摘 要

随着Web信息的爆炸式增长，如何构建Web信息集成系统来有效地组织和管理分布于世界各地海量的Web数据，从中有效的获取有用信息，成为人们最关注的问题。本文在深入分析和讨论Web信息集成系统的研究现状和发展趋势基础之上，提出了一个基于本体和服务发现的Web信息集成系统，并围绕系统中的几个主要关键技术，如：在领域本体构建和数据源的服务封装和服务描述以及基于服务发现的查询分解技术等方面进行了研究，其主要内容如下：

1）改进了传统的虚拟集成方法，在Mediator和数据源之间增加了一个Web服务库层，包含了用Web服务技术封装的各数据源包装器的服务，并采用语义Web服务本体描述语言(OWL-S)对各数据源服务进行语义描述，形成了中介器和数据源的松散耦合结构，使得数据源访问更具透明性。

2）在Mediator实现上，系统采用了带语义的动态服务发现机制实现LAV模式，满足了Web信息集成系统对数据源的动态扩展需求；在查询分解方面提出了一种语义匹配与选择算法，它采用语义相似度的计算，实现服务选择，完成查询分解。此外，系统使用基于领域本体构建全局和局部视图的策略，避免领域中概念的语义冲突。

3) 设计和实现了集成系统的领域本体、基于领域本体的数据源服务包装与语义描述和Mediator模块，包括查询处理、服务发现与选择策略等模块。并介绍了主要的本体结构和查询分解策略的实现情况。

关键词：Web信息集成　本体　Web服务　服务发现　语义匹配

Abstract

With the rapid development of the web information, how to construct the Web Information Integration System(WIIS) to organize and manage such distributed Web information effectively, and to find out useful information efficiently, is becoming a challenging but exciting research topic.

This dissertation presented a new web information integrated system, based on the overview of current research and implementation instances for WIIS. This dissertation focuses on several problems, which are: how to build effective domain ontology, how to wrap data source (such as: web service, service description) well, and how to explore the high performance for resolving query decomposition based on web services discovery. The main work of this dissertation is as follows:

First,add a services layer between Mediator and Wrapper,which includes all the various data source services which wrapped as web services. The Semantic Web Ontology Language (OWL-S) is used to describe the data source services, which forms the loose coupling architecture of the mediator and data source and makes the visit more transparent.

Second, on the realization of the Mediator, the system adopts the LAV model with a semantic dynamic service discovering mechanism. It is easier to support the extending of the dynamic data sources; Decomposition made inquiries in connection with a semantic matching algorithm--Semantic Match and Choice algorithm using the semantic similarity, Inquiries option to achieve complete decomposition. Furthermore, because global view and local view are all constructed based on domain-based Ontology, the conflict of semantic field be avoided.

Third, a domain-based Ontology, Ontology-based wrapping service and semantic description of the data sources and mediator modules, including query processing, service discovery and selection strategy are designed and implemented. Then a brief introduction on the main structure of Ontology and the achievement of decomposition algorithm is given by this dissertation.

Keywords: Web Information Integration Ontology Web Service Service Discovery Semantic Matching

创新性声明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果；也不包含为获得西安电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中做了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名：吴振锋　　日期2007年3月14日

关于论文使用授权的说明

本人完全了解西安电子科技大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属西安电子科技大学。本人保证毕业离校后，发表论文或使用论文工作成果时署名单位仍然为西安电子科技大学。学校有权保留送交论文的复印件，允许查阅和借阅论文；学校可以公布论文的全部或部分内容，可以允许采用影印、缩印或其它复制手段保存论文。（保密的论文在解密后遵守此规定）

本学位论文属于保密在___年解密后适用本授权书。

本人签名：吴振锋　　日期2007年3月14日

导师签名：　　日期2007年3月14日

第一章 绪论

1.1 研究背景

如今网络已成为人们进行信息传递和共享的一种重要工具，Internet上海量的Web信息资源成为了全球最大的知识仓库，网络技术作为一种新的环境资源为新技术开辟了新的领域——Web信息集成。Web信息集成的目标是将不相容的、不同模式的分布式Web信息源的数据进行有效的集成。用户可通过统一的访问模式透明地对分布式信息源进行访问以完成对信息搜索，而无需了解具体的分布式信息源的信息结构和访问接口。如何构建集成的Web信息系统来有效地组织和管理分布于世界各地的海量数据，提高信息获得速度并合理、高效地利用这一巨大的信息资源已经成为信息集成研究领域的一个热点问题。

传统的信息集成数据源多集中在异构数据库、异构多文档等方面的集成，其数据源有数据模式相对稳定，数据源的数量变化不大的特点。如今，随着信息源分布的越来越广泛，Internet上的各站点信息已经成为一种重要的信息源，然而，当前Web信息来源多种多样，有来自HTML网页、电子邮件、电子表格、文本文件及语音邮件等等，与传统数据源相比这些Web数据源具有自己独特的特点：首先，Web数据源分部广泛，其数量成爆炸式增长；其次，数据源的建立、运行与维护相互独立；再次，Web数据源中的数据属于非结构化或半结构化数据，每个数据源都有自己的数据模式，各个数据源之间缺乏一个统一的语义集；最后，Web数据更新和变化都非常频繁。

正是由于Web数据源所具有以上几个特点，而且内容和表现方式也在动态变化，结果使得Web数据处于杂乱无序的状态，数据集成性很差，给建立Web数据集成系统带来了很多挑战性的问题。现阶段，Web信息集成引起了众多研究者的兴趣，进行了大量的工作并取得了一定的成果，但是该领域仍然处于初级阶段，有很多关键的地方存在很多的困难。

1.2 研究内容

我们针对Web信息集成所涉及的各个方面，尤其是其集成的对象——Web信息源服务的语义描述以及集成中的查询分解技术进行了有效的研究。构建了一个基于本体的Web信息集成框架，其主要的研究工作有以下几点：

1) Ontology(本体)在Web信息集成中的应用，领域本体的建立，以及基于领域本体的Web信息集成系统的结构。

2) 数据源的Web服务封装，以及基于语义的服务描述方法。

3) 在相应的中间层中，在领域本体指导下采用服务匹配和选择策略解决查询分解等技术。

4) 采用上述技术实现了一个Web信息集成的原型系统，并通过实验对上述技术进行了验证。

1.3 论文结构

全文共分为七章，各章内容如下：

第一章简要介绍本论文的研究背景、研究目的和意义以及论文的主要工作和论文的章节组织。

第二章首先详细论述了Web信息集成的原理、内涵、主要方法、国内外研究现状；然后给出了集成相关的本体技术，涉及本体概念、分类、特点以及本体在信息集成中的应用；最后简要说明了另一个和集成相关的技术：Web服务的相关内容。

第三章提出一个基于本体和服务的Web信息集成系统框架，并详细说明了框架体系结构、结构中的各模块功能以及系统的设计特点最后给出了一个系统的查询处理流程。

第四章主要介绍此Web信息集成系统中一个主要的问题——领域本体的构建以及相关的内容。包括领域本体构建的元语、原则、构建方法和步骤以及使用的描述语言OWL、语言实例和构建工具Protégé的介绍。

第五章介绍此Web信息集成系统中另一个主要的问题——查询分解问题。主要内容有：本系统采用的查询分解方法的介绍，即带语义的服务发现技术，基于领域本体的Web信息源描述方法的详细说明，以及如何使用领域本体的语义相似度匹配实现服务发现和选择等内容。

第六章实现该Web信息集成技术的原型系统，并通过实例对所采用的方法进行说明和验证。

第七章总结与展望。总结所做的工作，展望未来的工作和下一步的研究。

第二章 相关研究内容及现状

2.1 Web信息集成

2.1.1 Web信息集成概述

Web信息集成就是在Web环境中实现不同(在横向或纵向上存在着差异)数据源之间的信息交互，并能够从这些不同的数据源中有效获取信息并加以融合，以支持对Web上多个信息源的统一查询的数据库技术。Web信息集成为全局应用和为用户提供统一、透明地访问一组已存在的自治、分布和异构数据源的方法，集成的数据源包括独立站点数据、XML数据集、文本数据集等结构化和非结构化信息。Web信息集成需要处理大量的、数目不定的源。如何在Web上各种各样存在形式的数据源中准确的选择合适的数据源并从中高效的找到查询需要的结果是Web信息集成的主要任务，它已成为数据库研究中的一个重要研究领域。

Web信息集成技术应用领域非常广泛，如电子商务，网络搜索，企业应用整合等。随着互联网的发展和各个机构信息化的完善，许多合作密切的企业或组织的桌面应用也会转变为Web整合的应用，这也将促使对Web信息集成需求的增长。

2.1.2 Web信息集成方法

Web信息集成系统具有多层体系结构(一般包括: 用户接口层，数据源层，中间层)，根据中间层的实现方法不同，信息集成系统可分为物化(materialized)集成和虚拟(virtual)集成两种[1]。

(1)物化集成技术

物化集成的典型代表是数据仓库方法。该方法将各信息源中的数据利用抽取器全部抽取出来，然后再用包装器、合成器将抽取的数据合成为一个全局模式，集中存储到数据仓库中，系统针对数据仓库进行数据维护和处理。该方法优点是对不同类型的数据源可以采用统一的管理，管理方式简单方便；缺点是数据仓库不允许更新，只能定期重建，并且存在例如数据类型不支持、数据结构无法映射、转换接口工作量大、系统网络传输和处理瓶颈等问题。只适合于数据刷新频率不高且集成规模不大的以决策分析和数据挖掘等为主要应用的系统中。该类系统的结构见图2.1。

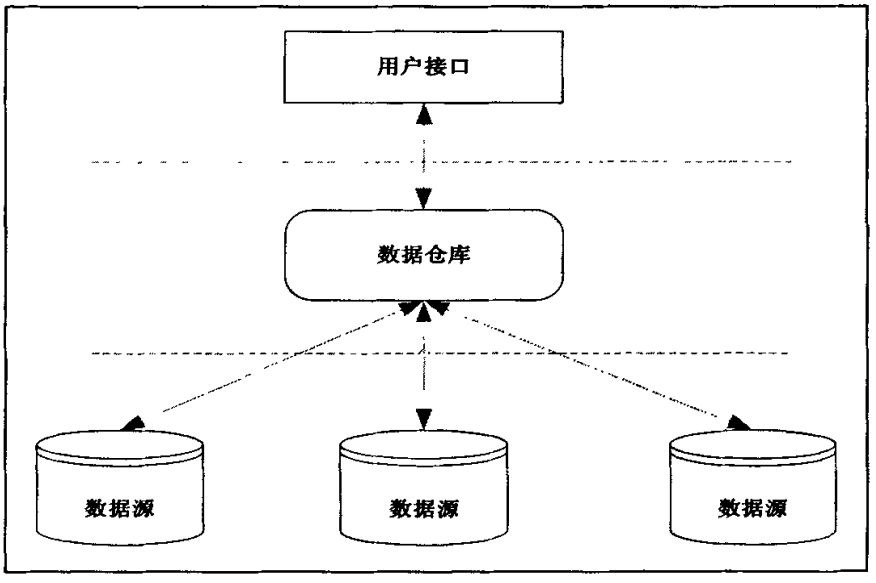

图2.1 数据仓库方法

(2)虚拟集成技术

在用户需要获得及时的查询结果，且数据源中的数据更新频繁的情况下，物化集成技术将不再是好的选择。这时候我们选择虚拟集成方法。虚拟集成方法又称Mediator/Wrapper方法。该类系统的结构见图2.2。

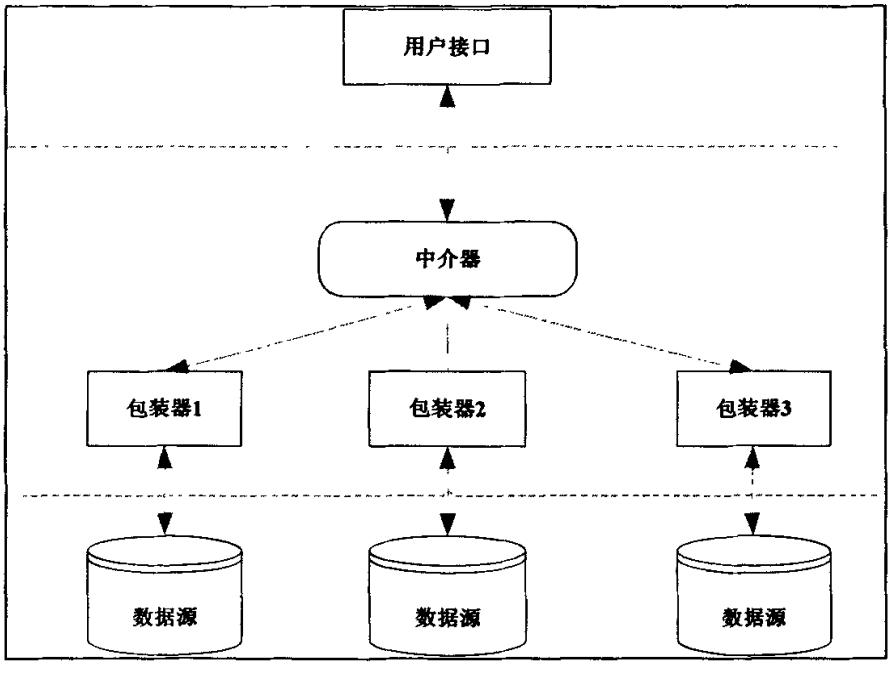

图2.2 Mediator/Wrapper方法

一般来说，在虚拟集成方法构建的信息集成系统中主要有中介器(Mediator)和包装器(Wrapper)两部分，其中包装器对特定数据源进行封装，将其数据模型转换为系统所采用的通用模型，作为其输出模式，并提供一致的物理访问机制。中介器不存储任何数据，而是采用支持虚拟视图的方式进行数据源操作。中介器的核心是全局查询处理、分解和优化，即如何把用户的查询分派成各数据源自身能接受的查询。中介器有一个使用通用模型描述的全局模式，它通过调用包装器或其它中间层来集成数据源中的信息，解决数据冗余和不一致性，提供一致协调的数据视图和统一的查询语言。通过在中介器和包装器之间分割处理任务，可以提高查询处理的并发性，减少响应时间。包装器既可与中间层处于同一位置，也可与数据源处于同一位置，这取决于系统的性能要求、数据源的归属关系及其访问控制权限。

虚拟集成系统有下列一些特点：

1）用户的查询提交给中间层。中间层具有一个其所集成数据源的全局视图，该视图叫做中间模式(Mediator Schema)，由一组虚拟的关系组成，其对应于下属数据源中的数据信息。中间层将用户的查询分解成对多个数据源的查询，并合并所有数据源提供的结果，形成最终的查询结果返回给用户。另外，为了进行查询分解中间层必须包含一组信息源模型，每个信息源模型描述了该信息源的内容、属性、内容完备性约束、可信度以及查询处理能力，查询分解的过程要以信息源的上述一系列属性为根本依据。

2）中间层不直接与数据源进行联系，它直接向Wrapper发送查询请求；Wrapper执行收到的查询，并将结果返回给中间层。

3）由于采用Mediator/Wrapper结构，这种信息集成系统不仅能够集成结构化数据源如关系型数据库、面向对象数据库等，而且可以集成半结构化数据如HTML，XML和无结构的数据如纯文本文件等。

虚拟集成技术有以下优点：

1）能够集成不同访问模式的数据源。简化了不同类型数据之间的统一存储，实现各种数据源的高度自治，便于扩展集成系统。

2）可以利用系统中现有的先进方法进行数据处理，优化了系统的性能。

3）支持用户的实时访问，并可以针对不同的用户提供不同的中间模式。

目前，很多的信息集成系统都采用了Mediator/Wrapper体系结构，国内外研究开发的基于Mediator/Wrapper的信息集成系统很多，例如：IBM阿尔马登研究中心与Stanford大学共同研究开发的TSIMMIS系统[3]和InfoMaster系统[4]，AT&T实验研究所研制出的Information Manifold(IM)系统[5]，南加利福尼亚大学的Craig Knoblock和Steve Minton等人设计的SIMS信息系统[6]以及在基于SIMS的面向互联网的WEB

信息集成系统Adriane[7]。国内的有东南大学研制的Versatile系统以及Galaxy系统等。

上述两种方法各具优势，数据仓库方法将被集成数据源的数据提取到本地存储，具有查询速度快、效率高、实现简单等特点，该方法对于命中的查询效率要比虚拟方法更高，但是不适用于动态变化强烈的数据源的集成；而虚拟方法更适应于数据源数目多、各局部数据源的自治性很高且局部数据经常变化的Web环境。

由于基于Mediator/Wrapper的虚拟集成技术可以集成范围广泛的数据源，且具有高度模块化和分布性，实现灵活、重用性、扩展性强等优点，因此在Web信息集成系统中通常采用虚拟集成技术。

2.1.3 国内外研究现状

Web信息集成是近些年才兴起的一个研究领域，在此之前，解决异构数据库交互的信息集成技术已有20多年的历史了。自1994年Stanford大学提出集成和共享Internet/Web上分布式信息源的Mediator/Wrapper架构以来研究者在分布式信息集成技术方面开展了大量研究工作，取得了许多研究成果。国际上重要的数据管理学术会议如VLDB，SIGMOD，ICDE等都会经常有关于集成技术研究的会议；此外CIKM，MAIM，WISE，WIDM等知名会议也特别关注Web信息和数据管理的研究，重要的杂志有ACM的TKDE等。国外已有许多工作组，从事数据集成及Web信息集成领域的研究。比如University of Washington的Alon Halevy等人的工作组，Stanford的Jefrey Ullman领导的实验室；G.Ashish和H.Venky等人提出虚拟数据库(Virtual Database，简称VDB)技术等。他们提出了许多好的Web信息集成和数据集成方法，为该领域的发展奠定了一定的基础。在国内，目前已有相当一部分人员从事Web数据管理，尤其是Web信息集成方面的研究：清华大学的周立柱，复旦大学的施伯乐，中国人民大学的孟小峰等。此外，国内比较重要的会议：“全国数据库学术会议”，重要的杂志：软件学报、计算机学报、计算机研究与发展等也都关注着Web信息集成的发展。

2.2 本体技术

2.2.1 本体基本概念

本体是哲学概念，它是研究存在的本质的哲学问题，即研究“世界的本原”。它所要回答的问题是客观世界的本质是什么。本体具有两个特性：静态性和动态性。静态性指它反映的是概念模型，没有涉及动态的行为。动态性指它的内容和服务

对象是不断变化的，针对不同的领域，可以定义和构造不同的本体。近几年来，这个词被应用到计算机界，并在人工智能、计算机语言以及数据库理论中扮演着越来越重要的作用。然而，到目前为止，对于此概念，还没有统一的定义和固定的应用领域。在知识工程领域，对本体概念的理解先后出现了以下的几个代表性定义。

1）Neches等人在1991年首先指出："一个本体定义了组成主题领域的词汇的基本术语和关系，以及用于组合术语和关系以定义词汇外延的规则"[10]。

2）Gruber于1993年指出："本体是概念化(conceptualization)的一个显式的(explicit)规范说明或表示"[11]。

3）Guarino和Giaretta于1995年给出了如下定义，即"本体是概念化的某些方面的一个显式的规范说明或表示"[14]。

4）Borst于1997年给出了一个类似的定义"本体可定义为被共享的概念化的一个形式的规范说明"[15]。

5）William在1999给出定义：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[16]。

从以上定义我们可以知道，本体通过对于概念、术语及其相互关系的规范化描述，勾画出某一领域的基本知识体系和共享概念模型。这包含4层含义：概念模型(Conceptualization)，明确(Explicit)，形式化(Formal)和共享(Share)。

1）概念模型指通过抽象出客观世界中一些现象的相关概念而得到的模型。概念模型所表现的含义独立于具体的环境状态。

2）明确指所使用的概念及使用这些概念的约束都有明确的定义。

3）形式化指Ontology是计算机可读的(即能被计算机处理)。

4）共享指Ontology中体现的是共同认可的知识，反映的是相关领域中公认的概念集，即Ontology针对的是团体而非个体的共识。

Ontology的目标是捕获相关领域的知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇(术语)和词汇间相互关系的明确定义。目前，普遍认为一个本体包括以下五种元素[18]：类、关系、函数、公理和实例。

(1)类除了一般意义上的概念外，还可以是任务、功能、行为、策略、推理过程等。本体中的这些类通常构成一个分类层次。

(2)关系表示类之间的关联，一般情况下用$R:C_1\times C_2\times...\times C_n$表示类$C_1$，$C_2$，..，$C_n$之间存在n元关系R。

(3)函数是一种特殊的关系，其中第n个元素相对于前面n-1个元素是唯一的。一般情况下，函数用$F:C_1\times C_2\times...\times C_{n-1}\rightarrow C_n$表示。

(4)公理表示一些永真式，用于说明函数之间或关联之间存在的关联或约束。
(5)实例是指属于某个类的个体。

2.2.2 本体分类

由于本体的分类方法比较杂，研究本体的机构和组织也很多，目前还没有能够被广泛接受的分类标准，但是根据本体不同方面的属性(如形式化程度、目的和描述对象等)，可以对本体进行初步的分类[20]。

根据本体的形式化程度不同，可以把本体分为高度非形式化的(highly informal)、结构非形式化的(structured-informal)、半形式化的(semi-formal)和严格形式化的(rigorously formal)。

按照描述或刻画建模对象的详细程度可将本体分为参考本体和共享本体，详细程度高的称作参考本体(reference ontology)，详细程度低的称作共享本体(share ontology)。

下面介绍一种典型的分类方式：根据本体对领域的依赖程度不同可以把本体细分为顶级(top-level)、领域(domain)、任务(task)和应用(application)本体等4类。其中：

1）顶级本体：

描述的是最普通的概念及概念之间的关系，如空间、时间、状态、事件、过程、行为、部件等等，与具体的应用无关，其他种类的本体都是该类本体的特例。因此顶级本体定义的概念可以跨越几个领域通用。

2）领域本体：

描述的是特定领域(如金融、医药、地理等)中的概念及概念之间的关系。它针对特定的应用领域知识的结构和内容，包括各种领域知识的类型、术语和概念，同时对领域知识的结构和内容加以约束，形成描述特定领域中具体知识的基础。

3）任务本体：

任务本体描述的是特定任务或行为中的概念及概念之间的关系。领域本体和任务本体通过特殊化顶级本体来描述一般领域、任务或活动中的概念。

4）应用本体：

描述的是依赖于特定领域和任务的概念及概念之间的关系。通常，应用本体是一种概念的混合，这些概念来自领域本体和顶级本体，然而，应用本体可能包含特定方法和特定任务的扩展，是领域本体和任务本体的特殊化。

图2. 3表示了它们之间的层次关系：

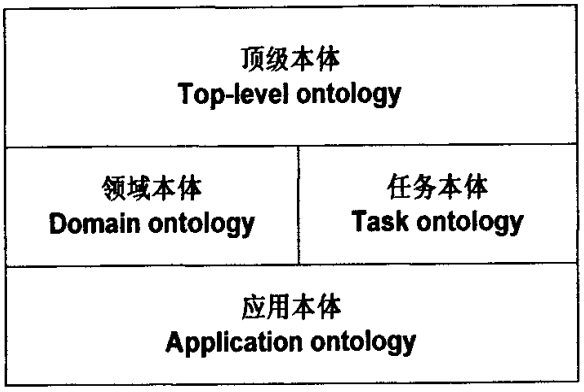

图2.3 本体层次关系

2.2.3 本体的特点

本体有以下突出的特点：

(1)本体可以在不同的建模方法、范式、语言和软件工具之间进行翻译和映射，以实现不同系统之间的互操作和继承。

(2)从功能上来讲，本体和数据库有些相似。但是本体比数据库表达的知识丰富得多。首先，定义本体的语言，在词法和语义上都比数据库所能表示的信息丰富得多；最重要的，本体提供的是一个领域严谨丰富的理论，而不单单是一个存放数据的结构。

(3)本体是领域内重要实体、属性、过程及其相互关系形式化描述的基础。这种形式化的描述可成为软件系统中可重用和共享的组件。

(4)本体可以为知识库的构建提供一个基本的结构。以描述对象的类型而言：有简单事实及抽象概念，这些可以描述成一个本体的静态实体部分，它们主要描述的是事物或概念的各个组成部分以及这些组成部分之间的静态联系；本体也可以描述事物或概念的运动和变化。应用本体，知识库就可以运用这类结构去表达现实世界中浩如烟海的知识和常识。

(5)对于知识管理系统来说，本体就是一个正式的词汇表。本体可以将对象知识的概念和相互间的关系进行较为精确的定义。在这样一系列概念的支持下进行知识搜索、知识积累、知识共享的效率将大大提高，真正意义上的知识重用和知识共享也能成为现实。

(6)本体适合表示抽象的描述，而领域模型是人们对领域内概念的抽象描述，因此在领域逻辑建模中，本体的使用可以帮助我们清楚地理解特定领域的相关元素、关系和概念，让知识表达更加准确便捷，帮助人们进行更好的各种处理。

2.2.4 本体在信息集成中的应用

当前的计算机正在从单一的设备向进行信息交换和事务处理的世界范围网络转变。如今，支持数据、信息和知识的交换、重用和共享成了当今计算机技术要迫切面临的任务。由于本体有以上六大特点，所以，目前，Ontology已成功地运用于知识工程，知识表示，知识标准化，自然语言处理，数据库设计，信息检索与抽取，智能信息集成和知识管理，电子商务等领域。

在信息集成方面，本体常用于将某个或多个特定领域的概念和术语规范化(比如软件领域，医学领域，机械工程领域)，为异构数据源数据集成提供统一的概念和术语标准，并且本体可以为其在该领域或领域之间的实际应用提供便利。

基于本体的信息集成系统由于能提供查询和资源描述所必需的语义信息，并通过领域本体知识库为信息源提供必要的语义标注信息，从而使系统对领域内的概念、概念之间的联系及领域内的基本公理知识有一个统一的认识，减少了因为数据源采用不同命名造成的语义冲突，进一步提高了系统的联想能力和精确性，为用户提供更有价值的信息。

2.3 Web服务

2.3.1 Web服务简介

（一）定义：

Web服务是部署在Web上的、分布式的、自包含的模块化软件组件，它为其他应用程序提供功能，执行特定的任务，遵守一系列的技术规范，这些规范使得Web服务可以在网络中被描述、发布、查找和调用。本质上说，Web服务是一种革命性的分布式计算技术。它使用基于XML的消息处理作为基本的数据通信标准，消除了不同操作系统、编程语言、应用架构和不同组件模型之间存在的差异，解决了异构系统相互访问的问题，使各系统下的功能能够成为计算网络的一部分协同运行。

（二）Web服务技术的优点：

Web服务技术之所以成为解决当今分布式异构系统之间的集成及互操作问题的最佳解决方案，是由于这一技术具有一些传统技术所不具备的特点。同时这一技术也有其适用性。

(1)封装完好：Web服务是一种部署在Web上的对象，它具备对象的良好封装性。对于用户而言，他能且仅能看到该对象提供的功能信息、访问信息和Web服务提供的网络编程接口。

(2)松散耦合：Web服务技术的一个基本特点就是透明性，即当一个服务的实现内容或者实现方式发生变化之后，调用它的用户不会发觉其中的变化，即服务的内容对用户是透明的。松散藕合特性主要由相关协议来保证，XML/SOAP协议正是目前最为适合的消息交换协议。

(3)互操作性：基于接口定义语言和协作协议，任何Web服务都可以与其它Web服务进行交互，真正实现了平台和语言独立性。开发者可以使用任何语言来编写Web服务，无需更改他们的开发环境就可生产和使用Web服务。

(4)高度的可集成能力：由于Web服务采取简单的、易理解的标准Web协议作为组件界面描述和协同描述规范，完全屏蔽了不同软件平台的差异，无论是CORBA，DCOM还是EJB都可以通过这一种标准的协议进行互操作，实现了在当前环境下最高的可集成性。

2.3.2 Web服务模型

Web服务模型基于三种角色（服务提供者、服务注册中心和服务请求者）之间的交互，见图2.4。交互具体涉及到发布、查找和绑定等操作。服务提供者首先提供某种功能服务的实现模块，然后定义了详细的服务描述，最后把服务发布到服务请求者或服务注册中心。服务请求者使用查找操作从本地或服务注册中心搜索服务描述，然后使用服务描述与服务提供者进行绑定，并调用相应的服务实现。

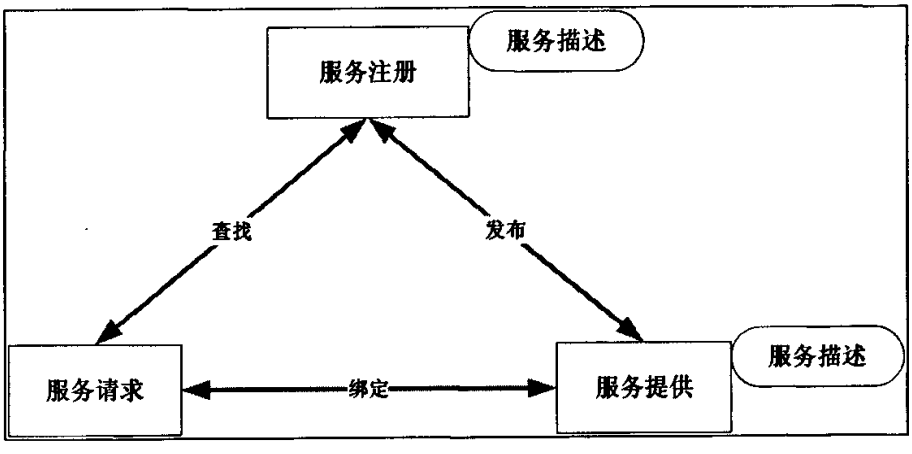

图2.4 Web服务模型

服务提供者：提供服务，并对服务进行注册以便于服务的利用。

服务请求者：向服务注册中心请求服务，并调用这些服务以完成具体的任务。

服务注册中心：这是可搜索的服务描述注册中心，服务提供者在此发布他们的服务描述。

在静态绑定开发或动态绑定执行期间，服务请求者查找服务并获得服务的绑定信息(在服务描述中)。对于静态绑定的服务请求者，服务注册中心是体系结构中的可选角色，因为服务提供者可以把描述直接发送给服务请求者。同样，服务请求者可以从服务注册中心以外的其它来源得到服务描述。

2.3.3 Web服务在信息集成中的应用

Web服务的设计初衷就是为了屏蔽异构系统的差别，异构的数据源可以使用Web服务对本数据源的数据服务进行封装，并发布服务到注册中心，供集成系统或用户调用。由于Web服务与生俱来的完好封装性、松散耦合性以及规范的协议、高度可集成能力等优点非常适合用于信息集成领域。因此，基于Web服务的信息集成方案已经成为构建Web信息集成系统较为理想的技术选择。

2.4 本章小结

本章一开始给出了Web信息集成的定义，详细的论述了Web信息集成的两种方法：物化集成方法和虚拟集成方法，说明了虚拟集成方法更适合用于Web信息集成，然后讨论了Web信息集成的国内外研究现状。接下来，我们介绍了本体的相关概念、分类和特点以及本体在信息集成中的应用。在本章最后，我们给出了Web服务的相关介绍，包括Web服务的概念、优点、模型以及Web服务在信息集成中的应用。在下一章节我们将介绍本论文提出的基于本体和服务的信息集成框架。

第三章 基于本体和服务发现的Web信息集成系统

3.1 体系结构

由于Mediator/Wrapper的集成方法(2.1.2节已经介绍)更适应于数据源数目多、各局部数据源的自治性很高且局部数据经常变化的Web环境，因此Web信息集成中主要采用此方法。

本论文提出的信息集成系统在传统的信息集成方法：Mediator/Wrapper方法上加以改进，增加了一个Web服务库层，包含了所有用Web服务技术[21]封装的各数据源包装器的服务，采用语义Web服务本体描述语言(OWL-S)对各数据源的抽取服务进行语义描述，形成了中介器和数据源的松散耦合结构，使得数据源访问更具透明性，同时也使系统满足了扩展性、维护性以及数据源数据更新的需求。在Mediator实现上，系统采用了带语义的动态服务发现机制实现LAV模式[22]，使系统非常容易地支持数据源的动态扩展。此外，本系统使用基于领域本体构建全局和局部视图的策略，避免了领域中概念的语义冲突。

系统包含领域本体、用户查询接口、中介器、Web服务库和包装器，其中中介器包含查询处理、服务发现与选择、服务调用、结果处理等功能；Web服务库，包括Web服务注册和服务调用等功能；包装器包含数据源包装器生成、Web服务生成等功能模块。体系结构如图3. 1所示。

3.1.1 领域本体

领域本体包含所有数据源的通用语义模型。该语义模型规定了待集成各数据源的数据对象的模式和数据的语义，从用户查询处理、全局数据视图的定义、各数据源局部数据视图的定义、数据源描述的生成到基于领域知识的信息检索与匹配，可以说本集成系统的各个环节都要以领域本体为指导。

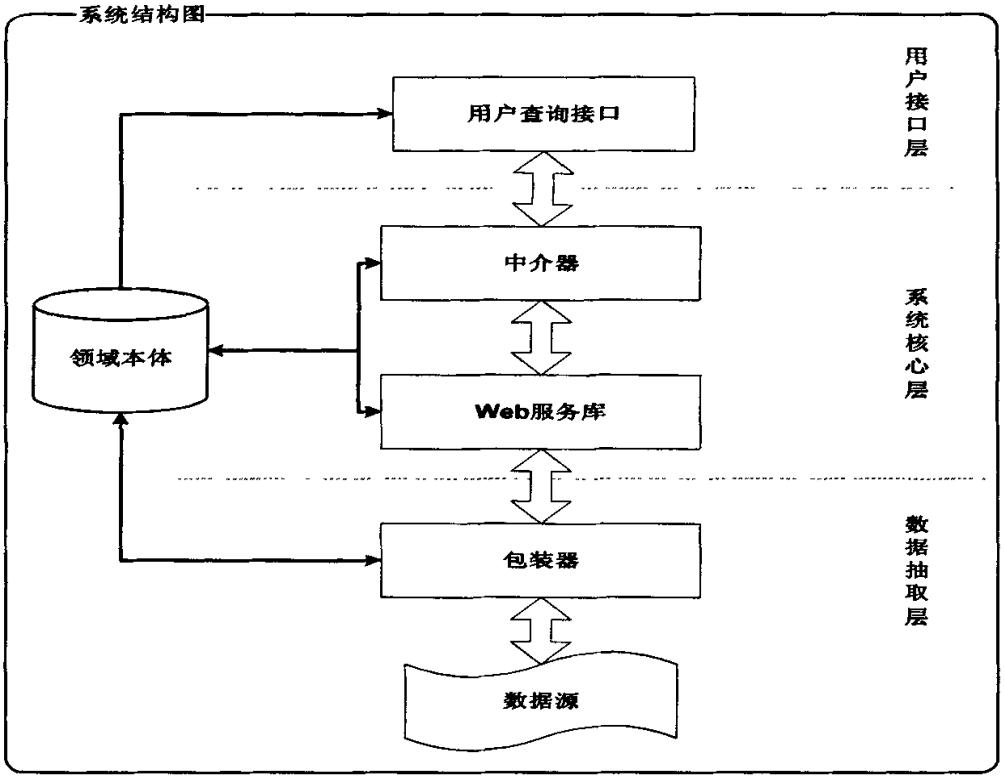

图3.1 系统结构图

3.1.2 用户查询接口

用户查询接口负责所有的用户查询的预处理。用户查询接口分析其收到的查询，依据领域本体信息分析查询条件，对用户的查询信息进行标准化和过滤，保留应用于查找的条件和约束信息，剔除与领域内容无关的查询条件，使形成新的用户查询描述。然后把处理过的查询描述转换成语义表示的查询条件。用户查询接口还接收中介器返回的查询结果，或直接提交用户，或做转换处理(比如使用XSLT把结果XML文件转换为其他所需格式)后提交给用户。

3.1.3 中介器

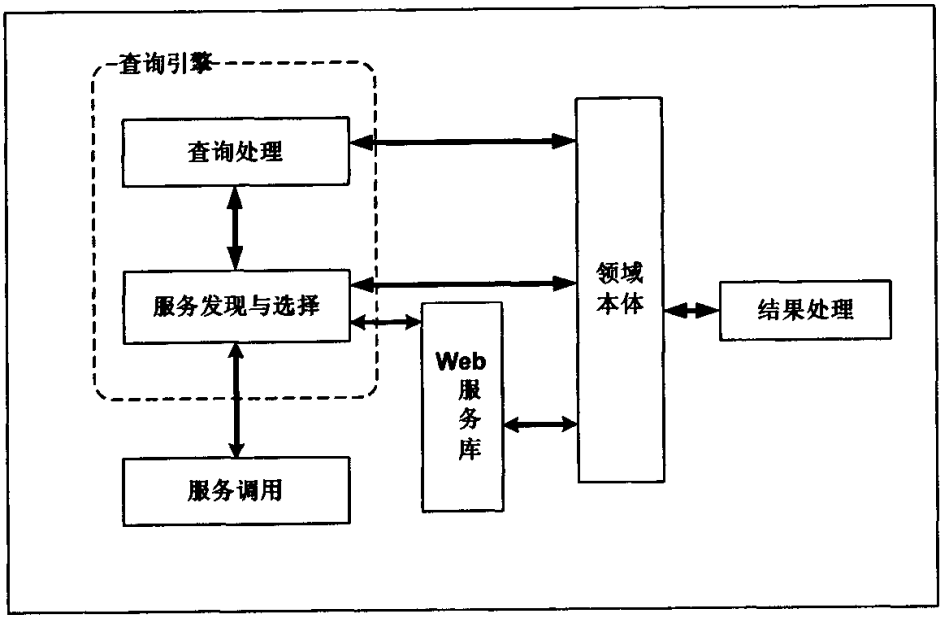

图3.2 中介器结构图

中介器是本系统的核心，中介器包含查询处理、服务发现与选择、服务调用、结果处理等模块，中介器结构图见图3.2，其中查询处理模块根据领域本体框架信息，分析用户查询接口发来的查询条件包含的概念及其依赖关系，得到相关本体的属性信息；服务发现与选择模块把Web服务库中的所有Web服务提供的语义信息与查询处理模块得到的语义条件进行语义匹配(匹配的语义包括服务的输入条件和输出信息的语义模型)，根据语义匹配算法得出匹配结果集，再从语义匹配结果集中按匹配度的高低进行排序，然后使用某种选择策略选择合适的服务；服务调用模块根据选取的各个服务所需的查询输入要求进行查询重写即把用户的查询条件传递给服务的输入条件，并通过服务绑定进行服务调用，完成所需信息的抽取工作。由于一个查询可能涉及到多个Web数据源，各个数据源之间的模式都不尽相同，所以抽取的结果也可能有多种模式的XML文档，因此，中介器还需要结果处理模块对这些结果集数据文档进行处理，生成一个完整的查询结果。

3.1.4 Web服务库

Web服务库包含有所有已注册的服务的详细信息，它向上提供库中已有的数据源数据抽取服务，向下封装数据源的包装器。一个Web服务对应一个数据源包装器，负责调用包装器抽取数据源上的某一类信息。注册时，Web服务使用带语义的描述

语言详细说明了被封装的数据源提供信息时的必要输入条件和其他限制信息，以及该数据源提供数据的模式，最后提供了如何访问该服务的方法。

3.1.5 包装器

包装器是根据一定的需求规则从特定的半结构的或无结构的数据源抽取数据并形成结构化数据的模块。它接收从中介器发来的查询条件，进行查询分析，采用特定的抽取方法[23]从数据源中按照所需结果的数据模式抽取合适的数据并返回给中间器。在抽取过程中，还要结合领域本体对抽取词汇进行修正，避免数据命名冲突。

为了方便数据源的动态加入和修改，系统采用包装器的Web服务封装技术[24]使系统具有完好封装、规范协议、高度可集成能力等特性。每个Web服务向外提供服务描述、查询的输入接口和输出数据的模式并把它发布到中介器的服务注册中供中介器选用。

3.2 设计特点

系统采用Web服务技术包装数据源，中介器与数据源是一种松散的耦合结构，系统充分利用了Web服务可以在分布式计算环境下被描述、发布、发掘和动态调用，并且不受平台和语言限制的优点。因此，新数据源只需在注册器注册成服务就可以融入系统，原有系统无需改动；已注册数据源发生变动只要提供服务的接口不变，数据源变更对系统是透明的。

系统采用独立的服务发现与选择模块来完成查询分解功能，因此在实践中可更换发现算法来提高发现的准确性，同时选择策略也属模块设计，留有接口，可以非常方便的更新和替换原有策略也可加入新策略来加强服务选择的正确性。

3.3 查询处理过程

在上述体系结构中，用户进行查询的主要过程如下：

a)对用户提交的查询条件进行预处理，得到合法的查询条件；

b)对查询条件进行查询处理，基于领域本体生成全局的查询视图；

c)采用语义的服务发现方法完成查询分解，并按选择策略选取完成此次查询任务的Web服务；

d)执行被选择了的服务得到各数据源中的数据；

e)对得到的结果进行合并，并将合并后的结果作为查询结果返回给用户。

经过上述过程，就可以实现用户只需根据领域本体就能够查询到不同数据源中感兴趣的信息，而不必关心查询结果是从哪个数据源中得到并经过怎样的处理。

在信息集成应用中，使用本文提出的体系结构必须解决三个主要问题：第一是如何构建领域本体；第二如何对数据源的封装服务进行语义的描述，第三是如何选取合适的服务完成查询分解。这些问题的解决我们将通过第四、五章详细说明。

3.4 本章小结

本章针对Web信息集成的目标，提出了一个基于本体和服务的Web信息集成框架，详细论述了框架中：领域本体、用户查询接口、中介器、Web服务库和包装器的设计，然后讨论了该系统的特点，最后给出了一个查询处理的过程。接下来的第四章将详细说明系统构建中的第一个主要问题——如何构建领域本体。

第四章 构建领域本体

领域本体在Web信息集成系统中扮演着重要的角色。本文的Web信息集成是以领域为背景进行的，它为整个系统建立了语义标准和知识标准。我们用领域本体定义该领域的知识框架。该知识框架规定了待集成的数据对象的模式和数据的语义，信息集成过程中的数据源描述、局部数据视图的定义、全局数据视图的定义到基于领域知识的查询分解、语义匹配等所有的关键环节都需要全局数据视图提供语义和知识的帮助，可以说构建领域本体是建立整个Web信息集成的最重要环节之一。

领域本体的构造是一个非常复杂的过程，除了必要的计算机技术之外，还需要具有相关领域的知识。本体的构建必须经过精心的设计。本节将介绍本体构建的建模元语、相关准则和方法。

4.1 构建元语及原则

4.1.1 构建元语

Perez等人采用分类法来组织本体[25]，归纳出本体5个基本的建模元语(Modeling Primitives)：

[1] 类(classes)或概念(concepts)

类含义广泛，可指任何事物，如工作描述、功能、行为、策略和推理过程等等。从语义上讲属于对象的集合的抽象表示，其定义一般采用框架(frame)结构，包括概念的名称，与其他概念之间的关系的集合，以及用自然语言对概念的描述。

[2] 关系(relations)

代表了领域中概念之间的交互作用，形式上定义为n维笛卡儿积的子集:R:$C_1 \times C_2 \times C_3 \times C_n$。如子类关系(subclass-of)，在语义上关系表示为对象元组的集合。本体关系中有五类：逆性质(Inverse)、传递性质(Transitivity)、对称性质(Symmetry)、关系的自反性(Reflexivity)、等价性(Equal)。

[3] 函数(functions)

一类特殊的关系。该关系的前n-1个元素可以唯一决定第n个元素。形式化的定义为F: $C_1 \times C_2 \times \cdots \times C_{n-1} -> C_n$。如Brother-of就是一个函数，Brother-of(x,y)表示x的兄弟是y，显然兄弟关系是个对称关系，y有个兄弟是x。

[4] 公理(axioms)

代表永真断言，比如概念乙属于概念甲的范围。

[5] 实例(instances)

代表元素。从语义上讲实例表示的就是对象，而概念表示的则是对象的集合，关系对应于对象元组的集合。

另外，从语义上讲，基本的关系共有4种：part-of，kind-of，instance-of和attribute-of。

Part-of：表示概念之间的部分与整体的关系；

kind-of：表示概念之间的继承关系，类似于面向对象中的类与子类之间的关系；

instance-of：表示概念的实例与概念之间的关系，类似于面向对象中的对象与类之间的关系；

attribute-of：表达某个概念是另外一个概念的属性。例如概念学历可以作为概念简历的一个属性。

在实际应用中，本体的构造也不一定要严格地按照上述5类元语。同时概念之间的关系也不一定仅限于上述的四种关系，可以根据特定领域的具体情况定义相应的关系。

4.1.2 构建原则

构建本体需要方法学的指导，以下是本体构建的5个主要原则：

- 清晰(Clarity):本体必须有效地说明所定义术语的意思。定义应该是客观的，与背景独立的。当定义可以用逻辑公理表达时，它应该是形式化的。定义应该尽可能的完整。所有定义应该用自然语言加以说明。
- 一致(Coherence)：本体应该是一致的，也就是说，它应该支持与其定义相一致的推理。它所定义的公理以及用自然语言进行说明的文档都应该具有一致性。
- 可扩展性(Extendibility)：本体应该为可预料到的任务提供概念基础。它应该可以支持在已有的概念基础上定义新的术语，以满足特殊的需求，而无须修改已有的概念定义。
- 编码偏好程度最小(Minimal encoding bias)：概念的描述不应该依赖于某一种特殊的符号层的表示方法。因为实际的系统可能采用不同的知识表示方法。
- 本体约定最小(Minimal ontological commitment)：本体约定应该最小，只要能够满足特定的知识共享需求即可。这可以通过定义约束最弱的公理以及只定义通讯所需的词汇来保证。

4.2 构建步骤与方法

4.2.1 构建步骤

总体上来说，构建一个领域本体的过程可分为领域分析、概念化和实现三个阶段。在领域分析阶段，主要是以文档的形式说明开发本体的目的和领域知识；在概念化阶段要确定领域概念以及概念之间的关系；在实现阶段主要使用形式化语言对概念化阶段产生的领域概念模型进行编码，使计算机能够理解和处理概念模型。概念化阶段作为本体建模的核心阶段，将对领域概念化分析的结果以适当的形式明确地记录下来。概念化的分析过程涉及到本体的四个基本性质，即同一性、完整性、严格性和依赖性。在具体分析过程中可以从这四个基本性质入手，澄清分类关系，识别层次结构，生成更为合理的本体。

领域本体建立的一般步骤是：

(一)明确建立本体模型的目的和使用范围

明确构建的本体涉及的专业领域、本体的目的和使用范围，以及本体的开发、维护和应用对象，这些因素和领域本体的建立过程有着很大的关系，所以应当在开发本体前加以明确。

(二)获取领域知识

针对该学科领域的内容和知识点，结合专家的理解和经验，如果该领域存在已建成的旧版本体，或者部分本体，则可以参考已有的本体，获得领域知识，并粗略地描述出领域知识之间的关系。

(三)建立本体框架

枚举出系统想要陈述的或想要向用户解释的全部概念。对于枚举出的概念按照一定的逻辑规则进行分类，形成不同的类别，使在同一类别的概念有较强的相关性。此外，对领域中的每一个概念进行重要性评估，选出关键性概念，摒弃不必要或者超出领域范围的概念，尽可能准确而精简，从而形成领域本体的框架结构。

(四) 细化概念和概念间的关系分析

由于主要使用类的形式来定义概念，所以概念和概念之间的关系主要体现在类和类之间的层次关系。首先，确定哪些属性是用来描述概念的。其次，通过基本概念，基本知识和基本技巧确立类之间的关系。第三是从基本概念，基本知识和基本技巧确定衍生的概念或知识。第四是，给出衍生的概念或知识的性质及其关系。最后，给出上述概念知识和关系的形式化描述，得出“概念—关系词典”，即领域层概念及其关系知识库。

(五)对领域本体编码实现

选用合适的本体描述语言对上述建立的领域本体进行编码实现。W3C组织已经提出了本体描述语言(OWL)的规范以支持本体建模。本体模型的形式化可以提供比自然语言更严格的格式以增强机器的可读性，便于本体模型自动进行逻辑推理及检验。

(六)本体的评估

本体的编码实现后，是否满足提出的需求，是否满足构造准则，本体的定义是否清晰完整，这些都需要我们在本体建立后进行进一步的评估。

需要说明的是，领域本体的建立是一个不断重复和修改的过程，没有哪一个时刻的本体是完善的，它的建立是逐步迭代完善的过程。作为领域知识的一种模型，领域本体需要不断的改进才能接近人对客观世界的认识。领域本体中的概念应该贴近于要研究的专业领域中的客观实体和关系法则。

4.2.2 构建方法

现阶段通常是借助领域专家对本行业的信息进行归纳和提取，以手工的方式构建。本体的手工构建是一项工作量巨大并且异常繁杂的任务，在本体的构建过程中一般使用以下方法：

（一）骨架法[26]

Mike Ushold & Micheal Gruninger的骨架法(Skeletal Methodology)该方法提供开发本体的指导方针。在建设过程中认为评价方法应该是其中的一个环节。包括如下步骤：

(1) 确定目的和范围：在此阶段需要弄清为什么要建立本体、建好后使用该本体的用户是谁，用途是什么。

(2) 建设本体：这一阶段主要包括三个步骤：本体捕获、本体编码以及本体的集成。在本体捕获阶段，主要任务是相关领域内的关键概念和概念之间关系的识别；对这些概念和关系的精确无二义的文本定义的生成；表达这些概念和关系的术语的识别。在本体编码阶段，主要是利用某种形式化语言显式地表现上一阶段的概念化结果。在本体集成阶段，指集成现有的本体成果。

(3) 评价：没有提出自己的评价方法，只是认为这应该是整个方法论的一个环节。

(4) 文档化：把本体中定义的主要概念、元本体等落实到文档中。

(5) 提炼出每个阶段的指导方针：指把设计本体的初始的指导方针总结为设计标准。

（二）METHONTOLOGY[27]

该方法是Mariano Fernandez & GOMEZ-PEREZ等人在马德里大学开发人工智能图书馆时使用的，它是一种建设本体的结构化方法。基本流程如下：

(1) 建立规格说明书(Specification)：产生一份以自然语言编写的非形式化的、半形式化的或者形式化的本体规格说明书。书中至少包含实现本体的形式化程度、范围。说明书要求简洁(每个术语都是相关的，没有无关或者重复的术语)、部分完整(包含术语的覆盖范围、每个术语的问题和粒度)、一致性(每个术语及含义都在领域内有意义)。

(2) 知识获取(Knowledge Acquisition)：知识的来源很多，可以是专家、书籍、手册、数字、表格、甚至是其他的本体。从这些数据源获得知识的关联技术包括头脑风暴法、访谈、文本的形式化/非形式化的分析以及一些知识获取工具。

(3) 概念化(Conceptualization)：将领域知识组织成概念模型，用规格说明书中识别的领域词汇表描述问题和解决方案。生成的概念模型允许最终用户确定一个本体是否有用，并且对于某个给定应用不需要查看源代码就是可用的；比较数个本体的范围、完整性、可重用性、共享性。

(4) 集成(Integration)：重用其他本体中已经建好的定义时，既可以查看源本体，选择适合自己概念模型的；也可以选择和自己概念模型中的语义和实现一致的术语定义。

(5) 实现(Implementation)：用任何一种形式化的语言编码实现本体。需要一套开发环境的支持，至少包括：词法和语法分析器、翻译器、编辑器、浏览器、搜索器、评价器、自动维护工具。

(6) 评价(Evaluation)：评价是指在本体生命周期的每个阶段和阶段之间，利用某种参考框架对本体、软件环境、文档进行技术判断。评价包括正确性(verification)和有效性(validation)。

(7) 文档化(Documentation)：指在本体建设的每个阶段都应该有对应的文档。

目前讨论的创建ontology的方法一般是基于手工的，是专家知识一种映射，按照特定的结构和形式显示地表示出来，可以借助于protégé，OntoEdit等辅助工具，这些辅助工具的研究对研究ontology的构建有非常重要的意义。同时，学者们还在探索自动和半自动的ontology的构建方法，即借助于基于统计学的语义分析方法和人工智能技术来自动创建ontology，如果有必要再用人工对ontology进行调整和完善。这种方法存在很大的难度，目前仅仅是在实验和探索阶段，还没有成熟的软件，已有的实验性系统效果也不能让人满意。

4.3 描述语言

Internet迅猛发展，带动了人们对Internet数据和元数据描述的研究，人们在不断积累经验的基础上推出了各种描述标准。W3C组织所推出的资源描述框架(Resource Description Framework，RDF)就是一个很好的例子，资源描述框架是W3C在XML的基础上制定的一套可描述知识概念、概念间的关系和实例的规范标准，用于表示任何的资源信息。RDF提出了一个简单的数据模型用来表示任意类型的数据，方便地描述对象(或者资源)以及它们之间关系。RDF的数据模型实质上是一种二元关系的表达，由于任何复杂的关系都可以分解为多个简单的二元关系，因此RDF的数据模型可以作为其他任何复杂关系模型的基础模型。然而，由于RDF本身并没有提供专门的机制来描述属性以及关系本身，因此，W3C组织又推出了资源描述框架定义集(Resource Description Framework Schema RDFS)，RDFS是描述RDF资源中属性和类的词汇表，并带有这些属性和类的泛化层次的语义，从而增强了RDF对资源的描述能力。后来，人们在这些描述语言中加入了逻辑推理的成分，推出了DAML和OIL，DAML和OIL继承了RDF(RDFS)的表达能力，还从描述逻辑中借鉴了很多表达方式，从而具备了较强的领域知识描述能力。最后，W3C组织以DAML+OIL为基础，在考察Web Ontology的需求之后，确立了OWL(Web Ontology Language)作为本体描述标准。相对于RDFS、DAML+OIL，OWL拥有更多的机制来表达语义。因此，基于OWL的知识表示成为了研究和应用的热点。以下我们重点介绍W3C组织提出的本体论Web语言OWL。

4.3.1 OWL语言介绍

OWL是本体论Web语言(Ontology Web Language)的字母缩写，其设计的最终目的是提供一种可以用于各种应用的语言，这些应用是需要处理信息的内容而不是仅向人类呈现信息的应用。OWL能够被用于清晰地表达词汇表中的词条的含义以及这些词条之间的关系，而这种对词条和它们之间的关系的表达就称作本体。OWL通过提供更多具有形式语义的词汇，使之在Web内容的机器可理解性方面要强于XML、RDF和RDF Schema(RDF-S)等所能达到的程度。OWL有三个表达能力递增的子语言：OWL Lite，OWL DL和OWL Full。

这三种表达能力递增的子语言，以分别用于特定的实现者和用户团体。

- OWL Lite[28]用于提供给那些只需要一个分类层次和简单约束的用户。例如，虽然 OWL Lite 支持基数限制，但只允许基数为 0 或 1。提供支持 OWL Lite 的工具应该比支持其他表达能力更强的 OWL 子语言更简单。相比 OWL DL，OWL Lite 还具有更低的形式复杂度。

- OWL DL 用于支持那些需要最强表达能力而又需要保持计算完备性(computational completeness，即所有的结论都能够确保被计算出来)和可判定性(decidability，即所有的计算都能在有限的时间内完成)。OWL DL 包括了OWL 语言的所有语言成分，但使用时必须符合一定的约束，例如，一个类可以是多个类的子类时，但它不能同时是另外一个类的实例。OWL DL 这么命名是因为它对应于描述逻辑，它是一个研究作为 OWL 形式基础的逻辑的研究领域。
- OWL Full 支持那些需要尽管没有可计算性保证，但有最强的表达能力和完全自由的 RDF 语法的用户。例如，在 OWL Full 中，一个类可以被同时看为许多个体的一个集合以及本身作为一个个体。它允许在一个本体增加预定义的(RDF OWL)词汇的含义。这样，不太可能有推理软件能支持对 OWL FULL 的所有成分的完全推理。

在表达能力和推理能力上，每个子语言都是前面的语言的扩展。这三种子语言之间有如下关系成立，但这些关系反过来并不成立。

- 每个合法的 OWL Lite 本体都是一个合法的 OWL DL 本体；
- 每个合法的 OWL DL 本体都是一个合法的 OWL Full 本体；
- 每个有效的 OWL Lite 结论都是一个有效的 OWL DL 结论；
- 每个有效的 OWL DL 结论都是一个有效的 OWL Full 结论。

使用OWL的本体开发，首先要考虑哪个子语言最符合需求。选择OWL Lite还是OWL DL主要取决于在多大程度上需要OWL DL提供的表达能力更强的成分。选择OWL DL还是OWL Full主要取决于在多大程度上需要RDF Schema的元建模(meta-modeling)机制(如定义关于类的类和为类赋予属性)；使用OWL Full相比于OWL DL，对推理的支持是更难预测的，因为目前还没有完全的OWL Full的实现。

OWL Full可以看成是对RDF的扩展，而 OWL Lite和OWL DL可以看成是对一个受限的RDF版本的扩展。所有的OWL文档(Lite，DL，Full)都是一个RDF文档；所有的RDF文档都是一个OWL Full文档，但只有一些RDF文档是一个合法的OWL Lite和OWL DL文档。因此，用户在把RDF文档转换到OWL文档时必须谨慎。当OWL DL或OWL Lite的表达能力认为是适当时，必须注意原来的RDF文档是否满足OWL DL或OWL Lite对RDF的一些附加的限制。其中，每个作为类名的URI必须明确地声明为类型为owl:Class (属性也类似)，每个个体必须声明为属于至少一个类(即使只有owl:Thing)，用于类、属性、个体的URI必须两两不相交。

4.3.2 OWL术语

此节将介绍OWL中常用的术语。rdf:和rdfs:前缀表示的术语是RDF和RDF Schema中的术语，其它的则是OWL引入的。

(1) OWL的类

Class (http://www.w3.org/TR/owl-guide/#owl_Class)：一个类定义了具有某些相同属性而属于同一群体的那些个体的集合。例如，Windows和Unix都同属于类SoftWare。多个类也可以用"子类"（subClassOf）关系组织为一个特定的层次结构。一个内置的类被称为Thing，它是所有个体的类，因此是所有OWL类的父类。下面用一个'软件'类的例子来说明OWL中类的定义：

```
<owl:Class rdf:ID="#SoftWare">
    <rdfs:comment >
        This is a 'SoftWare' class.
    </rdfs:comment >
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <owl:disjointWith rdf:resource="#HardWare"/>
</owl:Class>
```

<owl:Class>与</owl:Class>之间就是类的定义，用rdf:ID='SoftWare'来声明类的名字是'SoftWare'，它是Thing类的子类并且和HardWare类不相交。

(2) OWL的属性

一个属性是一个二元关系。有两种类型的属性：

一是数据类型Property，它是类实例与RDF文字或XML Schema数据类型间的关系称为owl:DatatypeProperty。二是对象Property，它是描述两个类的实例间的关系称为owl:ObjectProperty。

在我们定义一个属性的时候，可以通过指定property的domain和range以及定义subproperty来约束一个property。OWL中的属性有五种特征：传递属性(TransitiveProperty)、对称属性(SymmetricProperty)、函数属性(FunctionalProperty)、逆属性(Inverse0f)、逆函数属性(InverseFunctionalProperty)。下面是一个传递属性的例子：

```
<owl:TransitiveProperty rdf:ID="isBefore_version">
  <rdfs:domain rdf:resource="#SoftWare"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:inverseOf>
    <owl:TransitiveProperty rdf:about="# isAfter_version "/>
```

```
    </owl:inverseOf>
    <rdfs:range rdf:resource="# SoftWare "/>
  </owl:TransitiveProperty>
```

isBefore_version属性就是软件类对象之间一个传递属性，它和isAfter_version是反属性，现在我们可以通过两个事实描述得到其中蕴涵的传递知识。

```
    <Word rdf:ID="Word 97">
        <isBefore_version rdf:resource="#Word 2000"/>
    </Word>
```

以上实例说明Word97是Word2000的之前版本。同样可以定义Word2000是Word2003的先后关系。

```
    <Word rdf:ID="Word 2000">
        <isBefore_version rdf:resource="#Word 2003"/>
    </Word>
```

因为isBefore_version具有传递性因此我们可以推导得出Word97是Word2003的之前版本。

下边是一个数据类型属性的例子：

```
<xsd:schema xmlns:xsd "http://www.w3.org/2001/XMLSchema"
    xmlns=" http:/ /www.ardragon.org/software.xsd">
    <xsd:simpleType name="managerNumber">
        <!－year is an XMLS datatype based on integer－>
        <xsd:restriction base="xsd:decimal"/>
            <xsd:minInclusive value="1"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
<owl:Class rdf:ID="SoftWareManager"/>
    <owl:DataTypeProperty rdf:ID=" SoftWareManagerLimit ">
        <rdfs:domain rdf:resource="# SoftWareManager "/>
        <rdfs:range rdf:resource="&dt;managerNumber "/>
    </owl:DataTypeProperty>
```

managerNumber是利用XML Schema datatype定义的简单类型，限制最小数量为1；而后的数据类型属性引用该限制到一个SoftWareManager的类上，意味着软件管理者至少有一个。

(3) 个体

个体代表（领域中）我们实际感兴趣的那些对象，是类的实例，property可以被用来把一个individual和另一个individual关联起来。

< SoftWareManager rdf:ID="王二"/>

这里定义了一个叫王二的软件管理人员类的实例。

(4) 属性限制

除了能够指定属性特性，我们还能够使用多种方法进一步在一个明确的上下文中限制属性的值域，这是通过“属性限制”来完成的。下面描述的allValuesFrom和someValuesFrom就是两种约束限制。owl:allValuesFrom属性限制要求：对于每一个有指定属性实例的类实例，该属性的值必须全是由owl:allValuesFrom从句指定的类的成员。而owl:someValuesFrom则是部分的属性值是someValuesFrom从句指定的类的成员。

```
<owl:Class rdf:ID="#ApplicaionSoftWare">
   <rdfs:comment >
      This is a "应用软件类"。
   </rdfs:comment >
    <rdfs:subClassOf rdf:resource="#SoftWare"/>
      <owl:Restriction>
          <owl:onProperty rdf:resource="#hasProducer" />
          <owl:someValuesFrom rdf:resource="#Lab" />
      </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

allValuesFrom与 someValuesFrom的限制是局部，它们仅仅在包含它们的类的定义中起作用。上例中someValuesFrom限制仅仅应用在ApplicaionSoftWare的hasProducer属性上。SystemSoftWare类的制造商并不受这一局部限制的约束。owl:allValuesFrom限制与之相似。在上个例子说明在所有的ApplicaionSoftWare中至少有一个软件个体是Lab类的实例开发的，如果我们用owl:someValuesFrom替换owl:allValuesFrom那就意味着所有的ApplicaionSoftWare类实例的hasProducer属性都指向一个Lab类的个体的。

(5) 本体映射

OW L的本体映射分为以下三个方面：

(1) 等价类(equivalentClass)和等价属性(equivalentProperty)：等价类是指两个类可以被声明为等价，即它们拥有相同的实例。等价性可以用来创建同义类。例如，类Car可以被说成是类Automobile的等价类。两个属性也可以被声明为等价，

相互等价的属性将一个个体关联到同一组其它个体。它也可以被用来创建同义属性。例如，HasLeader可以被说成是hasHead的等价属性（equivalentProperty）。

(2) 个体之间的等价(sameAs)：可以把两个个体声明为相同个体。可以被用来创建一系列指向同一个个体的名字。

(3) 不同个体(differentFrom)和完全不同(AllDifferent)：differentFrom是指可以声明一个个体与其他个体不同。AllDifferent以指出一定数量的个体两两不同。

下面定义"办公自动化软件"的等价类" OA "。

```
<owl:Class rdf:ID="办公自动化软件">
    <owl:equivalentClass rdf:resource="&OA"/>
</owl:Class>
```

(6) OWL中复杂类的定义

(a) 集合操作(SetOperators)

OWL中集合操作包括三种情况：owl:intersection0f(交集)、owl:unionOf(并集)和owl:complementOf(补集)。

假设要描述概念："安全工具是杀毒工具和防火墙的并集"

```
<owl:Class rdf:ID="SecurityTools">
    <owl:union Of rdf:parseType="Collection">
        <owl:Class rdf:about ="#Antivirus"/>
        <owl:Class rdf:about ="#Firewall"/>
    </owl:union0f>
</owl:Class>
```

(b) 枚举类(Enumerated Classes)

OWL提供了通过直接枚举类的成员来详细说明一个类的描述，它通过owl:oneOf构造器来实现。

```
<owl:Class rdf:ID="Office2003">
    <rdfs:subClassOf rdf:resource="#MicrosoftOffice"/>
        <owl:oneOf rdf:parseType='Collection">
            <Office 2003 rdf:about ="#MicrosoftWord2003"/>
            <Office 2003 rdf:about="#MicrosoftExcel2003"/>
            <Office 2003 rdf:about="#MicrosoftVisio2003"/>
        </owl:one0f>
</owl:Class>
```

其它复杂类中还包括了不相交类(Disjoint Classes)。其例子参看上述提到的OWL类的例子。

4.4 构建工具Protégé

Protégé由斯坦福大学开发的一种创建和编辑ontology知识库的与平台无关的工具，开发语言为Java，为开放源码软件。由于其优秀的设计和众多的插件，Protégé成为目前使用最广泛的本体论编辑器之一。它能够构建领域模型和基于本体的知识的应用。Protégé的主要功能实现了丰富的知识模型结构集和行为，这些集合和行为支持各种形式的本体创建，可视化以及操作。此外，Protégé还为编辑ontology所涉及的类、属性、实例提供一个友好的图形界面，方便用户使用。目前Protégé的最新版本为3.2beta版。当前版本可以在相关插件的支持下用于创建基于Protégé默认文档、默认数据库、OWL数据库、OWL文档、RDF文档等多种存储格式的ontology，各种格式之间还可以彼此转换；同时，Protégé 通过plug-in架构和基于java的应用程序接口进行工具扩展和应用扩展。

Protégé的开发环境如图4. 1所示，它的图形界面统一以tab页的方式呈现给用户，Protégé本身及其插件提供了为数众多的tab页，用户可以自己配置是否显示相关的tab页，例如OWLClasses tab页、Properties tab页、Individuals tab页等等。这些tab页可以协同完成创建ontology等任务，用户可以利用它完成以下操作：1.创建和编辑ontology所涉及的类、属性及其之间的关系；2.输入实例数据；3.对已有的ontology和实例数据进行浏览和检索。

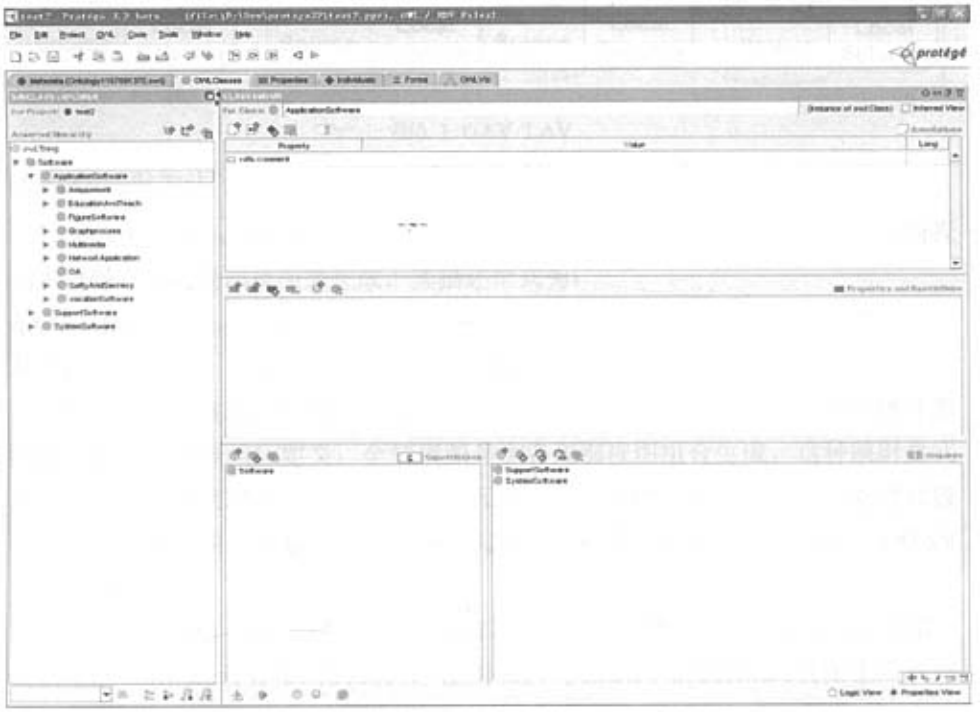

图4. 1 Protégé开发环境

4.5 本章小结

本章首先给出了领域本体构建的原则、方法、本体元语以及构建步骤；随后介绍了本文使用的OWL-Web本体语言和OWL语言中主要的术语：类、属性、约束、映射的概念以及他们的使用实例；最后，介绍了构建本体的工具Protégé，关于具体使用我们将在第六章给出。下一章将详细讨论构建Web信息集成中的另外一个主要问题——查询分解。

第五章 查询分解

查询分解问题一直是信息集成中的核心问题，查询分解主要解决在全局视图下的用户查询要求如何能够反映到各个具有局部视图的数据源上。即解决全局视图到众多局部视图的映射。目前，解决此问题有两种基本方法一种是GAV(Global-as-view，全局作为视图)和LAV(Local-as-view，局部作为视图)，如图5.1。

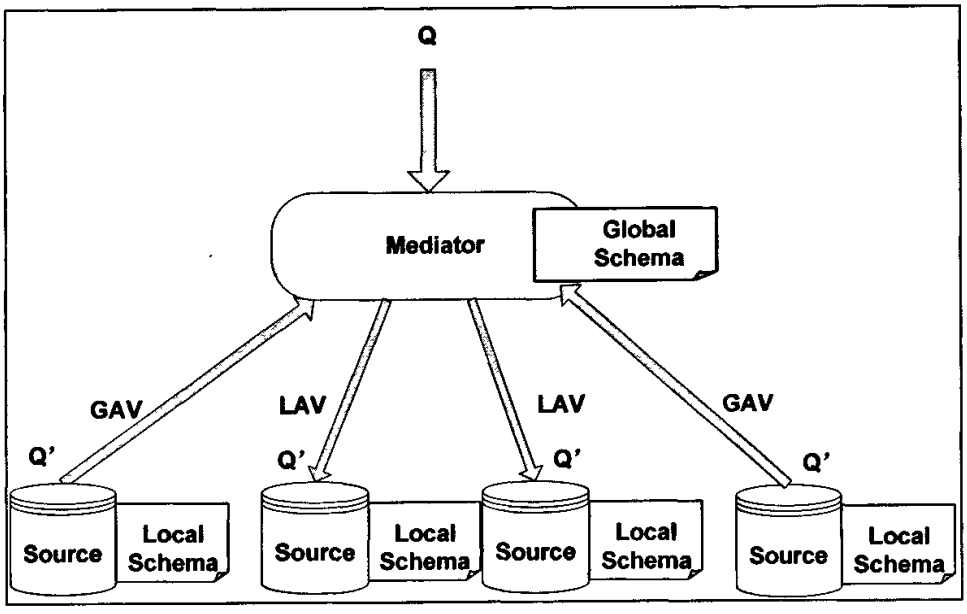

图5.1 GAV/LAV

(1) GAV映射方法

GAV(Global as View)：全局视图定义成多个数据源上的视图的合，即全局视图由已经存在的局部视图来生成，其描述形式为：

$r(X) \leftarrow s_1(X_1,Y_1) \wedge s_2(X_2,Y_2) \wedge s_3(X_3,Y_3) \cdots\cdots \wedge s_n(X_n,Y_n)$

其中r为全局模式，s_i为数据源中的关系，$X=\cup X_i$。

从建模的角度看，GAV映射方法的基本思想是全局模式中每个元素应该在数据源上的视图找到相应定义。全局视图是由各局部视图组合生成。这种映射显式地告诉了系统如何对全局模式上的各个元素求值。GAV映射方法的查询处理比较简单，因为这种映射关系告诉了系统如何使用数据源获取数据。下面是一个GAV模式映射的实例：

如图5.2所示，存在数据源LS1和LS2，两数据源内都有各自的Software视图，试图中的Software含有不同的属性，使用GAV方法GS中的Software视图生成规则为：

Software(id,名称,类别,版本,大小,功能描述,开发语言,开发单位,录入时间)=

[{a,b,c,d,e,f,g,h,i }|< a,b,c,d,e,f,g,_,_,_>∈Software1n

<_,_,_,_,_,_,_,h,I>∈Software2n

Software1. b= Software2. b]

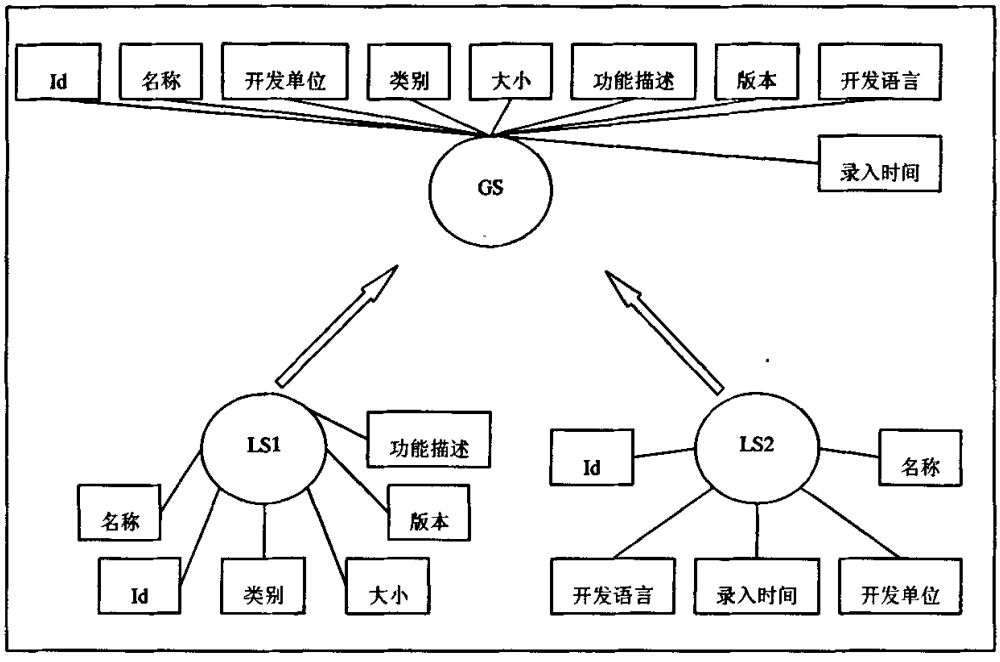

图5. 2 GAV的Software视图

由此我们可以看出，GAV方法可以建立不同层次的全局模式，查询分解就是视图的分拆，即可以根据全局模式的视图定义将查询直接分配到各数据源。

(2) LAV映射方法

LAV(Local as View)：每个数据源定义成全局模式上的视图，即局部视图参照全局视图构建，其描述形式为：

$S(X) \leftarrow r_1(X_1,Y_1) \wedge r_2(X_2,Y_2) \wedge r_3(X_3,Y_3) \cdots\cdots \wedge r_n(X_n,Y_n)$

其中，r为全局模式，S_i为数据源中的关系，$X=\cup X_i$；

从建模的角度来看，LAV映射方法的基本思想是数据源中每个元素的内容应该由全局模式上的查询来定义。各数据源的模式依照全局模式来生成，每个数据源的数据模式均为全局模式的一个子集。当数据集成系统建立在一个稳定的、已经构建好的全局模式之上的时候，使用LAV映射方案比较有效。在LAV的数据集成系统中，添加一个新数据源只需要添加一个映射规则即可，不需要作任何其它修改，因此数据源的可扩充性比较好。下边是一个LAV模式映射的实例：

如图5.3所示，全局视图中已有视图Software、Developer（开发单位，单位名称，单位所属）和Sort（分类，分类名称，使用平台），现存在数据源LS1、LS2，现采用LAV映射方法构建数据源的局部视图：

LS1.App_Software(id，名称，版本，大小，使用平台) =

[{a, b, c, d, e}|<a, b, _, c, d, _, _, _, _, _> ∈Software∩

<_, _, e>∈Sort∩Software.类别=Sort. 分类∩Sort. 分类名称=“应用软件”]

LS2.Dev_Information(名称，开发时间，单位名称，单位所属) =

[{a, b, c, d}|< _, a, _, _, _, _, _, _, b>∈Software∩

<_, c, d>∈Developer∩Software. 开发单位= Developer. 开发单位]

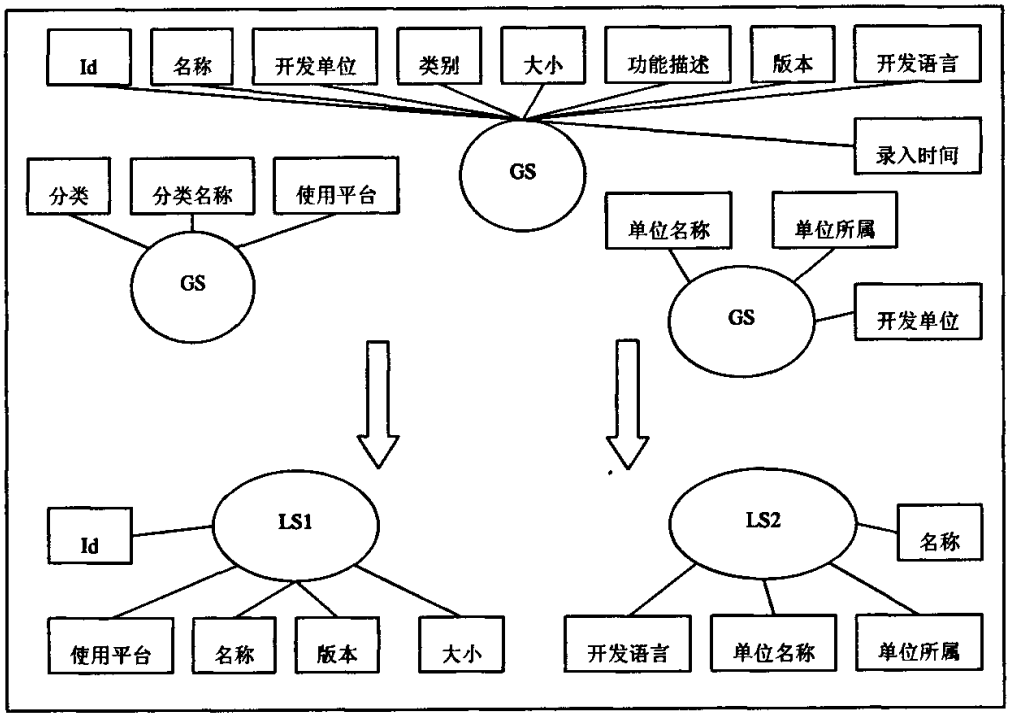

图5.3 LAV的Software视图

(3) 两种映射方案的比较

GAV映射方案的特点是：

1) 系统的质量依赖于数据源同全局模式的映射；

2) 当数据源改变或者新数据源加入的时候，需要重构全局模式；

3) 比较适合于建立在比较稳定的数据源集合之上的数据集成系统；

4）查询处理依靠全局模式映射，比较容易。

LAV映射方案的特点是：

1) 系统的质量依赖于对数据源定义的好坏；

2) 数据源加入或更新时只不影响全局模式，数据源的可扩充性比较好；

3) 查询分解处理比较复杂，需要专门的分解算法。

4）比较灵活，增加、删除和修改数据源中的信息只增删修对应的数据源模型，因此特别适用于WWW上的动态环境。

传统的LAV实现均采取映射文件固定全局视图与局部视图作法，此方法需要静态固定映射，在映射发生变化或者新数据源加入时需要人工维护映射，耦合性大，缺乏动态应变能力。本文针对Web信息源的自身特点，采用了LAV方法实现查询分解，但在具体实现技术方面更多考虑了降低中介器与数据源的耦合，选择使用Web服务包装数据源信息，以带有语义的服务查找的方式完成LAV中的查询分解。因此，为了实现语义的查询，在数据源服务描述方面，本文采用W3C组织最新发布的Web本体语言的服务描述标准(OWL-S)描述数据源提供的服务，大大加强了服务的描述能力，使得带语义的服务发现策略可以动态完成查询分解。在此基础之上，本文提出了一种带语义的服务发现和选择方法动态匹配全局查询需求，以实现查询分解的策略。下面将详细论述此实现方法。

5.1 基于本体的服务描述

在Web服务的标准协议栈中W3C组织定义了WSDL(Web service description language)来描述服务，描述包括服务的数量类型、消息格式、传输协议、位置、使用条件和服务质量等信息。然而，由于WSDL描述能力不强。在WSDL中，消息和操作等被描述得比较抽象，影响了重用、扩展和响应；并且，用WSDL基本上无法描述前件、后件及结果等逻辑关系。此外，WSDL缺乏对应用的语义描述，WSDL重心放在了描述服务的基础上，虽然它也支持输入、输出，但是它并不支持输入、输出之间的逻辑约束。因此，现行的Web服务机制难以详尽描述服务提供数据的模式同时也难以实现基于语义的逻辑化的服务发现，有鉴于此，人们提出了Web服务本体描述语言OWL-S[29]，OWL-S是以OWL语言为基础的Web服务描述框架，提供了对服务的语义描述，包括三类本体如图5.4：

1）Web 服务做什么，例如服务实体、服务可以实现的功能，以及服务的性能参数等。这方面的描述通过“ServiceProfile”来实现。ServiceProfile中包含有服务的输入输出描述，服务的输入/输出可以反映出该服务包装下的具体数据源中能提供的数据的数据模式，因此根据不同服务的输入/输出以及它们的类型就可得到相对应的服务能提供数据的数据模式。为了使输入/输出类型可以进行有意义的比较，所有输入/输出必须出自某个的公共的参数本体或其扩展部分。基于输入输出描述，服务请求者(人或代理程序)可以发现满足特定功能需要的 Web 服务，可以确定需要满足

哪些条件才能调用该服务。此外，服务请求者也可以遵循“ServiceProfile”来描述自己的服务请求。

2）Web 服务如何执行，包括服务执行的先后顺序、过程流程等，这方面的描述通过“ServiceModel”来实现，服务请求者利用“ServiceModel”可组合多个服务以完成复杂任务，同时在服务执行过程中，可以利用“ServiceModel”来协调参与各方的动作。

3）如何实际调用 Web 服务，描述具体的绑定信息，例如服务地址、通信协议及消息格式等。这方面的描述通过“ServiceGrounding”来实现。

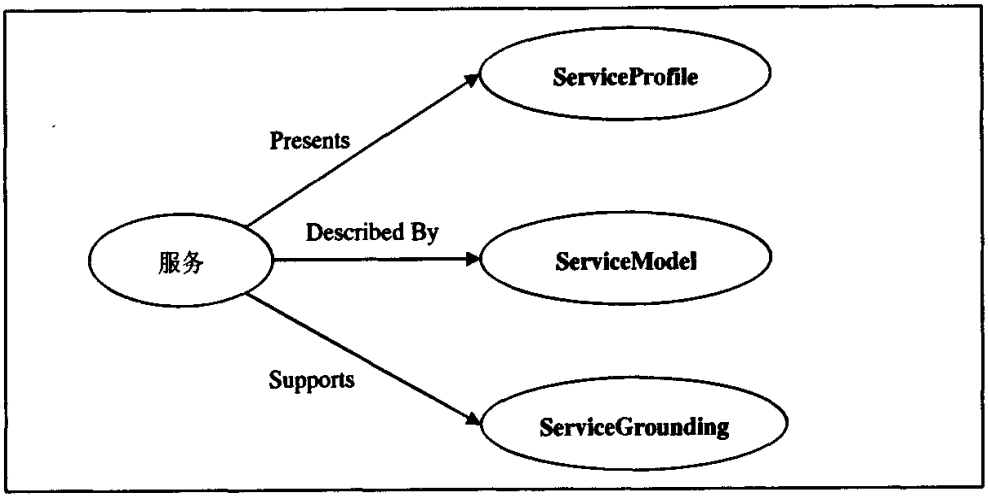

图5. 4 Web服务描述框架

用OWL-S描述Web服务，不仅使得基于语义的Web服务发现成为可能，同时也向外说明了服务能提供数据的数据模式。本系统采用Web服务本体描述语言OWL-S在领域本体的指导下对各个数据源服务进行语义描述，下面是ServiceProfile关于软件价格查询服务的输入输出本体描述片断示例，此Web服务名为“BuySoftware”，它要求有两个输入，一个为“Producer”，即软件生产商的名字，并且规定了输入参数类型为String型；另一个输入是软件发布日期“IssueDate”，它的类型是“Date”；此外，服务还说明了服务的输出信息“Software”信息和“Price”信息，前者的参数类型是String型，后者是float型。

```
<process:AtomicProcess rdf:ID="BuySoftware">
  <process:hasInput>
    <process:Input rdf:ID=" Producer ">
      <process:parameterType rdf:resource="&xsd;#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasInput>
```

```
<process:Input rdf:ID=" IssueDate ">
    <process:parameterType rdf:resource="#Date"/>
  </process:Input>
</process:hasInput>
  ..............
<process:hasOutput>
  <process: Output rdf:ID="Software">
  <process:parameterType rdf:resource="&xsd;#string "/>
  </process: Output>
</process:hasOutput>
<process:hasOutput>
  <process: Output rdf:ID=" Price">
  <process:parameterType rdf:resource="&xsd;#float"/>
  </process: Output>
</process:hasOutput>
```

5.2 服务发现与选择

在此，所谓的Web服务发现与选择，就是根据查询的全局视图条件，在已注册服务中查找最优匹配需求的服务或服务集，实现查询分派到各数据源的过程。基于语义的服务发现是在语义层上进行需求描述和匹配判断的。系统主要以OWL-S框架里ServiceProfile中关于服务输入参数和输出参数的本体描述为参考，根据语义匹配与选择算法SMC，选出符合此次用户查询所需的服务进行数据抽取。

在介绍匹配算法前，我们先来明确算法中需要用到的定义：

服务表示：一个Web服务可以表示为：$WS_i(I_i, O_i)$其中WS_i是Web服务的名字；I_i是该服务的输入集合；O_i是该服务的输出集合。

输入/输出属性：根据本体文件，把输入/输出参数的相关属性称为“输入/输出属性”（rdfs:domain / rdfs:range元素）。

扩展属性：在同一本体领域内，和输入属性有父子、包含与被包含等关系的属性称为“扩展属性”。

相似度：相似度是指两个概念语义符合的程度，在信息集成中，相似度更多的要反映文本与用户查询在意义上的符合程度，即用户请求和服务之间的相似度，用户请求和服务间的相似度越高，此服务符合查询要求的可能性越大。相似度是一个数值，一般取值范围在[0,1]之间。一个词语与其本身的语义相似度为1。如果两个词语在任何上下文中都不可替换，那么其相似度为0。

语义距离（**Semantic Distance**）：语义距离是度量两个词语关系的另一个重要指标。语义距离是领域本体中的两个不同的类之间存在的继承关系或二元关系链中最短的关系链的度量。一般而言，语义距离是一个零到无穷之间的实数。

语义距离与概念相似度之间有着密切的关系。相似度与对应的语义距离类似与反比关系。两个词语的距离越大，其相似度越低；反之，两个词语的距离越小，其相似度越大。二者之间存在一种简单的对应关系。这种对应关系有以下几个特点：

1) 两个概念的语义距离越大，其相似度越小；

2) 两个概念的语义距离为无穷大时，其相似度为0；

3) 两个概念的语义距离为0时，其相似度为1。

假设W_1和W_2分别表示两个概念，我们记其相似度为$Sim(W_1,W_2)$，其语义距离为$Distance(W_1,W_2)$，那么我们可以定义一个满足以上条件的简单的转换关系：

$$sim(W_1,W_2)=\frac{\alpha}{\alpha+\mathrm{Distance}(W_1,W_2)} \qquad (5\text{-}1)$$

其中α是一个可调节的参数。α的含义是：当相似度为0.5时的语义距离值。

语义距离的计算方法：语义距离常见的计算方法是根据领域本体（Ontology）来计算。因为本文构建的领域本体都是将所有的词组织在一棵或几棵树状的层次结构中，所有概念都拥有最高的父类Thing类，我们知道，在一棵树形图中，任何两个结点之间有且只有一条路径。于是，这条路径的长度就可以作为这两个概念的语义距离的一种度量。如图5. 5所示，黑色线段表示了Thing到Class13的距离：

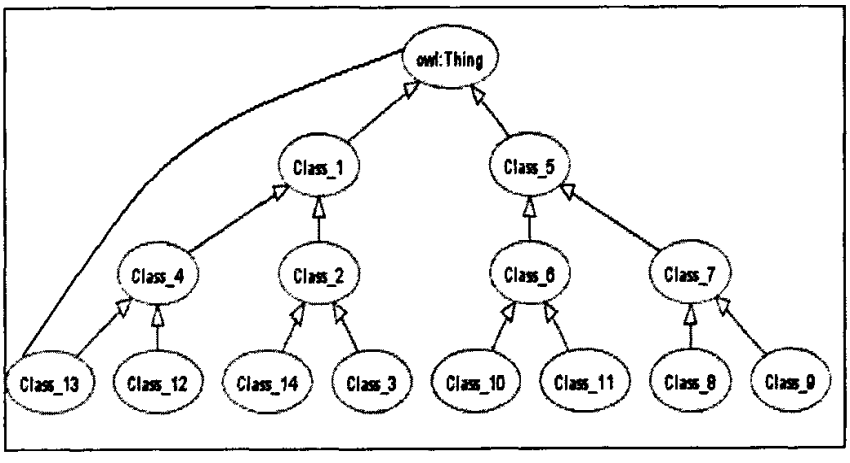

图5. 5 语义距离的计算

Semantic Match and Choice算法（简称SMC）是在领域本体的基础上，将基于全局本体的查询请求条件和查询请求输出与注册服务的参数进行相似度计算，在此，相似度采用依据语义距离SD来计算，记为SD_{mn}=Distance(m,n)，m和n是类，

Distance函数求出的是m和n在领域本体体系中的路径长度，语义距离越短，表明两个类的相似度越高。系统在领域本体的指导下计算，算法的具体步骤如下：

1) 得到用户查询需求：QIC{$class_{I1}$,$class_{I2}$, ...$class_{Im}$}，其中每个 class 都是用户关心的概念，概念类可以有多个，系统把 QIC 中的所有类与每个注册 Web 服务的所有输入条件 GIC{$class_{R1}$,$class_{R2}$, ...$class_{Rn}$}依次按照对应顺序如：($class_{I1}$,$class_{R1}$)，($class_{I2}$,$class_{R2}$)分别进行语义距离的计算，表达式为：SDI_{ii}=Distance($class_{Ii}$,$class_{Ri}$)。若 m>n，则对应的 $class_{Ri}$，(n+1<=i<=m) 取空值；反知，若 m<n，对应的 $class_{Ii}$，(m+1<=i<=n) 取空值。
2) 与输入条件类似，计算查询请求输出条件：QOC {$class_{O1}$,$class_{O2}$,...$class_{Om}$}与每个注册服务输出条件 GOC{$class_{G1}$,$class_{G2}$,...$class_{Gn}$}的语义距离如：SDO_{jj}=Distance($class_{Oj}$,$class_{Gj}$)。
3) 根据 SDI 和 SDO 计算输入、输出条件的相似度 SimI 和 SimO，计算采用公式（5-1）；然后，把 SimI 和 SimO 求和得到查询条件与此服务的总相似度 WS_k.Sim。
4) 依次计算所有注册服务与查询条件的总相似度。

SMC算法伪代码如下：

输入：用户查询条件，包括输入条件QIC和输出条件QOC
输出：满足一定要求的匹配服务或者服务集合

```
SMC(QIC,QOC){
  For each WSk in the registered services do{
    int i=0,j=0
    For each classIm ∈QIC and each classRn∈WSk.GIC do{
    //计算输入参数的语义相似度，其中α是一个可调节的参数。
    //α的含义是：当相似度为0.5时的语义距离值。
    // WSk.GIC是：注册中的第k个service的输入要求。
    If(i<=m&&i<=n)
      SDIii=Distance(classIi,classRi)
    Else if(i>m)
      SDIii=Distance(NULL,classRi)
    Else
      SDIii=Distance(classIi, NULL)
      SimIii = α / (α + SDIii)
    }
    SimIk = ∏_{i=1}^{max(m,n)} SimIii
```

For each $class_{Om}$ ∈QOC and each $class_{Gn}$∈GOC do{

//计算输出参数的语义相似度

If(j<=m&&j<=n)

SDO_{jj}=Distance($class_{Oj}$,$class_{Gj}$)

Else if(j>m)

SDO_{jj}=Distance(NULL,$class_{Gj}$)

Else

SDO_{jj}=Distance($class_{Oj}$,NULL)

$$SimO_{jj} = \frac{\alpha}{\alpha + SDO_{jj}}$$

}

$$SimO_k = \prod_{j=1}^{\max(m,n)} SimO_{jj}$$

//计算第k个服务的总相似度Weight1，Weight2是输入输出的权值参数

WS_k.Sim＝$SimI_k$×Weight1＋$SimO_k$×Weight2

//把计算过相似度的服务加入一个服务列表

WSList.add(WS_k)

}

//返回计算过相似度的服务列表

return WSList

}

SMC算法解决了服务选取的凭证计算，下面我们给出采用该算法完成查询分解的完整步骤：

i. 对用户查询条件调用SMC算法，计算得出服务列表。

ii. 按相似度高低对服务进行排序。

iii. 采用某种选择策略（如只选取高于给定相似度阈值的服务）对排序过的服务进行选择，取出此次需要的服务并对选取的服务进行调用完成查询分解。

iv. 若没有得到任何匹配用户要求的数据源服务，则对用户的原始查询条件进行推理得到新的输入条件（如用户条件为安全工具软件，由于没有哪款软件类别是安全工具类，所以系统对安全工具类进行推理得到子类防火墙类和杀毒工具类，由此用户的条件变成了查询防火墙软件和杀毒工具软件）再执行步骤i。

//查询分解伪代码

Match(QIC,QOC){

```
    WSList=SMC(QIC,QOC)
    Sort(WSList)//根据相似度进行服务排序
    // Choose()完成进行服务选择
    // condition给定相似度阈值，matches为按选择策略选出的服务或服务集
    matches= Choose(WSList ,condition)
    return matches
}
QueryDecompose(QIC,QOC){
    matches= Match(QIC,QOC)
    if(mathes==null){
        QIC'=GetNewCondition(QIC)
        QOC'=GetNewCondition(QOC)
        matches= Match(QIC', QOC' )
    }
    //返回最终结果服务或服务集
    return matches
}
```

5.3 本章小结

本章首先介绍了查询分解的主要的两种方法GAV发和LAV法，并给出两种方法的比较，说明了LAV方法更适合与Web信息集成系统；然后提出了一种实现LAV的思路，即采用基于本体和服务发现的方式来实现LAV。接下来，本章详细说明了该方法的实现，首先给出了如何用OWL－S语言实现Web服务的语义描述，然后详细论述了如何用服务发现与选择策略实现查询分解。

第六章 实现与测试

基于前面所述的理论和方法，我们设计并实现了本文第三章提出的WEB信息集成系统的一个原型系统。该原型系统主要使用java语言，平台采用J2SE v1.5.2，本体构建采用Protégé工具。原型系统实现了本文提出的系统框架的主要模块，如：构建了一个软件领域的本体、数据源的服务包装和基于本体的描述以及查询分解等模块。由于时间关系，信息抽取、数据合并以及在一些实际应用中需要考虑的细节问题我们的系统中没有顾及。

6.1 软件领域本体模块实现

基于开发需要我们把软件按层次划分为：系统软件、支持软件和应用软件三大类，系统软件分为：操作系统软件、系统实用软件、系统扩充软件；支持软件分为：软件开发、软件测评、软件管理、数据库管理、网络支持软件等；应用软件分为：文字处理软件、图形软件、图像处理软件、游戏娱乐软件、网络应用软件、安全与保密软件、教育教学等软件，其中层次分类的缩略图，如图6. 1

领域内的部分类定义如下：

```
<owl:Class rdf:about="#Software">
 <!—软件类定义-->
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Company">
<!—公司类定义-->
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Date">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="CurrencyPrice">
<!—价格类定义-->
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#SystemSoftware">
  <rdfs:subClassOf>
```

```
        <owl:Class rdf:ID="Software"/>
      </rdfs:subClassOf>
      <owl:disjointWith>
        <!—系统软件类与其他两种软件相独立-->
        <owl:Class rdf:about="#SupportSoftware"/>
       <owl:Class rdf:ID=" ApplicationSoftware "/>
      </owl:disjointWith>
    </owl:Class>
  <owl:Class rdf:about="#ApplicationSoftware">
      <rdfs:subClassOf>
        <owl:Class rdf:ID="Software"/>
      </rdfs:subClassOf>
      <owl:disjointWith>
        <owl:Class rdf:ID="SupportSoftware"/>
        <owl:Class rdf:ID="SystemSoftware"/>
      </owl:disjointWith>
    </owl:Class>
    <owl:Class rdf:about="#SupportSoftware">
      <rdfs:subClassOf>
        <owl:Class rdf:ID="Software"/>
      </rdfs:subClassOf>
      <owl:disjointWith rdf:resource="#ApplicationSoftware"/>
      <owl:disjointWith rdf:resource="#SystemSoftware"/>
    </owl:Class>
```

领域内的部分对象属性定义代码如下：

```
    <owl:ObjectProperty rdf:ID="run_At">
      <!—run_At属性把应用软件和操作系统相关联-->
      <rdfs:domain rdf:resource="#ApplicationSoftware"/>
      <rdfs:range rdf:resource="#OperationSystem"/>
    </owl:ObjectProperty>
    <owl:ObjectProperty rdf:ID="hasABargainor">
      <!—hasABargainor属性说明软件有贩卖商-->
      <rdfs:range rdf:resource="#Bargainor"/>
      <rdfs:domain rdf:resource="#Software"/>
    </owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="hasAPrice">
  <!—hasAPrice属性说明软件都有价格-->
  <rdfs:range rdf:resource="#CurrencyPrice"/>
  <rdfs:domain rdf:resource="#Software"/>
</owl:ObjectProperty>
```

领域内的部分数据类型属性定义代码如下：

```
<owl:DatatypeProperty rdf:ID="Name">
    <!—名称属性定义-->
  <rdfs:domain rdf:resource="#Software"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
   </owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Version">
    <!—版本属性定义-->
    <rdfs:domain rdf:resource="#Software"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="issueTime">
    <!—发布日期属性定义-->
  <rdfs:domain rdf:resource="#Software"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</owl:DatatypeProperty>
```

领域内的部分关系定义代码如下：

软件有一个发布日期、一个卖出日期、一个软件制造者。

```
<owl:FunctionalProperty rdf:ID="hasIssueTime">
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#IssueDate"/>
  <rdfs:domain rdf:resource="#Software"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="hasASaleTime">
  <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:domain rdf:resource="#Software"/>
  <rdfs:range rdf:resource="#saleDate"/>
```

```
        </owl:FunctionalProperty>
        <owl:FunctionalProperty rdf:ID="hasAProducer">
          <rdfs:domain rdf:resource="#Software"/>
          <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
          <rdfs:range rdf:resource="#Producer"/>
</owl:FunctionalProperty>
```

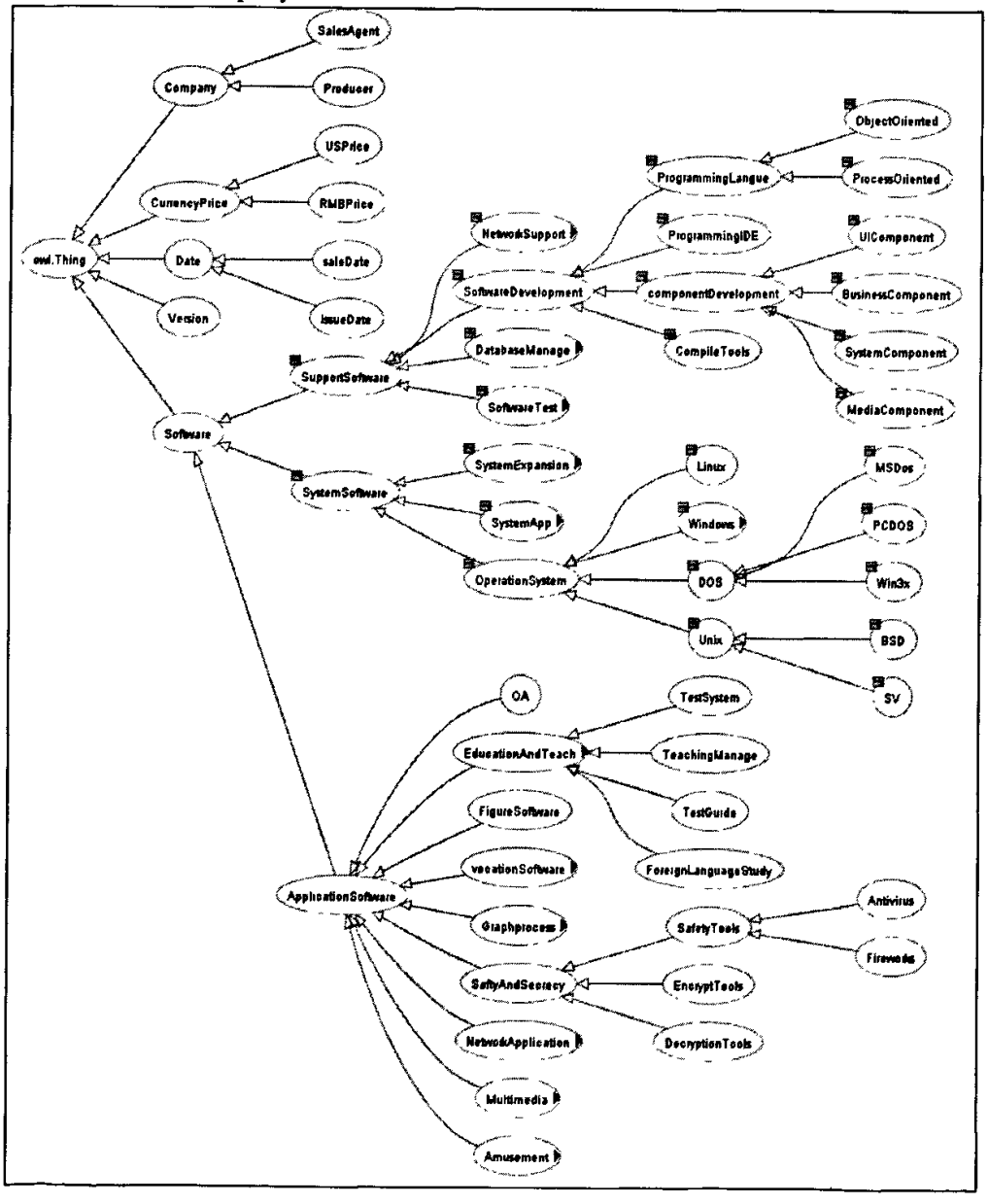

图6.1 本体分类图

6.2 服务描述模块实现

系统仿真了若干提供软件信息查询的服务如指定软件的价格信息查询，厂商查询，发布时间查询、销售查询等服务，不同数据源按各自所拥有的数据模式可能提供了相同的数据源服务，但他们能提供的信息项可能不同。我们用WS_i(I_i, O_i)形式描述服务的部分服务类的实例如下：

QuerySoftwareByProducerIssueTime(I , O)：

//通过软件厂商和发布日期查询软件信息

I：Producer and IssueTime

//输入条件是软件厂商和发布日期

O：Software and Price

//输出的是该服务包装的数据源中能提供的软件类及其软件价格类信息

QueryProducerBySoftware(I , O)：//查询软件生产商及版本信息

I：Software //输入条件是具体的软件类

O：Producer and Edition //输出软件对应的生产商和版本

下面是其中一个服务的Profile部分的描述实例。

```
<rdf:RDF
    xmlns:j.0="http://www.daml.org/services/owl-s/1.2/Profile.owl#"
    xmlns:j.1="http://www.daml.org/services/owl-s/1.2/Service.owl#"
    xmlns:j.2="http://www.daml.org/services/owl-s/1.2/Process.owl#"/>
<j.2:Output rdf:ID="Software">
    <j.2:parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
    http://www.source1.com/Schema1#Software </j.2:parameterType>
</j.2:Output>
<j.0:Profile rdf:ID="QuerySoftwareProfile">
 <j.1:presentedBy>
  <j.1:Service rdf:ID="QuerySoftwareService">
    <j.1:presents rdf:resource="#QuerySoftwareProfile"/>
      <j.1:describedBy>
       <j.2:AtomicProcess rdf:ID="QuerySoftwareProcess">
        <j.2:hasOutput rdf:resource="# Software "/>
          <j.1:describes rdf:resource="# QuerySoftwareService "/>
              <rdfs:label     rdf:datatype="http://www.w3.org/2001/XMLSchema
                #string ">QuerySoftwareProcess</rdfs:label>
```

```
<j.2:hasInput>
  <j.2:Input rdf:ID="Producer">
    <rdfs:label rdf:datatype=http://www.w3.org/2001/XMLSchema#
    string> Producer </rdfs:label>
     <j.2:parameterType
     rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
     http://www.ardragon.com/Software.owl# Producer
    </j.2:parameterType>
  </j.2:Input>
 </j.2:hasInput>
 <j.2:hasInput>
   <j.2:Input rdf:ID="IssueDate">
    <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema
      #date"> IssueDate </rdfs:label>
          <j.2:parameterType rdf:datatype="http://www.w3.org/
          2001/XMLSchema#anyURI">http://www.ardragon.com/S
          oftware.owl# IssueDate </j.2:parameterType>
     </j.2:Input>
  </j.2:hasInput>
 </j.2:AtomicProcess>
</j.1:describedBy>
</j.1:Service>
</j.1:presentedBy>
</j.0:Profile>
```

6.3 系统类实现

图6.2显示了系统主要类之间的关系，类OntologyDB封装了关于本体中类关系的操作，是本体对外的接口；类RegCenter是存放注册服务语义信息的容器；类UserQueryInformation封装了用户的请求，是系统的用户接口，封装好的输入信息提交到类QueryPretreatment，QueryPretreatment负责处理用户查询请求信息；类ServicesDiscovery是基于语义的服务发现引擎的实现类。控制类Mediator，它通过调用类OntologyDB类、ServicesDiscovery类和类RegCenter协同工作，完成服务发现、查询分解、服务调用等核心功能。最后，将发现结果封装在类Result返回给用户接口。

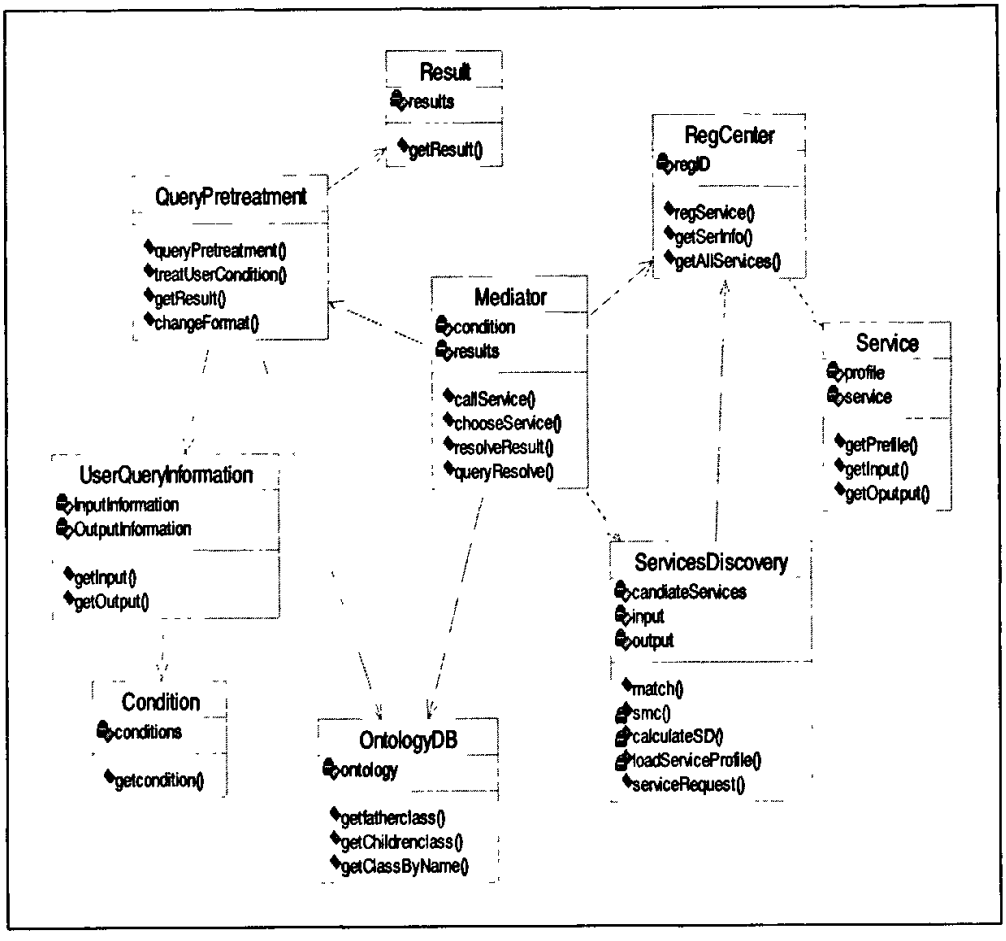

图6.2 系统类图

6.4 运行实例与分析

对于该系统实现，我们进行了跨数据源查询服务的实验。实验中我们自行设计了3个计算机软件相关的数据源，这些数据源上提供了两类服务：一类是实现不同的功能的服务，一类是实现了相同功能但提供数据信息不同的服务。三个数据源所提供的服务既有相似相同的类别（如基本的软件信息查询服务），又各自有侧重。数据源A侧重各种不同类别软件价格的查询服务；数据源B在软件厂商和代理商信息查询方面有优势；而数据源C的服务则比较综合。表6.1给出了数据源服务的示例：

	服务名称	输入类型	输出类型	说明
数据	WSA1	Amusement，Version	RMBPrice	服务给出查询娱乐类软件的人民币价格

源A	WSA2	Windows，Version	RMBPrice	服务给出查询Windows类软件的人民币价格
	WSA3	OperationSystem	USPrice RMBPrice	给出查询的操作系统类软件的美元价格和人民币价格
	WSA4	Producer，Version	Software	给出指定厂商指定版本的软件信息
数据源B	WSB1	Producer	Software Version	服务要求输入厂商信息输出数据源中对应的软件信息
	WSB2	SalesAgent	Software	给出指定代理商贩卖的软件信息
	WSB3	Producer，Game	SalesAgent	给出指定游戏软件的代理商
数据源C	WSC1	OperationSystem	USPrice	给出查询的操作系统软件的美元价格
	WSC2	Producer	Software Version	给出指定厂商生产的软件信息
	WSC3	BusinessComponent	Producer SalesAgent	输入商业组件类信息，输出相应的商业组件厂商和销售代理商信息

表6.1 数据源服务

当查询请求者的输入条件为Producer，输出条件为Software时，三个数据源中服务的相似度匹配如下表6.2：

	服务名称	输入相似度	输出相似度	综合相似度
数据源A	WSA1	0.0455	0.143	0.084
	WSA2	0.039	0.143	0.080
	WSA3	0.091	0.072	0.083
	WSA4	0.5	1	0.7
数据源B	WSB1	1	0.5	0.8
	WSB2	0.2	1	0.52
	WSB3	0.5	0.143	0.357
数据源C	WSC1	0.091	0.143	0.112
	WSC2	1	0.5	0.8
	WSC3	0.067	0.072	0.069

表6.2 相似度匹配

从上述相似度匹配结果可以看出，与用户查询条件不相关的数据源服务的相似度都在0.3以下，如WSA1服务的相似度为0.084、WSA3服务的相似度为0.083、WSC1服务的相似度为0.112；与用户输入条件部分相关的数据源服务的相似度在0.3到0.6之间；而与用户输入条件基本符合的服务相似度都在0.7以上，分别是数据源B的WSB1服务(相似度为0.8)、数据源C的WSC2服务(相似度为0.8)和数据源A的WSA4服务(相似度为0.7)。因此只要采取合适的选择策略，如我们的选择策略为选择相似度大于0.6的服务作为查询所需调用的服务，此次查询将选择服务WSB1、WSC2和WSA4进行调用，此时的查询分解结果将能够满足用户的查询要求，由此可见，采用前文讨论中所提出的设计能有效地完成查询工作。

第七章 总结与展望

本文的主要研究内容为基于本体和服务发现的Web信息集成技术。我们构建了一个软件领域的本体，提出了一个Web信息集成的系统结构，用OWL-S语言实现了数据源服务的语义描述，采用带语义的服务发现方法解决查询分解问题。其中，文中提出的系统采用了Web信息集成中的虚拟方法，充分继承了Mediator/Wrapper结构的精髓，并在数据源和中介器之间加入了一个服务库层，实现了中介器与数据源的松散耦合，满足了Web信息集成系统对信息源扩充的要求。在查询分解方面，本文在领域本体的基础上提出了采用带语义的服务发现方法和选择策略实现数据源的选取和查询的分解。

上述工作尽管为基于本体的Web信息集成提供了一个可行的方法，但我们应该清楚地认识到，这些工作与在构建一个实用、强健的基于本体的Web信息集成系统中所遇到的各种复杂和挑战性的需求仍还有很大距离，其中还有一些关键技术仍有待于进一步的探索和研究。以下两个方面将是我们进一步研究工作的重点：

[1]继续扩充和完善领域本体，使得基于此领域本体的各项应用更全面更准确。

[2]进一步改进查询分解策略，把更多的描述细节，如服务类型，服务个性化信息等加入到发现条件中；在服务选择考虑的因素中加入服务的其他条件（如Web服务的QoS功能：服务个性化等）等因素，以优化查询分解，提高查询分解的准确性。

致 谢

经过近一年来的资料收集、研究和实践，特别是近几个月来的论文撰写工作，终于完成了我的研究生学位论文，心头有种如释重负的感觉。尽管文中难免有不如意之处，但它毕竟凝结了我辛勤的汗水和不懈的努力。

在论文的最后，我要向所有帮助过我的人致以诚挚的感谢！

首先衷心感谢我的导师徐学洲教授在整个研究生学习阶段对我的栽培，师恩浩荡，不言而喻。他在处理繁忙的业务时，还对我悉心关怀和谆谆教导。他言传身教，使我对人生有了更进一步的认识，这些我将铭记在心。徐老师高瞻远瞩的学术思想、体察事物的独特思维方式，使我受益匪浅。徐老师对事业的执著追求的精神，渊博的学识、敏捷的思维、严谨的治学风范更是深深地感染着我，鼓舞着我！

感谢我的指导老师陈静玉的悉心指导，在其完成自己日常大量的工作之余，给我的论文撰写提出了大量的有建设性的意见。

感谢我的父母及我的姐姐，他们为我的成长付出了巨大心血。他们依然辛苦工作，支持我读研究生。他们对我无私的关爱和莫大的支持是我克服困难的最大动力，是有了他们的默默奉献才成就了我今天的学业。

感谢我的师兄弟们：张慎武、王宁、向红、程彪、黄听、陈国松、江佳、邓岳、付健等。他们无私的帮助和关心使我能够拥有一个愉快的生活氛围，同时，在论文写作过程中给了很多有价值的建议。

感谢西安电子科技大学，我在这里学习生活了近三年，不但学到了丰富的理论知识，也认识了好多好朋友。这将是我一生中非常宝贵的一段记忆。

感谢辅导员苗亚斌老师及其他2004级硕士研究生同学，他们的帮助、支持和鼓励，使我度过了难忘而有意义的研究生生活。

参考文献

[1] S.Abiteboul, P.Buneman, and D.suciu, Data on the Web-From Relations to Semi-Structured Data and XML, Morgan Kauffmann Publishers, 2000.

[2] 孟小峰. WEB信息集成技术研究. 计算机应用与软件，2003, No.11:32-36

[3] H.Garcia-Molina,Y.Papakonstantion,D.Quass.The TSIMMIS project: Integration of heterogeneous information sources. Journal of Intelligent Information Systems,1997,8(2):11 7-132.

[4] M.R.Genesereth,A.M. Keller,O.M.Duschka.Infomaster: an information integration system.In Proceedings of the ACM SIGMOD International Conference on Management of Data(ACM'97).no.539-542.1997.

[5] T Kirk. A.Y Levv.Y Sagiv,D.Srlvastava. The Information Manifold.In Proceedings of the AAA Spring Symposium on Information Gathering in Distributed Heterogeneous Environments, Stanford, CA, March, 1995.

[6] Y.Arens.C.A.Knoblok,W.Shen.Query reformulation for dynamic information integration.Journal of Intelligent Information Systems,1996,6(2-3):99-13.

[7] Ambite,C.A.Knoblock.Flexible and scalable cost-based query planning in mediators: a transformational approach[J].Artificial Intelligence, 2000(118): 115-161.

[8] D.McGuinness and F.Harmelen. OWL Web Ontology Language Overview. http://www.w3.org/TR/2004/REC-owl-features-20040210/.W3C Recommendation, 10 February 2004.

[9] Smith M K, Welty C, McGuinness D, OWL Web Ontology Language Guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210/

[10] NechesR.,FikesR.,FininT.,GruberT.,PatilR.,SenatorT.and SwartoutW.R ., Enabling technology for knowledge sharing,Al Magazine,1991,12(3):36 -56.

[11] Gruber,T.R.A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition,5,1993,199-220.

[12] B.Chandrasckaran, J.R.Josephson, and V.R.Benjamins,1999. What Are Ontologies ,and Why Do We Need Them. 1999 Jan/Feb:20-25.

[13] M.Uschold and M.Gruninger,1996.Ontologies:Principles,methods and applications. The Knowledge Engineering Review II(2).

[14] Guarino, N. and Giaretta, P. 1995. Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (ed.) Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing 1995. IOS Press, Amsterdam: 25-32.

[15] W.N.Borst,Construction Engineering Ontologies for Knowledge Sharing and Reuse,PhD Thesis,University of Twente, Enschede, The Netherlands, 1997.

[16] William, S. And Austin, T. 1999. Ontologies. IEEE Intelligent Systems, 1999 Jan/Feb: 18-19

[17] D.Martin, M.Burstein, J.Hobbs,et al .OWL-S: Semantic Markup for Web Services. http:/ /www.daml.org/services/owl-s/1.1/overview/.November,2004.

[18] 邓志鸿，唐世渭，张铭等. Ontology研究综述.北京大学学报 (自然科学版)，2002,9:38(5).

[19] OraLassila,Ralph R.Swick.Resource Description Framework(RDF) Model and Syntax Specification http://www.w3.org/IR/1999/REC-rdf-syntax-19990222

[20] Mike Uschold, 1998.Knowledge level modeling: concepts and terminology. The Knowledge Engineering Review，V ol.13 :1,1998:5-29.

[21] 岳昆，王晓玲，周傲英. Web服务核心支撑技术:研究综述软件学报，2004, 15(3):428-442

[22] A. Cali, D. Calvanese, G. De Giacomo, and M. Lenzerini. On the expressive power of data integration systems[A]. In Proceedings of 21st International Conference on Conceptual Modeling (ER2002), pages 338–350, Tampere, Finland, October 2002.

[23] Xiaofeng Meng, Dongdong Hu, Chen Li,Schema-Guided Wrapper Maintenance for Web-Data Extraction[A], ACM Fifth International Workshop on Web Information and Data Management (WIDM 2003),November 7-8, 2003, New Orleans, Lousiana, USA.

[24] Oded Shmueli.Architectures for internal Web services deployment[J]. In: Apers P, ed.Proc.of the 27th Int'l Conf.on Very Large DataBases. Roma: Morgan Kaufmann Publishers, 2001. 641~644.

[25] Perez A G，Benjamins V R,Fensel D.Overview of Knowledge Sharing and Reuse Components:Ontologies and Problem-Solving Methods.In:Stockholm V R,BenjaminsB,Cbandrasekaran A,eds. Proceedings of the UCAI-99 workshop on Ontologies and Problem-Solving Methods(KPR5)1999 1-15.

[26] Uschold M. Ontologies Principles,Methords and Applications. http://www.aiai.ed.ac.uk/project/pub/documents/1996/96-ker-intro-ontologies.ps Accessed:Nov,2004.

[27] Femandez M,et al. METHONTOLOGY:From Ontological Art Towards Ontological Engineering,AAAI-97 Spring Symposium on Ontological Engineering,Standford University,March 24-26th,1997:33-40

[28] http://www.w3.org/TR/owl-ref/#OWLLite.

[29] Martin D.et,OWL2S:Semantic Markup for Web Services[EB/OL]. W3C Member Submission, 22 Nov.2004.http://www.w3.org/Sub2mission/OWL-S/.

[30] SWARTZ A. MusicBrainz : A Semantic Web Services[J].IEEE Intelligent System,2002,17(1):76277.

[31] Jena Semantic Web Framework [EB /OL] , http://jena. sourceforge.net . 2005-08-26.

[32] 刘群，李素建. 基于《知网》的词汇语义相似度计算[J], Computational Linguistics and Chinese Language Processing, Vol.7, No.2, August 2002, pp.59-76.

[33] 柴晓路，梁宇奇. Web Services技术、框架和应用[M].电子工业出社，2003.

作者在读期间的研究成果

在硕士研究生期间取得的研究成果如下：

一、参加科研情况

1. 参与了基于Web Service的信息集成系统的框架的设计。主要负责该系统的整体框架设计、中介器的详细设计、异构信息源的服务描述、接口设计以及开发和测试工作。

2. 参与了软件资产管理系统的研发。主要负责入库需求分析和框架设计以及服务入库功能模块的实现和测试。

二、发表论文情况

1. 吴楠楠，徐学洲，陈静玉. 基于本体和服务发现的WEB信息集成 西安电子科技大学2006年研究生学术年会论文集. 已录用。