小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2010 年 5 月 第 5 期
Vol. 31 No. 5 2010

利用语义匹配度计算的 Web 服务发现方法

邹国兵[1,2],向 阳[1,2],甘杨兰[1],孙红雨[1,2],张 波[1,2]

[1](同济大学 电子与信息工程学院,上海 201804)
[2](同济大学 嵌入式系统与服务计算教育部重点实验室,上海 201804)
E-mail:guobing278@sina.com

摘 要:针对目前在 Web 服务注册与匹配过程中服务发现的查全率和查准率不高的问题,本文提出一种利用语义匹配度计算的 Web 服务发现方法.首先给出一个轻量级的 Web 服务功能接口和服务请求的语义描述模型 WS-SDM;然后利用领域本体作为语义知识表达方式,计算服务请求和服务描述中单个功能接口的语义匹配度;在此基础上,分别给出接口匹配算法和服务匹配算法.实验结果表明提出的服务发现方法能够获得较好的服务发现效果.

关 键 词:Web 服务;服务发现;领域本体;语义匹配度;服务匹配
中图分类号:TP311 **文献标识码**:A **文 章 编 号**:1000-1220(2010)05-0807-06

Web Services Discovery Method Utilizing Semantic Matching Degree Calculation

ZOU Guo-bing[1,2], XIANG Yang[1,2], GAN Yang-lan[1], SUN Hong-yu[1,2], ZHANG Bo[1,2]
[1](*College of Electronics and Information Engineering, Tongji University, Shanghai* 201804, *China*)
[2](*Key Laboratory of Ministry of Education on Embedded System and Service Computing, Tongji University, Shanghai* 201804, *China*)

Abstract: Current web service registration and matching process has led to low recall and precision for service discovery and has much potential improvement for their performance. This paper proposes a web service discovery method utilizing semantic matching degree calculation. Firstly, a light semantic-based description model WS-SDM is given to describe web service function interface and service request. Then, by adopting domain ontology as the way of expressing semantic knowledge, a semantic matching degree calculation method is proposed for single function interface between service request and service description. On the basis of the calculation method, interface matching algorithm and service matching algorithm are presented respectively. Finally, experimental results demonstrate that proposed method can achieve better service discovery effectiveness.

Key words: web service; service discovery; domain ontology; semantic matching degree; service matching

1 引 言

Web 服务作为一种新兴的 Web 应用模式,它是一种基于网络环境下的自适应,自包含,自描述,模块化的应用程序,提供了从简单请求到复杂商业过程的功能,已成为互联网中最为重要的一种计算资源.

Web 服务发现是面向服务的体系架构中一个重要的组成部分,如何为服务提供者发现合适的目标服务是实现服务共享和复用的关键问题.目前 Web 服务发现的方法可分为两大类.一类是基于 UDDI(Universal Description, Discovery and Integration)的服务注册与发现机制,由于该方法利用分类规范和关键词的查找方式进行服务发现,仅支持对服务语法层面的操作,缺乏服务功能描述的语义信息,服务匹配存在明显的不足,从而难以满足请求者对服务发现查准率的需求;另一类是基于语义的 Web 服务发现,该方法利用语义 Web 技术增强服务的语义描述能力.目前大多数研究采用基于 OWL-S[1] 本体描述语言实现语义推理和服务匹配,然而该方法以概念间的语义包含关系判断服务匹配程度,仍然是一种粗粒度层次的服务匹配,缺乏服务请求与 Web 服务描述间的精确匹配和量化计算.

本文提出了一种利用语义匹配度计算的 Web 服务发现方法.首先给出了 Web 服务功能接口和服务请求的语义描述模型 WS-SDM;然后以领域本体作为语义知识表达方式,计算服务请求和服务操作中单个功能接口的语义匹配度;最后在此基础上,分别给出接口匹配算法和服务匹配算法.通过实验表明提出的 Web 服务发现方法在查全率和查准率上具有较好的服务发现效果.

2 相关工作

目前,国内外研究学者在 Web 服务匹配方法上取得了相关的研究成果.卡内基梅隆大学的 Massimo Paolucci 等人[2] 提出了一种 Web 服务能力的语义匹配算法,利用 DAML-S[3]

收稿日期:2009-01-21 收修改稿日期:2009-04-03 基金项目:国家自然科学基金项目(70771077) 资助;国家"八六三"高技术研究发展计划项目(2008AA04Z106)资助;上海市科委项目(08DZ1122300)资助. 作者简介:**邹国兵**,男,1982 年生,博士研究生,研究方向为语义 Web 服务发现、本体工程;**向 阳**,男,1962 年生,教授,博士生导师,研究方向为数据仓库与数据挖掘、服务计算、智能决策支持系统;**甘杨兰**,女,1984 年生,博士研究生,研究方向为数据挖掘、生物信息学.

描述语言作为服务接口的描述模型,通过对本体中概念包含关系的语义推理,将 Web 服务语义匹配分为四个等级(精确匹配,可替代匹配,包含匹配和匹配失败). 但该方法仅从定性的角度判断服务功能接口的匹配等级,缺乏接口匹配的量化计算. Jeffrey Hau 等人[4]提出了一种基于 OWL-S 语义描述语言的 Web 服务语义相似度量的计算方法,该方法依据 OWL 描述语言中概念和属性固有的语义关系,采用基于推理的信息值度量方法,并通过给出的语义推理规则生成其推理的信息值,计算出服务接口中两个 OWL 对象的语义相似度. 该方法仅依据对象间结构信息的规则值进行语义推理,仍缺乏精确的语义匹配量化计算.

浙江大学的吴健等人[5]提出了一种基于本体论和词汇语义相似度的 Web 服务发现方法,该方法利用《WordNet》,《HowNet》和《同义词词林》等语义词典计算词语相似度,然后计算服务请求和服务描述在属性相似度,参量相似度和结构相似度上的度量值,最后综合计算服务匹配相似度. 该方法利用了词汇库中的词汇描述 Web 服务接口功能,给出了服务匹配程度的量化计算,但由于词汇库是对词汇通用的语义描述,直接用于描述特定领域的 Web 服务接口不是很全面,语义描述的领域针对性不强. 文献[6,9]中分别利用语义扩展和二分图匹配算法实现 Web 服务发现.

针对目前 Web 服务发现方法过程中存在的问题,本文避开利用概念间语义包含关系推理和实现服务匹配,造成服务发现查全率和查准率不高的情况. 考虑服务匹配过程中影响接口功能参数间语义距离的各种因素,并利用语义匹配度计算发现满足请求者需求的 Web 服务集.

3 Web 服务语义描述模型

Web 服务发现是根据服务请求者对目标服务的需求描述,通过服务匹配算法从 Web 服务库查找到与用户需求描述相匹配的服务. 因此,Web 服务的描述信息对于匹配结果至关重要. 本文参考文献[9]中的 Web 服务注册模型,给出一个轻量级的 Web 服务语义描述模型 *WS-SDM*,包含 Web 服务的语义描述和服务请求的语义描述两部分.

定义 1(Web 服务). Web 服务表示为一个四元组 $ws = \{wsId, wsName, wsDesp, OprSet\}$. 其中:*wsId* 是服务在服务库中的惟一标识符; *wsName* 是服务的名称; *wsDesp* 是服务功能的文本描述; $OprSet = \{opr_1, opr_2, \cdots, opr_s\}$ 是服务操作集合,$opr_i(1 \leqslant i \leqslant s)$ 为一个服务操作.

Web 服务的操作集合包含一系列相互关联的服务操作 $opr_1, opr_2, \cdots, opr_s$,每个操作可独立完成特定的功能. 通过分析 OWL-S 描述语言中的过程模型知,操作中输入/输出接口间具有绑定关系. 结合参数的关联因素,服务操作定义如下:

定义 2(服务操作). Web 服务操作表示为一个四元组 $opr = \{oprName, InSet, OutSet, fMap\}$. 其中: *oprName* 是服务操作名称; $InSet = \{inP_1, inP_2, \cdots, inP_l\}$ 是输入接口集合,且 $inP_i(i = 1,2,\cdots,l)$ 已被本体中的一个概念,实例或属性标注; $OutSet = \{outP_1, outP_2, \cdots, outP_n\}$ 是操作的输出接口集合,且 $outP_j(j = 1,2,\cdots,n)$ 已被本体标注.

fMap 是服务操作的输入和输出接口间参数的映射函数,$fMap: outP_j \rightarrow InSet', InSet' \subseteq Inset$, . 即对任意一个输出参数 $outP_j(j = 1,2,\cdots,n)$,由 *fMap* 映射为 *InSet* 的一个子集,表示一个输出接口与多个输入接口间隐含参数映射.

同样地,请求者提供他们对目标服务的需求描述.

定义 3(服务请求). Web 服务请求表示为一个三元组 $req = \{reqName, InReqSet, OutReqSet\}$.

其中,*reqName* 是服务请求者期望的服务名称; $InReqSet = \{inR_1, inR_2, \cdots, \}$ 是请求者对输入接口集的需求,且 $InR_i(i = 1,2,\cdots,k)$ 已被本体中的概念,实例或属性标注; $OutReqSet = \{outR_1, outR_2, \cdots, outR_m\}$ 是对输出接口的需求, 且 $outR_j(j = 1,2,\cdots,m)$ 已被本体标注.

4 Web 服务发现方法

4.1 本体形式化描述

在 Web 服务的语义描述中,服务请求者和服务提供者利用本体中的概念,实例和属性刻画服务操作中输入/输出接口. 领域本体(Domain Ontology, *DO*)是共享概念模型的明确的形式化规范说明[7]. 它描述了概念的内涵以及概念与概念之间的语义关系,具有良好的概念层次结构和对逻辑推理的支持[7]. 因此,本文以领域本体作为描述 Web 服务的语义环境,实现 Web 服务在语义层次上的量化计算和匹配.

定义 4(领域本体). *DO* 是一个四元组: $DO = \{C, A^C, R, I\}$. 其中,*C* 是领域内所有概念的集合,A^C 是概念属性的集合,*R*是概念,实例和属性间关系的集合,*I*是领域内所有实例集合.

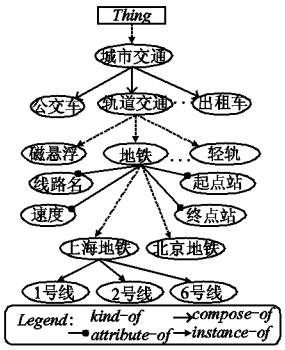

图 1 城市交通工具本体片段

Fig. 1 Ontology segment of city transportation tool

在定义 4 中,语义关系集合 $R = \{compose\text{-}of, kind\text{-}of, attribute\text{-}of, instance\text{-}of\}$. 其中,*compose-of* 是概念间的部分与整体关系; *kind-of* 是概念间的继承关系; *attribute-of* 是概念与其属性的关系; *instance-of* 是概念与所属实例的包含关系.

以本体中的概念,实例和属性作为节点,语义关系作为连接节点之间的弧线,构建一个本体有向概念结构树(*Ontology Directed Concept Structure Tree*, *ODCST*). 例如, 图 1 是交通工具分类本体中城市交通的本体片段. 地铁是一个本体概念,线路名和速度是概念地铁的属性, 1 号线为上海地铁的一个实例,城市交通和轨道交通是 *compose-of* 的关系.

4.2　语义匹配度计算

为了计算服务操作与服务请求间的语义匹配度,需度量本体中两节点在有向概念结构树中的语义距离. 由于在 *ODCST* 中两个节点存在不同的位置关系,因此在计算语义距离的过程中,针对不同的位置关系分别赋予两个节点间不同的语义关系权重或语义距离.

为便于描述语义匹配度的计算方法,首先给出四条规则. 在 *ODCST* 中,$Grandfather(c_1,c_2)$ 为一阶谓词,表示节点 c_1 与 c_2 满足祖父与子孙位置关系;$Descendant(c_1,c_2)$ 表示节点 c_1 与 c_2 满足子孙与祖父位置关系;$SameAs(c_1,c_2)$ 表示 c_1 与 c_2 属同一个节点;$\prec$ 为权重递减偏序关系.

rule 1. 在 *ODCST* 中,节点 c_1 与 c_2 满足祖父与子孙位置关系,w 为权重函数. 则语义关系权重规则为:

$$Grandfather(c_1,c_2)\rightarrow(1\leqslant w<2)\wedge(w(attribute-of)\prec w(compose-of)\prec w(kind-of)\prec w(instance-of)) \quad (1)$$

rule 2. 在 *ODCST* 中,节点 c_1 与 c_2 满足子孙与祖父位置关系,w'是权重函数. 则语义关系权重规则为:

$$Descendant(c_1,c_2)\rightarrow(0<w'<1)\wedge(w'(attribute-of)\prec w'(compose-of)\prec w'(kind-of)\prec w'(instant-of) \quad (2)$$

rule 3. 在 *ODCST* 中,$Semantic\_dist(c_1,c_2)$ 为距离函数,表示节点 c_1 与 c_2 的语义距离. 则语义距离规则为:

$$SameAs(c_1,c_2)\rightarrow(Semantic\_dist(c_1,c_2)=0) \quad (3)$$

rule 4. 在 *ODCST* 中,$InaccessiblePath(c_1,c_2)$ 表示节点 c_1 与 c_2 无可直达的路径. 则语义距离规则为:

$$(\neg SameAs(c_1,c_2)\wedge InaccessiblePath(c_1,c_2))\rightarrow(Semantic\_dist(c_1,c_2)=+\infty) \quad (4)$$

在规则 1 和 2 中,当节点 c_1 到 c_2 满足祖父与子孙(或子孙与祖父)位置关系时,则在计算语义距离的过程中,从 c_1 到 c_2(或 c_2 到 c_1)的路径上指定四种语义关系的权值为有序的高低关系和权重范围;若 c_1 与 c_2 表示同一个节点,规则 3 定义其语义距离为 0;若 c_1 与 c_2 不是同一个节点,且两个节点间无直接可达的路径,规则 4 定义其语义距离为 $+\infty$.

在 *ODCST* 中,对于任意两相邻节点 c_i 和 c_j,分别在以 c_i 为起点(或终点)和 c_j 为终点(或起点)的路径 $p(c_i,c_j)$(或 $p(c_j,c_i)$)上(如图 2 所示),其路径距离记作 $weight(c_i,c_j)$(或 $weight(c_j,c_i)$). 在计算路径距离时,除了规则 1 和 2 中的语义关系权重(如图 2(a, b)中的 *relWeight*)外,本文还考虑如下两种影响路径距离的因素:

(1) 节点深度权重. 在本体层次结构树中,节点所处的深度是计算语义距离时需要考虑的一个重要因素. 由于节点自顶向下由抽象逐渐变得具体,连接它们的路径边对语义距离计算的影响逐渐减小,因此应赋予较小的计算权值.

在 *ODCST* 中,对于任意两个相邻节点 c_i,c_j,节点深度权重为 $depWeight(c_i,c_j)=1/\alpha^{mindep(c_i,c_j)}$,其中 $mindep(c_i,c_j)$ 是节点 c_i 与 c_j 在本体结构层次树中深度的最小值,$\alpha\in N$ 且 $\alpha\geqslant 2$. 例如,在图 2(a)中,c_i 和 c_j 深度最小值为节点 c_i 的深度,则在计算其路径 $p(c_i,c_j)$ 的节点深度权重时 $mindep(c_i,c_j)=depth(c_i)$.

(2) 分布密度权重. 在本体结构层次树中,若一个概念节点分布越精确,表明该节点连接的路径边越具有重要的参考价值,计算语义距离时应赋予较低的权值.

在 *ODCST* 中,对于任意两个相邻节点 c_i,c_j,节点分布密度权重为 $disWeight(c_i,c_j)=1/\sqrt[\beta]{upperwid(c_i,c_j)}$,其中 $upperwid(c_i,c_j)$ 是 c_i 和 c_j 的路径上起始节点的分布密度,$\beta\in N$ 且 $\beta\geqslant 2$. 如在图 2(b)中,节点 c_i 和 c_j 在 *ODCST* 中起点节点为 c_j,则在计算其路径 $p(c_j,c_i)$ 的分布密度权重时 $upperwid(c_i,c_j)=width(c_j)$.

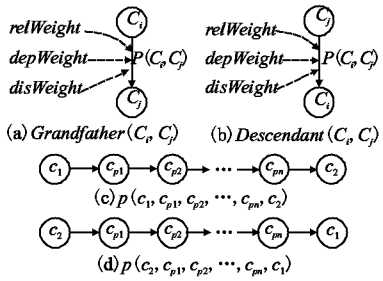

图 2　两个节点的位置关系和可达路径

Fig. 2　Position relation of two nodes and their accessible path

定义 5(语义距离). 假定节点 c_1 与 c_2 在 *ODCST* 中存在路径 $p(c_1,c_{p1},c_{p2},\cdots,c_{pn},c_2)$ 或 $p(c_2,c_{p1},p_{p2},\cdots,c_{pn},c_1)$,如图 2(c)和(d)所示. 则两个可达节点 c_1 与 c_2 的路径 $p(c_1,c_2)$ 或 $p(c_2,c_1)$ 的语义距离 $Semantic\_dist(c_1,c_2)$ 为:

$$Semantic\_dist(c_1,c_2)=\sum_p w^*(rel)\times depWeight\times disWeight \quad (5)$$

其中,$w^*(rel)$ 是规则 1(或规则 2)中的语义关系权重函数 w(或 w'),*rel* 是四种语义关系. *depWeight* 和 *disWeight* 分别为节点深度和分布密度权重. 语义距离为各路径距离之和.

在 Web 服务匹配过程中,语义匹配度表达了服务请求和服务操作中两个接口间的语义关联程度. 本文利用语义距离度量两个服务接口的语义匹配度.

定义 6(语义匹配度). c_1 与 c_2 为 *ODCST* 中任意两个节点,分别对应 Web 服务操作中的两个接口参数,则接口参数 c_1 与 c_2 的语义匹配度为:

$$seMatDeg(c_1,c_2)=\begin{cases}1-log_{\frac{3\gamma-1}{\gamma-1}}(Semantic\_dist(c_1,c_2)+1), & rule\ 1\sim 3\\ 0, & rule\ 4\end{cases} \quad (6)$$

在定义 6 中,$\gamma\in N$ 且 $\gamma\geqslant 2$,γ 是可调节参数. 若两个接口参数在 *ODCST* 中为同一节点或路径可达,则由语义距离计算其语义匹配度;否则语义匹配度为 0.

4.3　接口匹配算法

为了便于描述输入/输出接口的匹配算法,先给出匹配输出集(*MatchedOutSet*)的定义如下:

定义 7(匹配输出集). 给定一个服务请求 *req* 的输出接口参数集合 *OutReqSet* 和操作 *opr* 的输出接口参数集合 *OutSet*. 对于 $\forall outR_i\in OutReqSet(i=1,2,\cdots,m)$,若存在一个 $outP_j\in OutSet(j=1,2,\cdots,n)$,且满足 $seMatDeg(outR_i,outP_j)$ 为大于 0 的最大值,则匹配输出集为:

$$MatchedOutSet = \bigcup_{1\leqslant i\leqslant m}\{outP_j \mid Max_{1\leqslant j\leqslant n}(SeMatDeg(outR_i, outP_j))\} \quad (7)$$

服务请求的匹配输出集刻画了服务操作 *opr* 中与服务请求 *req* 的输出接口集语义相关度最高的输出接口集合.

给定一个服务请求 *req* 和服务操作 *opr*,算法 1 描述了输出接口匹配过程,计算服务请求的输出接口集 *OutReqSet* 和服务操作的输出接口集 *OutSet* 之间的语义匹配度.

算法 1. 输出接口匹配 *OutInterfaceMatch*.
输入: $OutReqSet = \{outR_1, ouR_2, \cdots, outR_m\}$,操作输出集 *Outset* 和输出权重数组 $W_o = \{w_1, w_2, \cdots, w_m\}$;
输出: 语义匹配度 *outMatchDeg*;
1. **Set** *outMatchDeg* =0; //初始化语义匹配度
2. **Define** *maxMatchArr*:Array[0..m-1];
3. **For** each $outR_i \in OutReqSet$ do { //循环请求接口
4. $maxMatchArr[i] = FindMaxMatch(outR_i, OutSet)$;
5. **If** ($maxMatchArr[i]$ = =0) //$outR_i$ 最大匹配失败
6. **Return** $maxMatchArr[i]$;
7. }
8. $outMatchDeg = \sum_{i=1}^{m}(w_i \times maxMatchArr[i])$;
9. **Return** *outMatchDeg*;

算法 1 以服务请求输出接口集,服务操作输出接口集和输出权重数组作为输入条件,返回输出接口的语义匹配度. 该算法调用(算法 2)单接口最大匹配算法,依次获取服务请求单接口的最大语义匹配度. 算法中利用一个最大匹配数组 *maxMatchArr*,存放每个服务请求输出接口的最大语义匹配度. 若最大语义匹配度为 0,则匹配失败,返回语义匹配度为 0;否则,将语义匹配度存入数组中. 最后,根据输出参数权重数组加权综合计算输出接口语义匹配度.

算法 2. 单接口最大匹配 *FindMaxMatch*.
输入: $outR, OutSet = \{outP_1, outP_2, \cdots, outP_n\}$;
输出: 最大语义匹配度 *maxMatchDeg*;
1. **Set** *maxMatchDeg* = *curMatchDeg* = *flag* =0;
2. **For** each $outP_i \in OutSet$ do { //循环操作接口
3. **If** ($SameAs(outR, outP_i)$) {//满足 *rule* 3
4. *maxMatchDeg* =1;
5. $MatchedOutSet.append(outP_i)$;
6. **Return** *maxMatchDeg*;
7. } **Else If** ($outR, outP_i$ 满足 *rule* 1 或 *rule* 2)
8. 计算 $Semantic\_dist(outR, outP_i)$;
9. **Else** *continue*; //无可达路径
10. 计算 $curMatchDeg = seMatDeg(outR, outP_i)$;
11. **If** ($curMatchDeg > maxMatchDeg$) { //更新最大值
12. $maxMatchDeg = curMatchDeg$; $flag = i$;}
13. }
14. **If** ($flag \neq 0$)
15. $MatchedOutSet.append(outP_{flag})$; //加入匹配输出集
16. **Return** *maxMatchDeg*;

算法 2 以服务请求的一个输出接口 *outR* 和服务操作的输出接口集 *OutSet* 为输入条件,返回 *outR* 在 *OutSet* 中的最大语义匹配度. 该算法以上节中的四条规则作为接口间的位置关系的判断条件,依次将 *outR* 与 *OutSet* 中的接口参数 $outP_i$ 相匹配. 若 *outR* 与 $OutP_i$ 是同一个节点(*line* 3),则最大匹配完成,将 $outP_i$ 存入匹配输出集,返回最大匹配度为 1;若 *outR* 与 $outP_i$ 满足规则 1 或 2 中的 $Grandfather(outR, outP_i)$ 或 $Descendant(outR, outP_i)$ 关系(*line* 7),利用公式(5)计算其语义距离;否则满足规则 4,继续匹配输出接口. 第 10 行利用公式(6)计算接口间的语义匹配度. 第 11-12 行更新最大匹配度,并将对应的匹配接口标记存放至 *flag*. 最后,若通过循环匹配出最大接口,将其存入匹配输出集,并返回最大语义匹配度.

文献[2]中的服务匹配算法要求服务请求与发布的服务在语义上是匹配的,首先必须满足服务请求的输入集包含发布的服务输入集;同时发布的服务输出集包含服务请求的输出集. 但该算法对请求者提供输入参数的要求过于苛刻,容易导致在许多情况下满足用户请求的服务因未能匹配成功而被过滤掉. 例如,对于一个操作 *opr* 含有三个输入参数$\{a,b,c\}$和两个输出参数$\{r,s\}$,服务请求 *req* 含有两个输入参数$\{a,b\}$和一个输出参数$\{r\}$. 由于服务请求的输入参数中不包含 *c*,因此在文献[2]的算法中 *req* 与 *opr* 匹配失败. 然而,如果操作中的输出参数 *r* 仅与输入参数 *a* 和 *b* 相关,则服务请求中的输入参数只要包含 *a* 和 *b* 就能够满足操作的输入接口匹配需求,从而服务操作 *opr* 满足服务请求 *req* 的匹配.

在输入接口的匹配中,本文考虑操作的输入/输出接口映射关系 *fMap*,先给出必备输入集(*EssenInSet*)的定义:

定义 8(必备输入集). 给定一个服务请求 *req* 和服务操作 $opr = \{oprName, InSet, OutSet, fMap\}$,且 *req* 对应服务操作 *opr* 的匹配输出集为 *MatchedOutSet*. 对于 $\forall outP_j \in MatchedOutSet$,存在一个输入/输出接口的参数映射集合 $fMap(outP_j) \subseteq InSet$,则必备输入集为:

$$EssenInSet = \cup\{fMap(outP_j) \mid outP_j \in MatchedOutSet\} \quad (8)$$

必备输入集是由操作中的输入/输出接口的参数映射关系,生成匹配输出集映射的操作中的输入接口子集,它描述了服务操作中满足服务请求必备的最小输入接口子集.

给定的一个服务请求 *req* 和服务操作 *opr*,由定义 7 和 8 给出的匹配输出集和必备输入集,计算 *req* 和 *opr* 在输入接口集上的语义匹配度. 首先,由输出接口匹配算法(本节中算法 1 和算法 2),生成服务请求 *req* 的输出接口集对应于服务操作 *opr* 的匹配输出集为 *MatchedOutSet*;然后,由操作中的输入/输出接口映射函数,计算匹配输出集对应的必备输入集 *EssenInSet*;最后,将必备输入集中的参数依次与服务请求 *req* 中输入接口集最大匹配后存入数组 *maxMatArr* 中. 若数组中有最大匹配度为 0,输入接口的语义匹配度为 0;否则,通过输入权重数组加权计算输入接口的语义匹配度.

$$inMatchDeg = \begin{cases} \sum_{j=1}^{|EssenInset|}(w_j \times maxMatArr[j]), & maxMatArr[j] \neq 0 \\ 0, & \text{其他} \end{cases} \quad (9)$$

4.4 服务匹配算法

给定一个服务请求 *req*,服务库 $WSB = \{ws_1, ws_2, \cdots, ws_t\}$,算法 3(*ServiceMatch*)描述了 Web 服务匹配的过程.

算法 3. 服务匹配 *ServiceMatch*.
输入:服务请求 *req*,Web 服务库 *WSB* 和匹配阈值 θ;
输出:服务排序列表 *wsRankedList*;
1. **Set** *wsRankedList* = NULL; *wsMatchDeg* = 0;
2. **For** each $ws_i \in WSB$ do { //依次匹配 *WSB* 中的服务
3. $wsMatchDeg = CalMatchDeg(req, ws_i)$;
4. **If** ($wsMatchDeg \geqslant \theta$) { //插入满足阈值的服务
5. 计算 ws_i 插入位置 *pos*;
6. *wsRankedList. insert*(ws_i, *pos*);}
7. }
8. **Return** *wsRankedList*;

算法 3 以一个服务请求 *req*,Web 服务库 *WSB* 和匹配阈值 θ 为输入条件,返回满足阈值条件且经排序的服务列表. 算法中依次计算 Web 服务库的每个 Web 服务 ws_i 与服务请求 *req* 的语义匹配度(调用算法 4). 对满足匹配阈值条件 θ 的 Web 服务,计算其在服务排序列表 *wsRankedList* 中的插入位置(*line* 5),并将其插入至服务排序列表中对应的位置(*line* 6). 最后,将匹配结果返回给服务请求者.

算法 4. 服务匹配度计算 *CalMatchDeg*.
输入:服务请求 *req*,服务 *ws* 和权重参数 $W = \{w_n, w_{in}, w_{out}\}$;
输出:服务匹配度 *wsMatchDeg*;
1. **Set** *wsMatchDeg* = *curWsDeg* = 0;
2. **Set** *nameDeg* = *inDeg* = *outDeg* = 0;
3. **For** each $opr_i \in ws.OPrSet$ do { //依次匹配服务操作
4. $nameDeg = Sim_{name}(req.reqName, opr_i.oprName)$;
5. 计算输出接口匹配度 *outDeg*;
6. **If** (*outDeg* = =0) *continue*; //输出匹配失败
7. 计算输入接口匹配度 *inDeg*;
8. **If** (*inDeg* = =0) *continue*; //输入匹配失败
9. $curWsDeg = w_n * nameDeg + w_{in} * inDeg + w_{out} * outDeg$;
10. **If** (*curWsDeg* > *wsMatchDeg*) //更新最大匹配操作
11. *wsMatchDeg* = *curWsDeg*;
12. }
13. **Return** *wsMatchDeg*;

算法 4 以一个服务请求 *req*,Web 服务 *ws* 和权重数组 *W* 为输入条件,返回服务的语义匹配度. 算法将服务请求参数与每个服务操作 opr_i 中对应的接口依次进行匹配. 首先计算服务请求名称和操作名称间的关键词匹配相似度(*line* 4);然后利用算法 1 和 2 计算 *req* 与 opr_i 在输出接口集上的语义匹配度(*line* 5). 若输出接口匹配失败,重新匹配服务操作(*line* 6);同样地,利用公式(9)计算 *req* 与 opr_i 在输入接口集上的语义匹配度(*line* 7). 若输入接口匹配失败,重新匹配服务操作(*line* 8);再综合加权计算服务语义匹配度(*line* 9). 最后,更新最大匹配的服务操作,并返回服务语义匹配度.

5　仿真实验与分析

5.1　实验环境与数据集

为了验证所提出的 Web 服务发现方法的有效性,本文设计实现了一个基于语义匹配度计算的 Web 服务发现原型系统 *SeMatDeg – DS*(Semantic Match Degree for Web Service Discovery System),并结合已存在的服务发现技术进行了仿真实验. 以信息检索中的查全率(*recall*)和查准率(*precision*)两个指标评价 Web 服务发现的效果. 服务查全率是指匹配出的相关服务数 S_{cn} 与服务库中所有的相关服务数量 S_{tcn} 之比;服务查准率是指匹配出的相关服务数量 S_{cn} 在所有匹配出的服务数量 S_{tmn} 中所占的比例. 服务查全率 S_{recall} 和服务查准率 $S_{precision}$ 的形式化表示,如公式(10)和(11)所示.

$$S_{recall} = \frac{\text{匹配出的相关服务数 } S_{cn}}{\text{服务库中所有相关服务数 } S_{tcn}} \tag{10}$$

$$S_{precision} = \frac{\text{匹配出的相关服务数 } S_{cn}}{\text{匹配出的所有服务数 } S_{tmn}} \tag{11}$$

在原型系统的实验配置环境中,使用的 CPU 为 Pentium IV 2.4 GHZ,内存为 1G,操作系统为 Windows XP. 本体编辑工具采用斯坦福大学生物医学信息研究中心开发的 Protégé 3.3.1,构建了一个约含 220 个概念,实例和属性的交通工具分类领域本体. 使用 Eclipse 3.1 作为实验集成开发环境,并利用 Java 和 JSP 程序开发语言设计和实现服务发现方法中的接口匹配算法,服务匹配算法和人机界面模块. 使用 Apache Tomcat 5.0.28 作为服务发现过程中人机交互的信息服务器. 以 Racer 作为语义描述逻辑推理机,实现算法中两个节点在本体结构树中的位置关系和计算它们之间的语义距离.

为了测试和比较 *SeMatDeg-DS* 和其它已有的服务匹配方法在服务发现中的查全率和查准率,本文从互联网中获取了 350 个交通工具领域以 WSDL 文件描述的 Web 服务作为实验测试的数据集,并分别形成基于 WSDL 和 UDDI 注册与发现机制的 JAXR Service Registry 服务库,文献[8]中基于 OWL-S 描述语言的 JUDDI Extended Registry 服务库和本文实验原型系统中基于 *WS-SDM* 语义描述模型的服务库. 其中,*WS-SDM* 语义描述模型通过定义 XML 模式实现,为 Web 服务中的每个服务操作创建输入/输出接口集的语义标注域,用于存放输入/输出接口集中参数的语义描述信息. 在服务匹配的过程中,利用语义标注域中的本体概念,实例和属性计算服务接口间的语义距离和语义匹配度.

5.2　结果比较与分析

实验 1:以城市交通工具领域相关的五组服务请求{R_1, R_2, R_3, R_4}作为

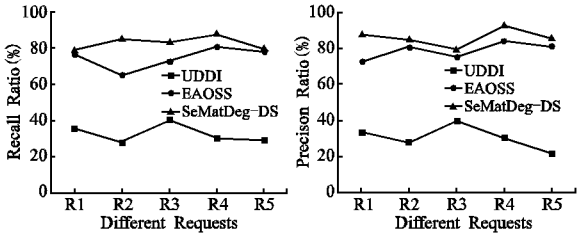

图 3　三种服务发现方法的查全率和查准率比较
Fig. 3　Comparison of recall and precision of three methods

实验测试条件,每组服务请求的输入/输出接口集中包含 3 ~ 5 个该领域的接口参数. 根据实验的效果比较和分析,设置以下几组实验参数:(1) $\alpha = \beta = \gamma = 2$;(2) $\theta = 0.75$;(3) $w_n = 0.2$, $w_{in} = 0.35$, $w_{out} = 0.45$;(4) $w_1 = w_2 = \cdots = 1/m$(或 $1/|Ess-$

enInset|). 以纳入该组请求的相关服务集作为服务发现对象,分别用 UDDI 表示基于关键词匹配的服务发现方法,EAOSS 表示文献[8]中语义包含关系推理的服务发现方法,SeMatDeg-DS 表示本文基于语义匹配度计算的服务发现方法. 三种服务发现方法的查全率和查准率,如图 3 所示.

由图 3 中的五组服务请求的查全率和查准率实验结果,计算出 SeMatDeg-DS 方法的平均查全率和平均查准率分别为 82.74% 和 85.82%;EAOSS 方法的平均查全率和平均查准率分别为 74.38% 和 78.80%;UDDI 方法的平均查全率和平均查准率分别为 32.36% 和 30.72%. 因此,在设定匹配阈值 θ 的情况下,本文提出的方法具有较好的服务发现效果.

实验 2. 在不考虑匹配阈值 θ 的情况下,其它几组实验参数同实验 1 不变,仍以实验 1 中的五组服务请求作为测试条件,计算三种服务发现方法在 *Top-k*($k=10,20,30$)下服务的平均查准率和综合平均查准率. 其中,*Precision@ Top-k* 为:

$$Precision@\ Top-k=\frac{\text{relevant services number in Top-k}}{\text{Top-k services number}} \quad (12)$$

公式(12)表示,在返回的服务发现列表的前 k 个服务中相关服务数所占的比例. Top-k 下三种服务发现方法的平均查准率和综合平均查准率比较结果,如表 1 所示.

表 1 三种服务发现方法的 Top-k 效果比较

Table 1 Comparison of Top-k effect among three service discovery methods

Method / Top-k	UDDI	EAOSS	SeMatDeg-DS
Mean Precision@ Top-10	0.4320	0.8160	0.9240
Mean Precision@ Top-20	0.3145	0.7605	0.8535
Mean Precision@ Top-30	0.2444	0.5889	0.8156
Synthetic Mean Precision	0.3303	0.7218	0.8644

综合分析图 3 和表 1 的仿真实验效果,本文利用语义匹配度计算的服务发现方法与其它两种方法相比较,具有较好的服务发现效果. 首先,在计算 Web 服务与服务请求的输入/输出单接口语义匹配度的过程中,充分考虑了影响语义距离的三种重要因素,使得单接口匹配结果更能够反映接口参数间的语义联系,从而提高了服务发现的查准率;其次,在服务请求与服务操作的输入接口集匹配过程中,依据服务操作中输入/输出接口间的映射关系,通过必备输入集匹配服务请求者的输入接口集,降低了服务匹配过程中对服务请求者提供输入接口集的要求,从而提高了服务发现的查全率.

同时,对利用语义匹配度计算的服务发现方法的时间效率进行了比较和分析. SeMatDeg-DS 方法介于标准 UDDI 服务发现方法和 EAOSS 服务发现方法之间. 因此,本文提出的服务发现方法具有有效性和可行性.

6 结束语

本文提出了一种利用语义匹配度计算的 Web 服务发现方法. 首先给出一个 Web 服务功能接口和服务请求的语义描述模型,表达服务的功能语义;然后考虑影响本体有向概念结构树中两个节点间语义距离的三种重要因素,并给出了服务请求和服务操作中单接口语义匹配度的计算方法;在此基础上,分别给出了输入/输出接口匹配算法和服务匹配算法. 该方法在功能接口的语义匹配计算过程中,依据节点间的语义关系权重,节点深度权重和分布密度权重等因素,能够更加精确地计算语义匹配度,提高服务的查准率;在输入接口集匹配过程中,考虑了输入/输出接口的映射关系,从而降低了服务请求者提供输入接口集的苛刻程度,提高了服务发现的查全率. 仿真实验表明提出的方法具有有效性和可行性.

本文对 Web 服务发现方法进行了初步的探索,仍有许多问题有待于进一步研究和解决. 下一步的工作重点是将服务质量(QoS)考虑到服务匹配和服务选择中,同时对服务发现过程中参数值的自动和有效选择进行深入研究,确保选择的实验参数能够使得服务发现效果达到最优化.

References:

[1] The OWL services coalition. OWL-S: semantic markup for web services[EB/OL]. http://www.daml.org/services/owl-s/1.0/owl-s.html,2004.

[2] Paolucci M, Kawamura T, Payne T R, et al. Semantic matching of web services capabilities[C]. In Proc. of the First International Semantic Web Conference, Sardinia, Italy: Springer-Verlag, 2002, 333-347.

[3] Burstein M H, Hobbs J R, Lassila O, et al. DAML-S: web service description for the semantic web[C]. In Proc. of the First International Semantic Web Conference. Sardinia: Springer-Verlag, 2002,348-363.

[4] Hau J, Lee W, Darlington J. A semantic similarity measure for semantic web services[C]. In Proc. of the 14th Intl. World Wide Web Conference (WWW 2005). Japan: ACM Press, 2005.

[5] Wu Jian, Wu Zhao-hui, Li Ying, et al. Web service discovery based on ontology and similarity of words[J]. Chinese Journals of Computers, 2005, 28(4):595-602.

[6] Zhang Zheng, Zuo Chun, Wang Yu-guo. Web service discovery method based on semantic expansion[J]. Journal on Communications, 2007, 28(1):57-63.

[7] Studer R, Benjamins V R, Fensel D. Knowledge engineering, principles and methods[J]. Data and Knowledge Engineering, 1998, 25(1-2):161-197.

[8] Srinivasan N, Paolucci M, Sycara K. An efficient algorithm for OWL-S based semantic search in UDDI[C]. In Proc. of the First Intl. Workshop on Semantic Web Services and Web Process Composition. San Diego, USA: Springer-Verlag, 2004,96-110.

[9] Deng Shui-guang, Yin Jian-wei, Li Ying, et al. A method of semantic web service discovery based on bipartite graph matching [J]. Chinese Journal of Computers, 2008, 31(8):1364-1375.

附中文参考文献:

[5] 吴 健, 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.

[6] 张 正, 左 春, 王裕国. 基于语义扩展到 Web 服务发现方法[J]. 通信学报, 2007, 28(1):57-63.

[9] 邓水光, 尹建伟, 李 莹, 等. 基于二分图匹配的 Web 服务发现[J]. 计算机学报, 2008, 31(8):1364-1375.