第38卷 第17期
Vol.38 No.17

计 算 机 工 程
Computer Engineering

2012年9月
September 2012

• 软件技术与数据库 •

文章编号：1000—3428(2012)17—0028—04 文献标识码：A 中图分类号：TP391

基于OWL-S过程模型的Web服务发现方法

曾 一，胡延强，洪 豪

(重庆大学计算机学院，重庆 400030)

摘 要：现有Web服务发现方法未考虑过程行为的相关信息，导致所发现的服务不能较好地满足用户需求且查准率较低。针对该问题，提出一种基于OWL-S过程模型的Web服务发现方法。将OWL-S过程模型转化为过程图，根据两过程图的相似度，判断两服务过程的匹配程度。实验结果表明，该方法能准确地匹配两服务的过程模型，具有较高的执行效率和较好的匹配效果。

关键词：Web服务；Web服务本体语言；过程模型；过程图；相似度；过程匹配

Web Service Discovery Method Based on OWL-S Process Model

ZENG Yi, HU Yan-qiang, HONG Hao

(College of Computer Science, Chongqing University, Chongqing 400030, China)

【Abstract】Existing Web service discovery methods don't consider the behavior of the process related information, therefore the found service can not meet the demand of user and has lower precision. Aiming at the problem, this paper proposes a Web Services process discovery method based on Web Ontology Language for Service(OWL-S) process model. It transforms OWL-S process model into process graph, then calculates the degree of matching of two service process models according to the similarity of two process graphs. Experimental result shows that this method can accurately match two service process model, and it has the high efficiency and good matching effect.

【Key words】Web service; Web Ontology Language for Service(OWL-S); process model; process graph; similarity; process matching

DOI: 10.3969/j.issn.1000-3428.2012.17.008

1 概述

Web服务本体语言(Web Ontology Language for Service, OWL-S)[1]作为一种带有语义的描述服务的本体语言主要由三部分构成ServiceProfile、ProcessModel和Service Grounding。ServiceProfile描述了服务提供者、服务功能、服务质量等相关信息。ServiceModel描述服务过程方面的信息。ServiceGrounding描述服务是如何被访问的，它说明了访问服务的细节。

现有的服务发现方法[2-5]主要基于服务名、输入参数、输出参数、前提条件、执行效果和服务质量，然而在多种情况均需考虑服务过程方面的信息。例如，集成图书订购系统时需要一个这样的服务：首先接收由客户发来的图书书名信息，然后接收送货地址信息，再次接收客户的支付信息，最后生成订单返回给客户。网络上可能存在首先接收由客户发来的图书书名信息，之后接收客户的支付信息，再次接收送货地址信息，最后生成订单返回给客户这样的一个服务。这2个组合服务虽然输入输出参数是一致的，但由于各个活动的执行顺序不一致，两者不能够相互替代。现有的服务发现方法因没有考虑过程行为的相关信息不能有效区分这2个服务。关于服务集成时对过程匹配的需求文献[6-7]均有研究。因此，在服务发现阶段需考虑服务过程方面的信息，使查询所得服务能够更好地满足用户的需求。

针对以上问题，本文提出一种基于OWL-S过程模型的Web服务发现方法。该方法将OWL-S过程模型转化为过程图，定义两过程图的相似度，并在此基础上设计一种过程图的匹配算法。

2 OWL-S过程模型到过程图的转换

OWL-S过程模型由原子活动和控制结构构成，它详细描述了服务的执行流程。

文献[8]提出一种将业务流程转化为过程图的通用算法，本文将该算法稍作调整用作OWL-S过程模型到过程图的转换。该算法使用自顶向下的方式遍历OWL-S过程模型的嵌套结构，同时对控制流程中原子活动和控制结构按照以下规则进行转换，将整个流程转化为过程图。

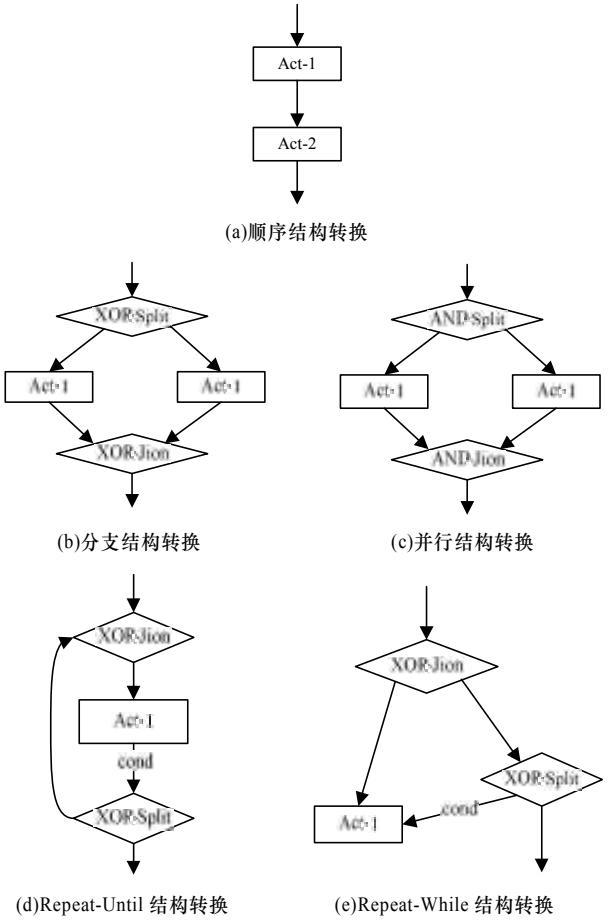
原子活动转换：在转换时将原子活动转化为活动节点，并将原子活动的活动名、输入参数集和输出参数集转化成活动节点的3个属性。

控制结构转换：将OWL-S过程模型中共8种控制结构(sequence、split、split-jion、any-order、choice、if-then-else、

作者简介：曾 一(1961－)，男，教授、CCF会员，主研方向：软件测试，软件工程；胡延强、洪 豪，硕士研究生

收稿日期：2011-10-26 **修回日期：**2011-12-19 **E-mail：**huyqg@126.com

repeat-until 和 repeat-while)转化成 AND-Split、AND-Join、XOR-Split 和 XOR-Join 这 4 种类型的连接节点。AND-Split 表示执行由该类型连接节点引出的所有分支，AND-Join 表示完成执行汇入该类型连接节点的所有分支，XOR-Split 表示执行由该类型连接节点的引出的分支之一，XOR-Join 表示完成执行由该类型连接节点的引出的分支之一。Sequence(Act-1, Act-2)按图 1(a)的方式转换，表示 Act-1、Act-2 依照文本出现的顺序依次执行。Choice(Act-1,Act-2)和 If-then-else (Act-1, Act-2)按图 1(b)的方式转换，表示 Act-1、Act-2 依照约束条件执行其中一个。split-jion(Act-1, Act-2)和 any-order (Act-1, Act-2)按照图 1(c)的方式转换，表示 Act-1、Act-2 均需执行，且执行顺序任意。Repeat-Until(Act-1)按图 1(d)方式转换，表示首先执行 Act，然后判断条件，若为真则继续循环，否则退出循环。Repeat-While(Act-1)按图 1(e)的方式转换，表示首先判断条件，如果为真则进行循环，否则退出循环。连接节点有 2 个属性:连接类型和节点原本对应的控制结构。



(a)顺序结构转换　(b)分支结构转换　(c)并行结构转换　(d)Repeat-Until 结构转换　(e)Repeat-While 结构转换

图 1　OWL-S 元素与图形元素之间的转换

3　基于 OWL-S 过程模型的 Web 服务发现

定义 1 过程图的活动节点 $a=(m, I, O)$，其中，m 表示活动节点的名字；I 表示活动节点的输入参数集；O 表示活动节点输出参数集。

定义 2 过程图的连接节点 $c=(t, s)$，其中，t 表示连接节点的类型；s 表示连接节点原本对应控制结构。

定义 3 过程图 $PG=(S, D, A, C, E, R, g)$，其中，S 表示起始节点；D 表示结束节点；$A=(a_1,a_2,\cdots,a_n)$表示所有活动节点构成集合；$C=(c_1,c_2,\cdots,c_n)$表示所有连接节点构成的集合；$E\in(S\cup D\cup A\cup C)\times(S\cup D\cup A\cup C)$；$R$ 表示所有控制条件构成的集合，g: E->R 为边到控制条件的一个映射。

定义 4 $PG_1=(S_1,D_1,A_1,C_1,E_1,R_1,g_1)$和 $PG_2=(S_2,D_2,A_2,C_2,E_2, R_2,g_2)$为 2 个过程图。令 $N_1=S_1\cup D_1\cup A_1\cup C_1$、$N_2=S_2\cup D_2\cup A_2\cup C_2$，映射 M: N_1->N_2 为一从 G_1 节点集到 G_2 节点集的部分单映射。令 $dom(M)=\{n_1|(n_1,n_2)\in M\}$, $cod(M)=\{n_2|(n_1,n_2)\in M\}$。给定一节点 $n\in N_1\cup N_2$，当且仅当 $n\in dom(M)\cup cod(M)$时，n 是可替换的。sum$n$ 是所有可替换节点的集合。用 S_1 表示可替换节点占总节点的比例:

$$S_1=\frac{|\text{sum}n|}{|N_1|+|N_2|} \tag{1}$$

给定一条边$(p_1,q_1)\in E_1$，当且仅当映射$(p_1,p_2)\in M$、$(q_1,q_2)\in M$ 且边$(p_2,q_2)\in E_2$，边(p_1,q_1)是可替换的。类似定义 E_2 中可替换的边。sum$e$ 表示 $E_1\cup E_2$ 所有可替换的边的集合。用 S_2 表示可替换边占总边数的比例:

$$S_2=\frac{|\text{sum}e|}{|E_1|+|E_2|} \tag{2}$$

设$(n_1,n_2)\in M$，Sim(n_1,n_2)表示 2 个节点的相似度。用 S_3 表示映射 M 中所有节点对的平均相似度:

$$S_3=\frac{2.0\times\sum_{(n,m)\in M}\text{Sim}(n_1,n_2)}{|\text{sum}n|} \tag{3}$$

过程图 PG_1 和 PG_2 在映射 M 下的图形相似度为:

$$S(PG1,PG2)_M=w_i\times S_1+w_j\times S_2+w_k\times S_3 \tag{4}$$

其中，w_i、w_j、w_k 分别为 S_1、S_2、S_3 的权重，$0\leqslant w_i\leqslant 1$、$0\leqslant w_j\leqslant 1$、$0\leqslant w_k\leqslant 1$，权重指定范围内可调。

定义 5 过程图 PG_1 和 PG_2 的节点集之间存在多种映射，令 F 表示 PG_1 的节点集到 PG_2 的节点集所有映射所构成的集合。定义 2 个过程图的相似度为:

$$S(PG_1,PG_2)=\max\{S(PG_1,PG_2)_M\mid M\in F\} \tag{5}$$

同时，将该相似度下所对应的映射称为 2 个过程图的最佳映射。$S(PG_1,PG_2)\in[0,1]$，$S(PG_1,PG_2)$的值越大，PG_1、PG_2 的匹配程度就越高。

由上述定义可知，根据两过程图的相似度判断两服务过程的匹配程度。本文采用贪心思想求得最佳映射进而求得两过程图的相似度。

算法的基本思路如下：首先，将 PG_1 的节点集 V_1 与 PG_2 的节点集 V_2 做笛卡尔积的结果放入集合 *pairs* 中，将集合 *map* 初始化为空。然后，算法进行迭代，在每一次迭代时，从 *pairs* 集合中选择一对能够使两图的相似度增加最大的顶点对(v_1,v_2)加入到映射 *map* 集合中，并从 *pairs* 集合中移除所有与顶点 v_1 或顶点 v_2 相关的顶点对。当 *pairs* 集合为空或者 *pairs* 集合中不存在加入集合 *map* 中能够使得两图相似度增加的顶点对停止迭代。此时，形成的 *map* 集合为最佳映射，在该映射下求得的相似度为两图的相似度。

算法 1 过程图匹配算法 *PGMatchmaking*(PG_1,PG_2)

输入 过程图 PG_1 和 PG_2

输出 过程图的相似度

(1)初始化 pairs≤V_1×V_2，map≤Φ；

(2)对于每一个节点对(v_1,v_2)∈pairs，IF 顶点 v_1 和 v_2 的种类不一样 THEN

将顶点对(v_1,v_2)从 pairs 中移除；

(3)IF(pairs 为空集或者 map 集合中的节点对数较上次迭代没有增加) THEN

输出 s(map)和 map 集合；

(4)IF 存在(n,m)∈pairs 使得 s(map∪{(n,m)})>s(map)并且不存在另一节点对(p,q)∈pairs 使得 s(map∪{(p,q)})>s(map∪{(n,m)})THEN

(4.1)map≤map∪{(n,m)}

(4.2)pairs≤{(p,q)∈pairs|p≠n,q≠m}；

(5) Goto 步骤(3)。

4 *Sim*(n_1,n_2)计算方法

OWL-S 过程模型转化的过程图中有 4 种节点：起始节点，结束节点，活动节点和连接节点。由于种类不一致的节点匹配没有意义在预处理步骤中已将其删除，因此算法 1 集合 *map* 中节点对的匹配只有 4 种情况：起始节点与起始节点匹配，结束节点与结束节点匹配，活动节点与活动节点匹配或者连接节点与连接节点匹配。

4.1 节点相似度的计算

算法 2 节点相似度的计算算法 *Sim*(n_1,n_2)

输入 节点对(n_1,n_2)

输出 节点对的相似度

(1)IF 2 个节点均为起始节点 THEN return 1；

(2)IF 2 个节点均为结束节点 THEN return 1；

(3)IF 2 个节点均为连接节点 THEN

return ConnectorSim(n_1,n_2)；//参考连接节点的相似度计算//算法

(4) IF 2 个节点均为活动节点 THEN

return ActivitySim(n_1,n_2)；//参考活动节点的相似度计算//算法

连接节点有 2 个属性：连接类型和节点所对应的控制结构。根据连接节点属性的异同计算连接节点的相似度。

算法 3 连接节点相似度计算算法 *ConnectorSim*(n_1,n_2)

输入 2 个连接节点 n_1、n_2

输出 2 个连接节点的相似度

(1)IF 2 个连接节点的连接类型不一致且所对应的控制结构也不一致 THEN return 0；

(2)IF 2 个连接节点的连接类型一致但所对应的控制结构不一致 THEN return 0.5；

(3)IF 2 个连接节点的连接类型一致且所对应的控制结构一致 THEN return 1.0；

4.2 活动节点相似度的计算

活动节点有 3 个属性：活动名，输入参数集和输出参数集。本文分别计算活动名的相似度、输入参数集和输出参数集的相似度之后加权求和求得相似度。

算法 4 活动节点相似度计算算法 *ActivitySim*(n_1,n_2)

输入 2 个活动节点 n_1、n_2

n_1: *Struct*($Name_1$,$InputSet_1$,$OutputSet_1$)

n_2: *Struct*($Name_2$,$InputSet_2$,$OutputSet_2$)

输出 2 个活动节点的相似度

(1)计算活动名的相似度

NameSimiarity=LS($Name_1$,$Name_2$)；

(2)计算输入输出参数集的相似度

(2.1)计算输入参数集的相似度

InSimilarity=SC($InputSet_1$,$InputSet_2$)；

(2.2)计算输出参数集的相似度

OutSimilarity=SC($OutputSet_1$,$OutputSet_2$)；

(2.3)计算输入输出参数集的相似度

IOSimilarity=(InSimilarity+OutSimilarity)/2；

(3)计算活动节点的相似度

return w_{io}×IOSimilarity+w_{name}×NameSimilarity

活动名是一串字符，计算活动名的相似度即计算两字符串的相似度。关于字符串相似度的计算主要有 3 种算法：NGram 算法，Check Synonym 算法，Check abbreviation 算法。这 3 种算法在计算字符串相似度时侧重点各不相同。NGram 算法主要根据两字符串中有多少相同的 q 长度的子串计算相似度，Check Synonym 算法在计算相似度时使用语义词典，Check abbreviation 算法在计算相似度时参照定制的缩写词汇的词典。本文综合 3 种方法计算字符串的相似度。

算法 5 字符串相似度计算算法 *LS*(str_1,str_2)

输入 2 个字符串 str_1、str_2

输出 2 个字符串的相似度

用 k_1 表示 NGram 算法计算的相似度，用 k_2 表示 Check Synonym 算法计算的相似度，用 k_3 表示 Check abbreviation 算法计算的相似度

(1)IF k_1,k_2,k_3 均为 1 THEN return 1；

(2)IF k_1,k_2,k_3 均为 0 THEN return 0；

(3)IF k_1=0,k_3=0 且 0<k_2<1 return k_2；

ELSE return (k_1 +k_2 +k_3)/3

4.3 参数集相似度的计算

输入参数集由多个输入参数构成，输出参数集由多个输出参数构成，由于输入输出参数均为某一领域本体中的概念，首先计算 2 个概念之间的语义相似度，进而计算输入输出参数集的相似度。

算法 6 概念相似度 *ConceptSim*(C_1,C_2)

输入 2 个概念 C_1、C_2

输出 2 个概念的相似度

IF C_1=C_2 return 1；

IF C_1≥C_2 return 1；

ELSE return OntoSim(C_1,C_2)；

在算法 6 中如果 C_1 和 C_2 完全一致，2 个概念的相似度为 1。如果 C_1 包含 C_2 2 个概念的相似度也为 1，因为，此种情况说明 C_1 至少具有 C_2 的所有属性。如果 C_2 能够满足服务的参数要求，那么 C_1 也必能满足。其他情况使用函数 *OntoSim*(C_1, C_2)计算 2 个概念的相似度。*OntoSim*(C_1, C_2)计算方法如下：

$$OntoSim(C_1,C_2)=\frac{\alpha\times(l_1+l_2)}{(Dis(C_1,C_2)+\max(|l_1+l_2|,1))} \tag{6}$$

其中，l_1、l_2 表示 C_1、C_2 在本体中的深度；*Dis*(C_1, C_2)表示 C_1 与 C_2 之间的距离；α 是一个可调节的参数，一般取

α>0。

在概念相似度计算的基础上给出 2 个概念集合相似度的计算方法，进而给出输入输出参数集的计算方法。

算法 7 概念集合相似度 $ParaSetSim(ParaSet_1, ParaSet_2)$

输入 2 个概念集合 $ParaSet_1$、$ParaSet_2$

输出 2 个概念集合的相似度

IF ($ParaSet_1=\varnothing \vee ParaSet_2=\varnothing$)　return 0;

ELSE

return　Max(ParaSetSim($ParaSet_1$–J,$ParaSet_2$–K)+ConceptSim(J,K)); // J ∈ $ParaSet_1$, K ∈ $ParaSet_2$

参数集合相似度计算如下：

$$SC(ParaSet_1,ParaSet_2)=\frac{ParaSetSim(ParaSet_1,ParaSet_2)}{\max(|ParaSet_1|,|ParaSet_2|)} \tag{7}$$

5　实验及结果分析

实验实现如图 2 所示的系统，它首先将服务的 OWL-S 过程模型转化为过程图，之后进行过程图匹配，最后输出两过程图的相似度。

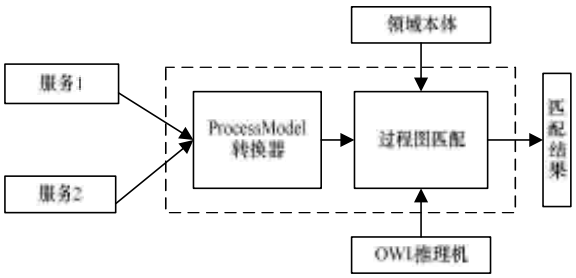


图 2　过程匹配框架

OWLS-TC4 标准数据集合在教育领域提供了 286 个用 OWL-S 描述的 Web 服务。在实验中从中选取 30 个服务作为测试样本。领域本体使用 OWLS-TC4 中提供的本体，使用 Wordnet 语义词典以及 Euler OWL 推理机。实验平台为双核 Intel 处理器、主频 2.80 GHz、2 GB 内存。另外，实验中使用参数为：在式(4)中，w_i=0.2，w_j=0.2，w_k=0.6，在算法 4 中，w_{name}=0.3，w_{io}=0.7，在式(6)中，α=0.5。

实验主要从执行效率和匹配效果 2 个方面测试算法的性能。实验方案如下：

(1)将测试样本与它自身做笛卡尔积得到 900 个服务对。

(2)依次计算这 900 个服务对的相似度，同时记录计算每一服务对所使用的时间。

将记录的相似度用下式处理：$\frac{1}{900}\sum_{i=1}^{n}[1.0-\frac{|y_i-x_i|}{x_i}]$($n$ 为 900，x_i 为 2 个过程图的真实相似度，y_i 为算法计算的 2 个过程图的相似度)作为算法结果的平均准确度。处理实验结果得到算法的平均精确度为 0.837。结果表明，本文算法能够准确计算出 2 个过程图的相似度。

对上述 900 服务对按节点个数对执行时间进行统计并求平均值，如图 3 所示。该图显示在节点数低于 25 个时算法有着较高的执行效率，由理论分析可知该算法的时间复杂度为 $O(n^3)$，说明该算法不会随着过程图的节点数的增加，耗时呈指数级急剧增长。实验虽选取了一个较小的测试样本，但基本能够验证该算法的可行性和有效性。

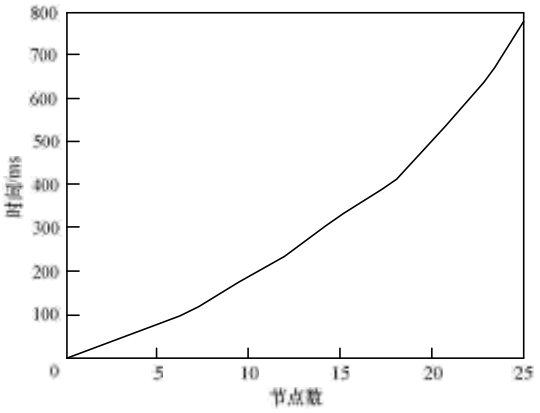


图 3　节点数与执行时间的关系

6　结束语

本文提出一种 OWL-S 过程模型的 Web 服务发现方法，实验验证了该方法的有效性。今后将提高算法的执行效率，以更好地满足实际需求，另外，本文在计算相似度时只考虑了控制流程的相关信息，在以后工作中可考虑其他数据信息，从而更准确反映出过程图的匹配程度。

参考文献

[1] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic Markup Language for Web Services[EB/OL]. (2004-05-18). http://www.w3.org/Submission/OWL-S.

[2] 邹金安. 基于 QoS 的语义 Web 服务发现研究[J]. 计算机应用, 2009, 29(10): 2844-2846.

[3] 吴　建, 吴朝晖, 李　莹, 等. 基于本体论和词汇语义相似度的 Web 服务发现[J]. 计算机学报, 2005, 28(4): 595-602.

[4] 仲　梅, 宋顺林. 一种语义 Web 服务的多层次匹配方法[J]. 计算机应用, 2007, 27(1): 199-202.

[5] 胡胜文, 石美林. 一种支持 QoS 约束的 Web 服务发现模型[J]. 计算机学报, 2005, 28(4): 589-594.

[6] Wombacker A, Mahleko B, Fankhauser P. Matching Making for Business Process Based on Choreographies[C]//Proc. of IEEE International Conference on E-technology, E-commerce and E-service. Taipei, China: [s. n.], 2004.

[7] Zdravkovic J, Johanesson P. Cooperation of Processes Through Message Level Agreement[C]//Proc. of International Conference on Advanced Information Systems Engineering. Riga, Latvia: [s. n.], 2004.

[8] Moon Jin-Young, Lee Dae-Ha, Park C, et al. Transformation Algorithms Between BPEL4WS and BPML for the Executable Business Process[C]//Proc. of IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises. Taejeon, South Korea: [s. n.], 2004.

编辑　陆燕菲