基于CWM的元数据集成中形式化推理技术的研究*)

赵晓非　黄志球
(南京航空航天大学计算机科学与工程系　南京 210016)

摘　要　在使用公共仓库元模型(CWM)进行元数据集成的过程中,参与集成的团体的不同经验以及描述数据的不同视角不可避免地带来元数据的冲突和冗余等问题,然而CWM的图形化特点使之缺乏精确的语义,所以如何在其上进行推理以自动发现这些问题至今没有得到很好的解决。本文研究了利用描述逻辑,一个一阶谓词逻辑的可判定子集,形式化CWM元模型和模型并进行推理的方法,将一致性检测分为水平一致性和演化一致性进行分别处理,在处理演化一致性的过程中对CWM元模型进行了扩展,使之支持元数据的版本能力从而能够推理由于演化引起的不一致问题,然后利用推理引擎LOOM对两种情形进行推理检测以发现不一致信息,取得了令人满意的结果,表明本文提出的方法是可行的。

关键词　公共仓库元模型(CWM),描述逻辑,元数据集成,元数据演化,水平一致性,演化一致性

Study on Formal Reasoning in Metadata Integration Based on CWM

ZHAO Xiao-Fei　HUANG ZHi-Qiu
(Department of Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016)

Abstract　During the metadata integration based on Common Warehouse Metamodel(CWM), the different experiences and views of describing data of organizations involved in integration bring metadata on some problems inevitably, such as inconsistencies and redundancies. However, reasoning on CWM metadata for detecting these problems automatically is difficult because CWM metamodel and metadata lack precise semantics. In this paper, an approach is proposed to formalize and reason on CWM metamodel and metadata in terms of a logic belonging to Description Logics, which are subsets of First-Order Logic. The approach distinguishes consistency into horizontal consistency and evolution consistency. To evolution consistency, extends CWM metamodel with version capabilities so that reasoning about inconsistency caused by evolution can be done. Then reasoning engine LOOM is applied to check consistency for above two situations, the results are encouraging.

Keywords　Common warehouse metamodel(CWM), Description logics, Metadata integration, Metadata evolution, Horizontal consistency, Evolution consistency

1　引言

如今,公共仓库元模型(CWM)[2,3]作为对象管理组织(OMG)提出的用于数据仓库领域的元数据集成标准,已经得到世界的公认,成为普遍接受和采用的标准。然而,在使用CWM建立元数据的过程中,由于:①建立元数据的不同团体拥有不同的经验和专业知识;②不同团体描述数据的侧重点不同;③元数据在建立过程中会不断演化;④CWM自身的特点,不可避免地会带来元数据的不一致问题,如元数据内容冲突或元数据内容违背元模型中的约束以及元数据演化过程中的误删造成的不一致等。因此,如果能在CWM元数据上进行推理,以发现甚至解决这些不一致的问题,能大大地提高元数据集成过程以及数据仓库系统的可靠性。

不幸的是,CWM元模型和作为元数据的模型都是以图形化的方式呈现给用户,而这种图形方式缺乏形式化的语义,因此如何有效地利用图形化的元模型提供的信息在元数据模型基础上进行推理成为一个棘手的问题,目前还没有很好的解决方案。为此,本文尝试采用逻辑学的方式来解决CWM元数据的形式化和推理问题。描述逻辑[1]作为一阶谓词逻辑的可判定子集提供了较强的描述能力,并配备了已实现的推理引擎如LOOM[13],RACER[10],Fact[19]等等,能够完成多种推理任务,成为我们的首选。

为了对CWM元数据进行全面的描述,本文中针对CWM元模型和模型的特点,采用了文[9]中提出的一种支持n元关系的描述逻辑,该描述逻辑对于CWM元模型和模型的结构化机制提供了很好的表现能力,能够为CWM元数据的描述和推理提供严格的形式化和推理框架。

相对于由于不同团体的不同视角等造成的不一致问题,与版本演化相关的不一致问题无论从成因上还是处理方式上都不同于前者,因此在本文中我们区分两种类型的一致性:水平一致性和演化一致性。由于目前CWM标准不支持元数据的版本能力,因此我们扩展了CWM元模型,使之能够支持元数据演化过程中轨迹信息的记录,然后分别研究了如何利用描述逻辑捕获两种情形的元模型和模型的结构化信息并利用LOOM的查询推理机制推理以发现不一致信息。

本文的结构如下:第2部分对我们采用的描述逻辑进行

*)航空科学基金(01I52058)资助。赵晓非　博士研究生,研究方向为数据仓库、语义Web、软件工程;黄志球　博士,教授,博士生导师,研究方向为数据库、数据仓库、软件工程。

了综述；第3、4部分分别研究了水平一致性检测和演化一致性检测中的形式化方法并分别举例了推理过程；第5部分介绍了相关工作；最后总结全文。

2 一种支持n元关系的描述逻辑

描述逻辑的基本元素是概念(concept)和角色(role)，分别用于描述领域中对象的类型和对象之间的关系，原子概念和原子角色可以通过构造算子组合成复杂概念和复杂角色。构造算子的集合决定了一种描述逻辑的表达能力。目前有多种描述逻辑可供选择。本文中针对CWM元模型和模型的特点，采用了Diego等[9]提出的一种支持n元关系的描述逻辑，简记为DL(n)。该描述逻辑是文[8]中提出的逻辑的一个变种。它的基本元素是概念(一元关系)和n元关系，用A和P分别表示原子概念和原子关系，任意概念C和任意关系R可以由以下规则建立：

$R::=\top_n \mid P \mid (i/n:C) \mid \neg R \mid R_1 \cap R_2$

$C::=\top_1 \mid A \mid \neg C \mid C_1 \cap C_2 \mid (\leqslant k[i]R)$

其中i表示关系R的第i个要素，是介于0和n_{max}之间的整数；n表示关系的元数，是介于2和n_{max}之间的整数；k是一个非负整数；$(i/n:C)$表示n元关系R相关联的第i个概念是概念C，如果n在上下文中很明确，则可简写为$(i:C)$；$\leqslant k[i]R$是对关系R的第i个要素相应于R的多重性约束。我们有以下约定：(1)只有元数相同的关系才能组成形如$R_1 \cap R_2$的表达式；(2)如果i表示n元关系的一个要素，则$i \leqslant n$。另外还有以下等价关系。

表1 DL(n)中的等价关系

$C_1 \cup C_2 \Leftrightarrow \neg(\neg C_1 \cap \neg C_2)$	$C_1 \Rightarrow C_2 \Leftrightarrow \neg C_1 \cup C_2$
$(\geqslant k[i]R) \Leftrightarrow (\leqslant k-1[i]R)$	$\exists[i]R \Leftrightarrow (\geqslant 1[i]R)$
$\forall[i]R \Leftrightarrow \neg\exists[i]\neg R$	

一个DL(n)知识库由TBox和ABox构成，其中TBox是描述领域结构的公理的集合，其中含有形如$R_1 \subseteq R_2$，$C_1 \subseteq C_2$的包含断言。除了包含断言，DL(n)还允许存在表达同一性约束的断言：

(id C $[i_1]R_1$, …, $[i_h]R_h$)

其中C是概念，每个R_j是一个关系，每个i_j表示R_j的一个要素。该约束用于表达如果概念C的两个实例都作为R_j的第i_j个要素关联到R_j，则二者是同一实例。

函数性依赖的断言：

(fd R i_1, …, $i_h \rightarrow j$)

其中R是关系，$h \geqslant 2$，$i_1, \cdots i_h$，j表示R的要素。该断言用于表达如果关系R的两个元组同时满足要素$i_1, \cdots i_h$，则也同时满足要素j。

由于一元函数性依赖(即$h=1$)能够导致推理的不可判定[9]，因此DL(n)不支持一元函数性依赖。另外尽管该约束的右半部分只包含单个要素j，然而也支持多个要素的情形，如果一个约束的右半部分包含多个要素，则它可被分解为多个右边为单一要素的约束。

DL(n)的ABox是描述具体情形的公理的集合，它由表示一个对象是否属于某个概念的概念断言和表示n个对象($n \geqslant 2$)是否满足一定关系的关系断言组成。

下面介绍DL(n)的语义，DL(n)知识库$\mathcal{K}$的一个解释$I=(\Delta^I, \cdot^I)$由一个解释的域Δ^I和一个解释函数$\cdot^I$构成，该函数将每个概念C映射为域Δ^I的一个子集C^I，将每个n元关系R映射为$(\Delta^I)^n$的一个子集R^I。更多的语义如表2所示。

表2 DL(n)的语义规则

$\top_n^I \subseteq (\Delta^I)^n$	$\top_1^I = \Delta^I$
$P^I \subseteq \top_n^I$	$A^I \subseteq \Delta^I$
$(i/n:C)^I = \{t \in \top_n^I \mid t[i] \in C^I\}$	$(\neg C)^I = \Delta^I \backslash C^I$
$(\neg R)^I = \top_n^I \backslash R^I$	$(C_1 \cap C_2)^I = C_1^I \cap C_2^I$
$(R_1 \cap R_2)^I = R_1^I \cap R_2^I$	$(\leqslant k[i]R)^I = \{a \in \Delta^I \mid \#\{t \in R_1^I \mid t[i]=a\} \leqslant k\}$

为了指定知识库的语义，我们进行如下定义：

(1)如果$R_1^I \subseteq R_2^I$($C_1^I \subseteq C_2^I$)，则解释I满足包含断言$R_1 \subseteq R_2$($C_1 \subseteq C_2$)；

(2)一个解释I满足断言(id C $[i_1]R_1$, …, $[i_h]R_h$)，如果对于所有的$a, b \in C^I$以及所有$t_1, s_1 \in R_1^I, \cdots, t_h, s_h \in R_h^I$，

有：$a=t_1[i_1]=\cdots=t_h[i_h]$，

$b=s_1[i_1]=\cdots=s_h[i_h]$，推出$a=b$

$t_j[i]=s_j[i], j \in \{1, \cdots, h\}, i \neq i_j$

(3)一个解释I满足断言(fd R i_1, …, $i_h \rightarrow j$)如果对于所有的$t, s \in R^I$，有$t[i_1]=s[i_1], \cdots, t[i_h]=s[i_h]$能推出$t[j]=s[j]$；

如果一个解释满足知识库$\mathcal{K}$中的所有断言，则称为$\mathcal{K}$的一个模型。

在DL(n)知识库上可进行几种推理操作，最常用的有知识库可满足性和逻辑蕴涵。一个知识库$\mathcal{K}$是可满足的如果$\mathcal{K}$存在一个模型。一个概念C是可满足的如果存在$\mathcal{K}$的一个模型使得C^I是非空。一个概念C_1被C_2所包含如果$C_1^I \subseteq C_2^I$对于$\mathcal{K}$的每个模型I都成立。一个断言a是逻辑蕴涵的如果$\mathcal{K}$的所有模型都满足a。

DL(n)配备了语义完备的推理算法。该算法使得以上提到的推理任务能够在可接受的有限时间内完成[9]。

3 水平一致性检测

本文区分了两种类型的一致性：水平一致性和演化一致性。水平一致性是指同一版本的元数据的不同部分之间的一致性；演化一致性指同一部分的元数据的不同版本之间的一致性。图1显示了两种类型的一致性之间的关系：水平面上的箭头描述了水平一致性，垂直面上的箭头描述了演化一致性。

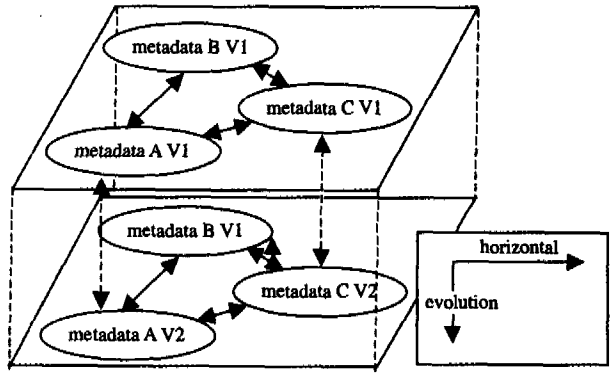

图1 水平一致性和演化一致性

3.1 水平一致性检测中的形式化方法

由于推理检测是利用元模型中显式表示的信息及推理所

得的隐含信息在作为元模型实例的元数据之上的推理，因此我们将元模型中的信息形式化为 Tbox 中的概念定义，而将作为实例的元数据形式化为 Abox。为了明确及简洁，以下均采用描述逻辑符号表示。

3.1.1 CWM 元模型的形式化

(1)元类

在 CWM 元模型中，元类也是一种类，因此在下文中我们不区分元类和类。元类被表示为图 2 所示的矩形。它被分为两部分：第一部分是类名；第二部分是类的属性，由属性名和一个相关联的类所表示，该类指示属性值的类型，例如属性 namespace：Namespace 表示属性 namespace 的值是 Namespace 类的实例。图中的"/"表明某个属性的类型属于 CWM 元模型中已存的元类(即属性所属的元类与属性类型的元类具有关联)。

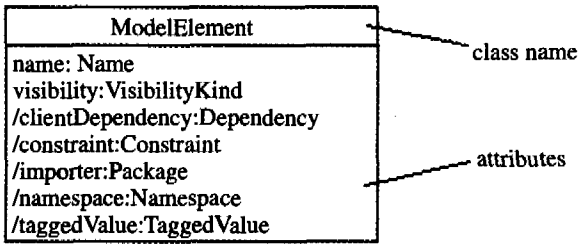

图 2 CWM 中的元类

由于 CWM 元类和 DL(n)概念都是用于描述实例的集合，因此我们很自然地想到用一个 DL(n)概念来表示一个 CWM 元类。

由于类 C 的一个类型为 C' 的属性 a 将 C 的每个实例关联到 C' 的实例，因此属性 a 可以认为是 C 的实例与 C' 的实例之间的二元关系，所以我们将属性 a 形式化为一个 DL(n)二元关系，此关系可以通过如下断言来表示：

$C \sqsubseteq \forall[1](a \Rightarrow (2:C'))$

该断言精确地指明了对于概念 C 的每个实例 c，所有通过 a 关联到 c 的对象都是 C' 的实例，并且反映了这样的语义：一个属性名在整个元模型中不必是唯一的，两个不同的元类可以有名字相同但是类型不同的属性(注意：尽管"/"后面的属性表示的都是 C 和 C' 之间的关联，然而不能通过关联的形式化代替属性的形式化，原因是 C 的一个属性可能对应 C 和 C' 之间的多个关联，因此可能会造成形式化过程中属性名信息的丢失)。

(2)聚合关联

CWM 元模型中的聚合关联如图 3(图中略去属性)所示，是两个元类的实例之间的二元关系，用于表明部分-整体的关系，例如 Classifier 元类与 Feature 元类之间的聚合关系表明每个 Classifier 的实例由一组 Feature 的实例组成。在 CWM 中聚合关联的名字是唯一的，即不能有两个聚合拥有同样的名字。

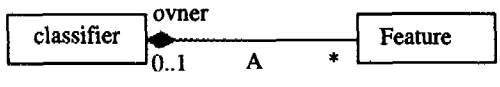

图 3 CWM 中的聚合关联

对聚合关联进行形式化的一般形式为，如果元类 C_1 通过聚合关系 A 聚合了 C_2，C_1 端的基数为 $m_1..m_2$，C_2 端的基数为 $n_1..n_2$，则将 A 形式化为 $DL(n)$的二元关系 A，在 Tbox 中添加断言：

$A \sqsubseteq (1:C_1) \sqcap (2:C_2)$

$C_1 \sqsubseteq (\geqslant n_1[1]A) \sqcap (\leqslant n_2[1]A)$

$C_2 \sqsubseteq (\geqslant m_1[2]A) \sqcap (\leqslant m_2[2]A)$

其中第二个式子表明了对于 C_1 的每个实例，C_2 的实例参与关系 A 至少 n_1 次，至多 n_2 次。聚合中的包含类与被包含类之间的区别并没有丢失，我们约定关系的第一部分是包含类。则图 3 中的聚合关联被形式化为：(Feature 端的基数 0..* 和 Classifier 端的 0 被省略)

$A \sqsubseteq (1:\text{Classifier}) \sqcap (2:\text{Feature})$

$\text{Feature} \sqsubseteq (\leqslant 1[2]A)$

我们并没有形式化角色名信息，如 Classifier 的角色名 owner，这是因为在实际推理中，利用代表 DL(n)关系的要素的缩写基本上已经够了，如果想保留角色名信息，可以采用定义角色名函数的方法，该函数对于给定的原子关系的每个要素，返回角色名集合中该要素对应的角色名，复杂关系的角色名函数可以通过构成它的原子关系的角色名函数变换而得。限于篇幅，本文略去此部分。

(3)一般关联

尽管目前的 CWM 版本不支持关联类的定义，然而每个一般关联在概念上仍有一个关联类与之对应，为了捕获一般关联的信息，在形式化中我们需要将每个一般关联形式化为一个 DL(n)概念。

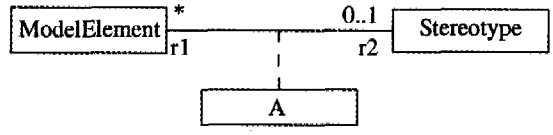

图 4 CWM 中的一般关联

例如图 4 中所示的一般关联可以形式化为一个概念 A 和两个二元关系 r_1，r_2，每个二元关系对应关联的一个要素。每个二元关系把 A 作为第一个要素，把 ModelElement 或 Stereotype 对应的概念作为第二个要素，因此有如下断言：

$A \sqsubseteq \exists[1]\, r_1 \sqcap \forall[1](r_1 \Rightarrow (2:\text{ModelElement})) \sqcap$

$\exists[1]\, r_2 \sqcap (\leqslant 1[1]\, r_2) \sqcap \forall[1](r_2 \Rightarrow (2:\text{Stereotype}))$

注意此处 r_1 和 r_2 的表示形式与聚合关联形式化所得的二元关系的表示形式有所不同，原因是与一般关联的要素对应的关系名仅仅相对于该关联是唯一的，在整个元模型中可能并不唯一。式中 $\exists[1]\, r_i (i \in \{1,2\})$表明概念 A 必须拥有关联 A 的所有要素 r_1，r_2；$\leqslant 1[1]\, r_2$ 表明相应的要素是单值的；$\forall[1](r_1 \Rightarrow (2:\text{ModelElement}))$表明 r_1 的第二个要素必须属于 ModelElement 类。此外，还要加上断言

$(\text{id}\ A\ [1]r_1, [1]r_2)$

指明概念 A 的每个实例表示相应关联的一个不同的元组。通过为 r_1 和 r_2 添加数量约束，我们可以形式化一般关联的多重性，然而由于与聚合关联不同之处是一般关联的要素对应的关系名在整个元模型中可能并不唯一，因此，形式化一般关联多重性的断言与聚合关联的情况略有不同，图 4 中的关联的多重性可形式化为：

$\text{ModelElement} \sqsubseteq (\geqslant 0\ [2](r_1 \sqcap (1:A))) \sqcap (\leqslant 1\ [2](r_1 \sqcap (1:A)))$

(4)泛化和继承

在 CWM 元模型中父类和子类之间的泛化表明子类的每个实例也是父类的实例。因此子类的实例继承了父类的属

性，但是它们还可以定义自己的属性。

泛化关系是被 DL(n)所支持的，如果一个 CWM 元类 Element 是 ModelElement 元类的泛化，我们可以将之形式化为 ModelElement ⊆ Element。

元模型中的继承关系可以通过 DL(n)中概念的继承关系来描述，这是由于"⊆"的语义是基于子集理论的。实际上，在 DL(n)中如果给定断言 $C_1 \subseteq C_2$，把 C_2 作为第 i 个要素的关系的每个元组也可以把 C_1 的实例作为第 i 个要素，因为它也是 C_2 的实例。因此在形式化中，C_2 的每个属性以及与 C_2 相关的每个聚合关联和一般关联都被 C_1 继承下来了。此外这种形式化方式也能完全捕获元类之间的多重继承关系。

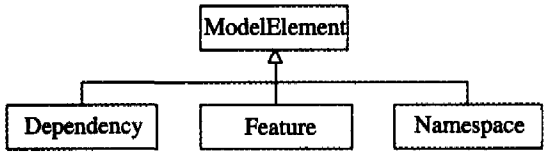

图 5　CWM 中的继承

图 5 所示的继承关系可以被形式化为：

Dependency ⊆ ModelElement

Feature ⊆ ModelElement

Namespace ⊆ ModelElement

3.1.2　*基于 CWM 的元数据的形式化*

CWM 元数据中的每个元素是元模型中相应元类的实例，元素之间的关系是元类之间相应关联的实例，因此 CWM 元数据必须被转化为 DL(n)知识库的 Abox。形式化的一般形式如下：

(1)如果元数据中元素 c 是元模型中元类 C 的实例，则可形式化为

$c : C$ 或 $C(c)$

(2)如果元数据中元素 c_1 以聚合的方式关联了 c_2，相应的元类 C_1(或其祖先)通过聚合关联 A 聚合了 C_2(或其祖先)，聚合关联 A 被形式化为 Tbox 中的二元关系 A，则可形式化为：

$\langle c_1, c_2 \rangle : A$

(3)如果元数据中元素 c_1 以非聚合的方式关联了 c_2，相应的元类 C_1(或其祖先)通过一般关联与元类 C_2(或其祖先)相联系，而该一般关联被形式化为概念 A 和二元关系 r_1, r_2，则 c_1 和 c_2 之间的关系可形式化为：

$a : A$

$\langle a, c_1 \rangle : r_1$

$\langle a, c_2 \rangle : r_2$

根据以上规则，图 6 所示的元数据可形式化为：

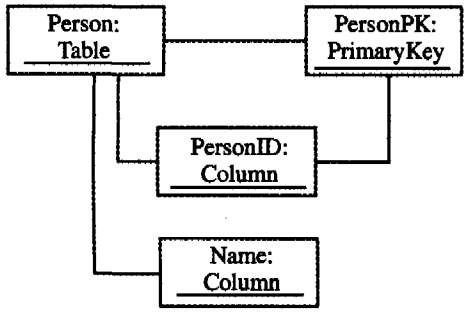

图 6　关系表的元数据

Person : Table

PersonID : Column

Name : Column

PersonPK : PrimaryKey

〈Person , PersonID〉 : ColumnSet-Column

〈Person , Name〉 : ColumnSet-Column

〈Person , PersonPK〉 : Table-PrimaryKey

ColumnVUniqueConstraint : ColumnVUniqueConstraint

〈ColumnVUniqueConstraint , PersonPK〉 : ColumnVUnique-Constraint-UniqueConstraint

〈ColumnVUniqueConstraint , PersonID〉 : ColumnVUnique-Constraint-Column

3.2　水平一致性检测推理举例

DL(n)知识库建立以后，利用推理工具的查询推理机制就可以对元数据进行查询及推理从而发现各种不一致信息。我们综合考虑了几种推理工具，由于 LOOM[13] 具有强表现力的概念定义语言，并提供了高效查询检索机制，其查询能力及产生式规则很适合于发现冲突信息，尽管它的分类算法不完备[18]，然而该算法在我们提出的知识库上是完备的，因此我们选择了 LOOM。以下举出利用 LOOM 发现约束被违背的例子。

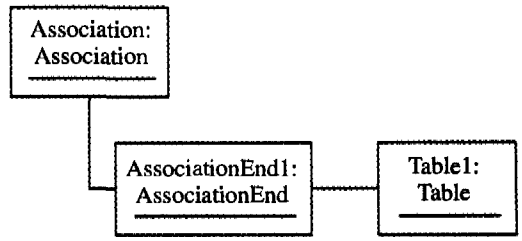

图 7　存在多重性冲突的元数据

图 7 所示为多重性约束被违背的情况，Association 元类的实例 Association 聚合关联了 AssociationEnd 的实例 AssociationEnd1，AssociationEnd1 与 Table 元类的实例 Table1 相关联。通过查阅元模型我们发现一个 Association 的实例通过 Association-AssociationEnd 关联至少聚合两个 AssociationEnd 的实例，而图中只聚合了一个，该元数据描述的实际情况可能是两个或更多个 Table 的实例相关联，因此多重性约束被违背，元数据出现了错误。

可以通过以下语句来发现多重性约束被违背的信息：

```
(defun multiplicity (? association ? metadata-A)
… //省略部分实现查询知识库，查询出? association 对应的元类 Association，以及 Association 通过聚合关联 Association-AssociationEnd 聚合了 AssociationEnd
(let* ((? count1 (length (retrieve (? associationEnd)
(:and
(Association-AssociationEnd ? association ? associationEnd)
(Namespace-ModelElement ? associationEnd ? metadata-A)))))
(? count2 (get-role-min-cardinality (get-concept 'Association)(get-relation 'Association-AssociationEnd))))
(if (< ? count1 ? count2)
(format t "Multiplicity conflict: ~S aggregates ~S elements, at least ~S is needed~%" ? association ? count1 ? count2))))
```

该函数根据给定的元数据范围和其中一个元素，通过检索 Tbox 获得元模型规定的与该元素相关的聚合关联的多重性下限，然后与检索 Abox 的结果比较，如果约束被违背，则用户被告知。利用该函数对我们人为引入图中所示错误的元数据进行检测，得出如下结果：

Multiplicity conflict: | 1 | ASSOCIATION aggregates 1 elements, at least 2 is needed

4 演化一致性检测

4.1 对 CWM 的扩展

为了捕获与元数据演化有关的推理所需的信息，需对 CWM 进行扩展。由于目前 CWM 并不支持元数据的版本信息，不同版本的元数据只能以不同文件的方式存放，元数据演化过程的信息很难被记录，因此必须扩展 CWM，使之能够支持演化信息的记录，从而使得维持演化过程中元数据的一致性成为可能。

我们扩展了 CWM 元模型，定义了 VersionedModel，CompositeModel，PrimitiveModel，Trace，HorizontalTrace，EvolutionTrace 六个元类，如图 8 所示。ModelElement，Model，Dependency 是核心包中的元类，为了简单起见，图中未画出 Model 和 ModelElement 之间的 Namespace 类和 Package 类，namespace 关联直接连到 Model 类上。Trace 类作为 Dependency 的子类用于记录模型元素的演化信息。为了记录元数据的版本、演化等信息，我们定义了 VersionedModel 类，它包括一个名-值对(version，Integer)用于指明元数据的版本信息。为了指明能够被水平一致性箭头和演化一致性箭头所关联的元数据的类型，VersionedModel 被分为两个子类 PrimitiveModel 和 CompositeModel，分别用于描述由单个团体建立的元数据和不同团体集成所得的元数据。CompositeModel 是 VersionedModel 的容器，即它可以包含属于同一版本的 PrimitiveModel。为了记录属于一个 CompositeModel 的元数据模型，我们引入一个名-值对(vmodel，Set(VersionedModel))，对于 VersionedModel，也需要一个名-值对(container，CompositeModel)，这两个名-值对在图 8 中都作为双向关联来表示。此外还有如下约束条件：

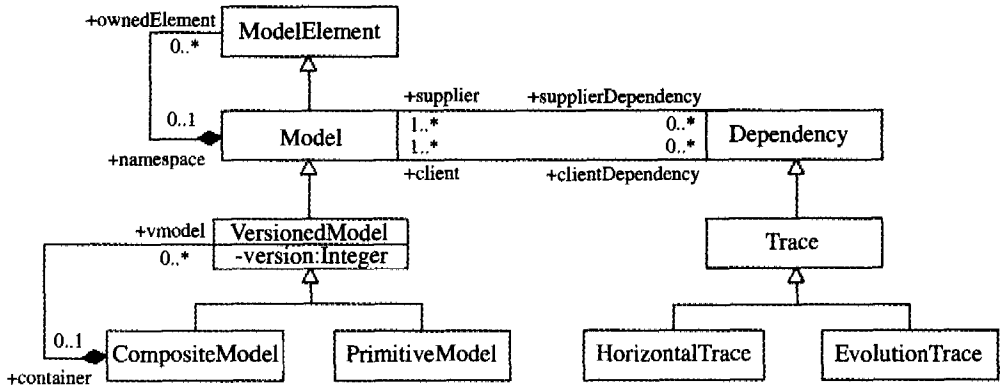

图 8 对 CWM 的扩展

(1)一个 CompositeModel 可以包含 0 个或多个 VersionedModels；

(2) HorizontalTrace 只能存在于属于同一 CompositeModel 的 PrimitiveModel 之间；

(3) EvolutionTrace 只能存在于同一 PrimitiveModel 或 CompositeModel 的不同版本之间；

(4)如果两个 models 通过一个 EvolutionTrace 相关联，则 client 模型的版本号必须大于 supplier 模型的版本号。

在使用扩展的元模型时，由不同团体建立的属于同一版本的元数据是 CompositeModel 的实例，元数据中的元素被作为名字空间的 CompositeModel 的实例所拥有，而 EvolutionTrace 和 HorizontalTrace 的实例通过 supplier 关联和 client 关联分别与同一元数据的不同版本相关联。

4.2 演化一致性检测中的形式化方法

演化一致性检测中元模型和模型的形式化过程需将 CWM 的扩展部分以及元数据中的相应部分添加进 Tbox 和 Abox 的定义中，除此以外与水平一致性检测中的相同，此处不再累述。

4.3 演化一致性检测推理举例

下面举出由于元数据演化造成的冲突的例子，如果某个关系表 Worker 拥有外键 WorkshopID，而以该外键作为主键的关系表 Workshop 已被删除或尚未被定义，则第一个关系表中的外键并不对应任何关系表的主键，造成冲突。

可以通过以下语句来发现上述冲突：

```
(defun foreignkey (? foreignkey ? metadata-v1)
(let * ((? count (length (retrieve (? table2)
(:and (Column ? column)
(Namespace-ModelElement ? column ? metadata-v1)
(ColumnSet-Column ? table1 ? column)
(Namespace-ModelElement ? table1 ? metadata-v1)
(Table-ForeignKey ? table1 ? foreignkey)
(ForeignKey-Column ? foreignkey ? column)
(Table ? table2)
(Namespace-ModelElement ? table2 ? metadata-v1)
(Table-PrimaryKey ? table2 ? primarykey)
(Namespace-ModelElement ? primarykey ? metadata-v1)
(UniqueConstraint-Column ? primarykey ? column))))))
(if (= ? count 0)
(do-retrieve(? metadata-v2)
(:and (Dependency ? dependency)
(Model-Dependency-c ? metadata-v1 ? dependency)
(Model-Dependency-s ? metadata-v2 ? dependency)
… //省略部分要求? table2 在? metadata-v2 中，拥有? column 作为主键，? table1 在? metadata-v2 中拥有? column 作为外键)
(format t "ForeignKey reference conflict: Foreign ~S of the
```

table ~S does not exist in ~S, while exists in ~S" ? column ? table1 ? metadata-v1 ? netadata-v2)))))

该函数根据给定的外键及其所在的元数据范围检索知识库，获得以该外键作为主键的表，如果不存在这样的表，则元数据出现冲突，然后根据演化轨迹在该元数据的相关版本中检索，如果成功，向用户输出有关冲突的信息。将该函数应用于我们人为引入例子中错误的元数据进行检测，得出如下结果：

ForeignKey reference conflict: Foreign | I | WORKSHOPID of the table | I | WORKER does not exist in | I | METADATA-A-V1, while exists in | I | METADATA-A-V2

5 相关工作

目前，利用CWM元模型提供信息推理元数据从而发现不一致信息的相关研究较少。Barton 等[4]和 Randall 等[5]阐述了元数据的建立和集成中可能遇到的不一致问题，并提出了解决策略。Finkelstein 等[16]详述了建立处理演化和一致性的工具时面临的一系列技术挑战，这些工具应该有助于建立、描述和推理形式化语言之间的关系，检测这些关系的一致性并提供诊断信息。Finkelstein 等[11]阐述了维持部分模型之间的完全的一致性不总是可能的，他们建议使用基于逻辑学的方法发现和处理不一致问题。在他们的研究中形式化机制被用于描述导致冲突的活动的顺序，而我们的方法是用逻辑来发现冲突信息。Grundy 等[14]阐述了支持冲突管理的关键需求是为开发者提供工具使他们能够配置冲突被发现、监视、存储和自动解决的时机及方式，文中介绍了他们建立支持冲突管理功能的多视角软件开发工具的实验。而我们的基于描述逻辑的方法也能够很容易地通过添加、删除或修改知识库中的逻辑规则被配置。Diego 等[8]利用支持 n 元关系的描述逻辑 DLReg 描述数据库模式并进行查询，他们详细论述了在约束之下查询包含的可判定性，他们在文[9]中扩展了DLReg，使之能够完全捕获概念之上的同一性约束和关系之上的函数性依赖，并证明了增加同一性约束和非一元函数性依赖之后推理过程是可判定的，以及在非二元关系上增加一元函数性依赖会导致推理的不可判定性，扩展后的描述逻辑可用于关系模型、实体-关系模型以及面向对象模型等多种数据模型的描述和推理。Simmonds[7]提出利用描述逻辑推理解决 UML 类图，顺序图，状态图之间的冲突的方法。Andy[6]和 Andrea 等[12]提出形式化并推理 UML 类图的方法，然而他们都没有有效利用元模型中提供的信息。此外，Francesco 等[17]对描述逻辑的描述能力和推理的计算复杂性之间的关系的相关研究进行了综述。Mens 等[20]深入研究了 UML 的扩展机制，并提出扩展 UML 元模型使之自动支持软件演化的策略。

结论 在本文中，我们提出了一种利用描述逻辑家族的一种特殊的形式逻辑形式化 CWM 元模型和基于 CWM 的元数据并进行推理以发现元数据中不一致信息的方法。该方法既可以自动发现与演化无关的不一致信息，也可以发现由于演化引起的不一致信息。该方法可用于开发支持自动推理 CWM 元数据的智能系统，从而提高元数据集成和数据仓库系统的稳定性。我们已经开始试验这样的系统，初步的试验结果是令人满意的。

未来的研究有以下几点：

(1)CWM 中存在一定数量的约束条件，用于以一种非形式化的方式来描述难以用元模型结构描述的信息，是以 OCL 的形式提供的，下一步我们将要研究形式化约束条件的可能性及方法。

(2)本文中用于检验该方法的元数据中的冲突信息是人为引入的，由于真实系统的元数据具有更自然的冲突情况和演化过程，因此下一步工作是在更真实的环境中检验我们的方法。

(3)该方法目前只研究了冲突信息的自动发现，下一步的工作是在发现冲突的基础上自动解决冲突。

参 考 文 献

1 Baader F, McGuinness D, Nardi D, et al. The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, 2003

2 Object Management Group. Common Warehouse Metamodel (CWM) Specification Version 1.1. November 2001

3 Poole J, Chang D, Tolbert D, et al. Common Warehouse Metamodel Developer's Guide. New York: John Wiley & Sons Inc, January 2003

4 Jane B, Sarah C, Hey Jessie M N. Building Quality Assurance into Metadata Creation: an Analysis based on the Learning Objects and e-Prints Communities of Practice. In: Proceedings 2003 Dublin Core Conference: Supporting Communities of Discourse and Practice - Metadata Research and Applications(DCMI), Seattle, Washington, 2003. 39~48

5 Hauch R, Miller A, Cardwell R. Information intelligence: metadata for information discovery, access, and integration. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of Data, 2005. 793~798

6 Evans A S. Reasoning with UML class diagrams. Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT98), 1998

7 Simmonds J. Consistency maintenance of uml models with description logic: [Master's thesis]. Vrije Universiteit Brussel, September 2003

8 Calvanese D, De Giacomo G, Lenzerini M. On the decidability of query containment under constraints. In: Proc of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems(POD'98), 1998. 149~158

9 Calvanese D, De Giacomo G, Lenzerini M. Identification constraints and functional dependencies in description logics. In: Proc. of the 17th Int Joint Conf. on Artificial Intelligence(IJCAI 2001), 2001

10 Haarslev V, Moller R. RACER system description. Proc of the Int Joint Conf on Automated Reasoning(IJCAR 2001), 2001

11 Finkelstein A, Gabbay D M, Hunter A, et al. Inconsistency handling in multi-perspective specifications. In: European Software Engineering Conference, 1993. 84~99

12 Cali A, Calvanese D, De Giacomo G, et al. A formal framework for reasoning on UML class diagrams. In: Proc. of the 13th Int Sym on Methodologies for Intelligent Systems (ISMIS'02), 2002. 503~513

13 Brill D. LOOM reference manual, version 2.0 edition. University of Southern California, Information Sciences Institute, December 1993

14 Grundy J C, Hosking J G, Mugridge W B. Inconsistency management for multiple-view software development environments. IEEE Transactions on Software Engineering, 1998, 24(11): 960~981

15 Object Management Group. Unified Modeling Language specification version 1.4. September 2001

16 Finkelstein A. A Foolish Consistency: Technical Challenges in Consistency Management. In: Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA), London, UK, September 2000

17 Donini F M, Lenzerini M, Nardi D, et al. Reasoning in description logics. In Principles of Knowledge Representation, Studies in Logic, Language and Information, 1996. 193~238

18 MacGregor R. Inside the LOOM description classifier. SIGART Bull, 1991, 2(3): 88~92

19 Horrocks I. FaCT and iFaCT. In: International Workshop on Description Logics(DL99), 1999. 133~135

20 Mens T, D'Hondt T. Automating support for software evolution in uml. Automated Software Engineering Journal, 2000, 7(1): 39~59

21 Miller J, Mukerji J. Model driven architecture (MDA). Draft ormsc/2001-07-01, Architecture Board ORMSC, July 2001