

# MINI——一种可减小变更影响范围的本体演化算法

刘 晨<sup>1),2)</sup> 韩燕波<sup>1)</sup> 陈旺虎<sup>1),2)</sup> 王建武<sup>1),2)</sup>

<sup>1)</sup>(中国科学院计算技术研究所网络与服务计算研究中心 北京 100190)

<sup>2)</sup>(中国科学院研究生院 北京 100049)

**摘 要** 本体演化会影响依赖本体的服务,使其重新修订和重新部署.面对同一变更需求,不同演化实现方法造成的影响范围差别很大.当前的本体演化研究主要集中在如何实现变更需求以及维护变更前后本体的一致性,对于如何降低演化影响范围关注甚少.文中提出了一种可以有效减小变更影响范围的本体演化算法 MINI.该算法首先分析了本体实体和服务之间的依赖关系并提出了量化变更影响范围的数学公式.根据这一公式,MINI算法将本体演化过程转变为图的启发式搜索过程,通过搜索一条影响值最小的变更路径来减小本体演化的影响范围.实验结果表明,MINI算法导致的平均变更影响范围大大小于现有的本体演化算法.该算法已在某实际项目中得以应用和验证.

**关键词** 本体演化;本体变更;变更路径;本体一致性;影响范围

中图法分类号 TP311

## MINI: An Ontology Evolution Algorithm for Reducing Impact Ranges

LIU Chen<sup>1),2)</sup> HAN Yan-Bo<sup>1)</sup> CHEN Wang-Hu<sup>1),2)</sup> WANG Jian-Wu<sup>1),2)</sup>

<sup>1)</sup>(Research Center for Grid and Service Computing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

<sup>2)</sup>(Graduate University of Chinese Academy of Sciences, Beijing 100049)

**Abstract** Ontology evolution is apt to impact on dependent services and cause them to be redeveloped and redeployed. However, different realizing methods for the same change requirements may result in very big different impact ranges. Today, researches of ontology evolution mainly focus on how to satisfy the change requirements and maintain the consistency of an evolving ontology. Unfortunately, few of them care about how to reduce impact ranges of ontology evolution. This paper proposes an ontology evolution algorithm called MINI, which can effectively reduce impact ranges for an evolution process. Firstly, through deeply analyzing dependent relations among ontology entities and services, the MINI algorithm establishes a math formula to quantify impact ranges. Based on this formula, the MINI algorithm transforms an ontology evolution process into a heuristic graph searching process. Through searching an ontology change path which has minimal impact value, the impact range of an evolution process is greatly reduced. The experiment results show that the average impact ranges caused by the MINI algorithm is greatly less than those of other evolution algorithms. In the context of some practical applications, MINI algorithm has been tested and evaluated.

**Keywords** ontology evolution; ontology change; change path; ontology consistency; impact range

收稿日期:2007-02-05;最终修改稿收到日期:2007-12-13. 本课题得到国家科技基础条件平台子项目“网络科技信息资源整合示范”(2005DKA64201)、国家自然科学基金(60573117)与国家“九七三”重点基础研究发展规划项目基金(2007CB310805)资助. 刘 晨,男,1980年生,博士研究生,研究方向为本体集成与本体演化. E-mail: liuchen@software.ict.ac.cn. 韩燕波,男,1962年生,教授,博士生导师,主要研究领域为面向服务的网格计算与软件集成. 陈旺虎,男,1973年生,博士研究生,研究方向为服务匹配与服务发现. 王建武,男,1980年生,博士研究生,研究方向为业务服务.

# 1 引 言

近年来,许多应用系统都采用本体作为语义支撑来满足不断增长的知识交换和知识集成需求.然而环境的变化和业务需求的多变性常常会引发支撑本体的演化<sup>[1]</sup>,这会影响依赖本体服务的正常运行.下面以一个实际项目为例来说明本体演化对服务的影响.某科技部门要打造中央级的科技资源集成门户,允许各地方科技厅的资源发布单位自主加入资源共享.本体的构建(本体片断见图 1)是实现该门

户的一个核心任务.依赖本体,平台可以以统一的方式接入和描述资源提供者提供的各类服务.由于事先不能确定加入共享的资源提供者,使得无法事先构建出一个能满足所有需求的全局本体.也就是说资源集成门户的本体会随着资源提供者的加入而不断演化<sup>[1]</sup>.但是,本体演化会对已注册的平台服务带来极大影响,每一次的本体演化都可能会引起已注册平台服务的重新修订和重新部署.因此,如何减轻本体演化对平台服务的影响就成为实现资源集成门户的关键.

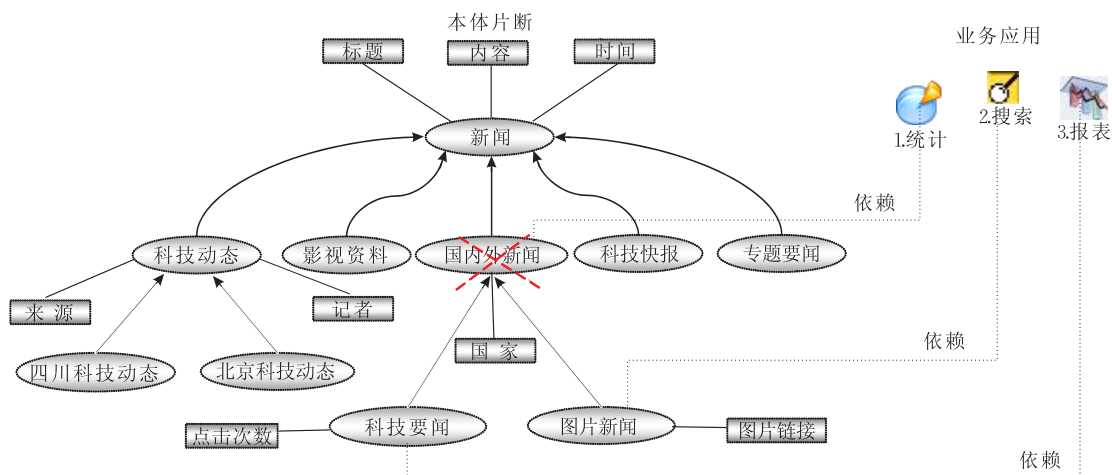


图 1 本体演化示例

从上面的例子分析得出,降低本体演化的影响可以从两个角度进行:(1)降低本体演化的影响范围,即当本体演化后受到波及的服务个数越少越好.(2)减轻服务的受影响程度,即如果某个服务不可避免的受到波及,那么是否可以控制本体演化的过程使其朝着有利于服务修订的方向进行,使其变得尽可能的简单易行.其中需求(2)的满足依赖于服务的具体实现方式,难以提出一个通用的算法.本文主要从满足需求(1)的角度出发来探讨如何降低本体演化的影响.

事实上对一个变更需求通常存在多种实现方法,不同实现方法所产生的影响范围差别很大.如图 1 所示,删除概念国内外新闻会导致概念科技要闻和图片新闻由于缺少父概念而处于不一致状态.为此,演化系统需要产生额外的附加变更(如删除概念科技要闻和图片新闻)使本体变为一致状态,这就导致了变更执行路径 1(删除概念国内外新闻→删除属性国家→删除概念国内外新闻和科技要闻间继承关系→删除概念科技要闻→删除属性点击次数→删除概念国内外新闻和图片新闻间继承关系→删除概念图

片新闻→删除属性图片链接)的产生.除此之外,还存在其它的演化实现方法.比如也可以将概念科技要闻和图片新闻变为新闻的子概念来使本体达到一致状态,这导致变更执行路径 2 的产生.显然这两条路径的影响范围却是完全不同的.如图 1 所示,服务 1、服务 2 和服务 3 分别依赖概念国内外新闻、图片新闻和科技要闻.因此,在执行路径 1 时,概念科技要闻和图片新闻被删除将会导致服务 1、服务 2 和服务 3 都需要发生变更.但是路径 2 在执行时保留了科技要闻和图片新闻,这使得服务 2 和服务 3 不需要做任何调整就能正常运行.显然路径 1 的影响范围要远大于路径 2 的影响范围.

当前本体演化研究<sup>[2]</sup>的焦点在于保证演化需求的实现以及维护演化前后本体的一致性,对于降低演化的影响关注甚少.但是,演化影响的降低不但可以显著减轻依赖本体应用系统的开发和维护的负担,还可以大幅增加系统的健壮性和灵活性.为此,本文提出了 MINI(Minimize Impacted Business Services)演化算法在保障本体一致性的前提下通过尽量缩小变更影响范围来降低本体演化的影响.

本文第 2 节将介绍当前本体演化方面的相关工作,指出它们的不足,明确 MINI 算法的改进方向;第 3 节将对 MINI 算法所依赖的本体模型及其一致性给出一个明确的定义;第 4 节将详述 MINI 支持的变更操作及变更影响的量化方法;第 5 节给出 MINI 算法的设计思想和实现步骤;第 6 节将对 MINI 算法进行理论探讨并给出实验评价;第 7 节形成结论,明确 MINI 算法的价值与贡献。

## 2 相关工作

当前,流行的本体编辑工具如 Protégé<sup>[3]</sup>、OntoEdit<sup>[4]</sup>、OilEd<sup>[5]</sup> 等均会提供执行本体变更的能力。文献[6]对这些工具进行了分析,指出它们所提供的变更执行能力还较为薄弱。这些本体编辑工具的变更执行策略通常较为简单,它们一般会为每一个支持的变更类型预定义一个执行策略,而且这个执行策略追求的往往只是逻辑上的简单性。这意味着变更在执行时不会考虑造成的影响等复杂因素,使得得到的结果往往并非是最优的。例如对于图 1 中所示的删除“国内外新闻”概念这一变更,绝大多数本体编辑工具都会选择删除所有子概念来保证变更后本体的一致性。这种解决方案虽然实现简单,但它通常意味着无法得到最好的结果。根据本章前面的分析,可以看出将“国内外新闻”的子概念变为其祖先的子概念可以有效减轻变更造成的影响,从而得到更好的变更结果。

Stojanovic 和 Maedche 等人在 2002 年提出了一种用户驱动的变更执行策略,并将该策略实现在 KAON 演化工具<sup>[7]</sup>中。该策略需要领域专家自始至终参与整个变更执行过程。在每一个变更执行前,一个变更的影响<sup>①</sup>列表被生成并同时被呈现给领域专家。领域专家需要理解这个列表的含义并确定是否执行或取消这个变更。该策略的缺陷在于过分依赖人工,这会使变更执行过程变得极为耗时而且容易出错。当本体规模较大时,我们不能假定领域专家能够完全理解整个本体及其组成元素间的相互关系。而且当出现分歧时,不同领域专家由于知识背景和认识的不同可能会做出不同的决策,这使得本体变更的结果是不可预知的。此外,过于依赖领域专家会给我们带来极大的工作负担,增大变更成本。本文提出的 MINI 算法可以回避这些缺陷,避免领域专家参与每一步的变更执行过程。领域专家仅需在执行过程开始前定义出变更需求和约束条件。变更执行过

程将会自动完成,领域专家仅需对最终结果进行审核和修改。这可以极大减轻领域专家的负担。

Stojanovic 和 Maedche 等人在 2003 年又从另一个视角审视了演化问题<sup>[8]</sup>。他们将变更执行过程描述为一个组件领域的重配置设计(Reconfiguration-Design)问题。他们关注两个核心问题。首先是变更需求的表达。他们将变更需求描述为做什么及不做什么两个组成部分。其次是变更需求的实现。他们将整个变更执行过程视为一个图的搜索过程,通过搜索一条包含变更需求并且保证一致性的执行路径来实现变更。本文提出的 MINI 算法受到了这个工作的启发。与这个工作相比,MINI 算法的贡献主要有以下两点。首先 MINI 算法分析了本体实体和服务之间的依赖关系并提出了量化变更影响范围的数学公式。其次,MINI 算法提出了新的启发搜索算子,通过搜索一条变更影响值最小的执行路径来减轻变更执行过程对现有服务造成的影响。此外,在搜索过程中,回溯的思想被引入以有效减少搜索的结点空间。

## 3 本体模型及一致性定义

本体演化的任务是根据变更需求执行相应的变更操作,并同时保证演化前后本体的一致性<sup>[9]</sup>。本节将首先介绍本文所依赖的本体模型和一致性定义。

### 3.1 本体模型

借鉴文献[9],本文所依赖的本体模型被定义为如下所示的一个二元组。其中,实体集合被描述为如定义 2 所示的 12 元组。

**定义 1.** 本体模型被定义为一个二元组:  
 $OntModel := (E, Imp)$ , 其中:

$E$ : 本体的实体集合;

$Imp$ : 导入的其它本体模型。

**定义 2.** 本体的实体集合  $E$  被定义为一个 12 元组: $E := (C, P, I, Hc, Hp, label, domain, range, mincard, maxcard, instconc, instprop)$ , 其中:

$C$ : 本体的概念集合。

$P$ : 本体的属性集合。属性集合还可以进一步划分为数据属性集合  $DP$  和对象属性集合  $OP$ 。数据属性刻画概念的性质,而对象属性刻画概念之间的关系。

$I$ : 本体的实例集合。

① 这里的影响不是指对服务的影响,而是指对本体内部其它元素的影响。

$Hc \subseteq C \times C$ : 概念的继承体系. 如果  $(c_1, c_2) \in Hc$ , 那么  $c_1$  是  $c_2$  的子概念,  $c_2$  是  $c_1$  的父概念.  $Hc^*$  是  $Hc$  的闭包, 它是自反、反对称和传递的.

$Hp \subseteq P \times P$ : 属性的继承体系. 如果  $(p_1, p_2) \in Hp$ , 那么  $p_1$  是  $p_2$  的子属性,  $p_2$  是  $p_1$  的父属性.  $Hp^*$  是  $Hp$  的闭包, 它是自反, 反对称和传递的.

函数  $label: E \rightarrow Literal$  获得实体的标签,  $Literal$  为文本的集合.

函数  $domain: P \rightarrow 2^C$  获得属性的定义域, 属性的定义域为概念的集合.

函数  $range: OP \rightarrow 2^C$  获得对象属性的值域, 对象属性的值域为概念的集合.

函数  $mincard: C \times P \rightarrow N_0$  概念属性对的最小出现次数.  $N_0$  为整数集合.

函数  $maxcard: C \times P \rightarrow N_0 \cup \{\infty\}$  概念属性对的最大出现次数.  $N_0$  为整数集合.

函数  $instconc: I \rightarrow C$  获得实例所属于的概念.

函数  $instprop: P \times I \rightarrow 2^{I \cup L}$  获得实例中某个属

性的值.

3.2 本体一致性

本体一致性刻画了本体在演化过程中需要遵守的不变性约束. 文献[9]将一致性定义为: 从本体的定义中无法推导出矛盾的结论. 该文献通过定义一个约束集合的方法来刻画本体一致性. 借鉴文献[9]的刻画方法, 根据定义 1 中所提出的本体模型, 本文把可能出现的矛盾情况描述成为一个约束集合  $M$ . 受篇幅所限, 表 1 仅列出了  $M$  中常用的约束定义. 根据该约束集合, 对本体一致性的判定被转化为判断该本体是否满足  $M$  中的每一条约束. 据此, 本体的一致性判定函数可以定义如下.

**定义 3.** 一个本体是一致的当且仅当它满足约束集合  $M$  中的每一条约束. 本体的一致性判定函数被定义为

$$isConsistent(O, M) = \begin{cases} \text{true}, & O \text{ 满足 } M \text{ 中所定义} \\ & \text{的所有约束} \\ \text{false}, & \text{否则} \end{cases}$$

表 1 一致性约束集合  $M$

约束名称	约束内容
唯一 ID 约束	所有的实体都只有一个唯一的 ID
概念继承体系约束	概念的继承体系是一个有向无环图
根不变性约束	存在一个根概念 $Thing \in C$ 且它是 $C$ 中所有其它概念的父概念
概念闭包不变性约束	本体中除根概念外的任何一个概念至少都有一个父概念
概念继承体系闭包不变性	父概念和子概念必须是概念集合 $C$ 的实体
属性闭包不变性	概念和属性之间可以建立 $domain$ 和 $range$ 关系
数据属性不变性	一个数据属性的值域不可以是概念
属性继承闭包不变性	父属性及其子属性必须是属性集合 $P$ 的实体
实例不变性	$I$ 中的每一个实例必须关联于一个概念
出现次数约束	最小出现次数必须小于最大出现次数
出现次数约束闭包	出现次数所对应的概念必须属于概念集合 $C$ , 属性必须属于属性集合 $P$

4 本体变更操作

本体演化算法通过顺序执行相应的变更操作来完成变更需求, 这将导致一条变更执行路径的产生. 分析本体演化的影响范围实质上就是分析变更操作及其组成的变更执行路径的影响范围. 本节首先给出 MINI 算法支持的变更操作, 接着分析并量化这些变更操作的影响范围.

4.1 本体变更操作的定义与分类

**定义 4.** 一个本体变更操作被定义为一个函数:  $OntCh(E) = E'$ , 其中  $E$  和  $E'$  分别是变更操作改变的本体实体集合及其改变后的结果.

以图 1 为例, 删除概念国内外新闻的变更操作被定义为  $RemoveConcept(\{\text{国内外新闻}\}) = \emptyset$ .

根据变更操作所影响的实体个数, 本文将其分为两类: 基础变更操作和复杂变更操作. 基础变更操作是指仅对本体中的一个实体进行修改的操作, 复杂变更操作则是指对本体中的多个实体进行修改的操作. 文献[9]中还指出对一个本体实体的变更操作可以分为 4 类: 添加、删除、重命名和设置操作. 后两种操作的执行效果可以由前两种操作的组合来完成. 如  $Rename(e) = e' \Leftrightarrow Remove(e) \circ Add(e')$ . 据此, 依赖定义 1 中所描述的本体模型, 我们可以定义基础变更操作的最小集合, 如表 2 所示.

**定理 1.** 表 2 所示的变更操作集合是完备的. 证明. 该定理等价于证明任何一个变更操作均可以分解为表 2 中所列出的基础变更操作, 下面采用反证法来进行证明.

假设存在某一个变更操作  $OntCh$  不能被分解为

表 2 中所示的基础变更操作,那么,给定本体  $O$  及其实体集合  $E$ ,根据定义 4 可知,存在  $E' = OntCh(E)$ .

设  $E = \{e_1, e_2, \dots, e_n\}$  且  $E' = \{e'_1, e'_2, \dots, e'_m\}$ ,那么  $OntCh$  可以被分解为  $Remove(e_1) \circ Remove(e_2) \circ \dots \circ Remove(e_n) \circ Add(e'_1) \circ Add(e'_2) \circ \dots \circ Add(e'_m)$ .

从而导出矛盾,于是可以证明表 2 所示的变更操作集合是完备的. 证毕.

表 2 基础变更操作表

基础变更	添加	删除
概念	$AddConcept$	$RemoveConcept$
概念继承关系	$AddSubConcept$	$RemoveSubConcept$
属性	$AddProperty$	$RemoveProperty$
属性定义域	$AddPropertyDomain$	$RemovePropertyDomain$
属性值域	$AddPropertyRange$	$RemovePropertyRange$
最小值约束	$AddMinCardinality$	$RemoveMinCardinality$
最大值约束	$AddMaxCardinality$	$RemoveMaxCardinality$
实例	$AddInstance$	$RemoveInstance$
概念实例	$AddInstanceOf$	$RemoveInstanceOf$
关系实例	$AddRelationInstance$	$RemoveRelationInstance$

**定理 2.** 表 2 所示的变更操作集合最小.

证明. 该定理等价于证明表 2 中任何一个变更操作都不能再分解为更小的变更操作,下面采用反证法来进行证明.

假定存在某一个实体  $e$ ,它的基础变更操作为  $OntCh(e)$ (这里的  $OntCh$  为添加或删除操作),且它可以被分解为更小的变更操作集合:  $OntCh(e) = OntCh_1(e) \circ OntCh_2(e) \circ \dots \circ OntCh_n(e)$ ,于是对于  $\forall OntCh_i$ ,都有

$$OntCh_i(e) \leq OntCh(e), \quad 1 \leq i \leq n \quad (1)$$

这里前一个“ $\leq$ ”表示变更操作的分解关系.

根据变更操作的定义,可知存在  $e'$ ,使得  $e' = OntCh_i(e)$ ,于是  $OntCh_i(e)$  可以被分解为  $Remove(e) \circ Add(e')$ ,也即

$$OntCh(e) \leq OntCh_i(e), \quad 1 \leq i \leq n \quad (2)$$

根据式(1),(2)可得  $OntCh_i(e) = OntCh(e)$ .

从而导出矛盾,于是可以证明表 2 所示的变更操作集合是最小的. 证毕.

根据定理 1,2 可以得到下面的推论 1.

**推论 1.** 任何复杂的本体变更操作均可以被分解为表 2 所示的基础变更操作.

该推论无疑是非常重要的.首先,应用该推论可以将复杂变更操作影响范围的计算转化为相应的基础变更操作影响范围的计算.其次,该推论还使得无论面对什么样的不一致情况,都可以通过产生表 2 中的基础变更操作来帮助本体回归到一个一致性状态.这可以简化 MINI 算法中附加变更产生策略的

设计和验证工作.

## 4.2 变更操作的影响范围分析

变更操作的影响范围主要取决于变更操作所改变的主体实体集合以及该集合中实体与服务间的依赖关系.也就是说,如果变更操作改变了某个服务正在依赖的本体实体,那么该服务就会受到影响.而且,如果该变更操作所改变的实体集合被越多的服务依赖,那么变更操作的影响范围也就越大.变更执行路径的影响范围则可看成是所有单个变更操作影响范围的叠加.本文以一个数学表达式来量化变更操作的影响范围.首先,本文定义服务和本体实体之间的依赖关系.接着给出变更影响范围函数来量化某一时刻一条变更执行路径的影响范围.

**定义 5(应用依赖函数).** 给定本体实体  $e$  以及依赖该本体的服务  $m$ ,定义函数:

$$\phi(e, m) = \begin{cases} 1, & \text{应用 } m \text{ 依赖 } e \\ 0, & \text{否则} \end{cases}$$

**定义 6(变更影响范围函数).** 设某一时刻共有  $u$  个服务依赖本体  $O$ ,此时针对某个变更需求生成的变更执行路径为  $path = OntCh_1, OntCh_2, \dots, OntCh_n$ ,其中  $OntCh_i$  改变的主体实体集合为  $E_i$ ,该变更执行路径的影响范围为

$$ImpactRange(path) = \sum_{k=1}^u \sum_{i=1}^n \sum_{j=1}^{|E_i|} \phi(e_{ij}, m_k) \quad (3)$$

然而,在实际应用中式(3)的计算还存在很大困难.这是因为式(3)要求实时统计出每个服务依赖本体的详细信息.它需要精确地知道每个服务都依赖了本体的哪些实体,而这在实际应用中往往是很难做到的.但是,注意到如果对式(3)进行某些变化则可以避免这个问题.其推导过程如下:

由定义 5 可知

$$0 \leq \phi(e, m) \leq 1 \quad (4)$$

根据式(3),(4)可得

$$\begin{aligned} \min ImpactRange &= \min \sum_{k=1}^u \sum_{i=1}^n \sum_{j=1}^{|E_i|} \phi(e_{ij}, m_k) \\ &\leq \min u^* \sum_{k=1}^u \sum_{i=1}^n \sum_{j=1}^{|E_i|} 1 \\ &= u^* \min \sum_{i=1}^n |E_i| \end{aligned} \quad (5)$$

注意到如果把本体完成一次演化过程中依赖本体的服务个数  $u$  视为一个常量,那么式(5)是与服务自身信息无关的.该式实际上定义了影响范围函数的一个上界.根据此公式,在 MINI 算法的设计过程中就可以通过压缩影响范围上界的方法来减小变更

影响. 也就是说, 在演化过程中, 改变的本体实体个数之和越小, 演化的影响范围也就越小. 据此, 本文把一个变更操作所能引发的变更实体个数视作它的影响范围. 其定义见 4.3 节.

4.3 变更操作的影响范围量化

对于基础变更操作而言, 对某一个实体的变更会直接引起其邻接实体的改变, 并随着变更的传播间接地影响到本体的其它实体. 也就是说基础变更操作的直接影响范围是变更实体的邻接实体. 因此, 基础变更操作的影响范围被量化为变更实体的邻接实体个数. 为了便于量化影响范围, 根据定义 1~3, 本文首先把一个本体描述为一个图模型.

定义 7(本体的图模型). 给定本体  $O$ , 它的实体集合为  $E = \{C, P, I, Hc, Hp, label, domain, range, mincard, maxcard, instconc, instprop\}$ , 那么本体  $O$  的图模型被定义为  $Graph := \{Nodes, Edges\}$ . 其中, 对于  $\forall n \in Nodes$  则必有  $n \in C \cup P \cup I$ ,  $\forall e \in Edges$  则必有  $e \in \{label, domain, range, mincard, maxcard, instconc, instprop\}$ .

根据图模型的定义, 表 2 中的基础变更操作可以直接转化为对图模型的结点和边的添加及删除操作. 下面首先给出一个基础变更操作影响范围的量化方法. 然后在其之上给出复杂变更操作影响范围的量化方法.

定义 8. 基础变更操作影响范围的量化: 给定一个基础变更操作  $OntCh(e)$ , 其中  $e$  是本体  $O$  中的一个实体,  $G$  是  $O$  对应的图模型, 那么,

如果  $e \in Nodes$ , 则  $ImpactRange(OntCh(e)) =$  与  $e$  邻接的边的个数.

如果  $e \in Edges$ , 则  $ImpactRange(OntCh(e)) =$

与  $e$  邻接的结点的个数.

定义 9. 复杂变更操作影响范围的量化: 给定一个复杂变更  $OntCh(\{e_1, e_2, \dots, e_n\}) = \{e'_1, e'_2, \dots, e'_m\}$ . 那么根据推论 1, 它可以被分解为  $Remove(e_1) \circ Remove(e_2) \circ \dots \circ Remove(e_n) \circ Add(e'_1) \circ Add(e'_2) \circ \dots \circ Add(e'_m)$ . 据此,

$$\begin{aligned} ImpactRange(OntCh(e_1, e_2, \dots, e_n)) = & \\ & ImpactRange(Remove(e_1)) + \\ & ImpactRange(Remove(e_2)) + \dots + \\ & ImpactRange(Remove(e_n)) + \\ & ImpactRange(Add(e'_1)) + \\ & ImpactRange(Add(e'_2)) + \dots + \\ & ImpactRange(Add(e'_m)). \end{aligned}$$

如图 1 所示, 在本体  $O$  中删除概念国内外新闻将会影响其邻接的 4 个实体:  $Hc$ (国内外新闻, 新闻)、 $Hc$ (科技要闻, 国内外新闻)、 $Hc$ (图片新闻, 国内外新闻)以及  $Domain$ (国家, 国内外新闻). 因此  $ImpactRange(RemoveConcept(新闻)) = 4$ .

5 MINI 演化算法

根据上面的分析, 本体演化问题可以被定义为寻找一条变更执行路径使得演化后的本体满足一致性约束且所改变的本体实体总数最小. 受文献[8]提出的图搜索算法启发, MINI 算法将演化问题转化为如图 2 所示的图搜索问题, 并提出新的启发策略来寻找影响值最小的变更执行路径. 其中, 图的结点是本体, 边是变更操作, 边的权值则是定义 8、定义 9 所给出的变更操作影响范围.

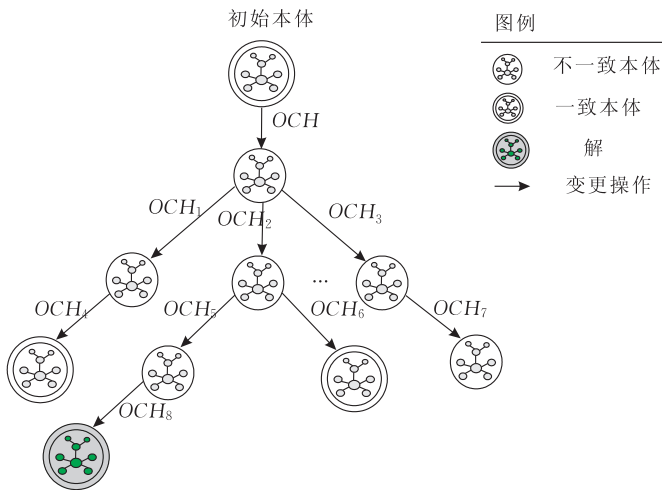


图 2 变更路径生成与图的搜索算法



如图 2 所示,当一个一致性本体应用某个变更操作后,将可能进入到一个不一致状态.通过推理器,MINI 算法可以得知违反了一致性约束的实体集合  $E$ .此时,MINI 算法将产生针对  $E$  的附加变更来促使本体进入下一个状态.执行一个附加变更操作后,本体将进入另一个状态.此时,如果本体变为一致的,那么它就有可能成为 MINI 算法追求的一个解.

依据算法的设计目标,本文提出一种最小搜索启发策略并应用于 MINI 算法以有效地减小解空间及加速算法的搜索过程.该启发策略被描述为:在图的搜索过程中计算已搜索路径的变更影响值,如果该值大于已经搜索到的最小值就不再继续搜索.搜索过程中,若本体进入到一个一致状态,且此时已搜索路径的影响值小于原先搜索到的最小值,那么将作为一个新的备选解. MINI 算法以一个待演化本体以及一个待执行的变更需求集合为输入,以一条最小变更执行路径和最小影响值为输出.它的实现如算法 1 所示.

在 MINI 算法的实现过程中,一个重要问题是如何针对某个变更  $OntCh$  导致的不一致实体  $e$  产生附加变更操作集合.附加变更操作的产生依赖于执行的变更  $OntCh$ 、不一致的实体  $e$  以及违反的不一致约束类型.附加变更操作的产生规则被抽象为一组附加变更产生策略.例如,如果实体  $e$  违反表 1 所述的概念闭包不变性约束,即  $(e \in C \wedge e \neq Thing \wedge e$  不存在父概念),那么在产生附加变更时,既可以选择删除孤立的概念结点,也可以选择将孤立概念结点挂接到本体中的某一概念结点成为它的子概念.据此,可以产生如下两个附加变更的产生策略:

首先,可以选择删除孤立的概念结点,即产生附加变更  $RemoveConcept(e)$ .

其次,如果  $OntCh$  是  $RemoveConcept$  变更,删除的概念为  $c'$ .  $c'$  的父概念集合为  $super(c')$ .那么,将可以产生附加变更  $\forall c'' \in super(c') AddSubConcept(e, c'')$ .

例如,如果实体  $e$  代表一个概念间的继承关系,它在执行变更  $OntCh$  后,违反了表 1 所述的概念继承体系闭包不变性约束,也就是说  $e$  的定义域概念或者值域概念有一个不属于概念集合  $C$ .据此,可以产生如下两个附加变更的产生策略:

首先,可以选择删除违反一致性约束的继承关系,即产生附加变更  $RemoveSubConcept(domain(e), range(e))$ .

其次,如果  $OntCh$  是  $RemoveConcept$  变更,删

除的概念为  $c'$ .  $c'$  的父概念集合为  $super(c')$ ,子概念集合为  $sub(c')$ .

若  $range(e) \notin C$ ,则产生附加变更  $\forall c'' \in super(c') AddSubConcept(domain(e), c'')$ .

若  $domain(e) \notin C$ ,则产生附加变更  $\forall c'' \in sub(c') AddSubConcept(c'', range(e))$ .

针对表 1 中所述的不一致约束集合,MINI 算法中定义了一组常见的附加变更产生策略.需要注意的是领域专家可以根据实际情况添加自定义的附加变更产生策略.根据推论 1,领域专家在制定附加变更产生策略时可以仅考虑基础变更操作.演化过程中,MINI 算法将根据变更影响范围的取值来动态选取最优策略,从而计算出一条影响范围最小的变更执行路径.下面以图 1 为例来简述 MINI 算法的执行过程.当概念国内外新闻被删除后,MINI 算法的主要执行步骤如下面算法 1 所示.

**算法 1.** 最小代价变更执行算法  $MINI(O, ChgReqs)$ .

输入:  $(O, ChgReqs)$ ,即待演进本体  $O$  和待执行本体变更集合

输出:最小影响值和最小变更执行路径

全局变量:  $(minValue, minPath)$ ,已经搜索到的最小影响值和最小变更执行路径  
 $(curValue, curPath)$ ,当前累积的影响值和正在搜索的变更执行路径

实现:

```
function MINI( $O, ChgReqs$ ) {
     $minValue = MAX\_VALUE$ ;  $minPath$  置空;
     $curValue = 0$ ;  $curPath$  置空;
    for  $\forall OntCh \in ChgReqs$  do
         $executeOneChange(O, OntCh)$ ;
        //顺序执行每一个变更需求
    end
    return  $(minValue, minPath)$ ;
}

function  $executeOneChange(O, OntCh)$  {
    if 超过最大搜索层数 then return;
     $value$  = 计算变更  $OntCh$  的影响值;
    if  $curValue + value > minValue$  then return;
     $curValue = curValue + value$ ;  $curPath = curPath \cup OntCh$ ;
     $O' =$  对本体  $O$  执行变更  $OntCh$ ;
    if  $isConsistent(O')$  then {
         $minValue = curValue$ ;  $minPath = curPath$ ;
        //寻找到的最小变更执行路径
    }
    else {
         $E =$  计算不一致实体集合;
```

```

for  $\forall e \in E$  do
    针对实体  $e$  产生附加变更操作集合  $OntChs$ ;
    for  $\forall OntCh \in OntChs$  do
         $executeOneChange(O', OntCh)$ ;
    end
end
}
 $curValue = curValue - value$ ;  $curPath = curPath - och$ ;
}

```

(1) 删除国内外新闻后,不一致的本体实体集合为 $\{Hc(\text{国内外新闻, 新闻}), Hc(\text{科技要闻, 国内外新闻}), Hc(\text{图片新闻, 国内外新闻}), Domain(\text{国家, 国内外新闻})\}$ . 这 4 个实体违反了表 1 中所列出的“概念继承体系闭包不变性”. 对于这 4 个实体,根据上面所述的附加变更产生策略,可以应用的变更操作为相应的 *RemoveSubConcept* 和 *AddSubConcept* 操作. MINI 算法将启动搜索过程,并首先选择其中的一个变更操作加以执行. 例如,首先选择 *RemoveSubConcept* (科技要闻, 国内外新闻) 加以执行. 根据定义 8, 这个变更的影响值为 2.

(2) 不一致的实体变为科技要闻. 它违反了表 1 中所列出的“概念闭包不变性约束”. 对于这个实体,根据上面所述的附加变更产生策略,产生的附加变更操作为 *RemoveConcept* (科技要闻) 和 *AddSubConcept* (科技要闻, 新闻). 然后,分别计算这两个变更操作的影响值. MINI 算法首先选择 *RemoveConcept* (科技要闻) 变更加以执行. *RemoveConcept* (科技要闻) 会直接影响一个邻接实体(属性点击次数),因此变更的影响值为 1. 此时,累积影响值为 3.

(3) 不一致的实体变为科技要闻和属性点击次数之间的定义域关系. 选择删除这个定义域关系后,累积影响值为 5. 随后,不一致的实体变为属性点击次数. 选择删除这个属性后,以概念科技要闻为根结点的子树中就不再存在不一致情况. 此时,MINI 算法将按照类似过程执行与 *Hc*(国内外新闻, 新闻) 和 *Hc*(图片新闻, 国内外新闻) 相关的变更操作. 执行完成后,将得到一条可保障本体一致性的变更执行路径,并将其保存到 *minPath* 列表中.

(4) 注意到在上述步骤中,MINI 算法均只选择了一个可能的附加变更操作加以执行,因此,MINI 算法还将向上回溯到存在其它附加变更操作的最近的实体上,选择不同的变更操作加以执行并累积变更的影响值. 此时,如果得到另一条满足一致性约束的变更执行路径且它的影响值更小,那么将用这条路径替换 *minPath* 中已经记录的最小执行路径.

(5) 当所有可能的变更执行路径均被搜索过后,MINI 算法结束,并返回最终结果.

## 6 评 价

### 6.1 时间复杂度分析

MINI 算法是基于图的搜索算法,而此类算法是一个 NP 完全问题. 对于一个图搜索算法来说,算法执行所耗费的时间主要取决于以下两个因素:

(1) 算法的搜索深度 *level*. 它表示该算法搜索到了图的第几层节点. 算法搜索的深度越深,算法执行所耗费的时间也就越多.

(2) 节点的分支数 *b*. 它表示图的每个节点最多允许产生多少个子节点. 每个节点产生的子节点个数越多,算法执行所耗费的时间也就越多.

当 *level* 和 *b* 值确定后,图搜索算法的时间复杂度可以记为  $O(b^{\text{level}})$ . 对于 MINI 算法而言,附加变更产生策略的个数是一个常数. 也就是说,在图搜索时,每个节点产生的子节点个数是一个常数. 因此,在不考虑阈值所起的加速作用时,MINI 算法的最差时间复杂度主要决定于搜索的层数.

阈值的引入可以改善 MINI 算法的最差时间复杂度. 阈值限定了算法的最大搜索层数,我们将其记为 *maxLevel*. 而此时,MINI 算法的最差时间复杂度变为  $O(b^{\text{maxLevel}})$ . 由于 *maxLevel* 是预先定义的一个常数,在很多情况下,它远小于完全搜索的最长搜索路径的层数,它为 MINI 算法的最差情况提供了保证.

### 6.2 应用评价

#### 6.2.1 实验内容

本文以引言中所介绍的实际项目为实验场景来验证 MINI 算法的应用效果. 该项目通过动态集成各地方科技部门的科技信息资源来实现中央级的科技资源汇聚. 本体的构建是实现资源汇聚的一个核心任务. 依赖本体,平台可以以统一的方式来接入资源提供者提供的如发布、搜索、统计以及审核等各种服务. MINI 算法被引入平台来保证每次本体变化所影响的平台服务数量最少. 下面以图 1 所示的科技资源本体中的科技新闻片断为实验数据. 该片断共涉及 10 种科技新闻资源以及与这些资源相关的查询、发布等共计 100 个服务.

根据引言中的分析,MINI 算法的提出是为了减小演化过程中受到影响而需要改变的服务个数. 因此,服务变更率这一指标被提出以衡量算法的有



效性. 该指标定义为

$$\alpha = \text{演化后需要改变的服务个数} / \text{服务总数} \quad (6)$$

采用 Protégé, KAON 中的变更执行算法和 MINI 算法分别对该本体执行演化操作, 并计算演化后服务的变更率. 由于删除操作对稳定性的影响最大, 因此实验内容主要以对图 1 中概念的删除操作为主. 部分实验内容和结果如表 3 所示. 从表 3 中可以看出, MINI 算法对服务的变更程度明显小于其它两种算法.

表 3 实验内容及结果

实验内容	改变率		
	Protégé	KAON	MINI
删除新闻概念	0.88	0.53	0.16
删除科技动态概念	0.45	0.59	0.28
删除四川科技动态概念	0.13	0.13	0.13
删除北京科技动态概念	0.15	0.15	0.15
删除影视资料概念	0.16	0.16	0.16
删除国内外新闻概念	0.49	0.35	0.22
删除科技要闻概念	0.26	0.37	0.26
删除图片新闻概念	0.11	0.27	0.11
删除科技快报概念	0.17	0.17	0.17
删除专题要闻概念	0.14	0.14	0.14
平均值	0.327	0.318	0.198

6.2.2 实验结果分析

表 3 反映出 MINI 算法的有效性主要体现在对于非叶节点的改变上. 非叶节点的层次越高, 其子节点和属性越多, MINI 算法的优势也就越明显. 对于叶节点的改变, 3 种算法所引起的服务变更率基本持平. 这是因为 MINI 算法的优势在于分析本体实体之间的依赖关系, 并在发生本体变更后能够尽量减轻它们之间的相互影响.

以删除“国内外新闻”概念为例进行说明. 在 Protégé 中, “国内外新闻”概念的删除会导致其子概念“科技要闻”和“图片新闻”的删除. 因此, 依赖这些实体的服务都需要发生改变, 即导致了 0.49 这一比较高的服务变更率. 而对于 MINI 算法来说, 当删除了“国内外新闻”概念后, 其子概念会变为概念“新闻”的子概念, 并且属性“国家”也会得到保留. 因此, 只需改变依赖“国内外新闻”概念的服务, 因此导致了 0.22 这一较低的服务变更率. 最后, 对于 KAON 算法来说, 如何对待已删除概念的子概念及其属性可以事先由领域专家手工配置. 不同的领域专家可能导致不同的变更结果, 因此导致的服务变更率往往在最坏和最好的情况之间徘徊. 它不能像 MINI 算法一样自动计算影响范围最小的变更执行路径. 本文所进行的实验选择了 KAON 算法的缺省配置来实现变更需求. 即在删除“国内外新闻”概念后, 其

子概念会变为概念“新闻”的子概念, 但“国家”属性会被删除. 因此, 依赖属性“国家”以及概念“国内外新闻”的服务都会受到影响, 这导致了 0.35 服务变更率.

此外, 从平均效果来看, Protégé 和 KAON 算法的服务变更率基本持平, 本文所提出的 MINI 算法则明显低于这两个算法, 仅为这两个算法服务变更率的 60% 左右. 通过这个实验结果可以充分说明本文所提出算法能够有效减小本体变更的影响范围.

7 结 论

开放环境下, 一个本体往往被多个服务所依赖. 本体变更通常会对服务产生巨大的影响. 不同的本体演化实现方法对服务所造成的影响范围差异很大. 传统的本体演化研究主要聚焦于如何实现演化需求并维护变更前后本体的一致性, 对于如何降低演化影响范围关注甚少. 本文的贡献在于深入分析了本体演化对服务的影响, 量化了演化的影响范围, 并提出了可以有效减小变更影响范围的 MINI 演化算法. 与传统算法相比, MINI 算法的优势在于能够有效降低本体变更所影响的服务个数, 显著减轻依赖本体应用系统开发和维护的负担.

参 考 文 献

[1] Liu C, Chen W, Han Y. DODO: A mechanism helping to dynamically construct domain ontologies for services integration//Proceedings of the International Workshop on Service Oriented Software Engineering in Conjunction with the 28th International Conference on Software Engineering (ICSE06). Shanghai, China, 2006: 13-18

[2] Peter H, York S. State-of-the-art on ontology evolution. Institute AIFB, University of Karlsruhe, Karlsruhe, Germany: Technique Report SEKT deliverable D3. 1. 1. b, 2004

[3] Noy N F, Klein M. Ontology evolution: Not the same as schema evolution. Knowledge and Information Systems, 2004, 6(4): 428-440

[4] Sure Y, Angele J, Staab S. OntoEdit: Multifaceted inferencing for ontology engineering. Journal on Data Semantics, 2003, 1(1): 128-152

[5] Bechhofer S, Horrocks I, Goble C, Stevens R. OilEd: A reasonable ontology editor for the semantic web//Proceedings of the Joint German/Austrian Conference on Artificial Intelligence (KI-01). Vienna, Austria, 2001: 396-408

[6] Stojanovic L, Motik B. Ontology Evolution within Ontology Editors//Proceedings of the OntoWeb-SIG3 Workshop at the

- 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02). Siguenza, Spain, 2002; 53-62
- [7] Stojanovic L, Maedche A, Motik B, Stojanovic N. User-driven Ontology Evolution Management//Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02). Siguenza, Spain, 2002; 285-300
- [8] Stojanovic L, Maedche A, Motik B, Stojanovic N. Ontology evolution as reconfiguration-design problem solving//Proceedings of the 2nd International Conference on Knowledge

Capture (K-CAP03). Sanibel Island, FL, USA, 2003; 162-171

- [9] Stojanovic L. Methods and tools for ontology evolution [Ph. D. dissertation]. University of Karlsruhe, Karlsruhe, 2004
- [10] Pinto S, Martins J. Ontologies: How can they be built? Knowledge and Information Systems, 2004, 6(4): 441-464
- [11] Liu Bai-Cong, Gao Ji. A study on ontology evolution management. Computer Science, 2004, 31(5): 9-12(in Chinese) (刘柏嵩, 高济. 本体演化管理研究. 计算机科学, 2004, 31(5): 9-12)



**LIU Chen**, born in 1980, Ph. D. candidate. His research interests include ontology integration and ontology evolution.

**HAN Yan-Bo**, born in 1962, professor, Ph. D. supervisor. His research interests include service oriented grid computing and software integration.

**CHEN Wang-Hu**, born in 1973, Ph. D. candidate. His research interests include service matching and service discovery.

**WANG Jian-Wu**, born in 1980, Ph. D. candidate. His research interest is in business service.

## Background

The work is supported by the R&D Infrastructure and Facility Development project (No. 2005DKA64201) and several other projects. The first project aims to integrate heterogeneous Web resources distributed on local science and technology offices. To unify representations of sharable resources, ontologies of local offices will be firstly integrated to form a global ontology. However, the global ontology will evolve to absorb new knowledge or remove out-of-date knowledge timely for responding changes of local offices' ontologies. Changes of the global ontology may impact on dependent services. Therefore, the proposed MINI algorithm is applied to reduce impact ranges when the global ontology evolved.

The traditional approaches for ontology evolution can be classified into following three categories. The first category is represented as ontology editors, which are tools that support the ontology development and maintenance task. The second category is called as "user-driven" strategy. The typical work is KAON. Different from ontology editors, in this strategy, users will direct the generation of change paths. The third category regards the ontology evolution as a reconfiguration-design problem and adapts the graph searching method to this problem. The benefits of this approach is that it allows the user to specify declaratively what she wants to do and not how to do that. This approach is implemented in the new version of the KAON system. All these researches only are focused on how to realize change requests and keep

the consistency of evolving ontology. Few of them care about the impact ranges of ontology evolution. However, decreasing impact ranges can remarkably alleviate the burden of services development and maintains.

To reach this goal, the paper proposes an algorithm called MINI. The main contributions of the paper are: (1) deeply analyze the dependent relations among ontology entities and services. The paper points out that if an entity is changed, then services depending on it will also be impacted. Furthermore, the more entities which have dependent services changed the greater the ranges of an ontology change. Based on these analyses, a math formula is established to quantify the value of impact ranges. (2) Transforms the problem of ontology evolution as a graph searching problem, which aims to searching an ontology change path that has minimal impact range value. A heuristic strategy is proposed and applied to accelerate the searching process. By applying this strategy, the MINI algorithm will accumulate the value of a searching path and decide whether it is less than the already found minimal value. If not, the MINI algorithm will try on another searching path to find best solutions. (3) Finally, through experiments, we find that the average impact ranges caused by the MINI algorithm is only 60% of the other current evolution algorithms. It proves that the MINI algorithm can be more effective to reduce impact ranges than other algorithms.