基于变化生成图的OWL本体协同进化方法研究*)

鲍爱华　姚　莉　张维明

(国防科技大学信息系统与管理学院　长沙 410073)

摘　要　随着本体规模的不断扩大，由于能力和知识的约束，单个知识工程师已经难以单独对本体进行改进，因此如何实现本体的协同进化成为了一个亟待解决的问题。本文提出了一个能够支持OWL本体协同进化的结构化方法，引入了一种有向的与或图——变化生成图来表示知识工程师的本体改进意图，通过发现变化生成图之间的冲突来找出多个知识工程师本体改进意图之间潜在的矛盾，并通过变化生成图的冲突消解来解决本体改进意图之间的矛盾，使得多个本体改进意图能够并存，从而实现本体协同进化的目标。

关键词　本体，本体进化，协同进化，OWL，变化生成图，与或图

Research on the Collaborative Evolution of OWL Ontologies Based on And-or Graph

BAO Ai-Hua　YAO Li　ZHANG Wei-Ming

(School of Information System and Management, NUDT, Changsha 410073)

Abstract　With the rapidly incensement of ontology size, it is very hard for a knowledge engineer to evolve ontology alone due to the knowledge and ability constraint. So, it is important for knowledge engineers cooperate to evolve the ontology, which we called collaborative evolution. In this paper, we propose a structural OWL ontology cooperative evolution method, and introduce a kind of directed and-or graph(Change Generation Graph) to describe the engineer's intention of ontology evolution. The confliction of change generation graphs is also analyzed, and then, this paper propose an algorithm to resolve the confliction, so that engineers' intention can coexist, and cooperative evolution is achieved.

Keywords　Ontology, Ontology evolution, Collaborative evolution, OWL, Change generation graph, And-or graph

1　引言

在语义Web的研究中，本体是一个研究热点和重点，它对应用领域中的知识进行了形式化的描述，使得机器能够理解并共享领域知识，是语义Web的基础。目前针对本体的大部分研究都集中在本体的创建和使用问题上，它们假设本体及其封装的知识不会经常变化。但是，在动态开放的环境中，领域知识是不断发生变化的[1]，这些变化包括领域知识的更新、系统功能的修改等等。因此，封装了领域知识的本体也需要进行相应的改进。

本体进化就是使本体不断地改进，以及时适应领域知识的变化，同时对本体的变化进行管理，从而保持本体的一致性[2]。由于本体内部概念之间的关系错综复杂，对于本体的任何修改都可能导致某个局部的不一致，因此本体进化是一个复杂的系统工程。它不仅需要实现知识工程师的本体改进意图，还需要对该变化所带来的本体不一致性进行分析，并执行额外的变化来保证本体的一致性。另外，还需要将发生的本体变化传播给依赖该本体的其他本体和应用，从而保证相关本体和应用的一致性。在这种情况下，完全由知识工程师手工完成本体进化是不现实的，需要一套方法和工具来支持知识工程师完成本体进化，保证演变后本体的一致性和正确性。

随着应用的深入，本体中所封装的领域知识越来越多，所涉及的应用领域越来越广，本体的规模也越来越大。因此，由于知识与能力的约束，单个知识工程师已经难以单独承担本体进化的任务，需要多个知识工程师互相协作来完成。所以，为了实现本体的进化，迫切需要对本体的协同进化方法进行研究，从而指导多个知识工程师协作进行本体进化，并指导协同进化工具的开发。

在本体表示方面，目前已经存在多种本体语言，如KAON[3]、DAML、OWL等，它们的语法和语义均存在着一定的差异。对于采用不同语言的本体，其进化方法也存在一定的差异。由于OWL本体语言已经成为W3C的推荐标准[4]，本文主要研究OWL(OWL DL和OWL Lite)本体的协同进化方法。

2　相关的研究工作

目前，已经有许多学者致力于本体进化的研究，并取得了一定的成果。其中，Stojanovic对本体进化的相关概念进行了定义[5]，并对KAON本体的进化问题进行了研究，提出了一个可行的本体进化方法[2]，并实现了一个支持该方法的原型系统。由于Stojanovic的工作较为系统，也具有一定的代表性，因此许多后续的研究工作均以Stojanovic的研究为基础，进行改进或扩充。

文[2]所提出方法的核心包含4个阶段：

1)本体变化表示阶段。需要将知识工程师的本体改进意图转化为一系列本体变化，并对这些变化进行形式化的表示；

2)本体变化语义分析阶段。主要对前一阶段所产生的变化进行分析，找出这些变化在执行后所造成的本体不一致，并且产生额外的变化来保证本体的一致性；

*)本课题得到国家自然科学基金(项目编号70371008)资助。**鲍爱华**　博士研究生，主要研究方向为分布式人工智能、语义Web；**姚　莉**　教授，博导；**张维明**　教授，博导。

3)本体变化传播阶段。需要将前两个阶段得到的本体变化传播到依赖于该本体的其他本体和应用中,从而保持相关本体和应用的一致性;

4)变化实施阶段。本体进化系统将由本体变化意图所得到的相关变化反馈给知识工程师,在知识工程师允许后执行这些变化操作,完成本体的进化。

在上述方法中,对本体变化进行语义分析依赖于本体所采用的本体语言,OWL 本体的变化语义分析与 KAON 本体有着一定的差异。在文[6]中,作者对上述方法进行了改进,着重研究了 OWL 本体的变化语义分析阶段,从而使得上述方法能够对 OWL 本体的进化提供支持。

在文[7]中,作者采用本体版本日志的方式来记录本体进化过程中的多个版本的本体,并且通过本体多个版本之间的区别来定义本体变化。例如,如果发现某个版本的本体比前一版本少了某个概念,那么说明发生了概念删除操作。在采用上述方法对本体进化进行分析后,作者能够消除在本体变化语义分析阶段中产生的多余的变化,从而提高本体进化的精确性。

上面提到的研究工作能够在一定程度上解决本体进化的问题,但是它们并不完全适用于多个知识工程师进行本体协同进化的情况。在协同进化的环境下,知识工程师的改进意图之间有可能存在潜在的冲突,这时就需要及时发现这些冲突,并且找出相应的解决方案提供给知识工程师,从而在实现多个知识工程师本体改进意图的同时保持本体的一致性。本文对文[2, 6]所提出的本体进化方法进行了改进,扩展了方法的第二阶段,即本体变化语义分析阶段,使之能够适应本体协同进化环境。另外,本文还引入了一种变化生成图(Change Generation Graph)来分析知识工程师本体改进意图之间的冲突,给出了这些冲突的发现和消解方法,并以一个实例来说明该方法的可行性。

3 OWL 本体协同进化方法

针对 Stojanovic 所定义的本体进化,本文的本体协同进化是指多个知识工程师相互协作所进行的本体进化,因此,本体协同进化带来了新的问题:

1)不同知识工程师所发起的本体进化需求本身可能出现冲突,例如,RemoveConcept(Student)与 AddInstanceOf(Peter, Student)是冲突的。这时,可能二者的变化都是合理的。但是如果同时执行,那么后者的变化将执行失败。

2)虽然变化本身并不冲突,但为了保持本体的一致性,产生的额外变化之间可能会发生冲突。这时,如果不进行处理,二者所执行的变化均有可能引起执行失败。

3)变化及引起的相关变化均没有冲突,但是在执行完成后,本体可能产生新的不一致,例如,如果$(b,c)\in Hc^* \wedge (d,a)\in Hc^*$[2],那么在执行了 AddSubConcept(a, b)与 AddSubConcept(c, d)之后,本体的概念层次就出现了闭环,而这是违反本体一致性模型的[6]。

由此可见,与普通的本体进化相比,本体的协同进化具有更高的复杂性,因此需要一个结构化的本体进化方法来指导。本文在文[2, 6]中所提出的进化方法的基础上对变化语义分析阶段进行了改进,使之能够满足协同进化的要求。改进后的变化语义分析阶段如图 1 所示。

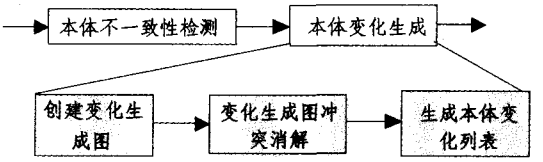

图 1 本体协同进化中的变化语义分析阶段

本体变化语义分析阶段包含两个任务,即本体不一致性检测和本体变化生成。本体不一致性检测的主要目的是找出本体在改变后可能存在的不一致情况,本体变化生成则是对不一致情况进行分析,并生成额外的变化来使本体保持其一致性。对于本体不一致性检测,文[6]通过定义本体一致性模型来发现本体的不一致,这在本体协同进化中同样适用,而本体变化生成任务则需要考虑更为复杂的情况。在这里,本文将本体变化生成任务分为 3 个步骤来完成:

1)创建变化生成图。采用本文所引入的变化生成图来表示本体改进意图,同时对本体变化执行后的本体一致性进行分析,生成额外的变化来保证本体的一致性。最后,将生成的额外变化添加到变化生成图中。

2)变化生成图冲突消解。对变化生成图的冲突进行定义,以此来发现知识工程师本体改进意图之间的冲突,并通过对变化生成图进行冲突消解来解决本体改进意图之间的冲突。

3)生成本体变化列表。遍历完成冲突消解的变化生成图,从而得到一个本体变化序列,并将该变化序列提供给知识工程师做参考。由于这里得到的变化序列既能够反映初始的本体改进意图,又不会与其他工程师的进化意图冲突,因而通过执行这些变化就可以达到本体协同进化的目标。

下面,本文将对本体变化生成任务的 3 个步骤进行详细的说明。在说明的过程中,我们以一个简单的 OWL 本体为例。由于 OWL DL 在描述能力上与描述逻辑 SHOIN(D)是等价的[8],因此这里我们采用 SHOIN(D)语法来表示 OWL 本体。

例 1 一个简单的 OWL 本体

$Professor\subseteq Person$, $Student\subseteq Person$(教授与学生都是人),$PhD\ Student\subseteq Student$,

$MSt\ Student\subseteq Student$(博士生与硕士生都是学生),$T\subseteq \forall\ teach^{-1}.\ Professor$,$T\subseteq \forall\ teach.\ Student$(*teach* 的域和取值范围分别是教授与学生),$Professor(peter)$, $PhD\ Student(tom)$(*peter* 是教授,*tom* 是博士生)

4 创建变化生成图

为了方便说明,我们首先给出几个定义。

***定义 1**[3](本体变化) 本体变化 $ch\in OntoCh$ 是本体与本体之间的映射,即 $ch: O\rightarrow O$。这里,*OntoCh* 代表所有本体变化的集合,O 代表所有本体的集合。

在后续的分析中,本文借鉴了 Stojanovic 在文[2]中对本体变化的分类方法,对本体的每个元素(如概念、属性等)均抽取出增性变化(如 *AddConcept*)和减性变化(如 *RemoveConcept*),共得到 28 个基本变化,任何复杂的变化均可由这些变化复合而成。

[2] $(x,y)\in Hc^*$ 表示概念 x 的父概念,下同。

[3] 这里,带 * 标注表示该定义参考自文[2],下同。

在对本体应用某个本体变化后，原有的本体可能会进入一个不一致的状态，因此我们需要对新本体的一致性状态进行分析，找出可能的额外的本体变化，从而使该本体保持一致性。创建变化生成图的主要目的就是找出这些额外的变化，并用变化生成图来对它们进行组织。

定义 2（变化生成图） 变化生成图（*Change Generation Graph*）是一种有向的与或图，它可形式化地定义为：$CGG:=(CHNodes,E)$。其中，

(1) $CHNodes=\{chNode_i\}, 1\leqslant i\leqslant|CHNodes|$ 是变化生成图的节点集合，每个节点代表了节点所包含的本体变化以及该节点的类型。节点的形式化定义为

$chNode_i=(ch_i,type), ch_i\in OntoCh\cup\{\text{“}emptyCh\text{”}\}, type\in\{\text{“}and\text{”},\text{“}or\text{”},\text{“}leaf\text{”}\}$

其中，“*emptyCh*”代表该变化不对本体进行任何改变。

(2) $E=\{E_k\}, 1\leqslant k\leqslant|E|$ 是变化生成图边的集合，每条边代表了相连节点所包含的变化之间的触发关系，即：为了保持本体的一致性，在完成 $chNode_i$ 所包含的变化 ch_i 之后，需要完成子节点所包含的变化 ch_j。变化生成图的边的形式化定义为

$E_k(chNode_i,chNode_j), chNode_i, chNode_j\in CHNodes, 1\leqslant i,j\leqslant|CHNodes|, i\neq j$

我们将类型为“*and*”的节点称为“与”节点，表示在完成该节点所包含的变化之后，为了保持本体的一致性，需要完成所有子节点所包含的变化，否则该变化不能执行；“*or*”节点表示在完成该节点所包含的变化之后，只需要完成某个子节点所包含的变化即可；“*leaf*”节点表示该节点的执行不会影响本体的一致性，不会引起其他相关的变化。

由变化生成图的定义可知，图的根节点代表了知识工程师所需要执行的变化，而图的其他节点则表示为了保持本体的一致性，在完成根结点所包含的本体变化后，可能需要执行的额外的本体变化。因此，变化生成图在反映了知识工程师的本体改进意图的同时，在执行后又保持了本体的一致性。例如，图 2(b)表示了在例 1 所示的本体中删除概念 *PhD Student* 所生成的变化生成图。

在给出变化生成图的定义后，我们将说明如何根据一个本体变化创建变化生成图。

4.1 创建变化生成图

为了保持本体的一致性，在将本体变化应用于本体时，需要考虑生成哪些额外变化才能保持本体的一致性，以及如何生成这些变化。这时，知识工程师将会遇到一些决策点，比如，在需要删除概念时，如何对待其子概念、如何对待与该概念相关的属性以及如何对待该概念的实例等等。下面，本文将具体说明如何根据本体进化决策点得到本体变化生成图。首先，我们给出基本进化策略的定义。

定义 3（基本进化策略） 基本进化策略 EES（*Elementary evolution strategy*）就是一系列能够帮助知识工程师处理本体进化决策点的本体进化策略。

在文[2]中，作者根据本体进化的具体情况分析得出了 9 种本体进化决策点和 23 个基本进化策略[4]，其中，每个本体进化决策点都对应着多个基本进化策略。例如，对于在删除概念后如何处理其子概念，知识工程师具有 3 个基本进化策略：(1)删除该概念所有的子概念；(2)将所有子概念转移为该概念父概念的子概念；(3)将所有子概念转移为根概念（*OWL:Thing*）的子概念。这里，本文将以这些决策点和基本进化策略为基础进行分析。

根据对本体进化决策点的分析和基本进化策略的定义，我们可以发现，如果某个本体变化在执行时会遇到某个本体进化决策点，那么该变化的执行将导致本体的不一致，从而需要产生额外的变化来保持本体的一致性，而需要产生的额外变化就蕴涵在基本进化策略中。所以，根据知识工程师所需要的本体变化、本体进化决策点和基本进化策略，我们就可以创建一个符合知识工程师本体改进意图的本体变化生成图。

下面，本文给出一个创建本体变化生成图的算法。

算法 1 创建本体变化生成图

输入：由本体改进意图转化得到的本体变化 *ontoCh*；

输出：表示本体改进意图的本体变化生成图 *CGG*。

①创建一个空的本体变化生成图 *CGG*，创建一个包含 *ontoCh* 的本体变化节点，设置该节点类型为“未知”，并将该节点添加到 *CGG* 中；

②在 *CGG* 中任取一个类型为“未知”的节点 *CN*。如果没有类型为“未知”的节点，那么转到步骤 7。

③如果 *CN* 所包含的变化 *ch* 是组合变化，那么找出组成 *ch* 的基本变化，并且为找出的变化创建类型为“未知”的变化节点，然后再将这些节点添加为 *CN* 的子节点。最后，设置 *CN* 的节点类型为“*and*”，转步骤 2。

④如果 *CN* 所包含的变化 *ch* 是基本变化，那么对该变化和相应的本体进行分析，找出在执行该本体变化时将会遇到的本体进化决策点，为每个本体进化决策点创建一个包含“*empty*”变化的本体变化节点，并添加为 *CN* 的子节点。如果 *ch* 在执行时不会遇到任何本体进化决策点，那么设置 *CN* 的节点类型为“*leaf*”，转步骤 2。

⑤对于步骤 4 中所创建的每个变化节点（设为 CN_1），重复进行步骤 5.1～5.3 的处理。

5.1 找出所有能够解决 CN_1 所对应的本体进化决策点的基本进化策略。

5.2 对于步骤 5.1 中所找出的每一个基本进化策略 *EES*，找出能够实现该策略的本体变化 *ch*。为 *ch* 创建一个类型为“未知”的本体变化节点，并添加为 *CN*1 的子节点。

5.3 设置 *CN*1 的节点类型为“*or*”。

⑥设置 *CN* 的类型为“*and*”，转步骤 2。

⑦对 *CGG* 中所有类型相同、包含的本体变化相同且所有子节点均相同的节点进行合并，完成本体变化生成图的创建。

例如，在例 1 的本体中删除概念 *PhD Student* 时，依据该算法所创建的变化生成图如图 2(b)所示。

4.2 变化生成图的求精

在创建变化生成图后，需要进行进一步的精化，这主要包括：

1)变化生成图中，如果两个类型相同的“*empty*”变化节点具有父子关系，那么需要将二者合并为一个“*empty*”变化节点；

2)变化生成图中可能会出现有向的闭环。很显然，为了防止变化之间出现依赖死锁，应该进行调整，去掉闭环。

这里，本文主要对如何去掉闭环进行讨论，首先我们给出闭环的定义。

[4]参见文[2]中的表格 6。

定义 4(闭环) 变化生成图 CGG 中的闭环 $ChCircle$ 的定义为

$ChCircle=\{chNode_i \mid chNode_i \in CHNodes, 1 \leqslant i \leqslant |ChCircle|\}$

其中

1) $\forall chNode_i \in ChCircle, 1 \leqslant i < |ChCircle|, (chNode_i, chNode_{i+1}) \in E$

2) $\forall chNode_j, chNode_k \in ChCircle, 1 \leqslant j,k < |ChCircle|, chNode_j \neq chNode_k$

3) $(chNode_{|ChCircle|}, chNode_1) \in E$

定义 5(不可删节点) 我们称变化生成图 CGG 的节点 $chNode$ 为不可删节点,如果 $chNode$ 为根节点,或者存在一个从根节点到该节点的路径,并且该路径中的节点除了 $chNode$ 外均是"*and*"类型节点。如果一个节点不是不可删节点,那么我们称之为可删节点。

调整闭环最主要的问题在于不能影响变化生成图的表现内容,这就要求在对变化生成图进行删减时,只能考虑移除变化生成图中的可删节点。下面,本文给出一个算法来移除变化生成图中的闭环。

算法 2 移除本体变化生成图中的闭环

输入:变化生成图 CGG 中的闭环 $chCircle$;

输出:闭环移除结果。

①在闭环 $chCircle$ 中,找出深度最浅的本体变化节点 CN,设 $CN_1=CN$。

②如果 CN_1 是可删节点,那么进行步骤 2.1～2.4 的处理;否则,转到步骤 3。

2.1 在 CGG 中删除 CN_1;

2.2 对 CN_1 所有的父节点进行分析。如果某个父节点 CN_2 是"*and*"类型节点,那么对 CN_2 执行步骤 2.1～2.3;如果某个父节点是"*or*"型节点,并且在 CN_1 被删除后只剩下一个子节点,那么修改该父节点类型为"*and*";

2.3 对 CN_1 所有的子节点进行分析。如果某个子节点 CN_3 是孤儿节点(父节点均被删除),那么对 CN_3 执行步骤 2.1～2.3。

2.4 转到步骤 4。

③考虑 CN_1 的父节点 CN_4,如果 CN_4 与 CN 等价,说明该闭环中没有可删节点,该闭环移除失败,返回 $FALSE$;否则,设 $CN_1=CN_4$,转步骤 2;

④闭环移除成功,返回 $TRUE$。

由上述的算法可知,如果最终无法破除所有的闭环,那么该变化生成图中将会存在本体变化依赖死锁,因此也就无法从该变化生成图中得到一个有限的本体变化序列,知识工程师的本体改进意图也就无法实现。

5 变化生成图的冲突消解

在多个知识工程师协作进行本体进化时,他们的进化意图可能存在着潜在的冲突,如果不进行冲突消解,那么很可能会使某些请求执行失败,或者带来本体的不一致。由于变化生成图反映了知识工程师进行本体进化的意图以及所可能引发的额外变化,因此本体变化生成图之间的冲突直接反映了知识工程师本体改进意图的冲突。所以,可以通过对变化生成图进行冲突消解来解决知识工程师本体改进意图之间的冲突。下面,本文首先对本体变化生成图的冲突进行定义。

定义 6(变化冲突矩阵) 变化冲突矩阵是一个二维对称矩阵,其行与列均代表了基本的本体变化,矩阵中的值表示了对应行与列中变化之间发生冲突的条件。如果两个变化在任何情况下均不会发生冲突,那么对应矩阵中的值为 0。

表 1 给出了部分变化冲突矩阵,根据变化冲突矩阵,我们可以很容易得到变化之间发生冲突的条件。例如,当 $a \equiv c$ 时,本体变化 $RemoveConcept(a)$ 与 $AddSubConcept(b, c)$ 很显然是互相冲突的。

定义 7(冲突的变化生成图) 我们称变化生成图 CGG_1 和 CGG_2 是冲突的,如果

$\exists chNode_i \in CGG_1, chNode_j \in CGG_2, conflictCondition[chNode_i, change][chNode_j.change]=TRUE$

也就是说,如果两个变化生成图中分别存在两个变化节点,并且这两个节点中所包含的变化满足变化冲突矩阵中本体变化冲突的条件,那么这两个变化生成图就是互相冲突的。

表 1 变化冲突矩阵(部分)

	AddSubConcepg (*b*,*c*)	*AddASubProperty* (*p*3,*p*4)	*AddPropertyDo* *main*(*p*3,*c*)	*RemovePropert* *yDomain*(*p*3,*c*)
RemoveConcept (*a*)	$a \equiv c$	0	$a \equiv c$	0
AddSubConcept (*a*,*b*)	$(b,c) \in Hc^* \wedge (d,a) \in Hc^*$	0	0	0
RemoveProperty (*p*1)	0	$p1 \equiv p4$	$p1 \equiv p3$	$p1 \equiv 3$
AddPropertyDomain (*p*1,*a*)0	0	0	$(a,c) \in Hc^* \vee (c,a) \in Hc^*$	0

这里需要注意的是:1)包含相同变化的两个本体变化节点不冲突;2)包含空变化的节点与其他任何节点均不冲突。

下面,本文给出一个算法来对两个本体变化生成图之间的冲突进行消解。

算法 3 对两个本体变化生成图进行冲突消解

输入:变化生成图 CGG_1 和 CGG_2;

输出:冲突消解结果。

①对于 CGG_1 中任意的变化节点 $chNode_i$、CGG_2 中任意的变化节点 $chNodej$,如果 $conflictCondition[chNode_i, change][chNode_j, change]=TRUE$,那么将节点对 $(chNode_i, chNode_j)$ 添加到冲突节点对列表 $conflictPairList$ 中。

②对 $conflictPairList$ 中的节点对 $(chNode_1, chNode_2)$ 进行遍历,并对每个节点对执行步骤 2.1～2.2 的处理。如果遍历完成,转到步骤 3。

2.1 如果在 CGG_1 中,$chNode_1$ 是不可删节点,那么转到步骤 2.2,否则对 $chNode_1$ 做如下处理:

2.1.1 在 $chNode_1$ 所属的变化生成图中删除 $chNode_1$，并将被删除的节点加入删除节点列表 *nodeList*；

2.1.2 对 $chNode_1$ 所有的父节点进行分析。如果某个父节点 $chNode_4$ 是“*or*”类型节点，并且在删除 $chNode_1$ 后，$chNode_4$ 只剩下一个子节点，那么修改 $chNode_4$ 的类型为“*and*”；如果某个父节点 $chNode_3$ 是“*and*”类型节点，那么对 $chNode_3$ 执行步骤 2.1.1～2.1.3；

2.1.3 对 $chNode_1$ 的子节点进行分析，如果某个子节点是孤儿节点(父节点均被删除)，那么对该子节点执行步骤 2.1.1～2.1.3；

2.1.4 对于任意的 $node \in nodeList$，对于任意的 $(chNode_k, chNode_m) \in conflictPairList$，如果 *node* 与 $chNode_k$ 或者 $chNode_m$ 等价，那么在 *conflictPairList* 中删除节点对 $(chNode_k, chNode_m)$，转到步骤 2。

2.2 如果在 CGG_2 中，$chNode_2$ 是不可删节点，那么转到步骤 2，否则对 $chNode_2$ 执行步骤 2.1.1～2.1.3。

③在对 *conflictPairList* 遍历处理完成后，如果该节点对列表不为空，即存在不可删除的冲突节点对，那么两个变化生成图冲突消解失败，返回 *FALSE*；否则，冲突消解成功，返回 *TRUE*，处理后的 CGG_1 和 CGG_2 就是两个没有冲突的本体变化生成图。

在第 7 节中，我们将举例说明如何根据算法 3 对两个变化生成图进行冲突消解。

6 生成变化序列

在创建变化生成图并对变化生成图进行冲突消解后，知识工程师可以得到一个与其他工程师不会产生冲突的本体变化生成图，利用这个变化生成图产生的任何变化序列都可以实现知识工程师的本体改进意图，并且保持本体的一致性。

由于变化生成图是一种与或图，而且在对变化生成图进行求精(4.2 节)后，图中的有向闭环已经被移除，因此，变化生成图实际上是一个不含圈的与或图，所以，我们可以采用经典的 AO* 算法对变化生成图进行遍历。由于篇幅原因，这里我们对遍历算法不进行详细的说明，AO* 算法的详细情况可参见文[9]。

生成变化序列可分为 3 个步骤：

1)使用 AO* 算法对变化生成图进行分析，得到一个可解的子图；

2)使用深度优先的遍历算法[10]对上述子图进行遍历，得到一个节点列表，并从节点列表中删除包含“*empty*”变化的变化节点；

3)将变化节点中所包含的本体变化按照顺序抽取出来，形成一个本体变化序列。

根据前面的分析可知，根据上述 3 个步骤得到的本体变化序列能够代表知识工程师的本体改进意图，在执行上述变化序列后，本体的一致性能够得到保持，同时不会与其他知识工程师产生冲突。也就是说，本文所提出的协同进化方法能够发现并解决多个知识工程师本体进化意图之间的冲突，从而能够支持本体的协同进化。

7 一个实例

下面，本文将通过一个实例来对文中所提方法进行验证。

假设有两个知识工程师同时对例 1 所示的本体进行进化操作，其中工程师 A 希望向本体中添加一个事实：*teach*(*peter*, *tom*)(*peter* 是 *tom* 的老师)，同时工程师 B 则需要删除概念“*PhD Student*”，那么根据算法 1 得到的代表二者本体改进意图的变化生成图分别如图 2(a)、图 2(b)所示。根据定义 7 可知，由于 *AddPropertyInstance*(*teach*, *peter*, *tom*)和 *RemoveInstance*(*tom*)互相冲突，因此前面这两个变化生成图存在着潜在的冲突，所以需要对他们进行冲突消解。利用算法 3 对二者进行处理，可得到两个无冲突的本体变化生成图，即图 0(a)和图 0(c)。

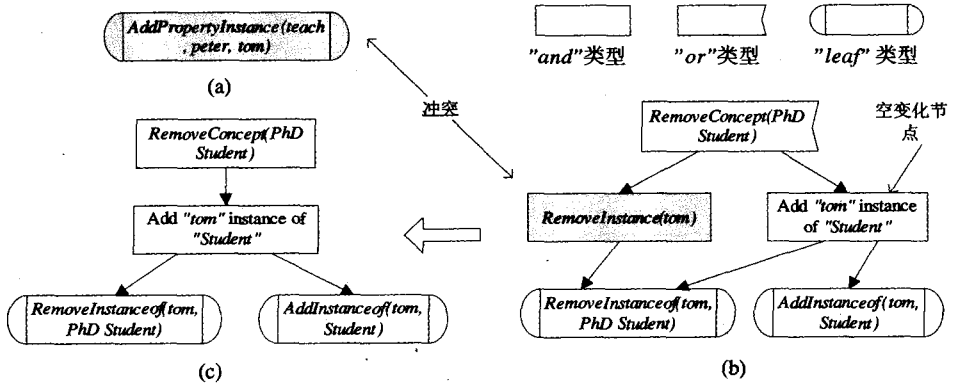

图 2 变化生成图

(a)向例 1 本体中添加事实公理 *teach*(*peter*,*tom*)所对应的变化生成图；(b)删除概念“*PhD Student*”所对应的变化生成图；(c)完成冲突消解后，删除概念“*PhD Student*”所对应的变化生成图。

最后，利用本体变化序列生成方法，可以从图 2(a)和图 2(c)中得到两个本体变化序列，即{*AddPropertyInstance*(*teach*, *peter*, *tom*)}和{*RemoveConcept*(*PhD Student*), *RemoveInstanceof*(*tom*, *PhD Student*), *AddInstanceof*(*tom*, *Student*)}。从上述两个变化序列可知，在执行完这两个序列后，两个知识工程师的本体改进意图均得到了实现，同时本体的一致性也得到了保持，更为重要的是，二者之间潜在的冲突得到了发现和解决，从而达到了本体协同进化的目标。

结束语 随着本体应用的深入，如何对本体进行改进、如何实现本体协同进化成为了一个亟待解决的问题，也是当前的一个研究热点。在本体协同进化中，知识工程师的本体改进意图可能存在着外在或潜在的冲突，由于本体进化本身的复杂性，现有的大部分方法都不能及时发现这些冲突，因此也就很难在本体协同进化环境中保持本体的一致性。

针对这个问题，本文对文[2, 6]所提出的 OWL 本体进化方法进行了改进，扩展了方法的第二阶段(本体变化语义分

析阶段),使之能够适应本体协同进化环境。本文引入了一种本体变化生成图来表示知识工程师的本体改进意图,通过寻找变化生成图之间的冲突来发现知识工程师本体改进意图之间潜在的冲突,并通过对变化生成图进行冲突消解来解决知识工程师之间的冲突,从而使得多个本体改进意图能够共存,最终达到了本体协同进化的目标。在下一步的工作中,我们将以本文所提出的算法为基础,设计并实现一个本体协同进化系统,从而进一步完善本体协同进化的研究。

参 考 文 献

1 Fensel D. Ontologies: dynamics networks of meaning. In: Proceedings of the 1st Semantic web working symposium, Stanford, CA, USA, 2001
2 Stojanovic L. Methods and Tools for Ontology Evolution: [PhD thesis]. 2004: University of Karlsruhe, 2004
3 Pierre G, Steen M V. Dynamically selecting optimal distribution strategies on web documents. IEEE Transaction on Computers, 2002, 51(6): 637~651
4 Horrocks I, Patel-Schneider P F, Harmelen F V. From SHIQ and RDF to OWL: The Making of a Web Ontology Language. Journal of Web Semantics, 2003, 1(1)
5 Stojanovic L, et al. User-driven Ontology Evolution Management. In: European Conf Knowledge Eng. and Management (EKAW 2002). Springer-Verlag, 2002
6 Haase P, Stojanovic L. Consistent Evolution of OWL Ontologies. In: Proceedings of the Second European Semantic Web Conference. Heraklion, Greece, 2005
7 Plessers P, Troyer O D. Ontology Change Detection Using a Version Log. In: 4th International Semantic Web Conference, ISWC 2005. Galway, Ireland, 2005
8 Horrocks I, Patel-Schneider P F. Reducing OWL Entailment to Description Logic Satisfiability. Journal of Web Semantics, 2004, 1(4)
9 Yaorui L. Introduction to Artificial Intelligence. Beijing: Tsinghua University Press, 1997
10 Weimin Y. Data Structure. Beijing: Tsinghua University Press, 1993

(上接第 175 页)

量会大大增加。所以在计算属性相似度时,可以先依据机器学习方法计算出属性的信息增益[8],并以此为依据来确定各个属性的优先级。最后,只选取几个信息增益大的属性进行相似度的计算,这样可以减少计算量。

2.2.3 基于关系计算概念相似度

本体中的概念之间都存在一定的关系,概念的关系对概念的描述也具有重要的作用。关系有关系名称、关系类型、关系实例数据等要素,因此判断两个关系是否相似主要从这 3 个要素的相似度进行考虑。

设概念 A 的关系为 r_i,概念 B 的关系为 s_j,两个关系间的相似度记为 $RSim(r_i, s_j)$。关系相似度计算公式如下:

$$RSim(r_i, s_j) = W1 * Sim(r_{i_{name}}, S_{j_{name}}) + W2 * Sim(r_{i_{type}}, S_{j_{type}}) + W3 * Sim(r_{instance}, S_{j_{instance}}) \quad (10)$$

其中 $w1$、$w2$、$w3$ 是权重,代表关系名称、类型、数据对关系相似度计算的重要程度,$w1+w2+w3=1$。

关系名称、关系类型本身都是字符串,因此可以采用字符串相似度计算方法进行判定。对于关系的实例数据,也可以采用基于实例的方法进行计算。为此把相似度的定义扩展到关系相似度,并用基于实例的方法来计算两个关系基于实例的相似度。例如两个关系 r_i 和 s_j,它们之间的相似度为 $RSim(r_i, s_j)$,即 $RSim(r_i, s_j) = P(r_i \cap s_j)/P(r_i \cup s_j)$。其中,$r_i$ 是本体 O_1 中的关系,$r_i \subseteq A_1 \times A_2 \cdots \times A_m$,$A_k(k=1,\cdots,m)$是本体 O_1 中的概念;s_j 是本体 O_2 中的关系,$s_j \subseteq B_1 \times B_2 \cdots \times B_n$,$B_k(k=1,\cdots,n)$是本体 O_2 中的概念。当 m 和 n 都等于 2 时,r_i 和 s_j 都是一个二元关系。概念中的一个关系会连接两个概念中的所有实例。对于关系 $r_i(A_1, B_1)$和 $s_j(A_2, B_2)$,利用它们对应的概念实例进行相似度计算。

设概念 A 和概念 B 之间共计算出 L 个 $RSim(r_i, s_j)$,并设置相应的权值 $w^l_{relation}$。概念 A 和概念 B 基于关系的相似度计算公式为

$$Sim_{relation}(A,B) = \frac{\sum_{k=1}^{l} W^{tu}_{k\,relation} RSim(r_i, S_j)}{\sum_{k=1}^{l} W^{k}_{relation}} \quad (11)$$

2.2.4 合并概念相似度

把基于实例、基于属性、基于关系计算得到的概念相似度进行合并,得到最后的概念相似度 $Sim(A,B)$,计算公式如下:

$$Sim(A,B) = w_{instance} Sim_{instance}(A,B) + w_{attribute} Sim_{attribute}(A,B) + w_{relation} Sim_{relation}(A,B) \quad (12)$$

其中,$w_{instance} + w_{attribute} + w_{relation} = 1$,权值的具体设置根据具体环境由用户确定。

3 性能分析

本文提出的综合的概念相似度计算方法通过概念名称的相似性过滤相关本体中某概念的候选概念集,可以大大减少相似度计算的次数。对于每一对基本相似的概念,采用了综合的相似度计算方法,虽然比单纯的基于实例的相似度计算公式计算量更多,但对于概念相似度的计算更能反映概念之间的相似关系。通过选择合适的权值,可以确保概念相似度的计算更全面、更准确。当然,在概念相似度的计算过程中存在大量的权值设定,可能对性能存在一定的影响。对于权值的选取,可以通过神经网络学习技术进行修正[7]。

结束语 本体是人和机器、程序间知识交流的语义基础,使用本体的目的是为了知识的共享和重用。然而由于各自建立的局限性和不同本体之间存在个体丰富性,本体间也就不可避免地存在着语义冲突,因而解决不同本体的概念间的语义冲突的本体映射成为本体研究领域的重要课题。本文针对目前本体映射中概念相似度计算所存在的问题,提出了一种综合的相似度计算方法。首先只对最相关的概念对计算相似度,减少相似度的计算;并从 3 个方面对概念进行相似度的计算,确保计算效果更加全面,计算结果更加准确。但是,计算过程中各个权值的设定还只是根据经验来给定,有一定的误差,应该对权值的设定做进一步的研究。另外,属性增益的计算在一定程度上增加了计算量,可以考虑采用其它更有效的机器学习方法来确定属性的优先级。

参 考 文 献

1 Doan A H. Learning to Map between Structured Representations of Data: [Ph thesis]. University of Washington, 2002
2 Noy N, Musen M. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In: Proc. AAAI2000. AAAI Press, 2000
3 Noy N F, Musen M A. SMART: Automated support for ontology merging and alignment [J]. In: Twelth Workshop on Knowledge Acquisition, Moeling, and Management, Banff, Canada, 1999
4 Dou D, McDermott D, Qi P. Ontology Translation by Ontology Merging and Automated Reasoning: [Ph thesis]. University of Yale, 2004
5 郑丽萍. 本体映射的研究. [硕士学位论文]. 山东科技大学, 2005
6 Kong C Y, Wang C L, Lau F C M. Ontology Mapping in Pervasive Computing Environment . URL. http://www. csis. hku. hk/~clwang/papers/EUC2004-laurel. pdf
7 蔡自兴, 徐光佑. 人工智能及其应用. 清华大学出版社, 2001
8 范明, 孟晓锋. 数据挖掘概念与技术. 机械工业出版社, 2001