定义 3(实体类) 一个实体类表示关系数据库的一个表。而实体类的每一个实例对应相应的关系表的一行，即一条记录。在关系表中的字段通过相应的实体类的实例变量来表示。通过映射规则可以将这些限制映射到实体类的限制。因此，实体类包含不同模型的模式限制。

定义 4(映射模型) 映射模型是在 Smalltalk 环境中访问数据库的核心，一个映射模型定义了在 OO 模型与关系型数据库模式之间的映射。映射模型规定了所有将要使用的实体类以及这些实体类是如何映射到关系表的，也规定了类和表之间的关系，这些关系可能是 1 : 1，1 : n 或者 m : n 的。

2.2 模式融合

2.2.1 主键-外键的融合

信息集成系统变得越来越复杂，特别是系统中复杂的元数据操作问题。主键-外键的融合是模式集成的重点。本文根据对象代理模型的特点，给出了如下的 M 算法：

```
Algorithm M(mapPair)/模式映射信息
Σ:={};
for{each pair of attributes(e1,e2)∈mapPair}do
k1:=primary key for e1;
k2:=foreign key for e2;
table1:=RelationOf (e1)  ⋈ …  ⋈ RelationOf(K1);
table2:=RelationOf(e2)  ⋈ …  ⋈RelationOf(K2);
Σ=Σ∪(πk1,e1(table1)=πk2,e2(table2);
return Σ
```

通过对象代理模型的Join操作[10]，可以实现主键-外键融合操作。例如：现在有两个表li和li1，表li的主键值等于表li1的外键值，要使两个表的模式融合，可用如下的操作：

```
JoinDeputyclass:#Li
select:{ Li.name:={li.name, li1.name}
…
Li.age:={li1.age};
from li, li1
where li.foreignkey=li1.key;
```

2.2.2 完整性融合

假设有两个表 priceOf(li,work)和 priceOf(li,work1)，通常的做法是对 priceOf(wang,work) ⇒priceOf(wang,work1)(两个 li 不是同一个人)进行重命名区分。但是在集成系统中有可能是一个人，他有一份兼职工作。对于这种情况，能用 Merge 操作实现多角色模式信息的融合。

定义 5(Merge 操作)

$C_1{}^s=<\{o_1{}^s\},\{T_{a1}{}^s: a_1{}^s: c_1{}^s\},\{m_1{}^s:\{T_{p1}{}^s:p_1{}^s\}\}>,...,$

$C_m{}^s=<\{o_m{}^s\},\{T_{am}{}^s: a_m{}^s: c_m{}^s\},\{m_m{}^s:\{T_{pm}{}^s:p_m{}^s\}\}>$为源类。

通过 Merge 操作派生的代理类 C^d=Merge($C_1{}^s$,…,$C_m{}^s$,*mmp*)。谓词*mmp*决定选择条件，$\{o^d|o^d\to\{o^s\}$,*mmp*$(\{o^s\})$==true$\}$。

(1)C^d的外延是 $C_1^s,...,C_m^s$ 实例的代理对象的集合，表示为 $\{o_1^d \left| o_1^d \to o_1^s\right.\}\cup...\cup\{o_m^d \left| o_m^d \to o_m^s\right.\}$。

(2) C^d的*attribute*集被定义为$\{T_a{}^d: a^d: c^d\}$，其从 $C_1^s,...,C_m^s$ 的共同*attribute* $\{T_{a^s} : a^s\}$继承。以$\{T_{a^d} : a^d\}$的形式继承 $\{T_{a^s} : a^s\}$ 的切换操作通过以下方式实现：

$read(o^d, a^d) \Rightarrow \uparrow fTa^s\to Ta^d\ (read(o^s, a^s))$

$write(o^d, a^d, v^d) \Rightarrow write(o^d, a^d, fTa^s \rightarrow Ta^d(v^d))$

$read(o^d, c^d) \Rightarrow \uparrow fTa^s \rightarrow Ta^d(read(o^s, c^s))$

$write(o^d, c^d, v^d) \Rightarrow write(o^s, c^s, fTa^s \rightarrow Ta^d((c_1.\min+c_2.\min) < c^d < (c_1.\max+c_2.\max))$

(3)C^d的*method*集被定义为$\{m^d : \{T_{p^d} : p^d\}\}$，其从$C^d$的共同*method* $\{m^s : \{T_{p^s} : p^s\}\}$继承。以$\{m^d : \{T_{p^d} : p^d\}\}$的形式从$\{m^s : \{T_{p^s} : p^s\}\}$继承的切换操作通过以下方式实现：

$apply(o^d, m^d, \{p^d\}) \Rightarrow \uparrow apply(o^s, m^s, fTa^s \rightarrow Ta^d(p^d))$

例如，通过读写操作可以很好地解决融合的问题：

read(d, price.c1) ⇒read (o, price.c1)

read(d, price.c2) ⇒read(o, price.c2)

Write(d, price.c) ⇒Write(o, (price.c1.min+price.c2.min) <price.c< (price.c1.max+price.c2.max))

3 系统实现

3.1 原型系统

本文的系统 Visualworks Smalltalk5i.4 建立在 Celetron 2.00 GHz CPU 和 256MB 内存的单机上；操作系统是 Windows2000。Oracle9i 和 Sybase11 分别安装在 3.00GHz CPU、1GB 内存的服务器上，操作系统是 Windows2000。Postgres7.2 装在另一台同样配置的服务器上，操作系统是 RedHat Linux 9.0。

3.2 系统应用实例

在银行贷款买房过程中，需要每个月的基本工资(定义在模式信息)和实际工资。每个月的基本工资之和是重要的参考指标，也就是模式信息的融合问题。可以用 Merge 操作得出其基本工资之和(如图 1)：

```
MergeDeputyClass:#Li
Select:{Li.name:={li.name,li1.name};
…
Li.salary.constraint:={li.salary.min+li1.salary.min<c<li.salary.max
+li1.salary.max};
from li, li1
where li.name=li1.name;
```

图 1 查询图

可以用决策树 ID3 算法挖掘出武汉地区贷款比例和兼职情况，从图 2 可以发现有兼职工作的人贷款比例较小。值得注意的是农民两项指标都很高，表明农民要想富裕就必须要兼职。

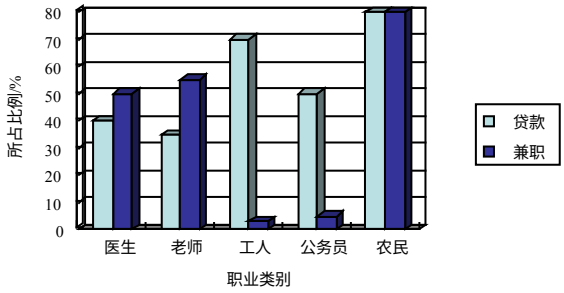

图 2 贷款和兼职关系

4 总结

模式融合问题是现今信息集成方案研究的重点和难点。对象代理模型能够很好地解决模式集成中的各种问题。本文主要对经济领域的数据进行集成和融合，今后的工作是建立预测数学模型，实现对生物数据的挖掘。

参考文献

1 Melnik S, Bernstein P A. Supporting Executable Mappings in Model Management[C]//Proc. of the 2005 ACM SIGMOD Int'l Conf. on Management of Data. 2005: 167-178.

2 Kim W, Choi I. On Resolving Schematic Heterogeneity in Multidatabase Systems[J]. Distributed and Parallel Databases, 1993, 1(3): 251-279.

3 Buneman P, Davidson S. Theoretical Aspects of Schema Merging[C]//Proceedings of the 3rd International Conference on Extending Database Technology: Advances in Database Technology, Vienna, Austria. 1992.

4 Melnik S, Rahm E. Rondo: A Programming Platform for Generic Model Management[C]//Proc. of the 2003 ACM SIGMOD Int'l Conf. on Management of Data. 2003: 124-134.

5 Hull R. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches[J]. SIGMOD Record, 1996, 25(2): 48-92.

6 Yang J, Karlapalem K. Algorithms for Materialized View Design in Data Warehousing Environment[C]//Proc. of Int'l Conf. on Very Large Data Bases. 1997: 137-146.

7 Peng Zhiyong, Li Qing. Using Object Deputy Model to Prepare Data for Data Warehousing[J]. IEEE Transaction on Knowledge and Data Engineering, 2005, 19(9): 124-135.

8 Peng Zhiyong, Kambayashi Y. Handling Conflicts and Replication During Integration of Multiple Databases by Object Deputy Model[C]//Proc. of the 20th Int'l Conf. on Conceptual Modeling. 2001: 85-98.

9 Karlapalem K, Li Qing. An Architectural Framework for Homogenizing Heterogeneous Legacy Databases[J]. SIGMOD Record,1995, 24(1): 15-20.

10 Peng Zhiyong, Kambayashi Y. Deputy Mechanisms for Object-oriented Databases[C]//Proc. of the 11th IEEE Int'l Conf. on Data Engineering. 1995: 33-40.