第32卷 第18期 计 算 机 工 程 2006年9月
Vol.32 № 18 Computer Engineering September 2006
·人工智能及识别技术· 文章编号：1000—3428(2006)18—0211—03 文献标识码：A 中图分类号：TP311

一种基于语义Web服务的服务自动发现的实现

沈玮韡，蔡鸿明，姜丽红
(上海交通大学软件学院信息化实验室，上海 200240)

摘 要：服务自动发现的实现中的核心问题是服务的自动匹配和定位。该文提出的解决方案是采用owl-s profile来提供发布服务属性和功能描述的广告，构建基于ontology的服务搜索引擎，采取服务参数匹配和输入参数匹配的分级匹配方式，实现了基于概念推理的服务自动定位。介绍了基于该方法开发出的一个电子商务系统原型，描述了它的系统架构和实现机制，分析了该解决方案的可行性。
关键词：语义Web服务；本体；服务自动发现；OWL-S

Implementation of Automatic Service Discovery Based on Semantic Web Service

SHEN Weiwei, CAI Hongming, JIANG Lihong
(Information System Lab, Software School, Shanghai Jiaotong University, Shanghai 200240)

【Abstract】The main point of automatic service discovery is locating services automatically. This paper introduces a solution to solve this problem. It uses owl-s profiles to publish service advertisement which describe service properties and functions, build a service match engine based on ontology, take a multi-step matchmaking method to implement automatic service discovery. It also describes the architecture and implementation of an e-commerce system prototype based on this solution, and analyses the feasibility of this solution.
【Key words】Semantic Web service; Ontology; Automatic service discovery; OWL-S

目前主流的Web服务描述语言和查询手段是WSDL和UDDI。它们并没有很好地实现自动化和互操作的目标[1]。WSDL对服务的描述是基于语法的，描述能力有限，仅仅提供了Web服务的接口描述，不能提供对服务功能和行为上的描述。UDDI是一套基于Web的、分布式的、为Web服务提供的信息注册中心的实现标准规范，它包含一组使企业能将自身提供的Web服务注册以使得别的企业能够发现的访问协议的实现标准，并且提供了一组基于标准的规范用于描述和发现服务。但是它同样不能表达服务的用途和能力，而且它对服务的搜索是基于字符串的[2]，因此可能查询到的服务不符合服务请求者的要求。

运用语义Web服务来实现服务发现的一个最显著的优势就是在于它能够在现有的Web服务标准的基础上为Web服务扩展语义信息[3]，以一种机器可解释的形式来标识用于服务发现的信息[4]，在可接受的时间和资源的限制下，自动操作和推理，极大地减少了人为参与。OWL-S，一种语义Web服务描述语言，能够提供比WSDL更为丰富的服务描述：服务的实体的联系信息，服务的输入、输出，服务执行的前置条件以及服务执行产生的预期的效果，服务所属的种类，服务的质量评价，以及一个不限长度的服务参数列表[4]。

通过匹配请求与service profile的输入输出参数来实现服务发现的方法[5]具有一定的局限性，因为无法保证请求中包含输出参数的信息(如电子商务系统)。因此，本文针对请求中不包含输出参数的情形，提出了一种基于profile服务参数和输入参数的匹配方式，并给出了输入参数匹配算法。

1 服务自动发现的实现

本节主要从用户需求的解析，寻找相似概念和服务的匹配3个方面来描述实现细节。

1.1 用户请求的解析

要发现满足用户需求的服务，首先要理解用户的需求，生成一个机器能够读懂的需求文档。用户是通过页面来输入自己的需求，这种需求是文本形式。因此，必须要对文本形式的用户需求进行解析，生成一个XML格式的文档。

定义1 需求集(RequestSet) 需求集是这样的一个集合：

{(Value(1),PropertyType(1),ClassName(1)),…,(Value(n),PropertyType(n),ClassName(n))}

其中集合的每一个元素是这样的一个三元组(输入值，属性，类名)。

定义2 需求类表(RequestClassList) RequestClassList是这样的一个集合：

{ClassName(1),ClassName(2),…,ClassName(n) }满足

$\forall$ element $\in$ RequestSet | element.3 $\in$ RequestClassList

集合Request的每一个元素的第三项值构成RequestClassList,且RequestClassList中的元素是不重复的。

根据如上的定义，将用户的需求解析成如下的Request.xml文档。

```
<RequestProduct>
<Customer>
    <CusName>CustomName</CusName>
    <CusAddress>CustomAddress</CusAddress>
    ….
```

基金项目：国家自然科学基金资助项目(70471024)
作者简介：沈玮韡(1981—)，男，硕士生，主研方向：信息化系统和电子商务；蔡鸿明，博士后；姜丽红，副教授、博士
收稿日期：2005-12-01 **E-mail：**sww811016@sjtu.edu.cn

```
</Customer>
<ClassName1>
        <PropertyType1>Value1</PropertyType1>
        <PropertyType2>Value2</PropertyType2>
…
</ClassName1>
….
```

其中，Customer 里面描述的是服务请求者的相关信息，Request.xml 中的每一个 ClassName 均定义在 RequestClassList 中。需求集中所有属于同一个 ClassName 下的属性值列在该 ClassName 里面。根据需求类表，生成该 request 对应的 request profile，如下所示：

```
<rdf:RDF   rdf:ID=”RequestService”>
                …
<profile:hasInput rdf:resource=”concept.owl#ClassName1”/>
<profile:hasInput rdf:resource=”concept.owl#ClassName2”/>
…
        </rdf:RDF>
```

1.2 匹配程度的定义

本文参考了文献[2,5]中定义的匹配程度，定义了 4 种匹配程度由高到低排列：完全匹配，可能匹配，部分匹配和不匹配。下面作简单说明：

(1)完全匹配(Full-Match)：指 2 个概念描述的是同一个事物，或者概念 2 包含在概念 1 之中。

(2)可能匹配[Possible-Match]：指第 1 个概念包含在第 2 个概念中。概念 1 是概念 2 的一个子概念，概念 2 不一定能够满足概念 1。

(3)部分匹配[Part-Match]：指第 1 个概念与第 2 个概念有交集但是不完全相同。是指概念 2 中存在一部分满足概念 1 定义的事物。

(4)不匹配[Not-Match]：两个概念描述的完全是两个不同的事物。

根据上面的匹配规则的定义，给出概念集的定义，用来描述两个概念的匹配程度。

定义 3 概念集(ConceptSet)：概念集是这样的一个集合：{(originalConcept，similarConcept1,matchDegree1).....}

概念集是一组三元组的集合，集合中列出所有与原概念相似的所有概念。其中，originalConcept 是原概念，similarConcept 是相似概念，matchdegree 指出它们的相似度。概念集中每个三元组的 originalConcept 相同，matchdegree 只能取完全匹配，部分匹配和可能匹配 3\种值。

1.3 服务的匹配

1.3.1 本体知识库的建立

在根据匹配规则进行匹配之前，首先要创建一个本体知识库。本体知识库是用来描述概念和概念，概念和实例之间关系的文件。只有建立了本体知识库，才能找到与用户需求中所提到的概念(或实例)相关联的概念(或实例)，这是进行匹配的前提。

1.3.2 服务描述

Web 服务采用 owl-s 来描述，在实现服务的发现的时候，主要是通过检索所有的 service profile 文件来进行匹配。之前的基于语义 Web 服务的服务发现通常是通过匹配 profile 里的 input 和 output 来实现的，但在电子商务中，用户需求中没有 output 信息，需求和服务之间的 output 无法进行匹配。解决办法是在描述每个服务的 profile 文件中增加 2 个 serviceparameter：ServiceType 和 ServiceContent，具体如下：

```
<rdf:RDF rdf:ID=”ProfileSample”>
…
  <profile:ServiceParameter>
    <addParam:Concept rdf:ID=”ServiceType”>
<profile:ServiceParamenterName>ServiceType</profile:Service
ParamenterName >
<profile:sParameter rdf:resource=”Concept.wol#SALE”/>
</ addParam:Concept >
</ profile:ServiceParameter >
…
</rdf:RDF>
```

ServiceType 定义服务的类型，如销售型服务、浏览型服务、物流型服务等。ServiceContent 是定义服务内容如对于销售型服务，ServiceContent 里面就是销售产品列表。ServiceType 只能取系统规定值，而 ServiceContent 只能取本体知识库中定义的概念(或者实例)。

1.3.3 服务匹配的实现方案

第 1 步：根据需求集生成对应的需求类表和概念集

通过本体知识库，运用匹配规则，找出每一个需求集中定义的概念(ClassName)相关联的概念，生成对应的概念集。

第 2 步：根据 Request.xml 来匹配 service profile 里面的 ServiceParameter——ServiceType 和 ServiceContent。从解析出来的需求文档中，可以获取用户需求类型和所需求的内容。例如：用户是要买 10 个鼠标。经过解析之后，可以知道 ServiceType=“Sale”&ServiceContent=“Mouse”。这样就可以从服务器中找出所有描述为销售类型，并且商品中包含鼠标产品的服务。但是，到这里匹配并没有结束，因为如果用户请求的输入参数与服务需要的输入参数不能匹配的话，这样的服务不能提供给用户。于是，还要做接下来的输入参数的匹配。

第 3 步：匹配概念集中的概念和 service profile 中的 input。

下面描述一个根据概念集来计算一个服务匹配度的算法(伪码)：

```
假设：需求类表中有 n 个概念，概念 i 对应的概念集为 conceptseti,
ServiceMatchDegree[i]=Not_Match   (0≤i<n)
for(k=0;k<n; k++)
{ isFailed =true;
for (p=0;p<conceptsetk.amout; k++) /* conceptsetk.amout 指概念集 k 中相似概念的个数*/
if(match(similarconcept(k,p),serviceinput)==Full_Match)
{ isfailed=flase;
ServiceMatchDegree[i]=MAX(ServiceMatchDegree[i],matchdegree(k,p));
}
if(isfailed=true) break; /*如果一个概念集里的概念在服务输入参数中都找不到，这个服务不匹配*/
}
```

其中，similarconcept(k,p)的意思是第 k 个概念集的第 p 个相似概念；matchdegree(k,p) 的意思是第 k 个概念集的第 p 个相似概念的相似度；match(similarconcept(k,p),serviceinput)函数的作用是将该相似概念与 serviceinput 中所有的参数进行匹配，返回匹配结果；算法中相似概念与服务输入参数的匹配必须是完全匹配，因为概念集中已经包括了所有与需求概念相似的概念，是一个闭包。

经过上述 3 个步骤之后，就能找出所有满足请求的 Web 服务。对于每一个服务来说，它的匹配程度取 MIN(Service

MatchDegree[1],…,ServiceMatchDegree[n])。

2 系统原型架构

目前开发的电子商务系统是一种 J2EE 架构，B/S 结构的原型系统。视图层由 JSP 页面组成，控制层由 Struts 框架描述和 servlets 实现，模型层和持久化层由 EJB 实现，封装了所有的业务逻辑的实现，后台数据库采用 SQL 2000。服务器容器采用 Tomcat+Jboss。服务描述语言采用 OWL-S，本体知识库由 OWL 语言描述。

该系统原型有 3 个主要模块(如图 1 所示)。

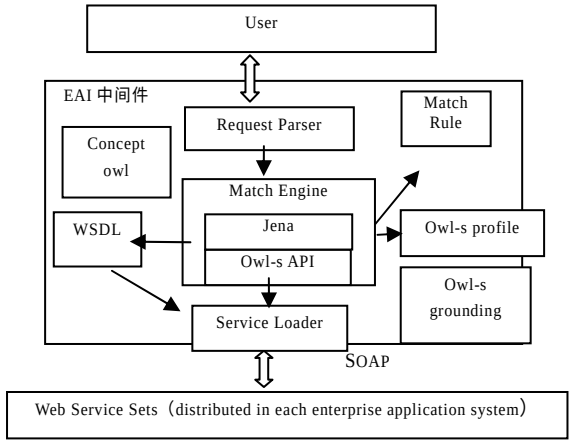

图 1 系统架构

(1)Request Parser 用来解析用户请求，它按照需求集和需求类表的概念分别生成 Request.xml 和 RequestProfile.xml，并将 RequestProfile.xml 发送到 Match Engine。

(2)Match Engine 是实现服务自动发现的核心组件，它能够分析 RequestProfile.xml，找出其中所有的概念(ClassName)。Jena 可以查询本体知识库 concept.owl，根据上面定义的匹配规则找出 RequestProfile.xml 中所有概念的相似概念，并判断二者的匹配程度，生成每个概念的概念集。OWL-S API 用于从 OWL-S profile 中读取参数信息，返回给 Java 主程序。

(3)ServiceLoader 会根据用户选择的服务对应的 profile 文件，找到其对应的 grounding 文件，绑定到对应到 WSDL 文件，最终实现服务的定位。最后装载 Request.xml 里的数值，调用该 Web 服务。

3 结束语

本文提出的服务发现实现方案适用于电子商务系统，实现是通过在服务描述文件 profile 中加入 ServiceType 和 ServiceContent 两个 serviceparameter，这样在匹配的时候就增加了 2 个约束条件，再通过输入参数的匹配能够找到用户所请求的服务。

当前的原型系统已经实现了请求解析模块和服务装载模块，搜索引擎中的匹配规则已经实现了完全匹配和可能匹配和不匹配，部分匹配规则还未在系统实现。此外采用 owl-s API 来导入 profile 参数信息时响应时间过长，系统的安全性能暂时还没考虑，这些都是接下来需要解决的问题。

参考文献

1 马应龙. 基于进化分布式本体的语义 Web 服务动态发现[J]. 计算机学报, 2005, 28(4): 603-614.

2 Li Lei, Horrocks I. A Software Framework for Matchmaking Based on Semantic Web Technology[C]. Proceedings of International WWW Conference, Budapest, Hungary, 2003.

3 史忠植, 蒋运承, 张海俊等. 基于描述逻辑的主体服务匹配[J]. 计算机学报, 2004, 27(5): 625-635.

4 OWL-S 1.1 Specification White Paper [EB/OL]. http://www.daml.org.

5 Massimo P, Terry R P. Semantic Matching of Web Service Capabilities[C]. Proceedings of International Semantic Web Conference, Sardinia, Italy, 2003.

(上接第 156 页)

表 1 的实验数据是函数 ParseRulesFile()对 93 条 icmp-info 类规则进行处理的时间值对比。由于需要构建规则头的十字链表和规则选项的十字链表，系统生成规则树时会相应增加处理时间。实验结果表明，修改后的程序增加了约 6.0%的处理时间。由于规则树的生成是在系统初始化时完成，不影响系统对数据报的匹配操作，因此其时间的增加对系统的整体性能无直接影响。

表 1 生成规则树的时间对比

组次	第 1 组	第 2 组	第 3 组	第 4 组	第 5 组	平均值
原程序（s）	0.341	0.332	0.326	0.332	0.344	0.335
修改后（s）	0.333	0.385	0.364	0.349	0.344	0.355

然后是规则匹配时的时间对比。在匹配函数 Detect (Packet*p)中增加时间函数。根据系统对 CheckTcpFlags()函数的调用次数，记录 5 组 10 000 个 TCP 数据包的总匹配时间并与原程序进行对比(见表 2)。

表 2 规则匹配时间对比

组次	第 1 组	第 2 组	第 3 组	第 4 组	第 5 组	平均值
原程序（s）	2.877	2.836	2.778	3.065	2.402	2.791 6
修改后（s）	1.907	1.745	2.087	2.053	1.787	1.911 8

实验数据表明，原程序的检测时间平均为 2.791 6s，修改后平均为 1.911 8s。通过引入参数十字链表驱动的匹配方式，提高检测过程的并行性，系统的匹配速度比原来提高了约 31.5%。

5 结束语

本文对Snort及其规则匹配的算法进行了系统地分析，针对其存在的问题提出了在规则头和规则选项两级匹配过程中，对于其条件处理函数引入参数十字链表驱动的方式。从而在规则匹配的过程中可以减少大量重复调用处理函数的系统时间，并充分利用参数内容之间的内在关系，及时地削减无效的规则。下一步的目标是在Snort2.0 基础上，对多模式匹配算法[3,4]进行改进，使之能够动态地对多个模式串进行删减，从而使本套方案能应用在Snort2.0 以上版本。

参考文献

1 Roesch M. Snort-lightweight Intrusion Detection for Networks[Z]. http://www.snort.org/docs/ lisapaper.txt, 2003-02-20.

2 Roesch M, Green C. Snort Users Manual[Z]. http://www.snort.org, 2004-08-11.

3 Coit J C, Staniford S, McAlerney J. Towards Faster String Matching for Intrusion Detection[C]. Proc. of DARPA Information Survivability Conference and Exposition, 2001: 367-373.

4 Norton M, Roelker D. Hi-performance Multi-rule Inspection Engine [Z]. http://www.snort.org, 2004-04.