基于语义近似逼近技术的多代理协作*)

吴克河[1,3] 马应龙[1,3] 郑 义[1] 周明全[2] 林鹏程[1]

(华北电力大学计算机科学与技术系 北京 102206)[1] (北京师范大学信息科学和技术学院 北京 100875)[2]

(电站设备状态监测与控制教育部重点实验室 北京 102206)[3]

摘 要 本体在多代理系统中起着重要的作用,它提供和定义了一个共享的语义词汇库。然而,在现实的多代理通讯的过程中,两个代理共享完全相同的语义词汇库是几乎不可能的。因为信息不完整以及本体的异构等特性,一个代理只能部分理解另外一个代理所拥有的本体内容,这使得代理间的通讯非常困难。本文就是探索利用近似逼近技术实现基于部分共享分布式本体的多代理通讯,从而实现多代理之间的协作查询。我们使用基于 OWL Web 本体语言的描述逻辑来描述分布式本体的近似查询技术。最终我们也开发了基于语义近似逼近方法的一个多代理协调查询系统。

关键词 本体,多代理,语义近似逼近

Multi-agent Coordination Based on Semantic Approximation Technology

WU Ke-He[1,3] MA Ying-Long[1,3] ZHENG Yi[1] ZHOU Ming-Quan[2] LIN Peng-Cheng[1]

(Department of Computer Science and Technology, North China Electric Power University, Beijing 102206)[1]

(School of Information Science and Technology, Beijing Normal University, Beijing 100875)[2]

(Key Laboratory of Condition Ministering and Control for Power Plant Equipment, Ministry of Education, Beijing 102206)[3]

Abstract Ontology plays an important role in multi-agent systems, and provides and defines a sharable semantic vocabulary. However, within multi-agent systems, it is almost impossible for multiple Web agents to completely share a same semantic vocabulary. Because of incomplete information and ontological heterogeneity, an agent can partially understand the contents of other ontologies. This makes multi-agent communication difficult. In this paper, we exploit and utilize semantic approximation for implementing better multi-agent communication based on partial shared distributed ontologies, and hence achieve coordinate query among multiple agents. We use description logic based on OWL ontology language for describing ontological information. We also develope a multi-agent coordination system based on semantic approximation.

Keywords Ontology, Multi-agent, Semantic approximation

1 引言

在语义 Web 查询系统中,为了执行一个给定的任务,代理之间应该相互协调合作,因此多代理系统的一个重要研究方面就是如何更好地实现不同代理之间的通讯。代理间良好的通讯依赖于代理之间的良好协调和互操作。本体提供了支持代理之间的通讯的技术,因为它提供了本体之间的通讯的语义词汇[1]。然而,现实的多代理通讯面临许多的问题。Uschold[2]认为基于本体的代理通讯面临的问题可以分成语言异构和术语异构等方面。我们这里主要强调术语异构性,而忽略语言异构的问题。实际上,代理经常使用以不同的方式定义的私有本体,不能直接共享。在这种情况下,如果要让两个代理相互理解,这两个代理所拥有的本体应该如何进行处理?这里有一些基本的方法处理基于本体的多代理之间的通讯问题:

合并。通常的做法是为这些不同的代理提供一个共享的全局的本体[3]。但是这种方法可扩展性特别差,因为当新的知识加入到本体中的时候,这个全局的本体就需要不断地被扩充,还要保证它的扩展是正确的。

映射。这种方法同合并不同的是,它保持每个本地本体的自治性,为了实现共享只要通过映射分布式本体之间的术语映射方式实现[4]。

但是,到目前为止,这些方法都几乎没有考虑到在进行本体合并或映射的时候存在信息冲突和信息不完整的现象,大多数的研究都是基于严格的语义约束进行的。在良构建立的单个本体语义之上,本体信息冲突可以有效地避免。但是问题在于分布式本体的设计者和创建者们不能保证在分布本体合并或者映射过程中不出现信息冲突。因为这些本体可能是松散建立的,可能由不同的人在不同的时间建立,而且也可能针对的是不同论域的(只是它们的本体信息可能部分相关而已),所以我们有理由相信分布式本体之间可能出现冲突或不完整的现象是普遍存在的。在这种情况下,会出现一些不能完全匹配的语义术语。术语不匹配就意味着不可能得到查询结果,因此需要一种有效的机制实际解决这个问题。目前大多数方法都是使用本体或描述逻辑在完整信息的框架下组织和描述信息。本文实现了一种术语的近似查询机制,用于处理多代理系统中的本体在术语不能完全匹配的情况下的多代理系统协调。这种方法不用对本体的逻辑语义体系进行扩展,简单易行,可操作性强。

本文第 2 节介绍了本体的概念及其开发工具;第 3 节介

*)国家 863 计划项目(2004AA1Z2450)。吴克河 教授,CCF 会员,主要研究方向:计算机网络及应用、智能软件技术;马应龙 博士,主要研究方向:软件工程、语义 Web、面向服务计算、形式化方法等;郑 义 硕士,主要研究方向:语义 Web、数据库和管理信息系统;周明全 教授,博士生导师,主要研究方向:计算机可视化技术、软件工程、中文信息处理。

绍了系统的功能和架构；第4节描述了语义匹配、分配律的应用和代理协调3个核心问题；第5节介绍了系统查询接口的实现；第6节是相关工作，最后是结论。

2 本体

2.1 本体的概念

虽然本体的研究日趋成熟，但在各种文献中，对于本体的相关概念和术语并不完全一致。这里通过给出几个有代表性的定义来解释本体的概念：

(1)本体是给出构成相关领域词汇的基本术语和关系，以及利用这些术语和关系构成的规定这些词汇外延的规则的定义[5]。

(2)本体是共享概念模型的明确的形式化规范说明[6]。

(3)本体是用于描述或表达某一领域知识的一组概念或术语。它可以用来组织知识库较高层次的知识抽象，也可以用来描述特定领域的知识[7]。

(4)本体属于人工智能领域中的内容理论，它研究特定领域知识的对象分类、对象属性和对象间的关系，它为领域知识的描述提供术语[8]。

从以上定义我们可以知道，本体通过对概念、术语及其相互关系的概念化、规范化的描述，来定义某一特定领域的基本知识体系和描述语言。其中“概念化”是指把客观世界中的现象抽象成概念(如实体、属性、过程)，并加以定义和相互关系的描述，它独立于具体的环境状态而存在。“规范化”指本体能被形式化地表示出来，从而能被计算机处理。“共享”指本体中体现的是共同认可的知识，反映相关领域中公认的概念集。

2.2 本体语言

目前在语义Web中描述本体的语言有很多，本文仅介绍其中最具代表性的RDF，RDFS，OWL三种语言。

RDF即资源定义框(Resource Description Framework)。RDF数据模型以XML语法为基础，把任何事物都看成是资源，并用三元组(主题、属性、对象)描述。RDF具有通用性，并不限定于某个领域的网络资源定义。

RDF Schema是RDF的定义机制。有了它，我们可以根据需要定义自己的词汇，而且不需要另外开发软件来解释自定义的词汇。RDF Schema的这个机制是应用程序都必须遵守的，所以应用程序对用它定义的词汇的文档会有统一的解析。

OWL(Web Ontology Language)吸取了DAML+OIL的设计和应用经验，是对RDF进行扩展的基础上发展起来的。OWL可以定义词汇之间的关系、类与类的关系、属性与属性之间的关系等等，从而可以使应用程序对内容信息进行处理，而非仅仅是表示信息。OWL是公认的未来的Web本体语言标准，由OWL Lite、OWL DL和OWL Full 3个表达能力递增的子语言组成。

这3种语言的表达能力依次由低到高，其中OWL的表达能力最强，本文在后面的系统设计中也是采用OWL语言表示本体。下面是一个描述软件公司的本体的简单例子，其中xmlns指本体的命名空间、owl:Class定义本体的类，rdfs:subClassOf定义本体的子类。关于这个本体的图形表示见图2中的代理1本体。

```
〈rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://www.owl-ontologies.com/softwareCom.owl#"
xml:base="http://www.owl-ontologies.com/softwareCom.owl"〉
 〈owl:Ontology rdf:about=""/〉
 〈owl:Class rdf:ID="staff"〉
  〈rdfs:subClassOf〉
   〈owl:Class rdf:ID="SoftwareCompany"/〉
  〈/rdfs:subClassOf〉
 〈/owl:Class〉
 〈owl:Class rdf:ID="DeviceMaintenence"〉
  〈rdfs:subClassOf rdf:resource="#SoftwareCompany"/〉
 〈/owl:Class〉
 〈owl:Class rdf:ID="Manager"〉
  〈rdfs:subClassOf rdf:resource="#staff"/〉
 〈/owl:Class〉
 〈owl:Class rdf:ID="Programmer"〉
  〈rdfs:subClassOf rdf:resource="#staff"/〉
 〈/owl:Class〉
 〈owl:Class rdf:ID="ComputerFittings"〉
  〈rdfs:subClassOf rdf:resource="#DeviceMaintenence"/〉
 〈/owl:Class〉
 〈owl:Class rdf:ID="Identifier"〉
  〈rdfs:subClassOf rdf:resource="#ComputerFittings"/〉
 〈/owl:Class〉
〈/rdf:RDF〉
```

2.3 本体开发工具

目前存在众多的本体开发工具，比较知名的有Protégé[9]，Oiled[10]，OntoEdit[11]等，这些工具都提供了创建、编辑和集成现有本体的开发环境。这里采用Protégé构建本体，来配合HP公司的Semantic Web Lab的Jena开发包[12]和SUN公司的JDK开发包进行系统开发。下面我们对其中最主要的Jena开发包进行简单介绍。

Jena是一个由HP实验室开发的表示和处理半结构化数据(主要是基于RDF的管理、查询等)的Java开源项目，包括的功能有RDF数据的读写和输出，RDF的编辑、合成和检索，RDFS、OWL、DAML+OIL等本体的操作，利用SQL数据库保存数据，利用检索语言RDQL检索，基于本体的规则的推理和验证等。

2.4 本体查询语言RDQL

RDQL是Jena模型中专门为查询RDF文件设计的查询语言，它同时支持OWL本体文件。一个RDF模型通常描述成三元组模式的图形。RDQL语句也是由一系列的三元组的图形模式组成，而每个三元组是由变量、关系和术语组成，RDQL就是通过定义变量的约束条件来进行查询的。以上面定义的本体为例，如果我们要查询staff的子类，通过本体文

〈? xml version="1.0"?〉

件我们可以清楚地看到 staff 的子类是 Manager 和 Programmer，描述成 RDQL 语句如下：

```
SELECT ? x
WHERE (? x, (rdfs:subClassOf),
(http://www.owl-ontologies.com/softwareCom.owl#staff))
```

在这个语句中，WHERE 子句中依次是变量、关系、命名空间下的术语，这里是通过关系和术语来约束变量的。

3 近似查询系统的设计

3.1 系统功能

本系统主要提供给用户近似查询的功能，即为用户提供一个输入框，用于输入查询字符串（字符串可以有一个或多个，多个时用一个空格隔开），然后把这个字符串解析成一个一个的子字符串，比如用户输入“ontology jena”，系统就会解析成“ontology”和“jena”，用户也可以通过在字符串的前面加“-”输入一个字符串的非值，比如要查找非 ontology 的内容，就可以输入“-ontology”。接下来根据解析好的一个或多个字符串在本地本体中进行取交集查询。如果遇到本地本体中没有的字符串，就连接到服务器查询其他本体，由服务器返回的值再在本地进行查询。最后将查询结果返回给用户。

在这个过程中，每个本体都需要有一个代理来处理用户的请求，这就涉及到代理协调的问题；如果需要查询的字符串在本地本体中不存在，需要对此字符串进行近似匹配；如果用户要查询的字符串很多，需要对查询的字符串进行运算后再做查询。我们在第 4 节中将对这些问题进行详细阐述，首先我们介绍一下本系统的结构。

3.2 系统结构设计

系统采用 3 层 b/s 结构设计。如图 1 所示，使用 jsp 技术设计表现层界面，主要用于和用户的交互，接收用户的查询信息，并把查询结果显示给用户；中间层采用 jena 开发包、socket 编程等技术，主要处理业务逻辑，包括字符串解析、字符串查询、代理消息传递、字符串匹配等；数据层我们在 owl 语言设计的本体中存放数据。

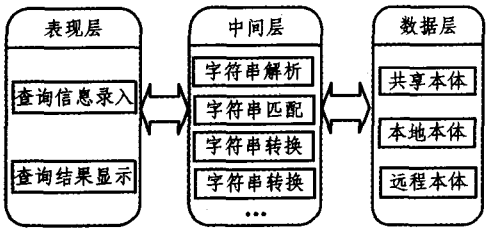

图 1 近似查询系统结构

这种设计方式可以使用户随时随地都可以进行查询，不受专用客户端的限制。可扩展性、可维护性、可重用性、共享性等多方面都大大增强，业务处理更加灵活。

4 关键技术

4.1 字符串的近似匹配

当我们要查询一个字符串时，如果这个字符串在本地本体中不存在，就需要对它取近似值。这时候我们需要询问其他代理，以确定该字符串是否在远程本体中存在。如果存在，就需要访问共享本体，查找出该字符串和本地本体中的字符串的关系（其中共享本体中存放着远程本体和本地本体之间类的映射关系），然后在本地本体中求出它的近似值。下面我们就字符串的近似匹配给出形式化描述。

这里我们假定存在两个系统代理 1 和代理 2，以及它们分别拥有的本体。这两个代理的共享本体就是通过映射建立的最小的共享本体，我们记为 SharedOnto。我们这里明确说明：如果这里建立的共享本体并不是最小共享本体，下面的定义依然有效。

定义 1（下界逼近） 设 LC 是所有本地代理的本体概念的集合，SC 是 SharedOnto 的概念的集合，且概念 $C\in LC$。在 SharedOnto 中的概念 $C_{glb}\in SC$ 称为概念 C 的下界，如果

1) $C_{glb}\sqsubseteq C$

2) 如果存在 $C'\in SC$ 使得 $C'\sqsubseteq C$，那么 $C'\sqsubseteq C_{glb}$

这里，令 $glb_{SharedOnto}(C)$ 代表概念 C 在 SharedOnto 本体的 SC 中所有的下界概念的集合。

定义 2（上界逼近） 设 LC 是所有本地代理的本体概念的集合，SC 是 SharedOnto 的概念的集合，且概念 $C\in LC$。在 SharedOnto 中的概念 $C_{lub}\in SC$ 称为概念 C 的上界，如果

3) $C\sqsubseteq C_{lub}$

4) 如果存在 $C'\in SC$ 使得 $C\sqsubseteq C'$，那么 $C_{lub}\sqsubseteq C'$

我们令 $lub_{SharedOnto}(C)$ 代表概念 C 在 SharedOnto 本体中的 SC 的所有的上界概念的集合。

在这些定义的基础上，基于共享本体 SharedOnto，我们可以确定一个概念的成员的一个实例 x 是不是一个本地本体的概念的实例。反过来这种方式也同时提供了基于本地术语的一种近似查询结果，也就是说：

1) 如果 x 是 SC 的概念 C 的下界的成员的实例，那么它同时也是 C 的实例；

2) 如果 x 不是 SC 的概念 C 的所有的上界的成员的实例，那么它也不是 C 的实例。

本文的这种方法首先是由 Selman 等人[14]针对理论逼近（Theory approximation）于 1996 年提出来的。为了使得我们更加清晰地实现基于描述逻辑的逼近查询，我们做进一步的定义。

定义 3（概念逼近） 设概念 LC 是本体代理的本体的概念，SC 是这个代理的共享本体 SharedOnto 的概念的集合，且 x 是 SC 中一个共享概念的实例，那么对于 $C\in LC$，我们定义一个映射 M，使得

$M(x, C)=1$，如果

$x:(\bigvee_{C'\in glb_{SharedOnto}(C)} C')$

$M(x, C)=-1$，如果

$x:\neg(\bigwedge_{C'\in lub_{SharedOnto}(C)} C')$

$M(x, C)=?$，否则

其中，在 $M(x, C)=1$ 的情形下我们称概念 C 的描述为非负，在 $M(x, C)=-1$ 的情形下称 C 的描述为负。进一步对上述定义中的逻辑符号做一些解释如下：我们使用 $x:C$ 表示个体 x 是概念 C 的实例。

$x:(\bigvee_{C'\in glb_{SharedOnto}(C)} C')$ 表示个体 x 是概念 C 的某个下界概念的实例。

$x:\neg(\bigwedge_{C'\in lub_{SharedOnto}(C)} C')$ 表示 x 不是 C 的任何一个上界概念的实例。

定义 4(术语重写规则) 从基于本地代理的查询中的概念到共享本体 SharedOnto 的共享概念的重写规则定义如下：

—如果概念 C 的表示为非负，那么使用

$\bigvee_{C' \in glb_{SharedOnto}(C)} C'$ 代替概念 C。

—如果概念 C 的表示为负，那么使用

$\bigwedge_{C' \in lub_{SharedOnto}(C)} C'$ 代替概念 C。

通过上面的定义我们可以得出这样一个结论：通过术语重写方法得到的查询结果集一定属于原本的查询结果集。但在本体不能包含查询字符串的时候，这样的结果已经是最优解。有关以上定义正确性的证明请参考文[13]。接下来，我们使用一系列的算法(算法 1 到算法 3)的 Java 实现来具体刻画这样的一个基于多代理系统的分布式本体术语的替换(重写)过程。通过这些算法，基于本体的代理之间可以相互协调完成一个共同的查询任务。其中算法 3 和算法 2 唯一的区别就是变量 queryString 的定义，这里只给出了 queryString 的初始化值的定义。

算法 1 术语替换算法 (Terminology Replacement Algorithm)

```
public String[] replacement(String node, boolean isComplement,String owlFilePath){
        if(isComplement)
            return lookupDirectSupers(node,owlFilePath);
        else
            return lookupDirectSubs(node,owlFilePath);
    }
```

算法 2 查找下界算法 (LookupDirectSubs Algorithm)

```
public String[] lookupDirectSupers(String node,String owlFilePath){
    String[] result=new String[50];
     String queryString = "SELECT ? x WHERE (? x, 〈rdfs:subClassOf〉,
                    〈http://www.owl-ontologies.com/unnamed.owl#"+node+")〉)";
    OntModel model=q.getOntoModel(owlFilePath);
    Iterator results=q.query(model,queryString);
    int resultNum=0;
    while(results.hasNext()){
      ResultBinding res=(ResultBinding)results.next();
        String x=res.get("x").toString();
        x=x.substring(x.indexOf('#')+1);
        result[resultNum]=x;
        resultNum++;
    }
    String[]s=new String[resultNum];
    for(int i=0;i<resultNum;i++){
      s[i]=result[i];
    }
    return s;
    }
```

算法 3 查找上界算法(lookupDirectSupers Algorithm)

```
.....
String queryString="SELECT ? x WHERE
        (〈http://www.owl-ontologies.com/unnamed.owl#
        "+node+"〉,
                        〈rdfs:subClassOf〉, ? x)";
.....
```

下面我们举例说明字符串是如何近似匹配的。如图 2 所示的例子，两个不同的法人实体为了实现它们的资源共享，采用多代理技术实现其共享应用。一个实体是关于软件开发公司的日常业务，比如软件开发公司下属设备维修部，专门负责维修公司的电脑配件，它同时也拥有不同工作性质的员工。另外一家实体是一个电子公司，它下属有销售部，负责销售公司库存的硬件设备，这些设备可能有一些 CPU、主板、外设等一些其它硬件设备。

本体中的实线代表类的继承关系，两个本体之间的实线代表类的映射关系，这里所说的映射在本体中表示为类的继承，箭头指向的一方为父类。我们的共享本体就是根据这种映射关系建立起来的。

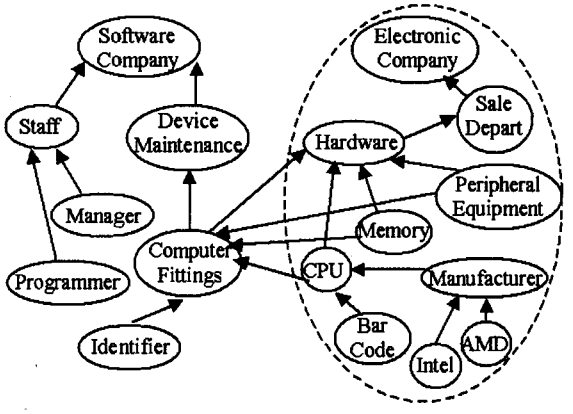

图 2 近似匹配本体实例

当软件开发公司需要新的电脑配件用于维护本公司电脑的时候，它可能会通过电子公司的代理去找出一些在型号、价钱等方面满足其要求的硬件配件。为了实现这样的查询，我们知道这两个公司的信息代理需要相互的协作。

例如：我们要通过代理 2 查询 Intel 公司生产的计算机设备(ComputerFittings)，它的形式化表述为：

$\text{ComputerFittings}^1(x) \wedge \text{Intel}^2(x)$。

我们发现代理 2 不知道计算机设备这个术语，这就需要同代理 1 协调共同完成这个查询。另外，代理 1 也没有同 Intel 完全匹配的术语，因此这种查询需要采用近似匹配的方法。

由图 2 中两个本体的映射关系我们发现：CPU、Memory、PeripheralEquipment 都是 ComputerFittings 的子类，也即

$\text{CPU}^2 \sqsubset \text{ComputerFittings}^1$；

$Memery^2 \sqsubseteq ComputerFittings^1$；

$PeripheralEquipment^2 \sqsubseteq ComputerFittings^1$。

所以，根据最小共享本体 SharedOnto 和概念逼近原则可以得出 ComputerFittings 的下确界为：

$glb_{SharedOnto}(ComputerFittings^1) = \{CPU^2, Memery^2, PeripheralEquipment^2\}$。

然后根据术语重写规则 $Computerfittings^1(x) \wedge Intel^2(x)$ 就可以近似改写成：$(CPU^2(x) \vee Memery^2(x) \vee PeripheralEquipment^2(x)) \wedge Intel^2(x)$。

同理，如果我们要查询 Intel 公司生产的非计算机设备，形式化描述成：

$\neg(ComputerFittings^1(x) \wedge Intel^2(x))$。

ComputerFittings 的上确界为：

$lub_{SharedOnto}(ComputerFittings^1) = \{Hardware^2\}$

因此 $\neg ComputerFittings^1(x) \wedge Intel^2(x)$ 可以改写成：$\neg Hardware^2(x) \wedge Intel^2(x)$。

4.2 分配律在字符串转换中的应用

本系统在做查询的时候，是利用 Jena 包为我们提供的 RDQL 查询功能，根据用户的查询字符串来构建 RDQL 语句。但是当用户输入较长的字符串，其中可能有多个子字符串需要用上述的近似匹配规则改写，这样最终会得到一个不规则的查询表达式，对构建 RDQL 语句造成不小的麻烦。而我们知道，在离散数学中，任何一个命题公式都可以转换成和它等值的析取范式。为了能够让查询语句有一个统一的简单易行的构建方法，这里我们使用分配律把查询表达式转换成等值的析取范式。例如上面的例子：

$(CPU^2(x) \vee Memery^2(x) \vee PeripheralEquipment^2(x)) \wedge Intel^2(x)$使用分配律转换成析取范式为：$(CPU^2(x) \wedge Intel^2(x)) \vee (Memery^2(x) \wedge Intel^2(x)) \vee (PeripheralEquipment^2(x) \wedge Intel^2(x))$。算法 4 给出了分配律的 java 代码实现。

算法 4 分配律算法（Allocation Algorithm）

```
public static List solve(List srcList) {
    if (srcList==null)
      throw new NullPointerException();
    List tempList=new ArrayList();
    tempList.add(new HashSet());
    while (srcList.size() > 0) {
      List tList=(List) srcList.get(0);
      tempList=Allocate(tempList, tList);
      srcList.remove(0);
    }
    return tempList;
}
public static List Allocate(List dst, List src) {
    List tempList=new ArrayList();
    int index=0;
    do {
      for (int j=0; j < src.size(); j++) {
        Set s=new HashSet((Set) dst.get(index));
        s.add(src.get(j));
        tempList.add(s);
      }
      index++;
    } while (index < dst.size());
    return tempList;
}
```

上述代码中 Allocate(List dst, List src)函数实现了把两个并集的交集转换成析取范式，solve(List srcList)函数通过循环调用 Allocate 函数实现把多个并集的交集转换成析取范式。

4.3 多代理之间的协调

在上面论述中我们提到，当术语不在同一个代理本体中时，需要多代理的相互协调，共同完成查询。我们这里假定有两个代理，代理协调过程如图 3 所示。此图描述的是一个术语的两个代理的协调过程。

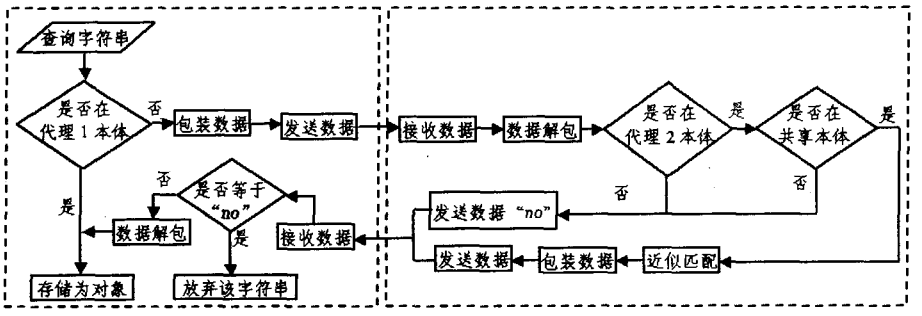

图 3 代理协调

1）当该术语存在于代理 1 本体中时，就直接把这个术语包装成对象，当术语不在代理 1 本体中时，就把它进行包装，发送到代理 2 中进行处理。

2）代理 2 接收到数据以后，首先对它进行解包，然后判断该术语是否在代理 2 本体和共享本体中。如果有一个本体中不包含这个术语，那么就返回给代理 1 一个“no”值，代表代理 2 不能理解这个术语。如果该术语在两个本体中都存在，就通过上述的近似匹配算法求出该术语的近似匹配值，返回给代理 1。

3）代理 1 接收到代理 2 返回的信息后进行判断，如果返回的是“no”，则意味着所有的代理都不能理解该术语，因此把它忽略。如果不是“no”，就对接收信息解包，包装成对象。

在这个过程中有几点问题需要说明。

①数据的包装格式：由于在近似匹配过程中只用到了两

个术语的属性:是否为非和术语的名称,因此我们把这两个值连成一个字符串,并用“,”隔开。

②代理间通讯:代理间使用 TCP 通讯协议,采用 java Socket 通讯技术,以确保数据能够准确传输。代理 2 采用多线程处理技术,保证每个客户端的请求都能及时响应。

③过程的最后把处理的结果包装成对象,是为了更好地为下一步分配律运算工作打好基础。对象的格式如下:

```
public class ListNode{
boolean complement;
String node;
String [] replace;
int nodeNum;
}
```

其中 complement 表示该术语是否为非;node 记录术语的名称;replace 记录术语的匹配值;nodeNum 记录术语或匹配值的个数。当 nodeNum=0,代表该术语不能被理解;当 nodeNum=1,代表该术语存在于代理 1 中或者匹配后的值有一个,这时候我们用 node 的值进行运算。当 nodeNum>1 代表该术语被近似匹配并且匹配值为多个,这时候我们用 replace 的值进行运算。

5 查询的实现

5.1 查询接口

查询是本系统的主要内容,系统中大部分模块的实现都是基于查询实现的,因此在系统中专门设计了一个查询接口,为其他模块提供最基本的查询服务。查询接口提供了如下几个基本功能:

求交集(queryIntersection)。对不定个数的术语的交集的查询;

求并集(queryMerge)。对不定个数的术语的并集的查询;

求非(queryComplement)。查询一个术语的补集;

存在判断(nodeIsExit)。判断一个术语在某个本体中是否存在。

为了实现动态查询,设置的方法通过接收用户的输入动态地构成 RDQL 查询语句。以求交集为例:如果用户的输入为“ComputerFittings Intel”,说明用户想查询 ComputerFittings 和 Intel 两个术语的交集。系统接收到字符串后,首先把这个字符串转化成长度为 2 的字符串数组,然后把这个字符串数组作为参数传给方法 queryIntersection,方法通过以下语句构建 RDQL 查询语句:

```
String queryString=
                "SELECT ? x WHERE";
for(int i=0;i<node. length;i++){
 queryString+=("(? x,〈rdf:type〉,〈"+nameSpace+"#"
    +node[i]+"〉)");
 if(i! =node. length-1)
  queryString+=",";
}
```

其中 node[]就是存放 ComputerFittings 和 Intel 的数组,nameSpace 是另一个参数,表示命名空间。最终得到的 RDQL 语句为:

```
SELECT ? x
WHERE (? x,〈rdf:type〉,
〈http://www. owl-ontologies. com/softwareCom. owl#ComputerFittings〉),
      (? x,〈rdf:type〉,
〈http://www. owl-ontologies. com/softwareCom. owl # Intel〉)
```

意思是查询既属于 ComputerFittings 又属于 Intel 的实例。

5.2 查询界面

为了给用户提供友好、简洁的操作方式,查询界面设计成 3 部分:查询输入框、查询按钮、查询结果栏,如图 4 所示。

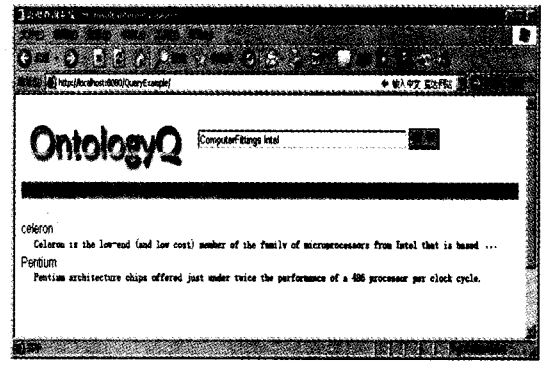

图 4 返回查询结果

图 4 中“请输入查询字符串”栏用于接收用户的查询信息,所要查询的术语可以有一个或者多个。当术语为多个时,要用空格隔开;后面的“查询”按钮用于执行查询;下面的空白处用于显示查询结果。如果不能得到查询结果,系统会显示“对不起,没有您想要的查询结果!”,如图 5 所示。

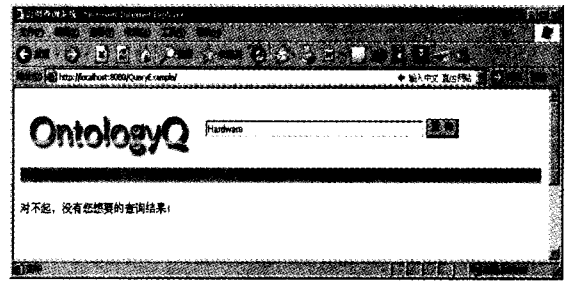

图 5 无匹配查询

6 相关工作

基于本体的近似查询技术的基础知识首先来源于知识汇编(Knowledge Compilation)[14]。它是一种在没有完全对应的匹配情况下所采用的一种近似匹配策略。目前的一些基于描述逻辑的术语近似查询方法[15~17]主要是为了实现信息集成和信息仓库的更好维护。它们并不是从术语的语义信息不完整,或者信息存在冲突的角度考虑近似查询的。正是因为这样,这些研究就降低了查询术语的难度。实际上,在分布式开放环境下,特别是语义 Web 环境下,充斥着大量的冲突信

息以及不完整的信息。因此，我们需要探讨在上述环境下的近似查询。本文考虑到多代理系统中不存在完整匹配的语义术语或者存在一些所谓的缺省信息和不完整信息的情况下如何实现正确的语义查询。Schaerf 和 Cadoli[18] 定义了一个良构的逻辑并提供了一个快速算法，用于近似推理。他们的查询方法特别依赖于具体参数的选择，这些参数是一个字母表的子集，用于确认近似逼近的程度。但是就这些参数来说，它们的哪些值能得到最好的逼近，是很难确定的。我们的方法可以通过术语的替换或者重写，将一个代理中不理解的术语通过协调交给另外一个代理去理解。我们的方法比较近似于文[19]的工作，但是我们的研究与它的区别主要有：1) 我们使用的是基于 OWL 本体语言的描述逻辑，OWL 目前已经成为标准。2) 通过上面的例子可以看出，我们的查询表达式中的概念可以具有概念并(disjunction)描述，而不是只具有交(conjunction)描述(文[19]只具有交描述)。3) 我们执行术语查询得到的所有结果（如果查询结果存在的话），一定是根据本体的语义而得到的用户想要的结果，因为替换术语的概念语义上比查询术语的概念是缩小的。而文[19]刚好相反，使用比查询概念语义范围上更大的概念替换查询术语，因此不能保证语义上总是正确的。

结论 本文提出了一种在多代理通讯情形下如何更好地实现代理之间的协调，为用户提供更好的查询能力。我们主要介绍了如何实现基于分布环境下的本地代理的查询，并提出了一系列相应的算法。值得注意的是，本文只是从代理通讯的本体级别考虑代理的通讯问题。但是实际上，多代理通讯至少包含本体级通讯、功能级通讯以及代理通讯语言 3 种层次的研究[20, 21]。我们只考虑第一种类型的通讯问题，从理论上提供了一种基于本体的代理通讯解决机制，并加以实现。

参 考 文 献

1 Baader F, et al. Rewriting concepts using terminologies. In: Proceedins of KR2000, Morgan Kaufmann, 2000
2 Uschold M. Barriers to effective agent communications. In: Proceedings of Ontologies in Agent Systems (OAS01), 2001
3 Steels L. The origins of ontologies and conmmunication conventions in multi-agent systems. Autonomous Agents and Multi-agent Systems, 1998(1):169～194
4 Hendler J. Agents and the Semantic Web. IEEE Intelligent Systems, 2001,16(2):30～37
5 Neches R, Fikes R E, Gruber T R, etal. Enabling Technology for Knowledge Sharing. AI Magazine, 1991,12(3):36～56
6 Studer R, Benjamins V R, Fense l D. Knowledge Engineering, Principlesand Methods. Data and Knowledge Engineering, 1998, 25(122):161～197
7 William S, Austin T. Ontologies. IEEE Intelligent Systems, 1999 Jan/Feb: 18～19
8 Chandrasekaran B, Josephson J R, Benjamins V R. What Are Ontologies, and Why Do We Need Them? 1999 Jan/Feb: 20～25
9 KMG. Using Protégé-2000 to Edit RDF: [Technical Report]. Stanford University, January 2001. http://protege. stanford. edu/protege-rdf/protege-rdf. html
10 Bechhofer S, et al. OilEd: a Reason-able Ontology Editor for the Semantic Web. In: Proceedings of KI2001, Joint German/Austrian conference on Artificial Intelligence, Vienna. Springer-Verlag LNAI, Vol 2174, 2001. 396～408
11 Sure Y, et al. 2002. OntoEdit: Collaborative Ontology Development for the Semantic Web. International Semantic Web Conference (ISWC02), LNCS, Vol. 2342, Sardinia, Italy, June 2002. 221～235
12 Hewlett-Packard Semantic Web Lab. Jena Semantic Web Framework. 2004. http://www. hpl. hp. com/semweb/
13 Ma Yinglong, Wu Kehe, Jin Beihong, et al. Approximate Semantic Query Based on Multi-Agent Systems. In: 2006 International Conference on Rough Sets and Knowledge Technology (RSKT2006), Lecture Notes on Artificial Intelligence, vol 4062, 2006
14 Selman B, Kautz H. Knowledge compilation and theory approximation. Journal of ACM, 1996,43(2):193～224
15 Stuchenschmidt H. Approximate information filtering with multiple classification hierarchies. International Journal of Computational Intelligence and Applications, 2002, 2(3):295～302
16 Stuchenschmidt H. Approximate terminological queries. In: Proceedings of FQAS02, 2002
17 Akahani J, et al. Approximate query reformulateion for ontology integration. In: Proceedings of ICWS2003
18 Schaerf M, Cadoli M. Tractable reasoning via approximation. Artificial Intelligence, 1995,74(2):249～310
19 Stuckenschimdt H. Exploiting Partially Shared Ontologies for Multi-Agent Communication. In: Proceedings of CIA'02, 2002
20 Payne R T, Paolucci M, Singh R, et al. Communicating Agents in Open Multi Agent Systems. In: First GSFC/JPL Workshop on Radical Agent Concepts (WRAC'02), 2002
21 Sycara K, et al. LARKS: Dynamic Mathmaking Among Heterogeneous Software Agents in Cyberspace. Autonomous Agents and multiagent Systems, 2002,5:173～203

(上接第 151 页)

4 Sycara K, Lu J, Klusch M, et al. Matchmaking among Heterogeneous Agents on the Internet. In: Proceedings of AAAI Spring Symposium on Intelligent Agents in Cyberspace, Stanford, USA, 1999
5 Hannebauer M. Autonomous Dynamic Reconfiguration in Multi-Agent Systems: Improving the Quality and Efficiency of Collaborative Problem Solving. LNAI 2427, Springer, 2002
6 De Palma N, Bellissard L, Riveill M. Dynamic Reconfiguration of Agent-Based Applications. In: Third European Research Seminar on Advances in Distributed Systems, Madeira Island (Portugal), 1999
7 Van der Hoek W, Wooldridge M. On the Dynamics of Delegation, Cooperation and Control: A Logical Account. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems, Utrecht, the Netherlands, July 2005
8 Klusch M, Gerber A. Issues of Dynamic Coalition Formation Among Rational Agents on the Internet. In: Proc. 2nd International Conference on Knowledge Systems for Coalition Operations (KSCO), Toulouse, France, 2002
9 Lermann K, Shehory O. Coalition Formation for Large Scale Electronic Markets. In: Proceedings of the Fourth International Conference on MultiAgent Systems, IEEE Computer Society, Boston, 2000. 167～174
10 Griffiths N, Luck M. Coalition Formation Through Motivation and Trust. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, Melboume, Australia, 2003
11 Wooldridge M, Jennings N. The Cooperative Problem Solving Process. Journal of Logic and Computation, 1999,9 (4): 563～592
12 Kraus S, Shehory O, Taase G. Coalition Formation with Uncertain Heterogeneous Information. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems, Melbourne, Australia, 2003
13 Lerman K, Shehory O. Coalition Formation for Large Scale Electronic Markets. In: Proceedings of the Fourth International Conference on MultiAgent Systems, Boston, 2000. 216～222
14 Liu J, Jin X, Tsui K. Autonomic Oriented Computing: From Problem Solving to Complex Systems Modeling. Springer, 2005
15 Liu J, Jin X, Tsui K. Autonomy Oriented Computing (AOC): Formulating Computational systems with Autonomous Components. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 2005,35(6):879～902
16 Liu J, Han J, Tang Y Y. Multi-agent oriented constraint satisfaction. Artificial Intelligence, 2002,136(1):101～144
17 陶丽，张自力. 一种面向自治计算的启发式联盟形成算法. 已投计算机学报
18 Liu C, Foster I. A Constraint Language Approach to Matchmaking. In: Proceedings of the 14th International Workshop on Research Issues on Data Engineering (RIDE 2004), Boston, 2004
19 Liu J, Tang Y Y, Cao Y. An Evolutionary Autonomous Agents Approach to Image Feature Extraction. IEEE Transactions on Evolutionary Computation, 1997,1(2):141～158
20 Tsui K, Liu J. Evolutionary Diffusion Optimization. In: Proceedings of the 2002 Congress on Evolutionary Computation, Honolulu, Hawaii, 2002
21 Russell S, Norvig P. Artificial Intelligence: A Modern Approach (2nd edition). Prentice Hall, 2002
22 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure (2nd ed.), Morgan Kaufmann, 2004
23 Zhang Z, Zhang C, Zhang S. An Agent-Based Hybrid Framework for Database Mining. Applied Artificial Intelligence, 2003, 17(5-6):383～398
24 Zhang Z, Zhang C. Agent-Based Hybrid Intelligent Systems: An Agent-Based Framework for Complex Problem Solving. LNAI 2938, Springer, 2004