基于 IO 与信息内容的语义 Web 服务发现①

马秀军, 陈继东, 李 坤

(中国石油大学 计算机与通信工程学院, 青岛 266580)

摘 要: 随着 web 服务数量大幅增长, 如何快速准确的发现并满足用户需求的服务已经成为一个亟待解决的问题. 现有的基于语义的 web 服务发现通常使用混合的方法, 先在本体层面上进行语义匹配, 当语义匹配失败的时候再采取其他的方法(基于关键字的匹配、基于结构分析)来弥补这个缺陷, 在补救的过程当中由于现有的方法并未准确的反应两个概念之间的相似性, 从而导致 web 服务的发现的准确率不高. 将信息内容语义相似度计算的思想考虑在内, 提出了采用基于服务的 IO(input, output)语义匹配和基于信息内容语义相似计算相结合的方法, 并以 owls-tc2.0 作为测试集合对该方法进行测试, 实验结果表明该方法能有效提高服务发现的准确率.

关键词: 语义 web; web 服务发现; 信息内容; 准确率

Method of Combining IO with Information Content for Discovering Semantic Web Services

MA Xiu-Jun, CHEN Ji-Dong, LI Kun

(School of Computer & Communication Engineering, China University of Petroleum, Qingdao 266580, China)

Abstract: In recent years, there is an exponential growth of the number of web services in the real world. In the web service research area, the most of concerning problem is how to discover the potential web services to meet users' needs fast and accurately. Nowadays, a lot of discovery methods based on semantic are employing the ontologies of concepts among the parameters of web services interface. With content of concept, when the semantic matching is fail, they take other methods such as keyword-based matching and structure-based analysis to make up for the defect. These methods improve the precision and efficiency in the web service discovery in a certain degree. However, all of these methods dismiss the semantic similarity of Information Content. In this paper, we propose a method which employs a combination of methods based on semantic web service discovery technology and the concept of semantic similarity of the IC. It uses owls-tc2.0 as a test set. The experiments show that our method is effective to improve the precision of web service discovery.

Key words: semantic web; discovery of semantic web; information content; precision

1 引言

随着 web 服务数量大幅增长, 如何快速准确的发现并满足用户需求的服务已经成为一个亟待解决的问题. 传统的 web 服务发现技术是通过语法层面的服务描述语言和基于关键字的匹配算法来实现的, 由于标准的 web 服务缺少必要语义描述信息, 无法准确的描述服务功能, 严重影响 web 服务发现的准确率. 而语义 web 服务通过利用语义 web 中的本体对 web 服务进行建模, 借助语义 web 服务的描述语言(SAWSDL[1], WSMO[2] 和 OWL-S[3])在语义层面对 web 服务各个组件进行描述, 实现了 web 服务功能的精确描述, 为提高 web 服务发现的准确率提供了可能.

目前, 基于语义 web 服务发现方法成为研究热点, 单纯基于语义匹配 web 服务发现的方法有: OWLS-UDDI matchmaker[4], RACER[5], MAMA[6]; 使用混合的方法的有: WSMO-MX[7], OWLS-MX[8], OWL-S Discovery[9]. 实践表明单纯使用基于语义匹配的 web 服务发现效果比较差, 原因是如果请求服务和

① 收稿时间:2015-05-20;收到修改稿时间:2015-06-15

web 服务在现实世界里是语义相关的, 但在本体概念树中的关系不是祖先与孩子节点之间的关系, 就会匹配失败; 而使用混合的方法是当语义匹配失败的时候则采取其他的方法来弥补. 在 RACER[5]中作者使用 DAML-S 作为 web 服务的描述语言, 只进行基于语义的服务匹配, 主要是利用本体概念之间的继承关系来实现 Web 服务 IO(input, output)的匹配, 这种方式与使用混合的语义匹配方法相比, 服务发现效果要差一些. 在 WSMO-MX[7]中作者使用自然语言描述文本句法分析抽取约束的思想, 并将本体定义的可能与概念关联的属性作为约束关系来抽取约束对象, 抽取的约束分为3类, 把web服务表示为三元组的形式. 首先利用概念匹配计算两个主体之间的匹配程度; 其次利用概念匹配计算两个元组的约束类型的语义匹配, 当两个元组的约束类型与主体的匹配程度均大于某阈值, 则利用概念匹配计算两个客体之间的匹配程度; 最后如果三元组的三个部分都能匹配成功, 则通过加权求和得到他们的匹配值. owls-mx[8]中使用基于逻辑的语义匹配和 IR(Information retrieval)相结合的方法, 在进行 web 服务发现的时候先进行 IO(input, output)语义匹配, 当 IO 语义匹配失败时再使用 IR 方法完善语义描述实现服务功能匹配, 其中, IR 使用了 cosine、extended Jacquard 和 Jensen--Shannon information divergence 等相似度度量. 在文献[9]中作者使用功能(IO)语义匹配和结构分析相结合的方式进行服务发现. 首先进行功能上的匹配, 作者定义了 4 种语义匹配级别, 功能匹配失败的时候再触发基于同义词字典的结构分析的算法. 在本文中, 我们提出了基于服务的IO(input, output)语义匹配和基于信息内容语义相似计算相结合的方法; 该方法在明确 web 服务的语义描述的基础上, 先进行 web 服务的 IO 语义匹配, 一旦语义匹配失败, 再进行基于 IC 的概念语义相似度算法发现所需服务, 实验结果表明所提方法能有效提高服务发现的准确率.

2　基于服务的IO语义匹配和概念信息内容相似度计算的方法

2.1 服务概念语义匹配

两个概念之间的语义相似度取决于其在本体分类层次中的逻辑包含关系. 首先对服务本体文件(owl\owls)进行解析得到服务 IO 的概念, 然后进行语义级别的匹配, 分别得到 Exact、Plug-in、Subsumes、Fail 这四个级别的语义匹配度[13]. 简单本体概念树示意图如图 1 所示:

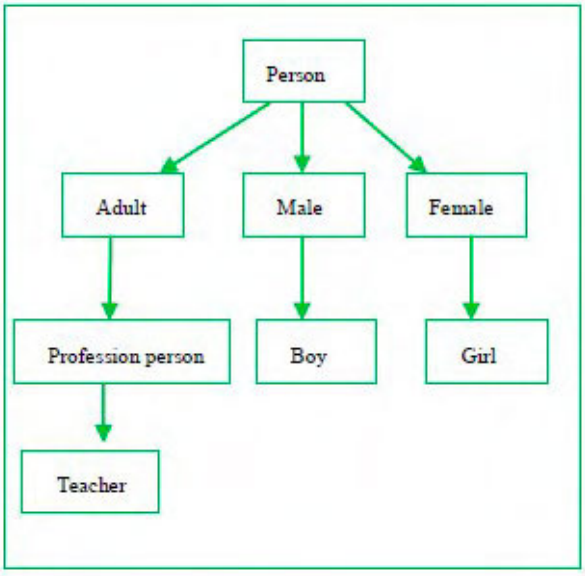

图 1　本体概念示例

Exact match : 服务 service 和请求服务 request 是 Exact 匹配即在本体中定义的概念, 服务 service 的 I 属性和请求服务 request 的 I 属性相同并且服务 service 的 O 属性与请求服务 request 的 O 属性也相同. 假设服务 service 的 IO 在图 1 中分别为 adult 和 male, 那么请求服务 request 的 IO 分别是 adult 和 male.

Plug-in match: 服务 service 和请求服务 request 是 Plug-in 匹配即在本体中定义的概念, 服务 service 的 I 属性和请求服务 request 的 I 属性相同或者是其直接孩子节点并且服务 service 的 O 属性是请求服务 request 的 O 属性的孩子节点. 假设服务 service 的 IO 属性在图 1 中分别对应为 adult 和 male, 那么请求服务 request 的 IO 属性分别是 adult 和 person.

Subsumes match: 服务 service 和请求服务 request 是 Subsumes 匹配即在本体中定义的概念, 服务 service 的 I 属性是请求服务 request 的 I 属性的祖先节点并且服务 service 的 O 属性是请求服务 request 的 O 属性的祖先节点. 假设服务 service 的 IO 属性在图 1 中分别对应为 adult 和 male, 那么对应的请求服务 request 的 IO 属性分别为 profession person 和 boy.

Fail match: 服务 service 和请求服务 request 进行 IO 语义匹配的时候上面三种情况都不满足就认为是服务的 IO 语义匹配失败.

根据自己的需求定义以上四种语义匹配级别, 在语义匹配过程中我们先进行 Exact match, 然后进行 Plug-in match, 最后进行 Subsumes match, 最终返回基于服务的 IO 语义匹配结果集. Exact match 在是语义匹配中最好的也是服务请求者最想要的匹配结果. Plug-in match 返回的结果虽然不是服务请求者最想要的结果, 但是可以帮助请求者查找出一些潜在的很有可能就是请求者想要的服务. Subsumes match 很有可能返回一些可以替代服务请求者最想要得到的服务. Fail match 返回的结果是服务请求者所不能接受的, 我们就认为 web 服务的语义匹配失败.

在现实世界中两个概念是同义词或者在语义上非常的接近, 但是在本体的定义中却不是祖先-孩子之间的关系. 在这种情况下, 使用基于语义匹配的方法就会失败. 本文允许出现语义匹配失败, 当语义匹配失败之后通过基于 IC 的概念语义相似度的计算来弥补.

2.2 基于 IC 的概念语义相似度计算

语义相似度是用来度量文档和术语的语义内容或涵义之间的相似度, 它在信息检索与抽取, 自然语言理解等领域中有着广泛的应用, 本文用来计算两个概念之间的相似度. 本文将同时考虑概念的信息内容(IC)以及两个概念在本体概念树中的距离信息, 使得该算法计算的相似度更高. 概念的 IC 值指的是概念所能提供的信息量, IC 表示为-lgp(c), 其含义是一个概念 c 的出现频率越大, 则该概念提供的自信息量就越小.

本文采用[12]中的IC计算模型, 该模型不仅考虑节点的子节点数, 同时还考虑了每个概念所处的深度, 目的是使每个概念都有一个准确的IC值. 由于本文的实验采用 owls-tc2.0 作为测试集合, 所以本文中的 IC 值是计算 owls-tc2.0 本体中定义概念的 IC 值. 如公式(1)所示:

$$IC(w) = k(1 - \frac{\lg(hypc(w)+1)}{\lg(node_{max})}) + (1-k)(\frac{\lg(deep(w))}{\lg(deep_{max})}) \quad (1)$$

其中 hypc(w)表示返回概念 w 的子概念, deep(w)表示概念 w 的所在概念树的深度, $node_{max}$ 表示分类树中概念的总数, $deep_{max}$ 表示分类树的最大深度, k 是一个可以调整的参数, 可以根据用户需要进行调整, 用于改变式(1)中两项在IC求解中的所占比例, 也就是改变节点的子节点数和深度两者在模型中所占比例. 本文中 k 取值为 0.5.

在计算相似度的时候将结合基于信息内容的语义相似度算法以及 Jiang 和 Conrath[12]的概念语义相似度算法, 即把信息内容及两个概念所处于分类树中的位置都考虑在内, 使用[18]中的计算方式. 如公式(2)所示:

$$sim(w1,w2) = 1 - k \times (\frac{\lg(len(w1,w2)+1)}{\lg(2 \times \max_{w \subset onto}(depth(w)))}) - (1-k) \times ((IC(w1)+IC(w2) - 2 \times IC(lso(w1,w2)))/2) \quad (2)$$

len(w1,w2)是概念 w1 和 w2 在概念树中的距离, 其中 k 是一个权值, 可以人为进行调节, 以使算法达到最佳性能, 本文中 k 的值为 0.5, lso(w1,w2)表示概念 w1 和 w2 的最小共同父节点. 该模型是将两个概念的路径因素也考虑到模型当中, 因为如果两个概念的共同父节点具有相同的值, 并且两个概念的 IC 值相同, 或者两个概念的 IC 值的和相同, 然而两个概念间的路径距离并不相等, 在这种情况下, 利用 Resnik[11] 、Jiang 和 Conrath[12]语义相似度算法则不能区别开来, 而本文所给出的计算模型则很容易的区别开来, 从而使得概念的语义相似度的计算更为准确.

在模型当中, 采用了 $\frac{\lg(len(w1,w2)+1)}{\lg(2 \times \max_{w \subset onto}(depth(w)))}$ 作为模型中的一项, 该项的选择是考虑了信息理论中的自信息的概念, 该项的物理意义是指两个概念 wl 和 w2 间的路径距离的自信息, 即概念对路径所带有的信息量, 其值域范围为[0, l]. 对于(2)模型, 其值域范围也为[0, l].

在实验过程中我们会遇见以下几种情况:

1)概念 w1 和 w2 如果是同一个概念, 那么用公式(1)计算出的 IC(w1)和 IC(w2)的值就相同, 再用公式(2)计算由于 len(w1,w2)值为零, 所以 sim(w1,w2)=1.0. 因此我们进行基于 IC 语义计算的时候先判断概念 w1 和 w2 是否为同一概念, 如果是就直接将 sim(w1,w2)的值设为 1.0, 而不进行基于 IC 语义计算.

2)概念 w1 和 w2 如果是不同领域的两个概念, 那么 w1 和 w2 相似度为 0. 此时我们也不进行基于 IC 语义计算而是直接将 sim(w1,w2)设为 0.

3)如果概念 w1 和 w2 不是同一个概念, 并且是同一个领域中的, 那么我们就用公式(1)和公式(2)计算 w1 和 w2 的语义相似度.

2.3 本文算法

算法先进行 web 服务的 IO 语义匹配, 当匹配失败之后再进行基于 IC 的概念语义相似度计算. 具体算法描述如图 2 和图 3, algorithmIO 是用来进行语义匹配的,

对于每一个 request 和 service 先进行 I 属性上的语义匹配再进行 O 属性上的语义匹配, 当语义匹配失败时使用 algorithmIC 进行基于 IC 概念语义相似度计算, 最后再将结果集显示在结果面板上. 在 algorithmIO 中第 8、9 行是进行 Exact 匹配, 10、11 行进行 Plug-in 匹配, 12、13 行进行 Subsumes 匹配, 如果请求服务 request 和服务 service 都不是以上三种情况就是 Fail Match. 在 algorithmIC 中, 第 4、5 行是判断 request 和 service 的 IO 概念是否是同一概念, 第 6、7 行是判断 request 和 service 的 IO 概念是否是同一个领域中的概念, 第 8 行调用 calculateSim 进行概念相似度的匹配, 第 15-18 行分别计算 w1 和 w2 的子概念数、在概念树中所处的深度, 第 19 行计算概念树中节点的总数, 第 20 行计算 w1 和 w2 之间距离, 第 21 行计算概念树的最大深度, 第 22 行计算两个概念之间的相似度.

```
MainAlgorithm:
1.  Service request;
2.  ArrayList  CollectionServces;
3.  ArrayList  IOCollections=
    algorithmIO(request,CollectionServices);
4.  ArrayList  ICCollections=
    algorithmIC(request,CollectionServices);
5.  将 IOcollections 和 ICCollections 显示在结
    果面板上。

    algorithmIO(request ,CollectionServices):
1.  service r = request;
2.  ArrayList c = CollectionServices;
3.  for(i=0;i<c.length;i++)          do
4.      matchIO(r,c.get(i))
5.  end for
6.  end algorithmIO
7.  matchIO(r,s)
8.  if(exactMatch(r,s))              then
9.      return exact;
10. else if(plug-inMatch(r,s))       then
11.     return plug-in;
12. else if(subsumes(r,s))           then
13.     return subsumes;
14. end if
15. end matchIO;
```

图 2　MainAlgorithm 和 algorithmIO

```
algorithmIC((request ,CollectionServices):
1.  Service r = request;
2.  ArrayList c = CollectionServices;
3.  for(i=0;i<c.length;i++)          do
4.  if(identifyConcept(r,c.get(i)))  then
5.      sim(r,c.get(i)) = 1.0
6.  else if(hasSameField(r,c.get(i)))  then
7.      sim(r,c.get(i)) = 0.0
8.  else sim=calculateSim(r,c.get(i))  then
9.  end if
10. end for
11. end algorithmIC
12. calculateSim(r,s)
13. w1 = r.getConcepts();
14. w2 = s.getConcetps();
15. hypcr = calculatehypc(w1);
16. hypcs = calculatehypc(w2);
17. deepr = calculateDeepr(w1);
18. deeps = calculateDeeps(w2);
19. maxnode = calculateMaxNode();
20. len = calculateLen(w1,w2);
21. maxDept=calculateMaxdept();
22. sim=
    countSim(r,c,hypcr,hypcs,deepr,deeps,maxno
    de,len,maxdept);
23. return sim;
24. end calculateSim
```

图 3　algorithmIC

3　实验结果及分析

我们的实验是使用 java1.6 实现的, 使用 Maryland 大学开发的 pallet 对本体进行解析和推理 (http://www.mindswap.org/2004/owl-s), 使用 xampp 作为本地服务器进行测试. 我们使用 OWLS-TC2.0 作为测试集合, 这个测试集合有 570 多个服务, 分别来自不同的 7 个领域(education, medical care,food, travel, communication, economy, weaponry). OWLS-TC2.0 中的服务是使用 OWL-S 描述服务的. 在实验中共有 20 个服务请求被评估, 对每种算法独立的运行检索以判定其效果和性能. 评估实验的查询性能使用准确度和召回率[16], 召回率是衡量某一检索系统从文献集合中

检出相关文献成功度的一项指标, 即检出的相关文献与全部相关文献的百分比. 准确率是检索出的相关文档数与检索出的文档总数的比率. 检索出的相关文档数为 RelevDoc, 检索出的文档数记为 RetriDoc, 准确度率和召回率定义如下:

$$\text{Precision} = (\text{RelevDoc} \cap \text{RetriDoc}) \big/ \text{RetriDoc}$$

$$\text{Recall} = (\text{RelevDoc} \cap \text{RetriDoc}) \big/ \text{RelevDoc}$$

Algorithm proposed 是本文提出方法, owls-discover 是文献[9]中提出的方法, owls-mx 是文献[3]中提出的方法. 实验结果如图 4 所示. Owls-discovery 是使用 IO 语义匹配和基于字典的结构分析相结合的方法, owls-discovery 得到的效果比 owls-mx 要好, 但是本文提出的方法比 owls-discovery 的效果要更高.

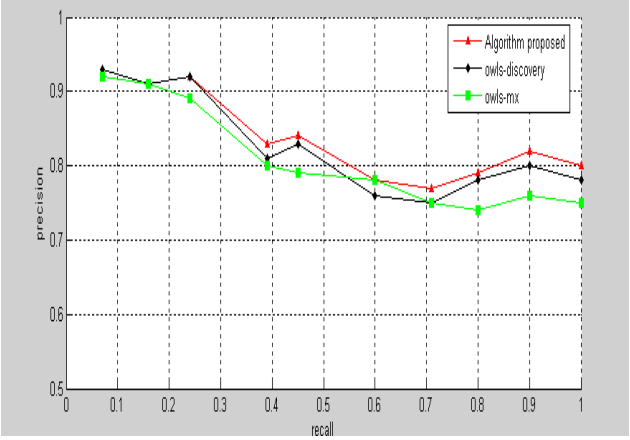

图 4 实验结果

4 结语

本文提出了一种基于 IO 语义匹配与信息内容相结合的 web 服务发现的方法. 首先进行 IO 语义匹配, 当语义匹配失败的时候进行基于 IC 的概念语义相似度计算. 本文首先描述了服务概念语义匹配, 然后描述了基于 IC 的概念语义相似度计算,给出了具体的基于 IO 语义匹配与信息内容相结合的 web 服务发现, 并通过实验验证了本文所提方法. 在接下来的工作中我们将改善所提算法的发现性能, 并且完善我们的 web 服务发现工具, 为进一步研究提供支撑.

参考文献

1 Kopecky J, Vitvar T, Bournez C, Farrell J. Sawsdl: Semantic annotations for wsdl and xml schema. IEEE, Internet Computing, 2007: 60–67.

2 Roman D, Keller U, Lausen H, de Bruijn J, Lara R, Stollberg M, Polleres A, Feier C, Bussler C, Fensel D. Wsmo-web service modeling ontology. DERI Working Draft, 2005: 77–106.

3 Martin D, Burstein M, Hobbs J, Lassila O, McDermott D, McIlraith S, Narayanan S, Paolucci M, Parsia B, Payne TR. OWL-S: semantic markup for web services. IEEE, Internet Computing, 2004: 72–81.

4 Sycara K, Paolucci M, Anolekar A, Srinivasan N. Automated discovery, interaction and composition of semantic web services. Web Semantics Elsevier. 2003.

5 Amorim R, Claro D, Lopes D, Albers P, Andrade A. Improving web service discovery by a functional and structural approach. 2011 IEEE International Conference on Web Services. 2011.

6 Klusch M, Kapahnke P. OWLS-MX3: an adaptive hybrid semantic service matchmaker for OWL-S. CEUR Proc. of 3rd International Workshop on Semantic Matchmaking and Resource Retrieval (SMR2). Washington, USA. 2009.

7 Wei D, Wang T, Wang J, Chen Y. Extracting Semantic Constraint from Description Text for Semantic Web Service Discovery. Springer, 2008: 146–161.

8 周宁,谢俊元.基于定性多用户偏好的 Web 服务选择. 电子学报,2011,39(4):729–736.

9 邱田,胡晓惠,李鹏飞,马恒太.基于 OWL-S 的服务发现语义匹配机制.电子学报,2010,38(1):42–47.