ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.8, August 2007, pp.1911−1921
DOI: 10.1360/jos181911
© 2007 by *Journal of Software*. All rights reserved.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

使用倒排索引优化面向组合的语义服务发现*

邝 砾, 邓水光+, 李 莹, 吴 健, 吴朝晖

(浙江大学 计算机科学与技术系,浙江 杭州 310027)

Using Inverted Indexing to Facilitate Composition-Oriented Semantic Service Discovery

KUANG Li, DENG Shui-Guang+, LI Ying, WU Jian, WU Zhao-Hui

(Department of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

+ Corresponding author: Phn: +86-571-87951647, Fax: +86-571-87953079, E-mail: dengsg@zju.edu.cn, http://www.cs.zju.edu.cn

Kuang L, Deng SG, Li Y, Wu J, Wu ZH. Using inverted indexing to facilitate composition-oriented semantic service discovery. *Journal of Software*, 2007,18(8):1911−1921. http://www.jos.org.cn/1000-9825/18/1911.htm

Abstract: It is proposed to establish inverted indexing for ontology-annotated outputs when services are registered in order to find the target services in a quick, accurate and efficient way. For each ontology-annotated output, there is a service list which records all the services in the registry that deliver the output. Based on the indexing, a composition-oriented service discovery algorithm is proposed, which greatly accelerates the filtering of irrelevant atomic services by making use of the inverted indexing, and increases the likelihood of finding a possible candidate by exploring service composition. The results of the extensive experiments show that the proposed algorithm provides better performance on response time than the sequential matchmaking, and better recall rate than the algorithms without the exploration of composition.
Key words: Web service; service discovery; service composition; inverted indexing; semantic similarity

摘 要: 提出为服务库中所有注册服务的输出建立倒排索引,以快速、准确、高效地发现目标服务.即为每个输出维护一个服务列表,用于记录在该服务库中所有能够产生该输出的服务.基于倒排索引机制,提出面向组合的服务发现算法.该方法利用倒排索引的优势,极大地减少了搜索空间,并通过挖掘服务组合提高服务发现的查全率.仿真实验表明,该方法能够在大规模服务库中快速、全面地响应用户请求.
关键词: Web 服务;服务发现;服务组合;倒排索引;语义相似度
中图法分类号: TP311 **文献标识码**: A

Web服务是一种基于Web环境的自包含、自描述、模块化且具有良好互操作能力的新应用,可以通过网络发布、定位和调用[1].近年来,随着Web服务标准的持续完善以及支撑Web服务的企业级应用平台的不断成熟,越来越多的企业和商业组织参与到软件服务化的行列中来,纷纷将其业务功能包装成Web服务发布出去,实现

* Supported by the National Natural Science Foundation of China under Grant Nos.60603025, 60503018 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z171 (国家高技术研究发展计划(863)); the National Key Technology R&D Program of China under Grant No.2006BAH02A01 (国家科技支撑计划); the Natural Science Foundation of Zhejiang Province of China under Grant No.Y105463 (浙江省自然科学基金)

Received 2007-03-01; Accepted 2007-04-26

© 中国科学院软件研究所 http://www.jos.org.cn

快速、便捷地寻求合作伙伴,挖掘潜在客户和增值业务的目的.Web服务在商业、金融和旅游等领域得到了广泛应用,随之而来的,是发布在网络上的服务呈现出爆炸性增长的趋势,这也使得如何以快速、准确和高效的方式发现目标服务成为一个迫切需要解决的问题.

基于UDDI[2]的服务发现机制使用关键字匹配技术,往往导致服务发现精度不够甚至失败.因此,近年来,国内外研究学者围绕基于语义的服务发现展开了广泛的研究.Paolucci等人[3]提出了第一种通过匹配服务概述(service profile)的基于语义的服务发现算法.Bansal和Vidal[4]提出了第一种通过分析服务模型(service model)的服务发现算法.由于语义的引入,本体的推理能力在服务发现中得到了灵活应用.史忠植等人[5]提出了基于描述逻辑的主体服务匹配算法.马应龙等人[6]提出了使用分布式描述逻辑扩展描述异构本体,并用优先分布式知识库描述本体的进化和更新,以适应语义Web服务的动态发现.这些服务匹配策略通常是通过顺序扫描服务库中所有注册服务的方式来实现的,这种方法也称为顺序查找.在使用这样的方法之前,服务库中的服务信息未做任何形式的预处理.当执行服务查询时,所有的注册服务逐一地与服务请求进行匹配比较.这种方法相对直观,容易实现,但是,当服务库中的服务数量达到一定数量级别时,这种方法的效率就会成为制约算法性能的瓶颈.

另一方面,大部分的服务匹配策略只检查单个服务能否满足服务请求.例如,假设服务库中仅有一个扫描服务和一个打印服务,而服务请求需要一个复印服务.许多现有的服务匹配算法将返回空的结果集,因为服务库中没有服务能够独立地满足该请求.然而,我们可以首先通过扫描服务将纸质文件转化为电子文件,然后通过打印服务将得到的电子文件转化为纸质文件,从而实现复印服务的功能.对于在服务发现中融入服务组合,已有一些研究工作有所涉及.Brogi等人[7]提出了一种面向组合的服务发现算法.该算法首先为服务库中的所有原子服务建立关系图,然后对于每个请求,算法通过分析该图来决定能否通过组合服务来满足该请求.该方法的潜在缺点在于其算法的高度复杂性,因为文中给出的仅有两个服务的服务库的构造和分析已经十分复杂,并且没有实验数据能够表明该方法的效率.Aversano等人[8]也提出了一种通过组合来优化现有服务发现的算法.该算法使用后向链的思想来验证能否通过组合服务来匹配请求.它以效率为代价,提高了发现可用服务的可能性,因为它需要经常顺序扫描服务库中的所有服务.Xie等人[9]提出了面向组合、基于与或图的服务搜索方法.该方法通过启发函数降低搜索空间,但是还需要进一步的实验评估来验证它的适应性和可行性.

基于以上分析,本文引入倒排索引的概念,即一种面向单词的索引机制,为所有注册服务的输出建立倒排索引.对于所有输出,都维护一个服务列表,用于记录在服务库中所有能产生该输出的服务.倒排索引可以快速地定位候选服务,降低搜索空间.基于这种机制,提出一种面向组合的服务匹配算法.该算法由原子级服务发现和组合级服务发现组成.原子级服务发现检查服务库中是否存在原子服务能够独立地满足服务请求,而组合级服务发现作为原子级发现的补充,检查能否即时组合原子服务来满足服务请求*.

本文第 1 节介绍如何为服务库建立倒排索引.第 2 节提出面向组合的服务发现算法.第 3 节是仿真实验和结果分析.最后是总结和展望.

1 建立倒排索引

对于原子服务库,一个服务可以被看成一个原子的、不可分解的流程,也就是我们所说的黑盒视图[10].在这种视图下,一个服务等同于一个操作,所有的内部过程被屏蔽,对外暴露的只有它的输入和输出接口.因此,我们可以抽取OWL-S服务概述的接口描述部分,将服务和服务请求定义成如下形式:

定义 1(服务). 服务是对服务所支持的一个操作的抽象描述,可以表示为一个三元组 $S=\langle N.In,Out\rangle$,其中:

(1) N 表示服务名称,作为服务的唯一标识;

(2) $In=\{In_1,In_2,\ldots,In_k\}$是服务的输入集合,其中每个输入都用本体进行标注;

(3) $In=\{Out_1,Out_2,\ldots,Out_l\}$是服务的输出集合,其中每个输出都用本体进行标注.

* 根据OWL-S[10]规范,Web服务可被划分为原子和合成服务.原子服务对应于通过一次交互完成的服务,而合成服务对应于需要多个动作完成的服务.由于原子服务和合成服务的不同特性和不同侧重,我们将针对原子服务和针对合成服务的发现策略区分开来.本文的服务发现策略针对的是原子服务.

© 中国科学院软件研究所 http://www.jos.org.cn

定义 2(服务请求). 服务请求是对接口需求的抽象描述,可以表示为二元组 $SR=\langle I,O\rangle$,其中:

(1) $I=\{I_1,I_2,\ldots,I_m\}$是可以提供的输入集合,其中每个输入都用本体进行标注;

(2) $O=\{O_1,O_2,\ldots,O_n\}$是需要的输出集合,其中每个输出都用本体进行标注.

相应地,基于黑盒视图的服务发现策略通常认为,判定一个服务能否服务请求,首先需要检验该服务能否提供请求所需的所有输出;如果能,则再检验服务请求能否提供调用该服务所需的所有输入.顺序查找的方法要求服务库里的所有注册服务都逐一地与服务请求进行这样的匹配比较.当服务库中有成千上万条记录时,很多时间会浪费在匹配比较一些毫不相关的服务上.

事实上,我们可以为所有注册服务的每个不同语义的输出分别创建一条倒排索引记录.如定义 3 所示,倒排索引由两部分组成:输出作为键,服务库中所有能产生该输出的服务列表作为键值.而服务列表中的每个元素又由〈服务标识,相关度〉二元组表示,其中,相关度表示该服务可以产生的输出与该输出索引键的相似程度,它的值域为(0,1).引入相关度是因为,一个服务不仅能产生它所声明的输出,也能产生该输出在本体中的所有父辈概念.例如,一个服务如果能提供轿车,就能满足需要交通工具的服务请求[3].因此,我们在服务库中维护一个领域无关本体,使其覆盖所有注册服务使用到的本体概念;当一个服务注册时,不仅生成从输出到服务的索引记录,同时也生成从该输出的父辈概念到服务的索引记录.为了区分该服务产生它所声明的输出和输出的父辈概念的能力的不同,我们定义从服务声明的输出到该服务的索引记录,其相关度为 1;从输出的父辈概念到该服务的索引记录,其相关度为输出与输出的父辈概念之间的语义相似度.

定义 3(倒排索引结构).

```
class ServiceNode{
    String service_id;
    float relativity;
}
class InvertedIndex{
    String output;
    ArrayList〈ServiceNode〉 service_list;
}
```

本体概念间的语义相似度计算是基于语义的服务匹配问题中服务与请求的相似度计算的基础.目前,学术界提出了很多种计算方法,并在一些特定领域被证明是可行的,如文献[11–14]等.这些方法可以被划分为两大类:基于信息共享量的计算方法和基于树的计算方法.第 1 类方法认为,两个概念的相似度取决于它们所包含的相同信息量的大小.一般而言,共享的信息量越大,二者的相似度越高;第 2 类方法将所有的本体概念构建成一棵概念树,利用两个概念在概念树中的最短路径、概念在概念树中的层高以及所在层高的节点密度等作为量度指标.如,文献[11]在综合考虑这些指标之后,提出了两个概念间相似度的计算公式:

$$SimCC(C_1,C_2)=\begin{cases} e^{-\alpha l}\cdot\dfrac{e^{\beta h}-e^{-\beta h}}{e^{\beta h}+e^{-\beta h}}, & \text{if } (C_1\neq C_2) \\ 1, & \text{if } (C_1=C_2) \end{cases} \tag{1}$$

其中,l 表示两个概念间的最短路径的长度;h 表示两个概念在树中最近的相同父辈概念在树中的高度;$\alpha,\beta\geq 0$ 是调节因子,据文献[11]中的实验,当$\alpha=0.2,\beta=0.6$ 时效果最佳.公式表明,两个概念的相似度随 l 递减,随 h 递增.该方法通用性较好,容易实现,因此在本文中我们选用该方法作为两个本体概念间的语义相似度的计算方法.值得注意的是,这里的语义相似度并不局限于某一种计算方法,任何相似度计算方法均可用在此处.

以如图 1 所示的一个OWL-S服务概述的接口描述部分为例[15],它描述了一个汇率转换服务,该服务以一个量值和两种货币代码作为输入参数,返回从一种货币转换到另一种货币得到的新值.当该服务注册时,提取出服务输出*finalAmount*以及服务标识*SontonCCWS*,生成一条从*finalAmount*到〈*SontonCCWS*,1〉的索引记录.如果*finalAmount*的索引记录不存在,则生成该索引键,并将〈*SontonCCWS*,1〉作为它对应的服务列表的第 1 个成员插

© 中国科学院软件研究所 http://www.jos.org.cn

入;否则,⟨*SontonCCWS*,1⟩将直接添加到*finalAmount*的服务列表的末端.同时,该输出的每一个父辈概念也将创建或更新索引记录.例如,*Number*是*finalAmount*在本体中的父辈概念,一条从*Number*到⟨*SontonCCWS*, *SimCC*(*finalAmount*,*Number*)⟩的索引记录也将生成.图 2 为上述服务注册时,服务库的倒排索引表更新的示意图.

```
⟨Information_Translation_Service rdf:ID="SotonCCWS"⟩
  ⟨profile:input⟩
    ⟨profile:ParameterDescription rdf:ID="OriginalAmount"⟩
      ⟨profile:parameterName⟩OriginalAmount⟨/profile:parameterName⟩
      ⟨profile:restrictedTo rdf:resource="#Amount"/⟩
      ⟨profile:refersTo rdf:resource="http://www.daml.ecs.soton.ac.uk/currencyconverterprocess.daml#originalAmountIn"/⟩
    ⟨/profile:ParameterDescription⟩
  ⟨/profile:input⟩
  ⟨profile:input⟩
    ⟨profile:ParameterDescription rdf:ID="OriginalCurrency"⟩
      ⟨profile:parameterName⟩OriginalCurrency⟨/profile:parameterName⟩
      ⟨profile:restrictedTo rdf:resource="http://www.daml.ecs.soton.ac.uk/ont/currency.daml#Currency"/⟩
      ⟨profile:refersTo rdf:resource="http://www.daml.ecs.soton.ac.uk/currencyconvertprocess.daml#originalCurrencyIn"/⟩
    ⟨/profile:ParameterDescription⟩
  ⟨/profile:input⟩
  ⟨profile:input⟩
    ⟨profile:ParameterDescription rdf:ID="FinalCurrency"⟩
      ⟨profile:parameterName⟩FinalCurrency⟨/profile:parameterName⟩
      ⟨profile:restrictedTo rdf:resource="http://www.daml.ecs.soton.ac.uk/ont/currency.daml#Currency"/⟩
      ⟨profile:refersTo rdf:resource="http://www.daml.ecs.soton.ac.uk/currencyconverterprocess.daml#finalCurrencyIn"/⟩
    ⟨/profile:ParameterDescription⟩
  ⟨/profile:input⟩
  ⟨profile:output⟩
    ⟨profile:ParameterDescription rdf:ID="FinalAmount"⟩
      ⟨profile:parameterName⟩FinalAmount⟨/profile:parameterName⟩
      ⟨profile:restrictedTo rdf:resource="#Amount"/⟩
      ⟨profile:refersTo rdf:resource="http://www.daml.ecs.soton.ac.uk/currencyconverterprocess.daml#finalAmountOut"/⟩
    ⟨/profile:ParameterDescription⟩
  ⟨/profile:output⟩
⟨/Information_Translation_Service⟩
```

Fig.1 An IO description segment of an OWL-S profile for "SontonCCWS" service

图 1 SontonCCWS 服务在 OWL-S 服务概述中的接口描述片断

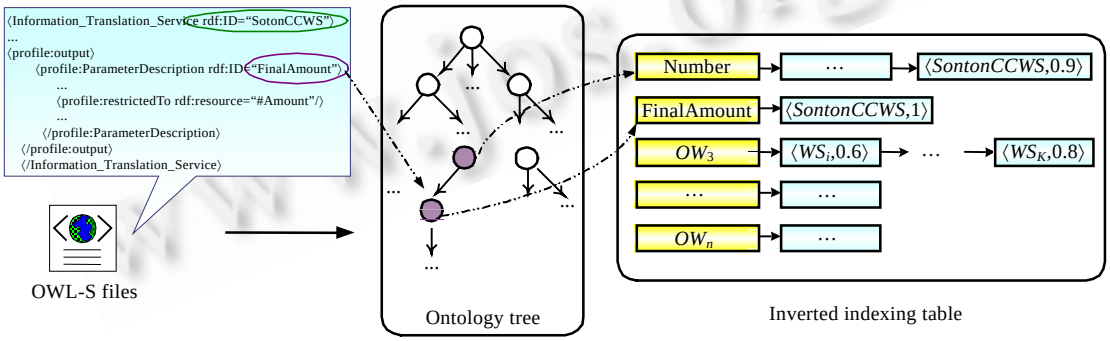

Fig.2 An example of updating the inverted indexing when the "SontonCCWS" service is registered

图 2 SontonCCWS 服务注册时倒排索引更新的示例

定义 4. 根据倒排索引表,得到所有能输出本体概念 C 的服务列表的函数可表示为 $S=f(C)$.

普通的服务库建立的是从服务到输出/输入的映射关系,在使用倒排索引技术之后,建立的是从输出到服务的映射关系.由于服务匹配通常从检查服务是否满足输出上的要求开始,倒排索引能够快速定位候选服务,从而

© 中国科学院软件研究所 http://www.jos.org.cn

更好地适应这种发现机制.

2 面向组合的服务发现算法

本文提出的服务发现算法的目标在于:第一,利用倒排索引的优势快速筛选掉不相关的服务;第二,通过挖掘服务组合找到更多合适的候选服务.这两个目标通过如下方式实现:该算法首先检查服务库中是否有原子服务能够独立地满足服务请求,也就是我们所称的原子级服务发现;如果没有,算法就检查是否能即时组合原子服务来满足服务请求,也就是我们所称的组合级服务发现.在给出详细的算法描述之前,我们首先定义 3 个 *ArrayList*⟨*ServiceNode*⟩上的操作.

定义 5(*ArrayList*⟨*ServiceNode*⟩的交集). $L_1,L_2\in ArrayList\langle ServiceNode\rangle$, 若 $\langle s,r_1\rangle\in L_1$ 且 $\langle s,r_2\rangle\in L_2$, 那么,$\langle s,\min(r_1, r_2)\rangle\in L_1\cap L_2$.即

$$L_1\cap L_2=\{\langle s,r\rangle|s:\langle s,r_1\rangle\in L_1 \text{ and } \langle s,r_2\rangle\in L_2, r:r=\min(r_1,r_2)\}.$$

例如,$L_1=\{\langle a,0.8\rangle,\langle b,0.7\rangle\}$,$L_2=\{\langle a,0.5\rangle,\langle b,1\rangle,\langle c,0.9\rangle\}$,那么,$L_1\cap L_2=\{\langle a,0.5\rangle,\langle b,0.7\rangle\}$.

定义 6(*ArrayList*⟨*ServiceNode*⟩的笛卡尔积). $L_1,\ldots,L_n\in ArrayList\langle ServiceNode\rangle$,从$L_1,\ldots,L_n$中分别挑出其中的元素$\langle s_1,r_1\rangle,\ldots,\langle s_n,r_n\rangle$,那么,$\left\langle s_1+\ldots+s_n,\frac{1}{n^2}\sum_{i=1}^{n}r_i\right\rangle\in L_1\times\ldots\times L_n$.即

$$L_1\times\ldots\times L_n=\left\{\langle s,r\rangle \mid s:s_1+\ldots+s_n,\langle s_i,r_i\rangle\in L_i, i\in\{1,\ldots,n\}, \text{ and } r:r=\frac{1}{n^2}\sum_{i=1}^{n}r_i\right\}.$$

例如,$L_1=\{\langle a,0.4\rangle,\langle b,1\rangle\}$,$L_2=\{\langle c,0.6\rangle\}$,那么,$L_1\times L_2=\{\langle a+c,0.25\rangle,\langle b+c,0.4\rangle\}$.

定义 7(*ArrayList*⟨*ServiceNode*⟩的并集). $L_1,L_2\in ArrayList\langle ServiceNode\rangle$,若$\langle s,r\rangle\in L_1\wedge\langle s,r_2\rangle\notin L_2$或$\langle s,r\rangle\in L_2\wedge\langle s,r_1\rangle\notin L_1$,那么,$\langle s,r\rangle\in L_1\cup L_2$;若$\langle s,r_1\rangle\in L_1\wedge\langle s,r_2\rangle\in L_2$,那么,$\langle s,\max(r_1,r_2)\rangle\in L_1\cup L_2$.

例如,$L_1=\{\langle a,0.4\rangle,\langle b,1\rangle\}$,$L_2=\{\langle b,0.3\rangle,\langle c,0.9\rangle\}$,那么,$L_1\cup L_2=\{\langle a,0.4\rangle,\langle b,1\rangle,\langle c,0.9\rangle\}$.

2.1 原子级服务发现

原子级服务发现由两个步骤组成:第 1 步,找到能够提供所有请求输出的候选服务.它可以通过对请求输出在倒排索引表里的键值取交集而求出.对于一个服务请求$SR=\langle I,O\rangle$,其中,$O=\{O_1,\ldots,O_n\}$是请求的输出集合,那么,通过初次筛选的候选服务集合可以表示为

$$S=\mathop{\mathrm{I}}_{i=1}^{n} f(SR.O_i) \tag{2}$$

如果该集合不为空,则表明服务库中有服务能够提供所有的请求输出.然后,原子级服务发现执行第 2 步,即检查服务请求能否提供调用该服务所需要的输入.如果能,则该服务成功匹配请求,添加到待返回服务集.如果原子级服务发现失败,则组合级服务发现开始工作.原子级发现算法伪代码如下所示.

算法. Atomic-Level Service Discovery.

Input: A service request $SR(I,O)$;

where $I=\{I_1,\ldots I_m\}$ and $O=\{O_1,\ldots,O_n\}$.

Output: A matched service list.

METHOD:

```
Set filterS=NULL;
FOR each O_i in SR.O
    tempL=f(SR.O_i);
    IF (filterS==NULL)
        filterS=tempL;
    ELSE    filterS=filterS∩tempL;
    End IF
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
    End FOR
    Set finalResult=NULL;
    FOR each service adv in filterS
      IF adv.In semantically similar with SR.I
        finalResult.add(adv);
    End FOR
    RETURN finalResult;
```

2.2 组合级服务发现

组合级服务发现的过程可以表示为一棵树的生长过程.服务请求的输出集合作为起始目标,表示成树的根节点.树的生长过程可以由以下两条规则定义.

定义 8(横向生长规则). 对于一个目标$T[O_1,\ldots,O_n]$,通过对各个$T.O_i$在倒排索引表里的键值取交集,求出那些能够提供目标T所需的所有输出的候选服务,该过程可表示为

$$S=\mathop{\mathrm{I}}_{i=1}^{n} f(T.O_i) \tag{3}$$

由于目标 T 初始为服务请求的输出集合 $SR.O$,第 1 次运算后的候选服务集合可以直接从原子级服务发现的第 1 步结果传值过来.集合 S 中的服务将表示为树的叶节点.如果集合 S 中有多个服务,则每个叶节点代表一个候选服务.

如果没有原子服务能够独立提供T中请求的所有输出,也就是说,集合S为空,则可以由几个服务并行组合提供这些输出,即并行组合中的服务分别负责目标T中的某些输出,而它们的并集能够提供目标T的所有输出.并行组合可通过以下步骤进行尝试:挑选出目标T中的任一输出,根据倒排索引表得到它所对应的服务列表,同时计算出目标T中其他输出对应的服务集合的交集.如果这两个集合都不为空,则它们的笛卡尔积即为所有的由两个服务组合能够提供所有输出的组合,其中一个服务负责产生$T.O_n$,另一个负责产生目标T除了$T.O_n$以外的其他输出.这样的并行组合也将被表示为树的叶节点.当两个服务的并行组合失败时,3 个服务的并行组合将以相似的方式进行尝试,以此类推,如公式(4)所示.

$$\begin{cases} \mathop{\mathrm{I}}_{i=1}^{n-1} f(T.O_i)\times f(T.O_n) \\ \mathop{\mathrm{I}}_{i=1}^{n-2} f(T.O_i)\times f(T.O_{n-1})\times f(T.O_n) \\ \cdots \end{cases} \tag{4}$$

定义 9(纵向生长规则). 对于一个叶节点$\sum_{i=1}^{n} S_i$,它可以是一个原子服务或是几个服务的并行组合.根据公式(5)计算出那些需要传入却不包含在服务请求的输入集合里的输入集,将它作为新的目标.顺序组合试图通过其他服务提供这些不被服务请求所提供的输入.如果新目标为空,则树在该节点上停止生长,否则,根据树的横向生长规则生成该节点的子节点.

$$T'=\bigcup_{i=1}^{n} S_i.In-SR.I \tag{5}$$

当新的目标不再产生或者尝试超过一定时间时,组合级服务发现算法停止.当组合级发现成功时,从叶节点到根节点路径上的所有服务节点的顺序组合即为满足服务请求的即时组合,它的匹配程度为该路径上所有节点匹配度之和的平均值.

2.3 一个示例

以如图 3 所示的例子来解释我们的服务发现算法.服务库中有 6 个注册服务,分别为$S1$~$S6$.当它们注册时,

© 中国科学院软件研究所 http://www.jos.org.cn

针对它们的每个输出建立倒排索引,如图中的倒排索引表所示.其中,O_6,O_7和O_8分别与I_6,I_7和I_8语义相似.

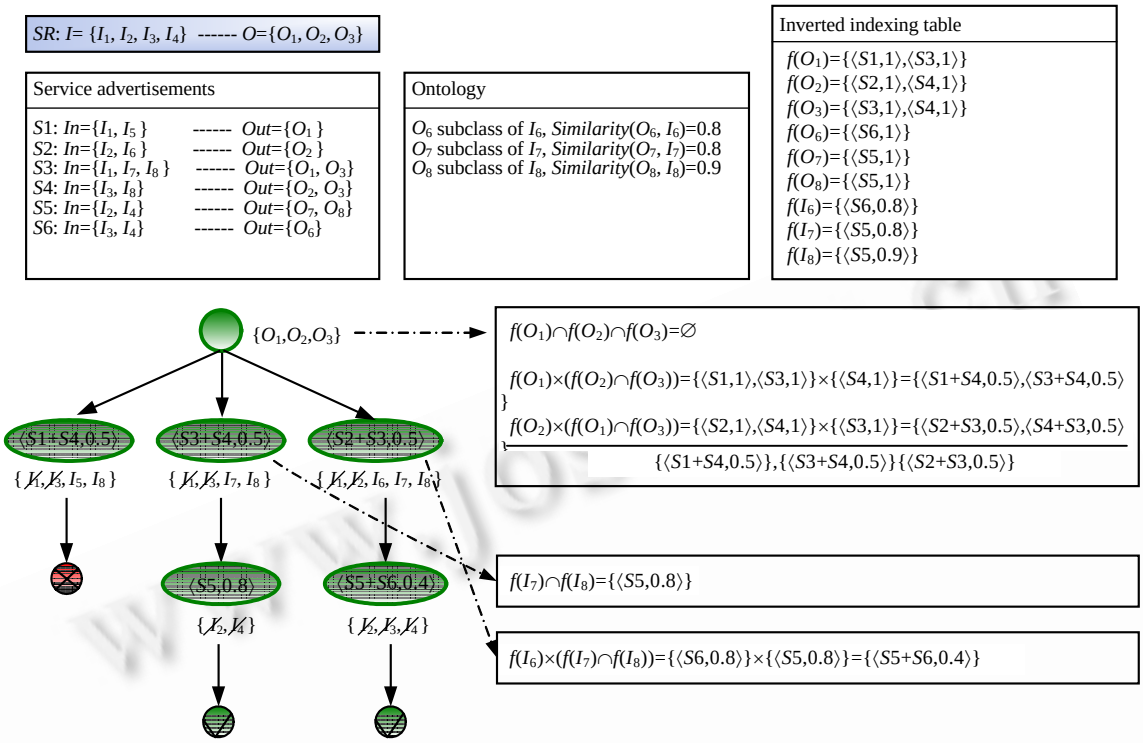

Fig.3 Matchmaking for an example request

图 3 示例请求的匹配过程

对于一个期望得到输出O_1,O_2和O_3并声称能够提供输入I_1,I_2,I_3和I_4的服务请求SR,首先原子级服务发现尝试找到一个服务能够同时提供这 3 个输出,然而该尝试失败了,因为这 3 个输出对应的服务集合的交集为空.然后,组合级服务发现开始工作.它的执行步骤如下:

1) 目标初始为请求输出O_1,O_2和O_3.由于这 3 个输出对应的服务集合的交集为空,尝试通过并行组合两个服务找到可能解,其中一个服务负责产生输出O_1,另一个服务负责产生输出O_2和O_3.根据公式(4),$\langle S1+S4,0.5\rangle$和$\langle S3+S4,0.5\rangle$是其组合结果.类似地,我们可以得到将目标分解成O_2和其余输出、O_3和其余输出的可能组合.因此,对于目标$[O_1,O_2,O_3]$,生成 3 个叶节点,分别为$\langle S1+S4,0.5\rangle$,$\langle S3+S4,0.5\rangle$和$\langle S2+S3,0.5\rangle$.

2) 对于叶节点$\langle S1+S4,0.5\rangle$,它们的输入集合的并集为$\{I_1,I_3,I_5,I_8\}$.由于请求SR仅提供了其中的I_1和I_3,对于$\langle S1+S4,0.5\rangle$,一个新的目标产生,即$\{I_5,I_8\}$.

   对于目标$\{I_5,I_8\}$,由于I_5对应的服务集合为空,服务节点不能继续生长,在这个节点上,树到达一个死节点.

3) 对于叶节点$\langle S3+S4,0.5\rangle$,它们的输入集合的并集为$\{I_1,I_3,I_7,I_8\}$.由于请求SR仅提供了其中的I_1和I_3,对于$\langle S3+S4,0.5\rangle$,一个新的目标产生,即$\{I_7,I_8\}$.

   对于目标$\{I_7,I_8\}$,I_7和I_8对应的服务集合的交集为$\langle S5,0.8\rangle$,因此,$\langle S5,0.8\rangle$作为$\langle S3+S4,0.5\rangle$的子节点生成.对于新生的叶节点$\langle S5,0.8\rangle$,它的输入集合为$\{I_2,I_4\}$,它完全包含在服务请求所提供的输入集合中.因此,无须再生成新目标,树在该节点上到达一个成功的终止.

4) 对于叶节点$\langle S2+S3,0.5\rangle$,它们的输入集合的并集为$\{I_1,I_2,I_6,I_7,I_8\}$.由于请求SR仅提供了其中的I_1和I_2,对于$\langle S2+S3,0.5\rangle$,一个新的目标产生,即$\{I_6,I_7,I_8\}$.

© 中国科学院软件研究所 http://www.jos.org.cn

对于新目标$\{I_6,I_7,I_8\}$,它们对应的服务集合的交集为空,然而,$S5$ 可以与$S6$ 并行组合起来,由$S6$ 负责产生I_6,$S5$ 负责产生其余输出.因此,$\langle S5+S6,0.4\rangle$作为$\langle S2+S3,0.5\rangle$的子节点生成.

对于新生的叶节点$\langle S5+S6,0.4\rangle$,它的输入集合为$\{I_2,I_3,I_4\}$,它完全包含在服务请求所提供的输入集合中.因此,无须再生成新目标,树在该节点上到达一个成功的终止.

5) 对于 SR,有两个组合可以满足它的要求,分别为 $S5.(S3+S4)$和$(S5+S6).(S2+S3)$,它们的匹配程度分别为 0.65 和 0.45.

2.4 算法复杂度分析

原子级服务发现的时间复杂度的计算包括 3 个部分:根据倒排索引表找出所有请求输出对应的服务列表;对这些服务集合作交集,以求出能够提供所有的请求输出的服务;最后对这个候选服务集中的服务做输入请求上的匹配比较.假设倒排索引表中的索引条目个数为k,那么,查找某个本体概念对应的服务列表的时间复杂度为$O(k)$.假设服务请求的输出集合的元素个数为n,则查找出服务请求的所有输出对应的服务列表的时间复杂度为$O(n\times k)$.给定两个分别有p和q个元素的无序集合,两个集合求交集的时间复杂度为$O(p\times q)$,由于$O(p\times q)\le O(((p+q)/2)^2)$,因此,我们假设所有输出对应的服务列表的平均长度为L,那么对n个服务集合作交集的时间复杂度为$O(n\times L^2)$.两个本体概念间的语义相似度计算的时间复杂度为$O(1)$,假设服务请求的输入个数为m,候选服务集的平均输入个数为s,则一个服务与服务请求在输入上的匹配比较的复杂度为$O(m\times s)$,而候选服务集的元素个数不会超过L,因此,第 3 部分的时间复杂度为$O(L\times m\times s)$.综合上述分析,原子级服务发现的时间复杂度为$O(n\times k+n\times L^2+L\times m\times s)$.

组合级服务发现的时间复杂度由横向生长规则和纵向生长规则的时间复杂度组成.对于一个目标$T[O_1,\ldots,O_n]$,横向生长规则的时间复杂度计算如下:挑选出一个输出$T.O_i$得到它对应的服务列表,同时计算出除了$T.O_i$之外的其他输出对应的服务列表的交集,然后计算出这两个集合的笛卡尔积,对于一个$T.O_i$做上述计算的时间复杂度为$O(n\times k+2n\times L^2)$.那么,对所有的$T.O_i$做上述计算之后再对结果求并集的时间复杂度为$O(n^2\times k+2n^2\times L^2)+O(n\times L^2)$,即横向生成规则的时间复杂度为$O(n^2\times k+2n^2\times L^2)$,其中,$n$为服务请求的输出集合的元素个数,$k$为倒排索引表中的索引条目个数,$L$为索引表中所有输出对应的服务列表的平均长度.纵向生长规则的时间复杂度计算包括:计算出候选服务集里的输入的并集,时间复杂度为$O(L\times s^2)$;再计算出该并集与请求输入集的差,并集的元素个数最大为$L\times s$,输入集的元素为m,求两个集合的差的时间复杂度为$O(L\times s\times m)$,因此,纵向生成规则的时间复杂度为$O(L\times s^2+L\times s\times m)$,其中,$s$为候选服务集的平均输入个数,$m$为服务请求的输入个数.

3 算法评测

3.1 实验准备

为了准备测试服务数据集,我们基于IBM XML Generator开发了一个符合第 2 节中提出的服务描述规范的服务文档生成工具.利用这个工具,我们生成了 5 组实验服务集,每组分别有 1 000,5 000,10 000,50 000 和 100 000 个服务描述文档.在每组数据生成过程中,为了更逼近实际数据形态,我们保证每组 25%的服务有 1 个输出,45%的有 2~4 个输出,25%的有 5~9 个输出以及 5%的有 10~15 个输出,对于输入也是同样的比例.我们从WordNet[16]中挑选出拥有大约 2 000 个本体概念的子本体,用于输入/输出的语义标注.另外,我们保证每组服务集的索引平均链长(即服务库中所有输出对应的服务列表的长度之和的平均值,以下简称为ALS)是相同的.

我们在每组数据集上进行了一系列实验.根据第 2 节中提出的服务请求描述规范随机生成 1 000 个服务请求.服务请求和服务描述文档共享同一个子本体.下列实验中记录的所有数据,都是对各个服务请求进行实验所得结果的平均值.我们的实验在 2.0-GHz Intel Xeon MP 处理器和 1GB 内存的 IBM x60 主机上运行,操作系统为 Redhat 9.0.

3.2 使用倒排索引对用户响应时间的影响

在这组实验中,我们将分别比较顺序匹配与原子级服务发现、顺序的面向组合的服务发现和组合级服务发

© 中国科学院软件研究所 http://www.jos.org.cn

现在用户响应时间上的表现.前一对服务发现策略是,当一个原子服务能够独立地满足服务请求时,它将添加到返回服务集中.后一对服务发现策略是,当一个服务能够部分满足请求在输出上的需求,或者是服务能够完全满足请求在输出上的需求,但仅能部分满足请求在输入上的需求时,它将尝试与其他服务组合共同实现服务请求的需要.

首先,我们考察当 ALS 增加而服务数量保持不变时各发现方法的用户响应时间将如何变化.我们使用 10 000 个服务的这组数据集.我们从该组数据集中分别抽取 100%,50%,10%,5%,1%和 0.01%的服务,并将服务库分别设置为 1,2,10,20,100 和 10 000 份所抽取的服务.因此,服务数量保持为 10 000,而 ALS 从 50 服务/输出增长到 10 000 服务/输出.

然后,我们考察当服务数量增长而 ALS 保持不变时各发现方法的用户响应时间将如何变化.我们使用生成的 5 组服务集作为测试数据.它们的 ALS 一致而服务数量从 1 000 到 100 000 不等.

图 4 显示了各服务发现算法的响应时间随着 ALS 的增加而产生的变化.从实验结果可以看出:1) 随着 ALS 的增长,原子级服务发现的响应时间逐渐增加,当 ALS 达到 10 000 时,原子级发现的响应时间接近但仍少于顺序匹配的响应时间;2) 顺序匹配的响应时间基本持平,但随着 ALS 的增长,仍有缓慢增长的趋势;3) 随着 ALS 的增长,两种面向组合的服务发现的响应时间都有下降的趋势;4) 比较原子级发现和顺序匹配的响应时间,前者由于使用倒排索引,响应时间总是小于后者,尤其是当 ALS 较小时,它们的差距尤为明显;5) 比较组合级发现和顺序的面向组合的服务发现的响应时间,前者由于使用倒排索引,响应时间总是小于后者.

可以看到,两种面向组合的服务发现不论使用倒排索引与否,它们的响应时间随着 ALS 的增长反而下降.这是因为 ALS 越大,服务库中不同的服务越少.比如,当 ALS 达到 10 000 时,服务库中的服务为同一个服务的 10 000 次备份,它们在接口描述上是一致的.而服务库中不同的服务越少,即时组合越容易失败.由于失败返回远快于一个成功的组合,所以,当 ALS 增加时,增加的失败的可能性反而导致响应时间上的减少.

图 5 显示了服务发现的响应时间随着服务数量增加而产生的变化.

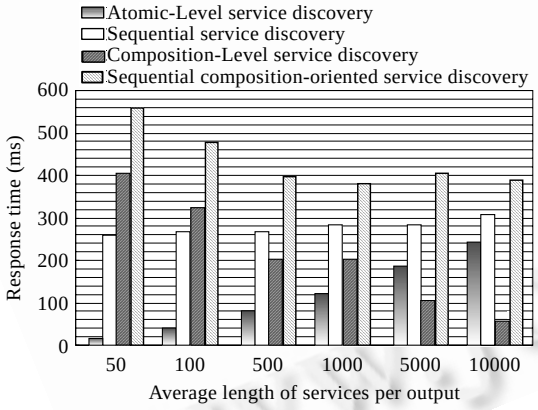

Fig.4 Changes of the response time of each algorithm with the increase of ALS while the number of services remains invariable

图 4 当 ALS 增加而服务数量保持不变时各算法响应时间的变化

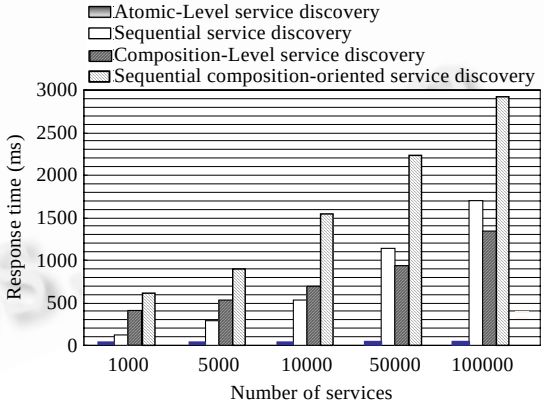

Fig.5 Changes of the response time of each algorithm with the increase of the number of services while ALS remains invariable

图 5 当服务库中服务数量增加而 ALS 保持不变时各算法响应时间的变化

从实验结果可以看出:1) 随着服务数量的增加,原子级服务发现的响应时间基本持平;2) 随着服务数量的增加,顺序匹配的响应时间迅速增长,当服务库中有 100 000 个服务时,响应时间长达 1 730ms,难以满足用户需求;3) 随着服务数量的增加,两种面向组合的服务发现的响应时间都在增长,但是组合级发现的增长速度明显比顺序的面向组合的发现的增长速度缓慢;4) 比较原子级发现和顺序匹配的响应时间,前者由于使用倒排索引,响应时间总是小于后者,特别是当服务数量很大时,两者的差距尤为明显;5) 比较组合级发现和顺序的面向

© 中国科学院软件研究所 http://www.jos.org.cn

组合的服务发现的响应时间,前者由于使用倒排索引,响应时间总是小于后者.

从以上两个实验中我们可以得出如下结论:使用倒排索引是一种提升服务发现效率的有效方法.使用倒排索引的服务发现在 ALS 较小而服务数量较多时的优势尤为明显.

3.3 挖掘即时组合对查全率和查准率的影响

在这组实验中,我们将原子级服务发现的查全率与查准率和组合级发现的查全率与查准率进行比较.查全率是指返回文档中相关的文档的比例.查全率是检索出的相关文档数和文档集中所有的相关文档数的比率.查准率是检索出的相关文档数与检索出的文档总数的比率[17].实验在 5 组测试集上进行.

表 1 显示,原子级和组合级发现的平均查全率分别为 86.58%和 91.82%,平均查准率分别为 95.28%和 89.84%.尽管组合级发现的查准率略低于原子级发现,挖掘即时组合仍不失为一种提高查全率的好方法.

Table 1 Average recall rate and precision for atomic-level and composition-level service discovery carried on five groups of test services

表 1 原子级和组合级服务发现在 5 组测试集上的平均查全率和查准率

		Group 1	Group 2	Group 3	Group 4	Group 5	Average
Atomic-Level (%)	Recall	82.3	86.4	90.6	85.5	88.1	86.58
	Precision	92.7	91.9	94.8	98.6	98.4	95.28
Composition-Level (%)	Recall	87.8	90.0	92.7	94.5	94.1	91.82
	Precision	86.8	89.8	87.5	90.2	94.9	89.84

4 结束语

本文提出使用倒排索引优化面向组合的语义服务发现方法.不同于顺序匹配,倒排索引能够快速定位候选服务.该算法利用倒排索引的优势大幅度提高了过滤不相关服务的效率,并通过服务组合方法的融合提高了发现更多候选服务的可能性.仿真实验的数据结果表明了该方法的良好性能.

文献[18,19]分别提出了基于语法和语义的倒排索引的服务发现和组合方法.与这些工作相比,我们的工作的优势在于:第一,倒排索引在服务发现和组合中的应用不仅仅局限于交集操作,我们引入笛卡尔积和并集操作,使得倒排索引与面向组合的服务发现方法能够更为理想地结合;第二,对于服务库中所有服务的不同语义的输出,在倒排索引中不仅记录所有能产生该输出的服务的集合,同时也记录这些服务产生该输出的能力,这样的倒排索引结构使得服务发现算法能够根据本体概念间的语义相似,对挑选出的服务进行排序;第三,对文中提出的方法做了仿真实验,实验数据表明,该方法能够在大规模服务库中快速、全面地响应用户请求.

今后的工作包括:研究如何将倒排索引的机制引入到针对合成服务的发现方法中,使其优化其发现方法的性能.我们的长期目标是建立一个支持服务发现的方法论.

References:

[1] Tidwell D. Web services: The Web’s next revolution. 2000. http://www-900.ibm.com/developerWorks/cn/education/webservices/wsbasics/tutorial_eng/index.shtml

[2] Bellwood T, Capell S, Clement L, Colgrave J, Dovey MJ, Feygin D, Hately A, Kochman R, Macias P, Novotny M, Paolucci M, von Riegen C, Rogers T, Sycara K, Wenzel P, Wu Z. UDDI version 3.0. 2002. http://uddi.org/pubs/uddi_v3.htm

[3] Paolucci M, Kawamura T, Payne TR, Sycara K. Semantic matching of Web services capabilities. In: Horrocks I, ed. Proc. of the 1st Int’l Semantic Web Conf. Chia: Springer-Verlag, 2002. 333−347.

[4] Bansal S, Vidal JM. Matchmaking of Web services based on the DAML-S service model. In: Rosenschein JS, ed. Proc. of the 2nd Int’l Joint Conf. on Autonomous Agents and Multiagent Systems. Melbourne: ACM Press, 2003. 926−927.

[5] Shi ZZ, Jiang YC, Zhang HJ, Dong MK. Agent service matchmaking based on description logic. Chinese Journal of Computers, 2004,27(5):626−635 (in Chinese with English abstract).

[6] Ma YL, Jin BH, Feng YL. Dynamic discovery for semantic Web services based on evolving distributed ontologies. Chinese Journal of Computers, 2005,28(4):603−615 (in Chinese with English abstract).

© 中国科学院软件研究所 http://www.jos.org.cn

[7] Brogi A, Corfini S, Popescu R. Composition-Oriented service discovery. In: Gschwind T, ed. Proc. of the Software Composition. Edinburgh: Springer-Verlag, 2005. 15−30.

[8] Aversano L, Canfora G, Ciampi A. An algorithm for Web service discovery through their composition. In: Zhang LJ, ed. Proc. of the IEEE Int'l Conf. on Web Services. California: IEEE Computer Society, 2004. 332−341.

[9] Xie XQ, Chen KY, Li JZ. A composition oriented and graph-based service search method. In: Mizoguchi R, ed. Proc. of the 1st Asian Semantic Web Conf. Beijing: Springer-Verlag, 2006. 530−536.

[10] Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne T, Sirin E, Srinivasan N, Sycara K. OWL-S: Semantic markup for Web services. 2004. http://www.w3.org/Submission/OWL-S/

[11] Li YH, Bandar ZA, McLean D. An approach for measuring semantic similarity between words using multiple information sources. IEEE Trans. on Knowledge and Data Engineering, 2003,15(4):871−882.

[12] Resnik P. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. Journal of Artificial Intelligence Research, 1999,11:95−130.

[13] Mihalcea R, Corley C, Strapparava C. Corpus-Based and knowledge-based measures of text semantic similarity. In: Poulin B, ed. Proc. of the American Association for Artificial Intelligence. Boston: AAAI Press, 2006.

[14] Jiang JJ, Conrath DW. Semantic similarity based on corpus statistics and lexical taxonomy. In: Carnie A, ed. Proc. of the Int'l Conf. Research on Computational Linguistics. Taiwan: Scandinavian University Press, 1997. 19−33.

[15] Payne TR. Soton currency converter. 2004. http://www.daml.ecs.soton.ac.uk/services/SotonCurrencyConverter.html

[16] Miller GA. WordNet: A lexical database for English. Communications of the ACM, 1995,38(11):39−41.

[17] Voorhees EM. Using WordNet for text retrieval. In: Fellbaum C, ed. WordNet—An Electronic and Lexical Database. Cambridge: MIT Press, 1998. 285−303.

[18] Huang S, Wang XL, Zhou AY. Efficient Web service composition based on syntactical matching. In: Chu XW, ed. Proc. of the 2005 IEEE Int'l Conf. on e-Technology, e-Commerce and e-Service. Hong Kong: IEEE Computer Society, 2005. 782−783.

[19] Xu B, Li T, Gu ZF, Wu G. SWSDS: Quick Web service discovery and composition in SEWSIP. In: Yu P, ed. Proc. of the 8th IEEE Int'l Conf. on e-Commerce Technology/3rd IEEE Int'l Conf. on Enterprise Computing, e-Commerce and e-Services. Palo Alto: IEEE Computer Society, 2006. 71.

附中文参考文献:

[5] 史忠植,蒋运承,张海俊,董明楷.基于描述逻辑的主体服务匹配.计算机学报,2004,27(5):626−635.

[6] 马应龙,金蓓弘,冯玉琳.基于进化分布式本体的语义 Web 服务动态发现.计算机学报,2005,28(4):603−615.

邝砾(1982－),女,湖南长沙人,博士生,CCF 学生会员,主要研究领域为面向服务的计算,工作流.

邓水光(1979－),男,博士,CCF 学生会员,主要研究领域为流程管理,面向服务的计算,社会计算.

李莹(1973－),男,博士,副教授,主要研究为中间件技术,软件体系架构,编译技术.

吴健(1976－),男,博士,副教授,CCF 高级会员,主要研究领域为 Web 服务,网格计算,数据挖掘.

吴朝晖(1966－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网格计算,嵌入式计算,普适计算.

© 中国科学院软件研究所 http://www.jos.org.cn