基于描述逻辑的CWM元数据冲突的检测和消解

赵晓非[1] 黄志球[2]
(南京信息工程大学计算机科学与技术系 南京 210044)[1]
(南京航空航天大学计算机科学与工程系 南京 210016)[2]

摘 要 元数据内容的冲突能够对数据仓库系统的稳定性和可靠性造成极大影响。在基于公共仓库元模型(CWM)建立元数据的过程中,参与建立元数据的团体的不同经验以及描述数据的不同视角不可避免地会带来元数据的这些冲突。然而,CWM的图形化特点导致了它缺乏精确的语义,所以如何自动检测和消解这些冲突,至今没有得到很好的解决。首先提出了一种支持概念之上的同一性约束的描述逻辑,用来对CWM元模型和元数据进行描述,然后阐述了利用该描述逻辑形式化CWM元模型和元数据的方法,接着研究了利用描述逻辑的查询推理能力检测元数据冲突的方法,最后研究了通过在知识库中定义冲突消解规则,消解元数据冲突的方法。利用推理引擎RACER所进行的实验结果表明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,冲突检测,冲突消解
中图法分类号 TP18,TP311.5 **文献标识码** A

Inconsistency Checking and Resolving of CWM Metadata Based on Description Logics

ZHAO Xiao-fei[1] HUANG Zhi-qiu[2]
(Department of Computer Science and Technology, Nanjing University of Information Science and Technology, Nanjing 210044, China)[1]
(Department of Computer Science and Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)[2]

Abstract The inconsistencies in metadata have remarkable influence on the stability and reliability of data warehouse system. During the metadata creation based on Common Warehouse Metamodel(CWM), the different experiences and views of describing data of organizations involved in metadata creation bring metadata inconsistencies inevitably. However, detecting and resolving these inconsistencies in CWM metadata automatically is difficult because CWM metamodel and metadata are rendered to users by graphs, which lack precise semantics. In this paper, we researched how to check and resolve CWM metadata inconsistencies in terms of a logic belonging to Description Logics. First, how to formalize CWM metamodel and metadata by means of the presented Description Logic DL_{id} which supports identification constraints on concepts was researched. Then the approach for checking metadata inconsistencies by query and reasoning mechanism of DL_{id} was researched. At last, the approach for resolving inconsistencies through defining inconsistency resolution rules in DL_{id} knowledge base was proposed. The results of the experiments with reasoning engine RACER are encouraging.

Keywords Common warehouse metamodel(CWM), Description logic, Metadata integration, Inconsistency checking, Inconsistency resolving

1 引言

如今,公共仓库元模型(CWM)[1]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;(4)CWM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,就能够为数据仓库系统各组件的开发提供支持,从而很大地提高元数据集成过程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的,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[2]作为一阶谓

到稿日期:2009-12-08 返修日期:2010-03-17 本文受国家自然科学基金(10402013)资助。
赵晓非(1978—),男,博士,主要研究方向为数据仓库、语义Web、软件工程,E-mail:zxf-first@nuaa.edu.cn;黄志球(1965—),男,教授,博士生导师,主要研究方向为数据库、数据仓库、软件工程。

词逻辑的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[3]，RACER[4]，Fact[5] 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，本文针对 CWM 元模型和元数据的特点，提出了一种支持概念之上的同一性约束的描述逻辑，简记为 DL_{id}。本描述逻辑对 CWM 元模型和元数据的结构化机制提供了很好的表现能力，并配备了可判定的推理步骤，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。通过将 CWM 元模型和元数据转换进 DL_{id} 知识库，我们研究了利用 DL_{id} 的查询推理能力检测元数据冲突的方法以及通过在知识库中定义冲突消解规则从而消解元数据冲突的方法。

2 描述逻辑 DL_{id}

描述逻辑的基本元素是概念(concept)和关系(relation)，分别用于描述领域中对象的类型和对象之间的关系。原子概念和原子关系可以通过构造算子组合成复杂概念和复杂关系。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。本文针对 CWM 元模型和元数据的特点，提出了一种支持概念之上的同一性约束的描述逻辑，简记为 DL_{id}。本描述逻辑可以看作是 Diego 等[12]提出的描述逻辑 DLR 的一个子集。它的基本元素是概念(一元关系)和角色(二元关系)。用 A 和 P 分别表示原子概念和原子角色，任意概念 C 和任意角色 R 可以由以下规则建立：

$$R::=\top_2 \mid P \mid (i/2:C) \mid \neg R \mid R_1 \sqcap R_2$$

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

式中，i 表示角色 R 的第 i 个要素，可取值为 1 或 2；k 是一个非负整数；$(i/2:C)$ 表示角色 R 相关联的第 i 个概念是概念 C，可简写为 $(i:C)$；$\leqslant k[i]R$ 是对角色 R 的第 i 个要素相应于 R 的多重性约束。

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

$$(id\ C[i_1]R_1,\cdots,[i_h]R_h)$$

式中，C 是概念，每个 R_j 是一个角色，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。

DL_{id} 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两个对象是否满足一定关系的角色断言组成。

下面介绍 DL_{id} 的语义。DL_{id} 知识库 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，将每个角色 R 映射为 $(\Delta^I)^2$ 的一个子集 R^I。更多的语义如表 1 所列。

表 1 DL_{id} 的语义规则

$\top_2^I \subseteq (\Delta^I)^2$	$\top_1^I = \Delta^I$
$P^I \subseteq \top_2^I$	$A^I \subseteq \Delta^I$
$(i/2:C)^I = \{t \in \top_2^I \mid t[i] \in C^I\}$	$(\neg C)^I = \Delta^I \setminus C^I$
$(\neg R)^I = \top_2^I \setminus R^I$	$(C_1 \sqcap C_2)^I = C_1^I \cap C_2^I$
$(R_1 \sqcap R_2)^I = R_1^I \cap R_2^I$	$(\leqslant k[i]R)^I = \{a \in \Delta^I \mid \#\{t \in R_1^I \mid t[i]=a\} \leqslant k\}$

为了指定知识库的语义，我们进行如下定义：

(i)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 $R_1 \sqsubseteq R_2$ ($C_1 \sqsubseteq C_2$)；

(ii)一个解释 I 满足断言 $(id\ C[i_1]R_1,\cdots,[i_h]R_h)$，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 $a=b$

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

如果一个解释满足知识库 K 中的所有断言，则称为 K 的一个模型。

在 DL_{id} 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 K 是可满足的，如果 K 存在一个模型。一个概念 C 是可满足的，如果存在 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 K 的每个模型 I 都成立。一个断言 a 是逻辑蕴涵的，如果 K 的所有模型都满足 a。

由于基本的描述逻辑 ALC[2] 中的推理是指数时间完全，而 DL_{id} 可以映射为 DLR[12] 的一个子集，在 DLR 中的推理也是指数时间完全，故 DL_{id} 中的推理是可判定的，且为指数时间完全。

3 CWM 元模型和元数据的形式化

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。

3.1 CWM 元模型的形式化

(1)元类

在 CWM 元模型中，元类也是一种类，因此在下面我们不区分元类和类。元类被表示成划分为两部分的矩形：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。

由于 CWM 元类和 DL_{id} 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_{id} 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_{id} 角色，该角色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。

(2)聚合关联

CWM 元模型中的聚合关联如图 1(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。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。

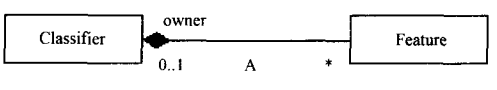

图 1 CWM 中的聚合关联

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 $m_1,\cdots,m_2$，C_2 端的基数为 $n_1,\cdots,n_2$，则将 A 形式化为 DL_{id} 的角色 A，在 Tbox 中添加断言：

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

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

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

第二个式子表明了对于 C_1 的每个实例，有至少 n_1 个、至多 n_2 个 C_2 的实例通过关系 A 与之相关联。聚合中的包含类与被包含类之间的区别并没有丢失，我们约定角色的第一部分是包含类，则图 1 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）

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

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

(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，在形式化中需要将每个一般关联形式化为一个 DL_{id} 概念。

例如，图 2 中所示的一般关联可以形式化为一个概念 A 和两个角色 r_1，r_2，每个角色对应关联的一个要素。每个角色把 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：

$A\sqsubseteq\exists[1]\,r_1\sqcap\forall[1](r_1\Rightarrow(2:\text{ModelElement}))\sqcap\exists[1]\,r_2\sqcap$

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

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 DL_{id} 角色的名字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 2 中的关联的多重性可形式化为

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

$(\leqslant 1\,[2](r_1\sqcap(1:A)))$

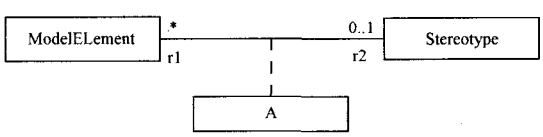

图 2　CWM 中的一般关联

(4)泛化和继承

泛化关系是被 DL_{id} 支持的。如果一个 CWM 元类 Element 是 ModelElement 元类的泛化，可以将之形式化为 ModelElement$\sqsubseteq$Element。

元模型中的继承关系可以通过 DL_{id} 中概念的继承关系来描述，这是由于“$\sqsubseteq$”的语义是基于子集理论的。实际上，在 DL_{id} 中如果给定断言 $C_1\sqsubseteq C_2$，把 C_2 作为第 i 个要素的角色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。

(5)约束

CWM 元模型中存在一定数量的约束条件，是以 OCL 的形式提供的，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。部分 OCL 约束可以被 DL_{id} 所捕获并且关于它们的推理是可判定的，如行为包中的 OCL 约束[C-4-4]：一个 Interface 只能够包含 Operation，可以形式化为：

$\text{Interface}\sqsubseteq\forall\,\text{Classifier}-\text{Feature.Operation}$

其它的 OCL 约束本质上对应于完全的一阶谓词逻辑公式，可能会导致推理的不可判定，因此我们不考虑这部分 OCL 约束。

3.2　CWM 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 元数据必须转化为 DL_{id} 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 中角色 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 和角色 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，如图 3 所示的元数据可形式化为：

Person :Table

PersonID :Column

Name :Column

PersonPK :PrimaryKey

〈Person,PersonID〉 :ColumnSet-Column

〈Person,Name〉 :ColumnSet-Column

〈Person,PersonPK〉 :Table-PrimaryKey

ColumnVUniqueConstraint :ColumnVUniqueConstraint

〈ColumnVUniqueConstraint,PersonPK〉 :ColumnVUniqueConstraint-UniqueConstraint

〈ColumnVUniqueConstraint,PersonID〉 :ColumnVUniqueConstraint-Column

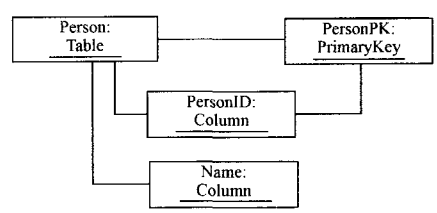

图 3　关系表的元数据

在推理引擎的选择上，我们综合考虑了市面上流行的几种工具 LOOM，CLASSIC，FACT 和 RACER。由于 LOOM 的推理算法不完备，而 FACT 不支持 Abox 和 Abox 推理，因此没有选择 LOOM 和 FACT。没有选择 CLASSIC 的原因是它所支持的描述逻辑的表达能力很弱，并且它的推理能力也

很有限。RACER 支持了强表达能力的描述逻辑，并且它为 Abox 之上的推理和检索提供了表达能力很强的查询语言，因此 RACER 成为我们的理想选择。

4 基于知识库查询语言的元数据冲突的检测

4.1 元数据冲突查询语言应满足的需求

DL_{nd}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。针对 CWM 元数据中冲突的特点和上述形式化方法，我们定义了一组需求。适合本课题需要的描述逻辑查询语言必须满足的这些需求。

查询语言应具有如下的查询格式：

$q(\vec{x}) \leftarrow body_1(\vec{x},\vec{y}_1,\vec{c}_1) \vee \cdots \vee body_n(\vec{x},\vec{y}_n,\vec{c}_n)$

其中应满足以下需求：

(1)查询体　查询体(即←后面的表达式)是 $body_i(\vec{x},\vec{y}_i,\vec{c}_i)$ 的析取。$body_i$ 是原子的合取。即查询体既可以包含析取，也可以包含合取。$\vec{y}_i$是体中出现的所有变量，$\vec{c}_i$是体中出现的常量。$\vec{x}$ 表示查询体中出现的变量和常量，并用于绑定查询的结果集合。

(2)概念查询原子　形如 $C(t)$的原子，其中 t 是属于$\vec{x}$，$\vec{y}_i$或$\vec{c}_i$的一个个体或变量。C 是一个概念，可以是一个原子概念或一个复杂概念。如原子 ForeignKey(x)表示原子概念 ForeignKey 的所有实例，这些实例将被绑定到变量 x。原子(Column⊓(→ForeignKey))(x)是一个复杂概念查询原子。

(3)角色查询原子　形如 $R(t,t')$的原子，其中 t 和 t'是属于$\vec{x}$，$\vec{y}_i$或$\vec{c}_i$的个体或变量。R 可以是原子角色、逆角色或传递角色。通过执行一个包含原子 ForeignKey-Column(x, column1)的查询(其中 column1 是一个常量)，并通过角色 ForeignKey-Column 关联到个体 column1 的个体将被绑定到变量 x。

(4)限定到明确提出的个体　RACER 是在开放世界假定之下进行推理的。在开放世界假定之下，一个角色的角色填充符的缺失并不被解释为它们不存在，这些角色填充符以后能被添加进 Abox。而在本课题中，我们查询 Abox 是想获得那些角色填充符断言被明确陈述的个体作为查询的结果。由此得出对该查询机制的一个需求是在被查询的 Abox 中，变量仅能被绑定到明确提出的 Abox 个体。

(5)真否定和失败即否定　开放世界假定支持真否定，例如，包含原子(→ForeignKey)(x)的查询将返回所有能够被证明为不是 ForeignKey 的个体作为结果集。然而，有时我们想要检索所有当前不知道是否是 ForeignKey 的个体，即推理引擎当前不能证明它们是 ForeignKey 的所有个体将被检索，这称为失败即否定。该查询语言应该支持失败即否定。

4.2 利用 nRQL 的元数据冲突查询

从 1.7.19 版以后，RACER 提供了表达能力很强的查询语言 nRQL[2]，它提供了对查询体和多种查询原子(一元概念查询原子、二元角色查询原子、二元约束查询原子、一元绑定个体原子、一元已知后继原子、否定原子)的支持，该查询语言满足上述所有需求：nRQL 支持查询体；一元概念查询原子对应于需求中的概念查询原子；二元角色查询原子对应于需求中的角色查询原子；已知后继原子仅检索明确提出的个体；否定原子满足拥有两种类型的否定即真否定和失败即否定的需求。因此，nRQL 是查询知识库，从而发现元数据冲突的理想工具。

以下举出检测外键引用冲突的例子。该冲突发生于一个关系表的外键引用的列在元数据的其它部分并不作为主键出现的情形。

为了检测外键引用冲突，可以定义下面两个 nRQL 查询。在执行这两个查询之前须给定外键 fk 及其所在的元数据范围 meta-context-1。第一个查询如下：

```
ans(table2)
  ← Column(col) ∧
    Namespace-ModelElement (meta-context-1,col) ∧
    ColumnSet-Column(table1,col) ∧
    Namespace-ModelElement(meta-context-1,table1) ∧
    Table-ForeignKey(table1,fk) ∧ Table(table1) ∧
    ForeignKey-Column(fk,col) ∧ Table(table2) ∧
    Namespace-ModelElement(meta-context-1,table2) ∧
    Table-PrimaryKey(table2,pk) ∧
    Namespace-ModelElement(meta-context-1,pk) ∧
    UniqueConstraint-Column(pk,col)
```

该查询根据给定的外键 fk 及其所在的元数据范围 meta-context-1 检索知识库，查找在 meta-context-1 中以该外键对应的列作为主键的表。其结果集合为 meta-context-1 中的以 fk 对应的列 col 作为主键的表 table2 构成的集合。如果 meta-context-1 中不存在这样的表，则结果集合为空。因此执行完该查询之后需判断结果集合是否为空，若为空则说明元数据中存在外键引用冲突。

执行第一个查询后若发现外键引用冲突，则执行如下的第二个查询：

```
ans(meta-context-2)
  ← Dependency(dep) ∧
    Model-Dependency-c(meta-context-1,dep) ∧
    Model-Dependency-s(meta-context-2,dep) ∧
    Namespace-ModelElement(meta-context-2,fk) ∧
    Table-ForeignKey(table1,fk) ∧ Table(table1) ∧
    Namespace-ModelElement(meta-context-2,table1) ∧
    ForeignKey-Column(fk,col) ∧ Column(col) ∧
    Namespace-ModelElement(meta-context-2,col) ∧
    ColumnSet-Column(table1,col) ∧ Table(table2) ∧
    Namespace-ModelElement(meta-context-2,table2) ∧
    Table-PrimaryKey(table2,pk) ∧
    Namespace-ModelElement(meta-context-2,pk) ∧
    UniqueConstraint-Column(pk,col)
```

该查询首先根据元数据的演化轨迹找到 meta-context-1 的相关版本 meta-context-2，然后查找在 meta-context-2 中是否有以该外键对应的列作为主键的表。如果成功，则返回所有满足条件的 meta-context-2 构成的集合，从而为追踪外键引用冲突产生的原因提供进一步的参考。

5 基于规则的元数据冲突的消解

5.1 采用基于规则的冲突消解方法的动机

在本课题中，我们借鉴了软件工程的冲突管理领域的研究思想，将如何消解冲突的问题称为冲突消解[10]，而将消解冲突的过程称为消解活动[10]。

我们采用了基于规则的冲突消解方法，主要基于以下几点原因：

(1)一个特定的冲突可以有多种消解方法，分别对应不同

的消解活动，选择哪一种消解活动依赖于冲突的成因。例如对于外键引用冲突，可能的消解活动有：(a)将外键引用的主键添加进元数据的相关部分；(b)将关系表的外键替换为元数据中已存在的主键；(c)删除列的外键属性。发生的可能原因有：外键引用的主键或其所在的关系表已被删除；或者该主键或其所在的关系表还未被加入到元数据中等等。冲突的成因不同，消解活动也不同。然而有时冲突的成因是很难判定的，因此在这些情况下还需要由用户决定执行哪些消解活动。而基于规则的推理将冲突的发现和消解活动封装在一起。每个冲突可以被不同的方法消解。冲突的消解方法的选择可以依赖于元数据的特定状态，也可以依靠消解者的主观判断，因此很好地解决了以上问题。

(2)如果发现一个特定的冲突，消解该冲突归结为改变该冲突涉及到的元类和元关联的实例。我们将消解活动分为3种原子消解活动：(a)添加元数据元素，即一个元类的实例化；(b)删除元数据元素，即某个元类的一个实例的删除；(c)改变元数据元素，即通过改变元数据元素的属性之一(即它所引用的其它元数据元素)，而改变该元数据元素。冲突消解机制必须允许上述消解活动组合。而基于规则的推理将冲突的发现和消解活动进行封装，并允许消解活动的自由组合。

(3)消解活动的执行可能会引起新的冲突。例如一个消解活动是将某一主键列的描述删除，而没有及时删除将该主键列作为外键的关系表的外键描述，就会产生外键引用冲突，即该冲突是由于其它冲突的消解活动的执行而产生的新的冲突。对于一个特定的冲突，它对应的不同消解活动可能导致的新的冲突是不同的。冲突消解机制必须能够处理由于消解活动执行而产生的新的冲突的情形。在非基于规则的编程语言中，考虑冲突之间的依赖将会使程序变得非常复杂并且难以维护。而在基于规则的系统中，规则仅需定义一次并且可以反复激活，因此基于规则的方法为规则的重用提供了很好的支持，从而解决了上述问题。

5.2 基于规则的元数据冲突消解机制应满足的需求

针对CWM元数据中冲突的特点和上述形式化方法及冲突检测方法，下面提出用于元数据冲突消解的描述逻辑规则语言应满足的需求。

冲突消解规则应该具有如下格式：

IF 冲突X出现在元数据M中

THEN 改变元数据M以便X被消解

对于一个特定的冲突来说通常有多种消解方法，每种方法由一个规则给出。因此，和某一冲突X相关的所有规则在条件中拥有相同的表达式：冲突X出现在元数据M中。

此外，还应满足以下需求：

(1)规则的条件是Abox查询和用户输入提示的合集。Abox查询用于发现冲突并检索额外的信息以便消解冲突。用户输入提示有两个作用：如果对于一个冲突有多个规则被激活，则它们让用户选择哪种消解方法；如果需要，则它们提示用户附加的输入。

(2)规则的结论是一系列Abox断言，它们对应一个或一组原子消解活动。这些断言消解已发现的冲突，并且仅仅可以使用在规则的条件中返回的变量或常量。

(3)规则的工作是基于Abox中明确陈述的知识。

5.3 利用nRQL规则的元数据冲突消解

RACER提供的nRQL不仅是一种查询语言，也提供了规则机制，允许用户对Abox进行简单扩充。

nRQL规则拥有条件和结论。条件是一个nRQL查询体，即第4部分介绍的Abox查询体。nRQL规则的结论是一组RACER Abox断言，这些断言可以引用在规则的前提中返回的变量，对它们进行删除和修改，也可以建立新的Abox个体，从而对Abox进行扩充。nRQL规则仅考虑Abox中明确陈述的知识。尽管nRQL规则的条件不允许包含提示用户输入的表达式，但我们可以采用其它方法来替代，即将描述用户选择的特定个体人为植入到Abox中，然后通过查询来获得用户的选择。经过这样处理，nRQL符合上述所有需求，可以利用nRQL来消解元数据冲突。

可以用下面的nRQL规则来消解外键引用冲突，采用的消解方法是删除列的外键属性：

```
(firerule
  (and (check-FKRI ? table ? fk ? col)
    (? u user-option-deletefk)
  )
  ((forget-role-assertion ? table ? fk Table-ForeignKey)
  (forget-role-assertion ? fk ? col ForeignKey-Column)
  (forget-concept-assertion ? fk ForeignKey))
)
```

其中，and表达式是规则的条件，而后3个表达式是规则的结论，它包含3个原子消解活动的组合。

6 相关工作

Thomas等[8]和Randall等[9]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[10]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，他们建议使用基于逻辑学的方法来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和消解冲突信息。Richard等[11]讨论了在模型驱动工程中对模型进行一致性检测的各种方法和思想。Diego等[6]提出了支持n元关系的描述逻辑DLR_{reg}，以便对关系模型、实体-关系模型以及面向对象模型等多种数据模型进行描述和推理。在文献[12]中，对DLR_{reg}的内部机制进行了进一步的分析，并讨论了在该描述逻辑中推理的可判定性问题和计算复杂度。Simmonds[13]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。Jordi[14]和Ken等[15]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，Franz等[16]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。

结束语 本文研究了利用所提出的支持概念之上的同一性约束的描述逻辑描述CWM元模型和元数据的方法，并提出了利用此描述逻辑的查询推理能力检测元数据冲突以及通过定义冲突消解规则从而消解元数据冲突的方法。本文中的方法可用于开发支持自动推理CWM元数据的智能系统，从而为数据仓库系统各组件的开发提供支持，以提高元数据集成和数据仓库系统的稳定性。利用推理引擎RACER所进行的实验的结果是令人满意的。

未来的研究有以下几点：

(1)目前我们对于OCL的形式化仅进行了初步的研究。由于OCL约束中有相当一部分需要用完全的一阶逻辑公式来表达，它们将导致推理不可判定，因此下一步的研究是区分出

不会导致不可判定推理的 OCL 约束并研究其形式化的方法。

(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)对于每种特定的冲突,都有多种可能的消解方法。如何有效地管理多种冲突以及它们的消解活动,是下一步要研究的问题。

参 考 文 献

[1] Object Management Group. Common Warehouse Metamodel (CWM) Specification Version 1.1[M]. 2003

[2] Baader F, Calvanese D, McGuinness D, et al. The description logic handbook: Theory, implementation, and applications (2nd edition)[M]. Cambridge: Cambridge University Press, 2007

[3] University of Southern California. Loom Knowledge Representation System 4.0[M]. 2004

[4] Haarslev V, Moller R, Wessel M. RacerPro User's Guide Version 2.0[M]. 2009

[5] Horrocks I. FaCT and iFaCT[C]// Proceedings of the 1999 International Workshop on Description Logics(DL99). 1999: 133-135

[6] Calvanese D, De Giacomo G, Lenzerini M. Identification constraints and functional dependencies in description logics[C]// Proceedings of the 17th International Joint Conference on Artificial Intelligence(IJCAI 2001). 2001

[7] MacGregor R. Inside the LOOM description classifier [J]. SIGART Bull, 1991, 2(3): 88-92

[8] Margaritopoulos T, Margaritopoulos M, Mavridis I, et al. A Conceptual Framework for Metadata Quality Assessment[C]// Proceedings of the 2008 Dublin Core and Metadata Applications Conference: Metadata for Semantic and Social Applications(DCMI). Berlin: Universitatsverlag Gottingen, 2008: 104-113

[9] Hauch R, Miller A, Cardwell R. Information intelligence: metadata for information discovery, access, and integration[C]// Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2005: 793-798

[10] Finkelstein A. A Foolish Consistency: Technical Challenges in Consistency Management[C]// Proceedings of the 11th International Conference on Database and Expert Systems Applications (DEXA). Heidelberg: Springer, 2000: 1-5

[11] Paige R F, Brooke P J, Ostroff J S. Metamodel-based Model Conformance and Multi-view Consistency Checking[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2007, 16(3): 1-48

[12] Calvanese D, De Giacomo G. Expressive description logics[A]// The Description Logic Handbook: Theory, Implementation and Applications [C]. Cambridge: Cambridge University Press, 2003: 178-218

[13] Simmonds J. Consistency maintenance of UML models with description logic[D]. Brussel: Vrije Universiteit Brussel, 2003

[14] Cabot J, Clariso R, Riera D. Verification of UML/OCL Class Diagrams Using Constraint Programming[C]// Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation. New York: IEEE Press, 2008: 73-80

[15] Satoh K, Kaneiwa K, Uno T. Contradiction Finding and Minimal Recovery for UML Class Diagrams[C]// Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering. New York: IEEE Press, 2006: 277-280

[16] Baader F, Lutz C. Description Logic [A]// The Handbook of Modal Logic[C]. Elsevier, 2006: 757-820

[17] Motik B, Grau B C, Horrocks I, et al. Representing Structured Objects Using Description Graphs[C]// Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008). California: AAAI Press, 2008: 296-306

(上接第 144 页)

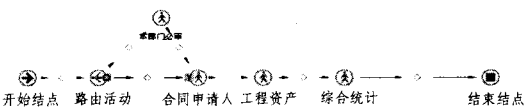

图 4 自动生成的分公司收验货工作流模板

结束语 本文研究了工作流流程定义的复用问题,提出了流程相似的判定依据,继而提出了流程相似性判定方法。本方法对流程进行结构和信息相似判定后自动生成工作流模板和表单。经测试,本方法运行的结果同专家手工创建的结果较为接近,证明流程相似性判定方法是有效的。在后续研究中,我们可以扩充流程定义的本体模型,细化流程相似的判定依据,以提高流程相似判定方法的准确率。

参 考 文 献

[1] 岳晓丽,杨斌,郝克刚. 信牌驱动式工作流计算模型[J]. 计算机研究与发展,2000,37(12):1513-1519

[2] van der Aalst W M P, Medeiros A K A, Weijters A J M M. Process Equivalence: Comparing Two Process Models Based on Observed Behavior[C]// International Conference on BPM'06. Lecture Notes in Computer Science, 2006(4102): 129-144

[3] Coalition W M. Workflow process definition interface-xml process definition language[R]. WFMC-TC-1025. Workflow Management Coalition, 2002

[4] van der Aalst W M P, Basten T. Inheritance of workflows: an approach to tackling problems related to change[J]. Theoretical Computer Science, 2002(270): 125-203

[5] 龚晓庆,刘锋,葛玮,等. 基于复用的工作流过程定义工具——PDTBR[J]. 计算机应用,2009,20(1):315-318

[6] Goderis A, Li P, Goble C. Workflow discovery: the problem, a case study from e-Science and a graph-based solution[C]// 2006 IEEE International Conference on Web Services. 2006: 312-319

[7] 张少华,向勇,沈浴竹,等. 网格工作流生成中的一种流程执行机制[J]. 通信学报,2008,29(6):43-50

[8] 韩韧,黄永忠,刘振林,等. OWL 本体构建方法的研究[J]. 计算机工程与设计,2008,29(6):1397-1400

[9] Toran J. On the hardness of graph isomorphism[C]// Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS). Redondo Beach, CA, Nov 2000: 180-186

[10] 李锋,商慧亮. 有向图的同构判定算法:出入度序列法[J]. 应用科学学报,2002,20(3):258-262

[11] 刘群,李素建. 基于知网的词汇语义相似度的计算[J]. 计算机语言学及中文信息处理,2002(7):59-76

[12] 徐德智,王怀民. 基于本体的概念间语义相似度计算方法研究[J]. 计算机工程与应用,2007,43(8):154-156

[13] 陶望龙,邵新宇,张国军,等. Web 环境下基于表单的工作流管理系统研究[J]. 计算机应用研究,2003,12:55-57