文章编号: 1001- 2486(2009) 03- 0097- 07

一种基于规则的 OWL-S 本体语法一致性维护方法*

鲍爱华, 张维明, 袁金平, 姚 莉
(国防科技大学 信息系统与管理学院, 湖南 长沙 410073)

摘 要: OWL-S 本体语法一致性维护是语义 Web 服务维护工程中需要重点研究的问题。提出一个一致性维护框架指导知识工程师进行 OWL-S 本体语法一致性维护, 定义并抽象出 OWL-S 本体基本变化, 提出变化关联矩阵分析基本变化间的依赖关系, 基于关联矩阵提出额外变化序列生成算法分析基本变化对全局的影响, 并分析了算法时间复杂性。对 OWL-S 本体语法一致性规则进行了形式化定义, 使机器能够自动检测这些规则, 辅助知识工程师完成 OWL-S 本体语法一致性维护工作。

关键词: OWL-S; 本体维护; 语法一致性; 规则; 关联矩阵; 变化生成算法
中图分类号:TP393 文献标识码: A

A Rule-based Approach for Maintenance of OWL-S Ontology Syntactic Consistency

BAO Ai-hua, ZHANG Wei-ming, YUAN Jin-ping, YAO Li
(College of Information System and Management, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: The maintenance of OWL-S ontology syntactic consistency is very important in the evolution of semantic web services. In this paper, the differences between OWL-S and domain ontology are analyzed, and a maintenance framework is proposed to guide engineers to maintain the syntactic consistency of OWL-S. After the elementary changes of OWL-S are defined and abstracted, the change relation matrix is given to define the dependency between them, then, an algorithm to generate potential changes based on relation matrix is shown, and the time complexity of this algorithm is also analyzed. The syntactic consistency rules of OWL-S are divided into two types: mandate rules and optional rules while the descriptions and formal definitions of these rules are also given. In so doing, the agent can check these rules automatically, and help engineers to maintain the syntactic consistency of OWL-S ontology.

Key words: OWL-S; ontology maintenance; syntactic consistency; rule; relation matrix; ontology change generation algorithm

语义 Web 服务[1] 是语义 Web 中的研究热点和重点, 它将语义与 Web 服务进行封装, 使 Web 服务成为计算机可理解、Agent 可处理和对用户透明的计算实体, 从而为用户提供自动的 Web 服务。在相关工作中, DARPA 组织所提出的 OWL-S 本体规范是比较典型的研究成果[2]。

语义 Web 是一个动态变化的复杂环境, 其中的 Web 服务需要随着领域知识和业务逻辑的变化而进行调整, 具体到技术层面就是要求对描述语义 Web 服务的 OWL-S 本体进行维护, 才能适应外部环境的变化。OWL-S 本体是一个结构复杂的复合体, 局部的修改会对本体的其他部分造成影响, 因此在对 OWL-S 本体进行维护时不仅需要考虑实现知识工程师的维护意图, 还需要从系统工程的角度分析维护操作的潜在效果, 在此基础上维护 OWL-S 本体的语法一致性。

目前针对 OWL-S 本体语法一致性维护的研究工作并不多见, 相关的研究通常散见于领域本体的演化中。Stojanovic 等[3-4] 定义了 KAON 本体变化操作, 采用演化策略和变化依赖图对 KAON 本体的语法一致性进行检测和修复。Haase 等[5] 考虑了 OWL Lite 本体的结构一致性问题, 提供了一套判定策略来处理 OWL Lite 本体中超出其语言表达能力的问题。但由于 OWL-S 本体论域的特殊性, 在其语法一致性维护的问题上, 目前的工作还需要进一步研究。

* 收稿日期:2008- 12- 05
基金项目:国家自然科学基金资助项目(70371008)
作者简介:鲍爱华(1981—), 男, 博士生。

本文以Web服务重组[6-7]为背景,对OWL-S本体的粗粒度修改操作进行了定义,并以OWL-S规范为基础对其语法一致性规则进行了描述和抽象,同时提出了一个变化关联矩阵来分析变化之间的依赖关系,并基于关联矩阵对OWL-S本体变化的潜在效果进行了分析。另外,本文也给出了一个OWL-S本体语法一致性维护框架来指导知识工程师对OWL-S本体进行修改和语法一致性维护。

1 相关概念

为了方便对后续的本体变化以及语法一致性规则进行严格的描述,首先依据OWL-S语法规范对相关概念进行明确定义。

定义1(语义Web服务) 语义Web服务 SWS 可以结构化描述为: $SWS:=(PF, SM, SG)$,其中:(1) PF 是服务概貌($Profile$)的集合,用于描述服务的功能;(2) SM 是 SWS 的过程模型($Model$),用于描述服务与请求者的交互模型;(3) SG 是 SWS 的访问基础($Grounding$)集合,说明了agent如何访问 SWS。

定义2(服务概貌) 服务概貌 $Profile$ 可以表示为四元组 $<Input, Output, Precondition, Result>$,其中 $Input$ 表示服务的输入集合,$Output$ 表示服务的输出集合,$Precondition$ 表示服务的前提条件集合,$Result$ 表示服务的条件输出结果集合。

定义3(过程) 过程模型中的过程 $Process$ 可以表示为四元组 $<Participant, Input, Output, Precondition, Result>$,其中 $Participant$ 表示过程的参与者集合,$Input$ 表示过程输入集合,$Output$ 表示过程输出集合,$Precondition$ 表示过程前提条件集合,$Result$ 表示服务的条件输出结果集合。

定义4(过程模型) 语义Web服务 SWS 的过程模型 SM 可以结构化定义为: $SM:=(P, C; CO, F, PER, DF)$,其中:

(1) P 表示 SM 中过程的集合,$P=P_a \cup P_b \cup P_c$,其中 P_a、P_b 和 P_c 分别表示原子过程、抽象过程和组合过程集合;

(2) C 代表 SM 中控制构造子集合,$C=C_{seq} \cup C_{split} \cup C_{splitjoin} \cup C_{anyorder} \cup C_{choice} \cup C_{ite} \cup C_{iw} \cup C_{ru} \cup C_{per}$,其中9种符号分别代表 $OWL\text{-}S$ 过程模型中9种类型控制构造子集合;

(3) $CO \subseteq P_c \times C$ 代表 SM 中组合过程与控制构造子构建关系集合,$\forall cp \in P_c, c \in C, <cp, c> \in CO \Rightarrow ComposedOf(cp, c)$;

(4) F 是 SM 中控制构造子集间的映射函数,表示控制构造子之间的组成关系。

$$\forall c \in C, F(c)=\begin{cases} \cong, & c \in C_{per} \\ (A, \prec), & c \in C_{seq} \\ \{c_{then}, c_{else}\}, & c \in C_{ite} \\ A, & c \in C \setminus (C_{per} \cup C_{seq} \cup C_{ite}) \end{cases}$$

,其中 $A \in 2^C$,$\prec$ 是建立在 A 上的偏序关系,$\forall c_1, c_2 \in A, c_1 \prec c_2$ 表示在序列 A 中 c_1 的执行优先于 c_2;c_{then} 和 c_{else} 分别表示 $If\text{-}Then\text{-}Else$ 中 $Then$ 和 $Else$ 所对应的控制构造子;

(5) $PER \subseteq C_{per} \times P$ 是 SM 中 $Perform$ 类型控制构造子与过程间的调用执行关系的集合;

(6) $DF=DF_{vd} \cup DF_{bind}$ 是 SM 中数据流集合,其中: DF_{vd} 是赋值型数据流,形式上 $DF_{vd}:=\{<val, per, i> \mid per \in PER \wedge i \in per.P.Input\}$; DF_{bind} 是绑定型数据流,形式上 $DF_{bind}:=\{<per_1, o_1, per, i> \mid per_1, per \in PER \wedge i \in per.P.Input \wedge o_1 \in per_1.P.Output\}$。

定义5(访问基础) 服务访问基础 $Grounding$ 可以结构化表示为三元组 $<p, InputMap, OutputMap>$,其中(1) $p \in P$ 表示该 $Grounding$ 所对应的过程;(2) $InputMap:=\{<wsdlInput, i> \mid i \in cp.Input\}$ 表示输入映射集合,其中 $wsdlInput$ 表示WSDL文件中定义的服务输入;(3) $OutputMap:=\{<o, wsdlOutput> \mid o \in cp.Output\}$ 表示输出映射集合,其中 $wsdlOutput$ 表示WSDL文件中定义的服务输出。

定义6(OWL-S本体变化) OWL-S本体变化 $Ch \in OntoCh$ 是OWL-S本体之间的映射,即 $Ch: O \overrightarrow{\ } O'$。形式上,$Ch=<name, args, preconditions, postconditions>$,其中:(1) $name$ 是OWL-S本体变化的名称;(2) $args$ 是OWL-S本体变化的参数;(3) $preconditions$ 是本体变化得以实施的前提条件;(4)

postconditions 是本体变化实施后所必须满足的条件, 表示该变化对本体修改的结果。如果变化 *ch* 实施后使得 OWL-S 本体中的元素集扩大, 那么该变化称为增性变化, 否则称为减性变化。

根据 OWL-S 语法规范, 以 OWL-S 基本元素为基础抽象出 22 种增性变化和 19 种减性变化, 如表 1 所示。由于篇幅限制, 本文忽略基本变化的前提条件和后继效果, 相关内容可参见文献[9]。

表 1 OWL-S 本体基本变化

Tab. 1 Elementary changes of OWL-S ontology

OWL-S 本体元素		增性变化	减性变化
Service Prfiles		*AddPrfile*(*pf*)	*RemovePrfile*(*pf*)
		AddPFInput(*pf*, *i*)	*RemovePFInput*(*pf*, *i*)
		AddPFOutput(*pf*, *o*)	*RemovePFOutput*(*pf*, *o*)
		AddPFPrecond(*pf*, *pre*)	– –
		AddPFResult(*pf*, *r*)	*RemovePFResult*(*pf*, *r*)
Process Model	*Process*	*AddProcess*(*p*)	*RemoveProcess*(*p*)
		AddParticipant(*p*, *par*)	*RemoveParticipant*(*p*, *par*)
		AddInput(*p*, *i*)	*RemoveInput*(*p*, *i*)
		AddOutput(*p*, *o*)	*RemoveOutput*(*p*, *o*)
		AddPrecondition(*p*, *pre*)	– –
		AddResult(*p*, *r*)	*RemoveResult*(*p*, *r*)
	Control Construct	*AddCtrlConstruct*(*c*)	*RemoveCtrlConstruct*(*c*)
		AddCtrlCstrCond(*c*, *cond*)	– –
		AddComposedOf(*p*, *c*)	*RemoveComposedOf*(*p*, *c*)
		AddPerform(*per*, *p*)	*RemovePerform*(*per*, *p*)
		AddSubCtrlCstr(*c*, *c*′)	*RemoveSubCtrlCstr*(*c*, *c*′)
		AddSeqCCRel(*c*, *c*′)	*RemoveSeqCCRel*(*c*, *c*′)
		AddElseCtrlCstr(*c*, *c*′)	*RemoveElseCtrlCstr*(*c*, *c*′)
	Data Flow	*AddDataFlow*(*df*)	*RemoveDataFlow*(*df*)
Service Groundings		*AddGrounding*(*g*, *p*)	*RemoveGrounding*(*g*)
		AddInputMap(*g*, *i*, *wi*)	*RemoveInputMap*(*g*, *i*)
		AddOutputMap(*g*, *o*, *wo*)	*RemoveOutputMap*(*g*, *o*)

2 OWL-S 本体语法一致性维护框架

总体而言, 本文所提出的 OWL-S 本体语法一致性维护框架(图 1)分 5 个步骤实现 OWL-S 本体的维护和语法一致性检查工作:

(1)前提条件检查。本步骤采用 OWL-S 基本变化(表 1)表示维护需求, 然后对变化操作的前提条件进行检查, 如果不成立, 则向知识工程师反馈错误信息。

(2)额外变化生成。本步骤首先基于关联矩阵对变化间的触发关系及条件进行分析, 并在此基础上采用变化序列生成算法对 OWL-S 本体变化的潜在触发变化进行分析, 得到一个新的变化序列。

(3)维护确认。根据步骤(2)所得到的变化序列, 知识工程师根据其维护需求对该

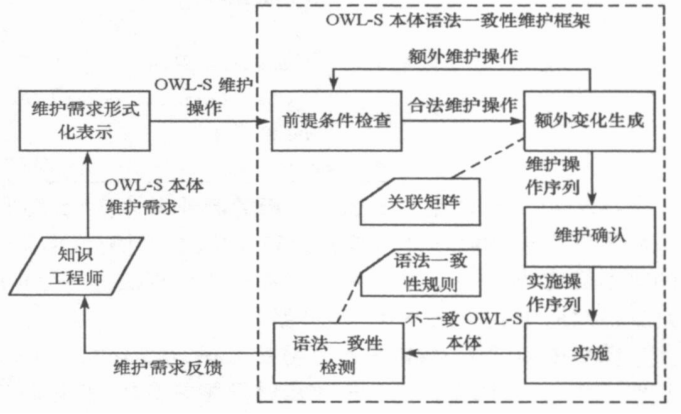

图 1 OWL-S 本体语法一致性维护框架

Fig. 1 OWL-S syntactic consistency maintenance framework

序列进行确认, 如果发现与其需求相悖的变化, 那么需要回溯, 对其维护需求进行重新分析。

(4) 变化实施。在本步骤中, Agent 根据维护操作序列自动对 OWL-S 本体进行修改。

(5) 语法一致性检测。本步骤主要基于 OWL-S 本体语法一致性规则对本体进行检测, 如果发现语法不一致, 则反馈给知识工程师, 产生新的维护需求, 否则语法一致性维护操作完成。

根据上述语法一致性维护框架, 知识工程师能将其 OWL-S 本体维护需求进行形式化表示, 并在机器的辅助下分析维护操作所引起的潜在变化, 从而系统掌握该变化所造成的影响; 在变化实施后机器对修改后本体的语法一致性进行检查, 并通过反馈使知识工程师产生新的本体维护需求, 维持本体的有效性。

3 基于关联矩阵的 OWL-S 本体变化序列生成算法

定义 7(变化关联矩阵) OWL-S 本体变化关联矩阵 CRM 是一个二维矩阵, 其行与列分别代表 OWL-S 本体基本变化(表 1) , 矩阵中的元素代表对应变化触发关系的条件, 即 $\forall ch_i, ch_j \in OntoCh$, $CRM[ch_i][ch_j] = true \Rightarrow ch_i \overrightarrow{\ } ch_j$。

变化关联矩阵 CRM 的部分内容如表 2 所示。由于 CRM 对任意两个基本变化间的触发条件进行了定义, 因此在对某个 OWL-S 本体变化所触发的额外变化进行分析时, 只要依次对 CRM 中的相关元素进行真值判断, 就可以得出该变化所触发的额外变化。在此基础上, 对这些额外变化进行嵌套分析, 就可以得到所有产生的额外变化序列。表 3 中的 CGA 算法对这一流程进行了描述。

表 2 变化关联矩阵(CRM)

Tab. 2 Part of change relation matrix (CRM)

	RemoveGrounding(g)	*RemoveDataFlow*(df)	*RemoveInput*(p_2, i_2)	*RemovePeform*(per_2, p_2)
AddProcess(p_1)	0	0	0	0
RemoveProcess(p_1)	$p_1 = g.p$	0	$p_1 = p_2$	$p_1 = p_2$
RemoveInput(p_1, i_1)	0	$i_1 = df.i$	0	0
RemovePerform(per_1, p_1)	0	$per_1 = df.per$	0	0

表 3 额外变化序列生成算法 CGA

Tab. 3 Additional change list generation algorithm (CGA)

输入: 待分析的 OWL-S 本体变化 ch; 生成的变化序列 *addList*

ChangeGeneration(ch, *addList*)

```
1  for all ch_i ∈ Ch do
2    if CRM[ch][ch_i] = false then
3      continue
4    else
5      addList.add(ch_i)
6      ChangeGeneration(ch_i, addList)
7    end if
8  end for
```

虽然 CGA 算法是一个简单可行的额外本体变化序列生成方法, 但该算法是嵌套计算过程, 因此在采用该算法对 OWL-S 本体变化进行分析时的计算性能是一个必须考虑的问题。这里, 本文对最为复杂的基本变化 *RemoveCtrlConstruct*(c) 在采用该算法进行分析时的算法复杂性进行分析, 复杂性的衡量指标是分析时所生成的额外本体变化数量。

图 2 中列举了 *CRM* 中所有与 *RemoveCtrlConstruct* 相关的变化, 其中背景透明的变化不会进一步触发其他变化, 而其他变化则会进一步触发相关变化, 变化间连线上的数字则表示触发的权重。假设 ai 为过程模型中过程平均包含的输入参数数目, ao 为过程平均包含的输出参数数目, n 代表控制构造子的子构造子平均数目, m 代表控制构造子 c 到叶构造子的平均深度, 那么采用表 3 中的算法对该变化进行分析时所产生的最大额外变化数目 ACC 为

$$ACC=[3+(ai+ao)+n+1]\times[1+(n+1)+(n+1)^2+\cdots+(n+1)^{m-1}]\approx[(ai+ao)+n+4]\times(n+1)^m$$

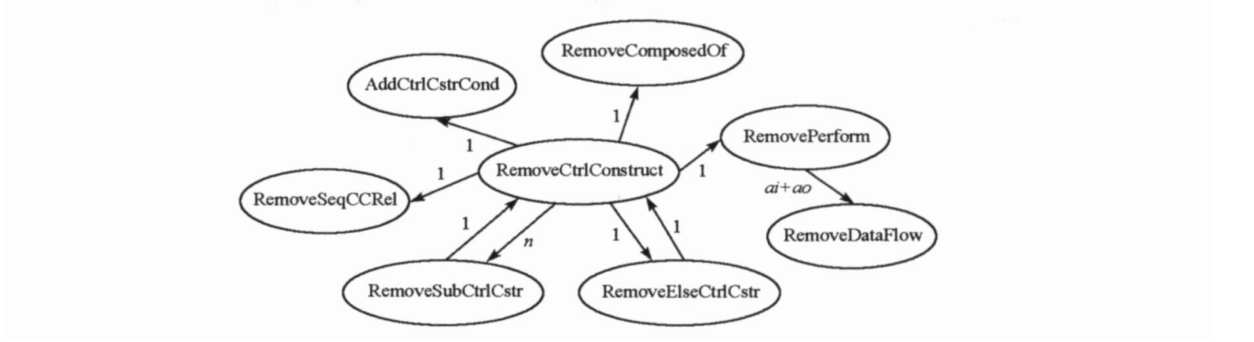

图 2 *RemoveCtrlConstruct* 关联变化图

Fig. 2 Related change graph of *RemoveCtrlConstruct*

如果删除的控制构造子 c 是 *Perform* 类型, 那么不会触发 *RemoveSubCtrlCstr* 等控制构造子组合层次的变化, 因此在上式中我们忽略该类型控制构造子删除变化的影响, 忽略 $(ai+ao)$, 即 $ACC\approx(n+4)\times(n+1)^m\approx(n+1)^m$。一般来说, 控制构造子平均所包含的子构造子的数目是有限的(如小于 5), 否则可以拆分为多个控制构造子; 控制构造子的平均深度也是有限的(如小于 10), 否则可以组合为多个组合服务。因此, 在对变化 *RemoveCtrlConstruct*(c) 的额外变化序列进行计算时, 所产生的额外变化数目是可控的。

4 面向服务的 OWL-S 本体语法一致性规则

定义 8(语法一致性模型) OWL-S 本体语法一致性模型 *GramCM* 可以定义为: $GramCM=MR\cup OR$, $MR=\{MR_i\}$, $1\leqslant i\leqslant 11$, $OR=\{OR_j\}$, $1\leqslant j\leqslant 3$。其中 MR_i 是强制一致性规则, 所有的 OWL-S 本体必须满足; OC_j 是可选一致性约束, OWL-S 本体在修改后可以选择性满足。

定义 9(语法一致性) OWL-S 本体被称为语法一致的, 当且仅当其满足 *GramCM* 模型。

4.1 强制一致性规则

MR_1: 实例唯一性规则。在 OWL-S 本体中, 无层次关系概念的实例集是不相交的。其形式化描述为: $\forall C_1, C_2\in C, (C_1, C_2)\notin Hc\Rightarrow\neg\exists i, C_1(i)\wedge C_2(i)$ ①。

MR_2: 原子过程参与者规则。在 OWL-S 过程模型中, 任意的原子过程包含且仅包含客户端和服务端两个参与者。该规则的形式化描述为: $\forall ap\in P_a, |ap.Participant|=2$。

MR_3: 组合过程构建规则。在 OWL-S 过程模型中, 任意的组合过程都必须由一个控制构造子构建而成。该规则的形式化描述为: $\forall cp\in P_c, \exists c\in C, ComposedOf(cp, c)$。

MR_4: 过程访问基础规则。在 OWL-S 过程模型中, 任意原子过程都存在唯一一个 *grounding* 对其访问提供支持; 任意抽象过程和组合过程都不与 *grounding* 对应。该规则的形式化描述为: $(\forall ap\in P_a, \exists grd\in SG, grd.p=ap)\wedge(\forall p\in P\setminus P_a, \neg\exists grd\in SG, grd.p=p)\wedge(\forall grd_1, grd_2\in SG, grd_1.p=grd_2.p\Rightarrow grd_1=grd_2)$。

MR_5: 控制构造子构建规则。对于下列类型的控制构造子, 至少由两个子构造子构建而成: *Sequence*、*Split*、*Split+Join*、*Any-Order*、*If-Then-Else* 和 *Choice*。其形式化描述为: $\forall c\in C\setminus(C_{rw}\cup C_{ru}\cup C_{per}), |F(c)|\geqslant 2$。

MR_6: *If-Then-Else* 构建规则。在 *If-Then-Else* 的构建上, *Then* 所对应的控制构造子不能为空, 即 $\forall c\in C_{ite}, F(c).c_{then}\neq null$。

MR_7: 循环构建规则。在 *Repeat-While* 和 *Repeat-Until* 的构建上, *While* 和 *Until* 必须与一个控制构造

① Hc 表示本体中概念间的层次关系, $(Conc_1, Conc_2)\notin Hc$ 表示概念 $Conc_1$ 和 $Conc_2$ 之间存在一条继承路径。

子相对应, 即 $\forall c \in C_{rw} \cup C_{ru}$, $|F(c)| = 1$。

MR_8: Perform 构建规则。在 OWL-S 过程模型中, 任意 *Perform* 控制构造子都必须与具体的非抽象过程相关联。该规则的形式化描述为: $(\forall c \in C_{per}, \exists p \in P \setminus P_s, <c, p> \in PER) \wedge (\forall p \in P_s, \neg \exists c \in C_{per}, <c, p> \in PER)$。

MR_9: 过程执行输入约束。OWL-S 过程模型中非抽象过程在执行时, 其输入至少拥有一个数据来源。该规则的形式化描述为: $\forall per = <c, p> \in PER, \forall i \in per.p.Input, \exists df \in DF, df.per = per \wedge df.i = i$。

MR_{10}: 访问基础约束。对于原子过程的任意输入与输出参数, 原子过程所对应的访问基础中均存在与之相匹配的 WSDL 输入与输出。该规则可以形式化描述为: $\forall ap \in P_a, \exists grd \in SG, grd.p = ap \Rightarrow (\forall i \in ap.Input, \exists iMap \in grd.InputMap, iMap.i = i) \wedge (\forall o \in ap.Output, \exists oMap \in grd.OutputMap, oMap.o = o)$。

MR_{11}: 服务描述规则。对于 OWL-S 本体所描述的语义 Web 服务 *SWS*, 在过程模型中必须存在一个过程来描述该服务, 即 $\exists p \in P$, $describedBy(SWS, p)$。

4.2 可选一致性规则

OR_1: 抽象过程关联规则。在 OWL-S 过程模型中, 抽象过程应该与一个原子过程或组合过程相关。该规则可以形式化描述为: $\forall sp \in P_s, (\exists ap \in P_a, realizedBy(sp, ap)) \vee (\exists cp \in P_c, expandsTo(sp, cp))$。

OR_2: 原子过程执行约束。在 OWL-S 过程模型中, 不存在冗余的原子过程, 即过程模型中所有的原子过程至少被执行一次。该规则可以形式化描述为: $\forall ap \in P_a, \exists per = <c, ap> \in PER, per.p = ap$。

OR_3: 孤立控制构造子约束。在 OWL-S 过程模型中, 任意的控制构造子或用于构建组合过程, 或者参与构建其他控制构造子, 不存在孤立的控制构造子。该规则可以形式化表示为: $\forall c_1 \in C, (\exists cp \in P_c, composedOf(cp, c_1)) \vee (\exists c_2 \in C, c_1 \in F(c_2))$。

本文所抽象三条可选一致性规则分别对 OWL-S 过程模型中的抽象过程、原子过程执行和控制构造子关联进行了约束, 采用可选一致性规则, 知识工程师能够对修改后的 OWL-S 本体进行优化, 减少本体中的冗余信息, 提高推理的效率。

5 案例分析

下面以 OWL-S 推荐标准实例——航空票务服务 OWL-S 本体[8] 为例, 说明如何使用本文所提出的方法对该本体进行语法一致性维护。在该服务中, 用户通过查询系统输入航班的基本信息来获取可用的航班; 然后对查询到的航班进行筛选得到最满意的航班; 登入订票系统, 对满意的航班进行订票并保存相应的信息(如票据等)。这是一个简单的组合服务, 其组成示意图如图 3 所示。

假设知识工程师需要删除组合过程中 p_1 的执行, 即取消获取航班详细信息的过程执行, 那么知识工程师需要向 OWL-S 本体维护系统输入本体变化 $RemoveCtrlConstruct(p_1)$, OWL-S 本体维护系统在接收到该变化请求后的执行行为是:

(1) 检查前提条件。$RemoveCtrlConstruct(p_1)$ 的前提条件是 $p_1 \in C_{per}$, 显然该条件成立。

(2) 额外变化生成。根据 CRM 及表 3 算法, 该变化可能触发的额外变化如图 4(a) 所示。

(3) 在变化序列中的变化经过确认与实施后, 采用一致性规则检测结果如图 4(b) 所示。

(4) 根据语法一致性检测的反馈结果, 知识工程师将产生新的维护需求, 使新的本体能够满足规则 MR_9 和 OR_2, 从而完成 OWL-S 本体语法一致性的维护工作。

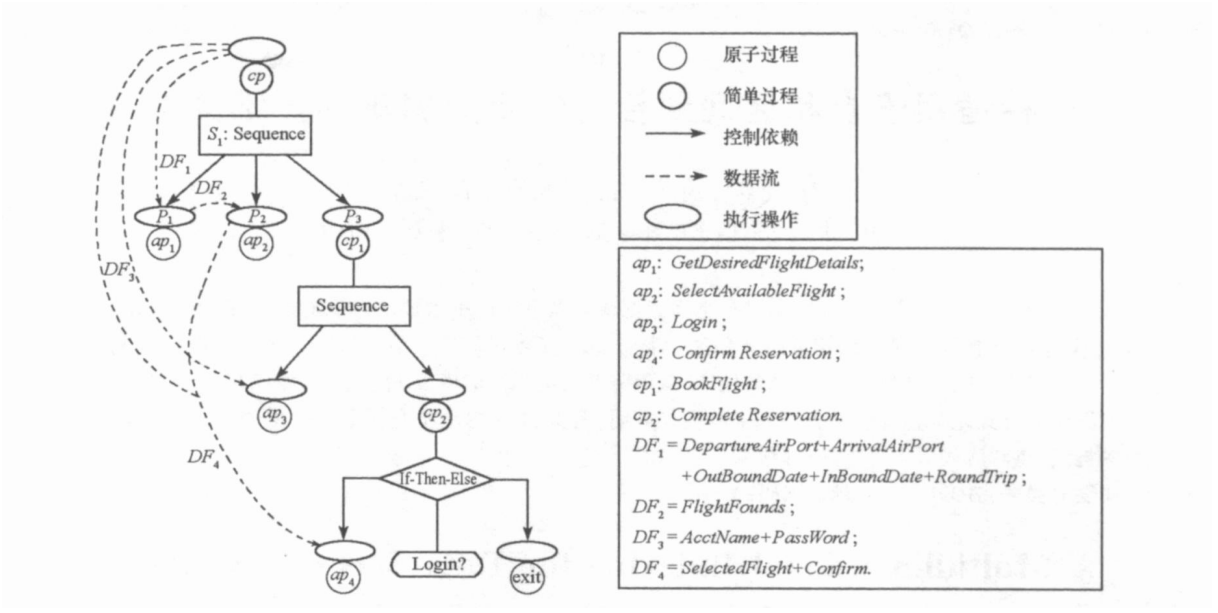

图 3 Bravo Air 本体过程模型
Fig. 3 Process model of Bravo Air

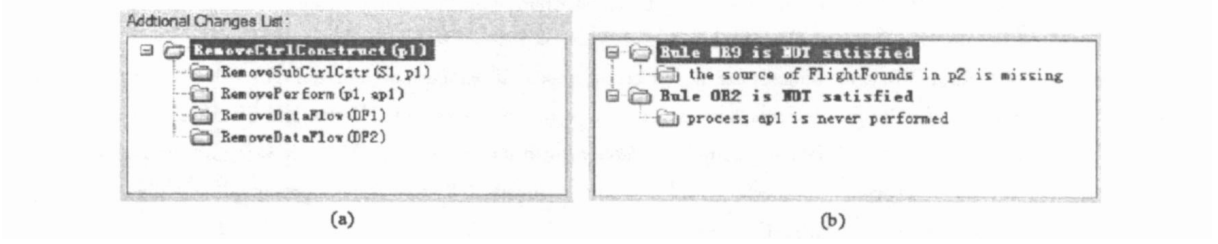

图 4 (a)额外变化生成; (b) 语法一致性规则检测
Fig. 4 (a) Additional change generation; (b) Syntactic consistency checking

6 总结与展望

采用 OWL-S 本体对 Web 服务进行语义标注的主要目的在于使 Web 服务的发布、发现、组合、调用和执行监控能够自动化完成[10], 因此在对 OWL-S 本体进行维护时, 其语法一致性的检测和维护是一个必不可少的过程。针对这个问题, 本文首先分析了 OWL-S 本体与领域本体在语法一致性维护上的差异, 在此基础上提出了一个 OWL-S 本体语法一致性维护框架, 并对框架中的额外本体变化序列生成算法和 OWL-S 本体语法一致性规则进行了详细说明。以该框架为指导, 知识工程师能够在机器的辅助下完成复杂的本体变化效果分析、本体修改及其语法一致性检查, 并在机器反馈信息的帮助下完成 OWL-S 本体进一步的修改。下一步工作将在 OWL-S 本体维护原型系统中完善本文所提出了额外变化生成算法和语法一致性规则的检测, 并进一步研究如何进行 OWL-S 本体语义一致性的检测与维护。

参 考 文 献:

[1] Mcllraith S, Son T C. Semantic Web Services [J]. IEEE Intelligent Systems, 2001, 16(2): 46-53.
[2] Martin D, Barsfin M. OWL-S: Semantic Markup for Web Services (V1.2) [Z/OL], available at http://www.ai.sri.com/daml/services/owl-s/1.2, 2006.
[3] Stojanovic L, et al. User-driven Ontology Evolution Management[C]//European Conf. Knowledge Eng. and Management (EKAW), 2002.
[4] Stojanovic L, Methods and Tools for Ontology Evolution [D]. PhD thesis, University of Karlsruhe, 2004.
[5] Haase P, Stojanovic L. Consistent Evolution of OWL Ontologies [C]//Proceedings of the 2nd European Semantic Web Conference, 2005.
[6] Hillman J, Warren I. An Open Framework for Dynamic Reconfiguration [C]//Proceedings of the 26th International Conference on Software Engineering, 2004.
[7] Kim Y H, Kim E K. Autonomic Service Reconfiguration in a Ubiquitous Computing Environment[C]//ISPA 2006: 584-593.
[8] OWL-S 1.2 Release: Examples [EB]. http://www.ai.sri.com/daml/services/owl-s/1.2/examples.html, 2006.
[9] Bao A H, Yuan J P, Yao L, et al. Approach to the Formal Representation of OWL-S Ontology Maintenance Requirements[C]//Proceedings of the 9th International Conference on Web-Age Information Management, 2008.
[10] Martin D, Burstein M, McDermott D, et al. Bringing Semantics to Web Services with OWL-S[J]. World Wide Web, 2007(10): 243-277.