一种基于本体的语义 Web 服务发现模型①

曹渝昆　丁明伟 (重庆大学 计算机学院 重庆 400030)

摘　要： Web 服务已经成为互联网中最为重要的一种计算资源和软件资产，Web 服务的大量涌现对服务发现提出了挑战。Web 服务发现的关键是 Web 服务的语义描述的准确性和 Web 服务检索引擎的检索效率。提出了一种基于本体的 Web 服务描述方法，采用 OWL-S 对 Web 服务进行语义描述，并提出了针对性的 Web 服务检索引擎。通过试验，该模型结合语义 Web 服务技术实现 Web 服务的动态查找与组合，可提高 Web 服务发现的查全率和查准率。

关键词： Web 服务；本体；OWL；OWL-S；UDDI

Discovering Model of Semantic Web Service Based on Ontology

CAO Yu-Kun, DING Ming-Wei

(Deptartment of Computer Architecture, Institute of Computer, Chongqing Universiy, Chongqing 400030, China)

Abstract: Web services have already become one of the most important computing resources and software assets in the Internet. With the increasing number of Web services, how to find the required services efficiently or how to gain the best service from the vast service sets become important. The semantic description of Web service and the efficiency of OWL-S retrieval model of semantic Web service are the key to the discovery of the Web service. This article proposes the OWL-S framework of semantic Web service, and adopts OWL-S to describe the Web service, and raises pointed matching algorithm. Through experiments, in this model, together with the semantic Web service technology, the dynamical searching and composition of Web service are implemented, so that the precision ratio and the recall ratio of Web service can be improved.

Keywords: Web service; ontology; OWL; OWL-S; UDDI

1　引言

Web 服务是一种基于网络环境的自适应的、自描述、模块化的应用程序，因其具备良好的互操作能力和可重用性而在电子商务、应用集成、流程管理等领域中扮演越来越重要的角色。近年来，随着 Web 服务相关标准的持续完善和支持 Web 服务开发的软件平台的不断成熟，Web 服务已经成为互联网中最为重要的一种计算资源和软件资产。然而，随着 Web 服务数量的不断增长，用户在众多的开放性的 Web 服务中如何发现适用的服务成为一个亟待解决的问题。Web 服务缺乏语义描述是制约 Web 服务准确发现的瓶颈问题。Web 服务标准 UDDI(Universal Description，Discovery， and Integration，统一描述、发现和集成)提供一种基于分布式商业注册中心机制来进行服务管理、注册和发现。基于 UDDI 的服务发现机制的不足在于服务发现时的匹配过程是基于框架的关键字匹配，无法提供服务功能性描述，即：服务的语义信息，也没有内在对服务自动发现和组合的支持，这显然无法满足用户对服务发现智能化的要求。因此，对 Web 服务进行语义化描述是解决问题的关键。针对此问题，本文通过对语义 Web 服务的 OWL-S (Ontology Web Language for Services，web 服务的本体语言)的研究，将语义 Web 技术应用到 Web 服务的发现过程中，提出了一种基于本体的 Web 服务描述方法，通过本体进行语义标注生成 Web 服务请求的 OWL-S 文档，并提出了相匹配的服务检索框架，继而

① 基金项目:博士后科学基金(20080430740,20080430744);重庆市自然科学基金(CSTC)(2008BB2194)
收稿时间:2009-08-11;收到修改稿时间:2009-09-07

在 OWL-S 语义扩展后的 UDDI 中实施语义匹配。

2　OWL-S技术

OWL-S(Web Ontology Language for Services)是用 OWL 语言描述的 Web Service 的本体，是一种具有显式语义的无歧义的机器可理解的标记语言(markup language)，可描述 Web Service 的属性和功能。

图 1 是 Service 的上层本体。在 OWL-S 中，一个 Service 由三部分来描述 Service Profile，Service Model，Service Grounding。简单来说，Service Profile 描述服务是做什么的，Service Model 描述服务是怎么做的，Service Grounding 描述怎么访问服务。一个 Service 最多被一个 Service Mode 描述，一个Service Grounding必须和一个Service相关联。Service Profile 描述一个服务，主要包含服务提供者的白页信息、黄页信息和服务的功能信息，可以提供服务所属的分类、服务的 QoS(Quality of Service，服务质量)信息，还提供了一种机制来描述各种服务的特性。Service Model 主要是服务提供者用来描述服务的内部流程。Service Grounding 可看作是与服务交互需要的服务描述元素规格从抽象到具体的映射，适合于服务自动选择 Agent 并设置与服务的通信，进而调用服务。

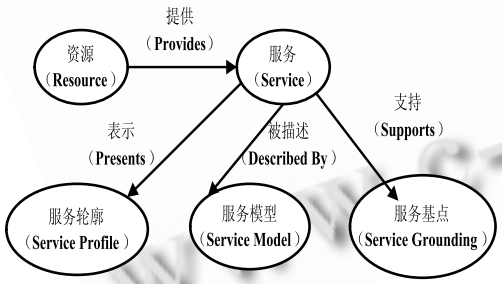

图 1　OWL-S 服务本体

3　基于OWL-S/UDDI的Web服务的发现框架

3.1 服务发现框架

应用代码驱动的语义 Web 服务的 OWL-S 框架主要以 Java 代码为起点，由代码产生 OWL-S 描述本体，这种方式适合 Web 服务的未来发展趋势。在这种情况下，开发者只需要编写代码来描述功能完善的 Web 服务，然后转化为语义描述。最重要的是，转化过程可通过相应的工具自动完成，使得开发难度相对降低和实施过程比较容易，并减少了产生错误的可能。例如，通过 Axis 和.net Web 服务框架可自动从 Web 服务的源代码生成 WSDL 描述文档；此外，服务的 OWL-S 描述也可以用 WSDL2OWL-S 工具自动生成。同样，Web 服务的 OWL-S 文档中的 Service Profile 可以通过 OWL-S2UDDI 工具自动生成与 UDDI 的发布广告信息进行映射，语义 Web 服务的 OWL-S 框架如图 2 所示。

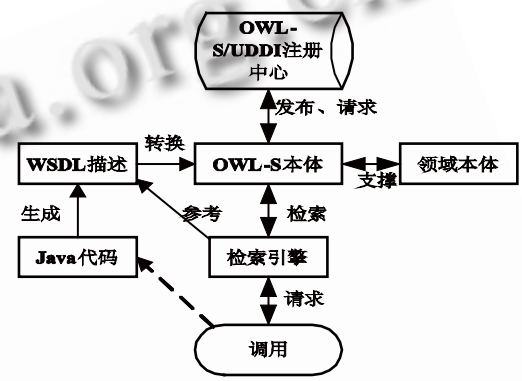

图 2　语义 Web 服务 OWL-S 框架

如图 2 所示，在语义 Web 服务 OWL-S 框架中，Java 代码开发的 Web 服务，其描述用 WSDL 文档，在相应的领域本体支撑条件下转换成 OWL-S 本体，共同对 Web 服务进行描述，并将转换后的 OWL-S 信息添加进 OWL-S/UDDI 注册中心。当服务请求者发送请求时，需经过检索引擎对查询请求进行处理，进而在注册中心进行匹配检索，匹配过程也可参考原有的 WSDL 服务描述，匹配结果经检索引擎筛选后反馈给服务请求者进行服务调用[1,2]。

3.2 服务检索引擎

用 OWL-S 代码驱动方式来进行语义 Web 服务的开发。服务描述广告和服务发布广告都用 OWL-S 本体描述，语义 Web 服务的发现机制 OWL-S/UDDI 则以 OWL-S 为搜索条件。采用基于服务提供者、服务注册中心和服务请求者三者间交互的 SOA(Service Oriented Architecture)体系架构和语义 Web 服务技术，涉及服务的发布、查找、组合和绑定操作。OWL-S 是连接语义 Web 和 Web 服务两大技术的桥梁。语义 Web 服务的 OWL-S 检索引擎主要围绕 OWL-S 展开。具体检索引擎如图 3 所示[1,2]。

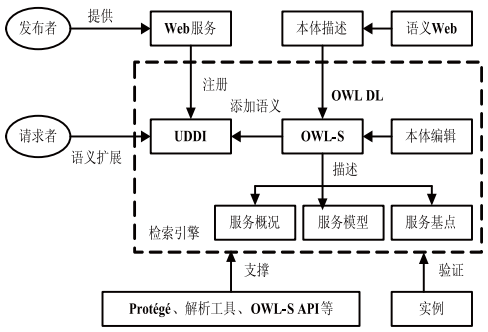

图 3　语义 Web 服务的 OWL-S 检索引擎

Web 服务注册于 UDDI 注册中心。语义 Web 服务用 OWL-S 本体描述 Web 资源并为其添加语义信息，形成 OWL-S 文档。语义 Web 服务的 OWL-S 检索引擎用 OWL-S 来描述 Web 服务的概况(Service Profile)、模型(Service Model)和基点(Service Grounding)，对 UDDI 添加语义形成 OWL-S/UDDI 的服务发现和服务匹配机制，并对检索请求进行语义标注处理，进而采用相应的匹配算法与 OWL-S/UDDI 中的服务的 OWL-S 描述进行语义匹配，达到语义检索的目的。服务的 WSDL 描述经 OWL-S API 转化成 OWL-S 文档，由于转化过程中会丢失数据、工作流等信息，使得描述不准确，所以要对 OWL-S 本体进行编辑。

OWL-S/UDDI 检索的依据是服务检索请求语义扩展后的 OWL-S 文档，它通过检索接口用 OWL-S VM 对服务进行匹配检索。OWL-S 解析器负责从 UDDI 中下载服务本体和 OWL-S 服务描述信息，并转换成推理引擎能够使用的推理断言(reasoning assertion)。基于 OWL-S/UDDI 的调用过程是 Web 服务在 Soap 协议语义 Web 服务的 OWL-S 描述及其应用上进行的 Web 服务交互，OWL-S VM 将 OWL-S 描述的 ServiceGrounding 信息映射成为 WSDL 描述，交给传统的 Web 服务调用框架(WSIF)调用。

4　检索引擎匹配算法设计

在本文提出的检索引擎中，采用了一个比较相似度(匹配度)的匹配方法，通过对服务请求和服务发布的三个主要特征进行比较来选择相近服务。相似度比较完成以后，按照三者匹配度的加权平均值来确定理想的服务，返回给服务请求者。这三个特征分别是文本描述、IO 描述和服务参数。在进行匹配之前，服务请求者提供一个服务请求的 Profile 来描述其理想的服务。然后按照此算法与注册中心所发布服务的服务描述进行比较，找到合适的服务并返回给服务请求者。具体算法如下：

在进行服务匹配前，先进行预处理，过滤与服务不同类别的服务发布.在服务请求的 serviceCategory 与服务发布的 serviceCategory 进行比对的时候，如果二者引用相同的分类法，那么可以直接比较 value，相等或者前者小于后者时说明二者有相同的分类。而当服务请求与服务发布采用的是不同分类方法时，需要在这两种分类之间建立具体类别间等价映射，通过等价映射可以将它们的 servieeCategory 和 Value 统一，这样二者的比对就转换为在同一种分类法中的比较。预处理完成以后，通过类别过滤的服务可以进行后续的服务匹配。

4.1　文本匹配

文本匹配的一种简单的方法是向量空间模型。向量空间模型把所要表示的对象抽象成向量的形式，对象可以是整个文本文件或者是其中的段落或者句群，也可以是一个多媒体文件。文本的内容通常用它所包含的语言特征来表示，也就是项(Term)。用于表示语言特征的单位可以是字、词、短语和句子。在通常的文本表示模型中通常是使用关键词作为特征项。文本 D 可以用项集表示为 D ={t1，t2，…，tn}，其中 n 是用于表示文本的特征项的个数。

对于 n 个项的文本 D ={t1 ，t2，…，tn}，项 tk (1<=k<=n)通常被赋予一定的权重 wk(Term-Weight)，用于表示该特征项对于文本的重要程度，即文本 D 表示为(t1，w1，t2，w2，…，tn，wn)，或者简记为{w1，w2，…wn}，若将 t1，t2，tn，视为坐标系，则 w1，w2，…，wn 就是相对应的坐标值，D{w1，w2，…，wn}就可以被视为为 n 维空间中的一个向量。我们称 D{w1，w2，…，wn}为文档 D 的向量表示，其权重的计算通过下面的公式(1)实现[3]：

$$w_{ij} = tf_{ij} * idf_j$$
$$idf_j = \log_2(N / df_j) \quad (1)$$

其中，w_{ij} 代表文本 i 中 j 的权重；tf_{ij} 代表项 j 在文档 i 中出现的次数；df_j 代表含项 j 的文档数。idf_j 是文档总数除以含项 j 的文档数。$\log_2$ 用以抑制 idf_j 的作用。

由此可以用两个向量分别表示服务请求与服务发布的描述文本。两个文本(R 代表服务请求的文本，Ai 代表服务发布 i 的文本)之间的内容相关程度(Degree of Relevance)，常常用它们之间的相似度 SD(R，Aj)来度量。当文本被表示为向量空间模型时，可以借助向量之间的某种距离来表示文本之间的相似程度，通常用两个向量之间的夹角余弦来表示[3]：

$$SD(A,R)=\frac{\sum_{j=1}^{i}W_j * d_{ij}}{\sqrt{\sum_{j=1}^{i}(d_{ij})^2}\sum_{j=1}^{i}(W_j)^2} \quad (2)$$

其中，$SD(A,R)$代表一个服务请求的文本 R 和文本 A 之间的文本相似度；W_j 是项 j 在文本 R 中的权重；d_{ij} 项 j 在文本 A 中的权重。

4.2 IO 匹配

IO 匹配是整个匹配策略中最重要的部分。对于 IO 匹配，假设服务请求的输出集(用 RO 表示)属于服务发布的输出集(用 AO 表示)时，即服务发布能提供服务请求指定的所有输出，输出匹配成功；当服务发布的输入集(用 AI 表示)属于服务请求的输入集(用 RI 表示)时，即服务清求能提供服务发布所需的所有输入，输入匹配成功。显然上述的匹配可认为是集合的包含问题。集合 A 包含集合 B 意味着集合 B 中的任意一个元素都被包含在集合 A 中，即对于 B 中的每一个元素 b，A 中总是存在一个相应的元素 a，使得 a=b。由于服务的输入、输出参数都是通过 OWL 定义的领域本体中的类进行描述的，“ a=b ”的意义可理解为 a 关联的类与 b 关联的类相同。但在基于功能的语义相似匹配中，要求 a，b 属于同一个 OWL 类这样的定义太强，更恰当地应理解为 a 关联的类 X 与 b 关联的类 Y 之间的语义相似程度，以此来判断 a 与 b 的匹配程度。而服务功能的语义相似度，可以通过计算本体类集合的相似度来获得.因此语义相似度的定义是关键。归纳上述分析，下面是计算输入集/输出集的匹配度流程(以输入集为例，服务请求的输入集 RI，服务发布的输入集 AI)：

步骤 1：引入语义距离的概念，计算类 A(RI 中元素 a 关联的类)与类 B(AI 中元素 b 关联的类)的语义距离 DS；

步骤 2：由语义距离得到 a 与 b 的匹配度；

步骤 3：由步骤 1、步骤 2 得到元素两两匹配的匹配度，计算 RI 与 AI 的匹配度。

下面，详细说明 3 个步骤中所使用的算法：

① 语义距离

类之间语义距离的提出，是为了计算类的匹配度。距离是匹配度的反函数。两个类之间的距离越大，那么她们的相似度越低，即匹配度越低。语义距离是指在一个本体中的两个不同类间存在的继承关系或者二元关系链中最短的关系链长度的度量。本文计算 A 与类 B 的语义距离 DS(A，B)公式如公式(3)所示[3]：

$$\begin{aligned}&DS(A,B)=\begin{cases}\infty & A和B不在同一本体\\ DC(A,B) & A和B在同一本体\end{cases}\\ &DC(A,B)=\begin{cases}1 & A和B为继承关系\\ a & A和B\ 为二元关系，a为二元关系权值\end{cases}\end{aligned} \quad (3)$$

② 相似度函数

概念间相似度函数是将语义距离转化成相似度，从而将语义距离应用于语义匹配[3]。

$$Sim(A,B)=\frac{1}{e^{DS(A,B)}} \quad (4)$$

其中，DS=0 时，Sim=1；Sim 是 DS 的减函数；Sim 值域为[0，1]。

③ 相似度计算

在本文中，给概念间的上下位关系赋予权重 1，而对概念间其它关系赋予权重 α(α>1)(因为相似度与距离成反比关系，这里的权重指的是计算距的权重，所以上下位关系的权重小于其它二元关系)。这样，要计算两个 Web 服务输入矢量元素所对应概念间的距离，就转化为一个带权重的有向图(因为考虑了各种各样的二元系，所以是图)中寻找最短路径的问题，对此可以运用图论中最短路径的算法(BFS 或 dijkstra)来计算两个概念间的距离。计算两个 Web 服务输入矢量元素所对应概念间的距的算法如下[3]：

Step1 输入 I 对应的标志符 class1 以及 I′ 对应的标志符 class2，try-list=Φ，abandoned-list=Φ；

Step2 把 class1 的权重初始化为 0，并加入到 try-list 中；

Step3 如果 try-list 不为空，则进行如下操作：取走 try-list 中具有最小权重的元素 u：

Step3.1 如果 Synset(u)包含 class2，则返回 u 具有的权重，算法结束；

Step3.2 利用推理机从本体库中得到与 u 具有

直接上下位关系的概念集合 H(u)；

Step3.3　如果 H(u)中包含 class2，则返回(u 的权重+1)，算法结束；

Step3.4　对 H(u)∩try-list 中的每一个元素 v，如果(u 的权重+1)<v 的权重，则把 v 的权重修改为(u 的权重+1)；对 H(u)、try-list、abandoned-list 中的每个元素 w，设定 w 的权重为(u 的权重+1)，并把它加入到 try-list 中；

Step3.5　利用推理机得到以 u 为定义域的所有直接二元关系的值域集合 B(u)；

Step3.6　如果 B(u)中包含 class2，则返回(u 的权重+α)，算法结束；

Step3.7　对 B(u)∩try-list 中的每一个元素 v，如果(u 的权重+α)<v 的权重，则把 v 的权重修改为(u 的权重+α)；对 B(u)、try-list、abandoned-list 中的每个元素 w，设定 w 的权重为(u 的权重+α)，并把它加入到 try-list 中；

Step3.8　把 u 加入到 abandoned-list 中。

在这个算法中，Synset(u)表示利用推理机获得概念 u 所有的同义概念；try-list 为一表，其中元素表示本体库中尚未遍历的概念；abandoned-list 也为一列表，其中元素表本体库中已经遍历过的概念。Step3.3 表示赋予上下位关系的权重为 1；Step3.6 表示赋予其它二元关系的权重为 α。经上述两个算法就得到两个 Web 服务关于输入和输出参数距离，进而得到了这两个 Web 服务关于输入和输出参数的相似度。

有了输入、输出集合之间的匹配相似度，就可以利用加权平均或几何平均在服务功能语义相似匹配的过程中计出服务请求与服务发布之间的 IO 匹配相似度了。一般把两个 Web 服务 W =<I ，O>和 W '=<I '，O'>相似度定义为[3]：

$$Match_{IO}(T,C) = a * S(O,O') + S(I,I') \qquad (5)$$

其中 α 是一个大于 0 的常量，一般建议把其设为大于 0.5，这样就可以加大输出参数的相似性在整个 Web 服务相似性计算中的权重。

4.3 参数匹配

参数匹配是使用 Profile 提供的信息，进行服务请求与服务发布的终端属性的匹配。服务提供者可以用标签<Profile:servlceParareter>来规定终端信息，例如，服务能够提供的地理区域，或者服务质量水平。参数匹配是为了获得最佳服务而在匹配过程中添加的附加条件。由于服务参数的值是一个涉及本体的类的 URI，语义匹配策略可以识别服务请求和服务发布之间两个成对比较的服务参数间的匹配度。定义完全匹配的匹配度是 1，插拔匹配的匹配度是 0.6，包含匹配的匹配度是 0.4，匹配失败的匹配度是 0。这里使用简单的线性规划可以得到所有两两参数匹配的最大匹配度，最后由其平均值决定参数匹配的匹配度。

三级匹配完成之后，得到了一个服务请求和一个服务发布的文本匹配度、功能匹配度和参数匹配度，采用 0-1 之间的数值表示服务请求 T 和服务广告 C 之间的相似程度，则 T 与 C 的服务匹配相似度如下公式(6)所示[5]：

$$Match(T,C) = u1 * Match_{Text}(T,C) + u2 * Match_{IO}(T,C) + u1 * Match_{Parameter}(T,C) \qquad (6)$$

公式(6)中，$Match_{Text}(T,C)$ 为文本匹配度，$Match_{IO}(T,C)$ 为 IO 匹配度，$Match_{Parameter}(T,C)$ 为参数匹配度。μ1，μ2，μ3 指用户对匹配的结果的侧重点，μ1+ μ2 +μ3 =1，0 <μ1，μ2，μ3 <1。

服务请求者首先要提供一个 OWL-S 文档，对所需服务进行描述。然后进行预处理，过滤掉不同类别的服务以后，首先进行文本匹配。这一步骤将会产生文本匹配度，然后进行功能匹配的比较产生 IO 匹配度，最后进行参数匹配，产生参数匹配度。由于终端信息是服务提供者的可选信息，服务提供者能够选择提供终端信息或者选择不提供，所以参数匹配有时候是不会进行的。最终的匹配度由这三个匹配度决定，可以作为识别发布的服务是否与请求之服务相关，将加权计算结果返回给请求者。

5　试验

本文试验在 Windows XP 环境下开发。采用 Sun 公司的 NetBeans5.0.2 作为开发工具。Protégé 3.3.1 作为领域本体建模工具。Protégé 是斯坦福大学医学院医学信息研究组开发的一个免费、开源的本体工具，它为知识工作者提供了一个可以构建领域本体的环境。

假设存在一个提供旅游业务的网站，该网站提供的有两个服务：W=<Company，SwimmingPool>为列出一个机构拥有大型游泳池的情况的 Web 服务，W=<City， SwimmingPool>为列出一个城市拥有大型游泳池情况的 Web 服务，另外假设两个服务<Room，SwimmingPool>，<Location，SwimmingPool>。Web 服务<Hotel， SwimmingPool>表示用户想查询某宾馆提供游泳池的情况。

5.1 所需的本体

本文试验本体中定义了旅行服务所用到的各种属性和概念关系。图 4 中是一个简化的服务分类本体。

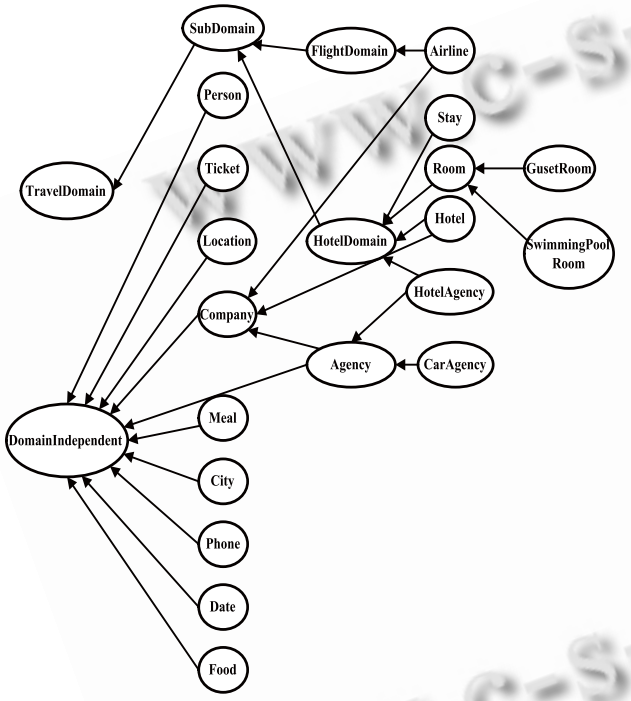

图 4 领域本体分类图

5.2 试验结果分析

从表 1 可以看出，对于服务 W=<Company，SwimmingPool>，Company 和 Hotel 具有直接上下位关系，因此语义距离为 1，对于 W=<City，Swimming Pool>，City 概念和 Hotel 概念之间几乎不存在上下位语义关系，因此只考虑语义距离情况下计算的语义距离较大。而每个 Hotel 都有一个 hasCity 的二元属性(表示该 Hotel 位于什么地方，hasCity 属性的值域为 City)，并且 Hotel 又是 Web 服务的输入概念，所以本文算法给出的语义距离很小。

表 1 语义距离计算结果

用户请求 Web 服务 W=<Hotel, SwimmingPool>		
Web 服务语义描述	只考虑上下位关系 IO 语义距离	本文计算语义距离
<Hotel, SwimmingPool>	0.0	0.0
<Company, SwimmingPool>	1.0	2.3
<City, SwimmingPool>	∞	0.45
<Room, SwimmingPool>	∞	0.50
<Location, SwimmingPool>	∞	1.51

尽管本例较为简单，但是基本上可以验证基于 OWL-S 的 IO 匹配算法的可行性。用户可以通过改变期望匹配级来改变模糊度，期望匹配级越低则返回的结果数量越多、期望的匹配级越高则返回的结果越少。

6 结语

目前，现在的 Web 服务缺乏语义描述是语义 Web 服务发现存在一个突出的问题。本文提出了一种基于 OWL-S/UDDI 的语义 Web 服务发现框架及有针对性的匹配算法，通过在服务查找中加入语义推理，增强了 UDDI 的语义 Web 服务发现能力。试验数据表明基于 OWL-S 的语义 Web 服务的发现模型有利于提高 Web 服务检索的查准率和查全率，取得了语义 Web 服务发现的效果。

参考文献

1 牟帅,黄映辉,李冠宇.语义 Web 服务的 OWL-S 描述及其应用.计算机技术与发展, 2009,19(1):13－16.
2 牟帅.语义 Web 服务的 OWL-S 描述及其应用[硕士学位论文].大连:大连海事大学, 2009.
3 刘涛.基于 OWL-S 语义 Web 服务发现技术的研究[硕士学位论文].武汉:武汉科技大学, 2008.