摘　要

本文以面向商家的托管式电子商务物流管理平台的研究与开发为应用背景，结合面向服务架构 SOA、Mediator/Wrapper 数据集成框架技术和 Agent 技术，深入研究了 TAR（任务-代理-资源）协同数据管理框架的设计，并采用面向 Agent 方法对其进行优化。通过工作流技术结合多 Agent 协作，实现了数据存取；采用本体树组织数据的存储结构，分别建立全局本体和局部本体以及之间的映射关系实现了数据存储。

首先，探讨了 TAR 数据协同软件架构的关键技术。一是研究了 SOA 和 SOA 开发技术 Web Service，给出了 TAR 数据协同管理软件基于 SOA 的层次结构模型。二是对面向 Agent 的软件开发框架进行深入研究，给出 KQML 的表示方法并结合应用背景对 KQML 原语进行扩充。三是给出了工作流建模方法与协同表示方法，并深入研究了角色协同和信息协同技术。最后建立了 Mediator/Wrapper 框架结构，给出 OWL 描述模板，探讨了全局本体和局部本体之间的映射关系，通过查询分解算法实现从语义查询到数据源查询的转换，实现数据的透明访问。

其次，对基于本体的 TAR 协同数据管理框架进行领域分析和领域设计。在领域分析中，首先结合面向商家的托管式电子商务物流应用背景给出了软件框架的功能模型，并对领域需求进行抽象与描述，给出软件用例模型。其次对关键用例进一步分解，给出基于工作流表示的物流订单管理活动模型和 BPEL 表示。最后构建全局本体和局部本体，提出基于相似度综合的本体映射方法，并对查询转换算法做了详细的分析。在领域设计中，首先建立对象结构模型与行为模型。其次运用 Agent 的技术对类模型进行了优化设计，对通信过程给出扩充的 KQML 原语表示。最后给出典型应用数据库设计和界面设计。

最后给出开发环境配置和典型应用实现。首先给出开发环境配置。其次给出数据管理软件的构件图，并采用 ACME 语言对模型进行可实现性描述。最后给出典型应用代码片段，展示了软件的实际开发过程。

本文所提出的 Agent 软件设计方法、Mediator/Wrapper 编程模式、本体映射方法和 TAR 数据协同框架都被应用到系统的开发中，具有一定的理论意义和工程实践价值。

关键词：协同数据管理，本体描述模板 OWL，本体映射方法， KQML 表示，多 Agent 协作，Mediator/Wrapper，TAR 框架

ABSTRACT

This paper presents the framework of cooperative data management software based on research and development of seller-oriented logistics management platform of e-commerce company.The software architecture,optimized by Agent-Oriented technology and ontology mapping method, implements by SOA/Web Service techonoloy,Mediator/Wrapper data integration framework and Agent-oriented technology.Workflow technology and Multi-Agent collaboration are used to achieve collaborative data access, ontology and mapping technology are to realize collaborative data storage.

Firstly,some important technology of TAR-oriented data cooperating are discussed,including SOA architecture and Web Service technology for establishing the sofeware model,Agent-oriented technology for class model optimization,extended KQML communicate method for Multi-Agent cooperation,workflow model method and role-oriented cooperative technology for the task scheduling,Mediator/Wrapper framework for the share of data,OWL description templates are used to establish the ontology tree,including global ontology and local ontology,and the ontology mapping is used to resolve the semantic heterogeneity.query process is to realize transparent access to data.

Next, the requirements analysis and design procedure of software model are presented.The sofeware functional model based on application background is set up.and at the stage of the functional model,the use case diagram and activity diagram are built.and then the global ontology and the local ontology are established based on OWL description template,in order to establish contacts between them,a mapping discovery algorithm based on the compositive concept similiarity computation and query transformation algorithm are proposed.The structure and behavior model are constructed by Object-Oriented methods in design phase. the Agent technology is to optimize the software architecture.

Finally, the toolkit and technique of software implementation is presented. The major components are presented and are described by ACME. Later on, the program implementation for the typical application in the system is presented.

The design and implementation of Agent-Oriented Optimization Method, Mediator/Wrapper framework and a mapping discovery algorithm, which are applied in the TAR Architecture of Cooperative Data Management, have theoretical significance and engineering practical value.

Key words: Cooperative Data Managerment, OWL, Ontology Mapping Method, KQML expression, Multi-Agent Communication, Mediator/Wrapper, TAR Framework

目 录

图表清单

注释表

英文简称	英文全称	中文注释
XML	Extensible Markup Language	可扩展标记语言
WSDL	Web Services Description Language	Web 服务描述语言
SOAP	Simple Object Access Protocol	简单对象访问协议
UDDI	Universal Description Discovery and Integration	统一描述、发现和集成协议
BPEL	Business Process Execution Language	业务流程执行语言
KQML	Knowledge Query and Manipulation Language	知识查询与处理语言
ADL	Architecture Description Language	体系结构描述语言
TAR	Task Agent Resource	任务代理资源
SOA	Service Oriented Architecture	面向服务的体系结构

第一章 绪论

1.1 研究领域的发展概况

近年来，协同计算一直是学术界讨论和研究的热点。协同计算是指通过先进的技术，比如计算机网络技术、计算机多媒体技术、通讯技术和群件技术共同构成协同计算的环境，使得不同时间和地域的人们可以协调一致地为了某个任务共同工作[1]。协同计算作为当前较新的软件研究的热点课题，它的协作不仅仅只是包括了人和人之间的，协同的思想具有更深入的内涵，除此之外还包括不同数据和资源、不同的终端的设备、不同的应用系统、不同的背景应用或者是机器和人相互之间等全面的协同。

数据管理是通过使用计算机的软硬件技术有效的对数据进行收集、存储、处理，最后提供应用的过程[2]。数据管理的目的主要是为了将数据的作用更高效地更充分地发挥。如何组织数据是高效地管理数据的关键步骤。伴随着计算机相关技术的不断地发展和扩充，数据管理最初是人工进行管理，随后采用文件系统的管理形式，后来又采用数据库系统的管理方式。数据库管理系统所构建的数据相关的结构，对数据与数据之间的内在的关系描述更加地充分和完备，方便了对数据进行必要的修改、及时的更新和扩展新的数据，大大降低了数据的冗余程度，一定程度上提高了管理数据的效率和共享数据的程度。20 世纪 60 年代后期以来，计算机管理的对象规模越来越大，应用范围越来越广泛，数据量急剧增长，同时多种应用，多种语言互相覆盖地共享数据集合的要求越来越高，在协同计算中协同数据的管理也越来越重要。

计算机系统结构沿着“单机单用户-单机多用户-多机系统-计算机网络-计算机互联、互操作-和协同工作”的方向发展[3]。而计算机互联、互操作和协同工作构成的网络计算和协同计算是计算机支持的协同工作的基础。从软件体系结构角度，传统的 C/S 系统结构中，服务器是网络的中心，而客户机是网络的基础，客户机依靠服务器获取所需要的资源，由于 C/S 结构软件的数据分布特性，客户端所发生的一切灾难性事件都成为可怕的数据杀手，C/S 结构的企业级的异地软件需要在各个区域安装多台服务器，且需要做数据同步的处理，因而只要有一个数据点上的数据出现安全隐患，整个应用的数据的安全都会受到威胁，因此 C/S 结构的异地软件对于大型应用的数据安全性是无法保证的且不是高效的。在数据一致性方面，一般大型的企业都是采用在各个区域安装服务器，接着再对各个区域的数据进行相关的同步操作。由此如果发生网络方面的故障，那么会引起部分区域的数据库的数据不能同步，数据无法在同一个时间点上保持一致性，因此不能较好地用于做决策性的分析。在数据实时性方面，C/S 结构不可能随时随地看到当前业务的发生情况，看到的都是事后数据。在数据溯源性方面，C/S 结构仅仅上传中间报表数据，在总部不可能查到各分支的原始单据。在服务器响应及时性方面，企业业务

流程、业务模式不是一成不变的，软件供应商提供的软件也不是完美无缺的，所以为了保证各程序版本的一致性，必须暂停一切业务进行更新，其服务响应时间基本是不可忍受的。

随着全球信息化的浪潮、信息化产业不断发展、延伸，SOA（Service-Oriented Architecture）系统架构的出现，将给企业信息化带来一场新的变革。在信息化的应用发展的历程中，出现过一系列的描述信息的标准如 XML、Unicode、UML，然而来自不同结构的系统的数据源的数据的格式还是保留各个独立的模式，因此在激励竞争且多变的市场环境下，企业的管理模式很难固化。应用传统的信息化软件，当企业要做出一些改动时需要面临巨大的挑战，SOA 的出现带来了新的改变，不再是各自独立的架构形式，能够轻松的互相联系组合共享信息。对于集成信息和数据资源是简便且有效的，将广域网或者是局域网上的网页、文档或者是目录轻松地进行集成，对于信息资源之间的协同性和相关性得到了加强[4]。与此同时也将原本成本高且复杂的集成转变为成本低而且较为简便的设定相关的参数，创造了一个全面集成的应用信息化的全新领域。SOA 是面向服务的架构，根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合以及使用。SOA 的实施可从企业外部访问，随时可用，粗粒度的服务接口升级，松散耦合，服务可重用，服务接口的设计，服务接口的标准化，支持各种消息模式，精确定义的服务契约[5]。

目前国内外的一些软件成品和研究论文都有关于协同数据管理框架研究的相关表述，结合文献资料，相应的关键技术总结如下：文献[6]提出将面向对象的设计思想用于计算机辅助的协同设计数据管理中，重点阐述数据的存储、查询、并发控制、版本管理等技术。文献[7]提出根据对分布式的产品数据管理进行分析后，构建面向协同服务平台的开发式的管理数据的框架结构，提出多个 PDM 之间的数据交互方法、数据安全性保障机制和数据的一致性解决策略。文献[8]根据面向对象的技术和 Web 产品数据管理的功能需求，提出一种以模块为核心技术，用以组织管理和发布信息的分布式的数据管理系统，同时对该系统的体系结构给以详细的分析。文献[9]提出一种采用多智能体技术的网络协同数据管理的模型，通过多智能体技术解决产品数据模型中数据异构性、协同性和网络化的问题，建立产品的数据管理模型和安全信息的体系结构，但是对于将多智能体如何作为软件模块有待进一步研究。

在语义网蓬勃发展的时代，特定的领域例如现代电子商务、供应链与物流和数字图书馆等领域，信息数据孤岛使得企业数据无法进行语义化沟通交流，因此如何利用数据协同管理提供领域内的语义化服务，解决缺乏语义描述的企业异构信息数据交换是亟待解决的问题[10]。

1.2 课题研究意义与主要研究内容

本文以现代电子商务平台应用为研究背景。现代电子商务平台包括以下模块：客户模块、商家模块、交易模块、支付模块、风险控制模块、物流模块和后台客服管理。本课题主要以面向商家的托管式电子商务物流管理平台为应用背景，主要是基于本体的 TAR（任务-代理-资

源）协同数据管理框架的设计与实现，重点解决系统中的基于本体的资源存储模式和基于 TAR 的资源存取管理框架的设计与实现，形成软件构架的研究专题。

1. 课题研究意义

现代电子商务领域将物流引入交易平台，不再是单纯的只提供给客户和商家交易的平台，还为双方提供相应的物流服务。托管式物流最核心的部分有两部分，一是面向客户发货和仓储功能，二是面向商家发货和仓储功能。本文主要是实现面向商家发货和仓储的功能，通过面向商家发货的方式，通过集中货量与供应商谈折扣的方式降低商家的物流成本，同时提高物流服务的质量，减少纠纷率。

本文使用本体技术将数据源使用本体树的形式表示，形成局部本体，从领域的角度构建全局本体，并提出一种综合相似度计算的本体映射方法将全局本体和局部本体关联起来。通过 Mediator/Wrapper 集成框架实现技术有效的集成平台中的异构数据源，采用查询分解与重写方法解决数据访问对任务请求的透明性。多 Agent 之间的协作实现了资源共享与调度的灵活性。面向服务架构 SOA 的实现技术 Web Service 实现数据信息的跨平台的交换与共享，为物流平台的请求用户提供统一的视图。采用面向 Agent 与面向对象相结合的软件开发方法，研究基于本体的 TAR 协同数据管理框架的设计与实现。因此具有一定的理论意义与工程实践价值。

2. 课题研究内容与方法

首先探讨面向商家的托管式物流领域知识和基于 SOA 的软件开发的理念，结合 SOA 架构和 Web Service 技术提出“任务-代理-资源”的协同数据管理框架，并运用面向对象建模方法进行 TAR 数据协同管理软件领域分析，构建 Mediator/Wrapper 数据交互共享的框架，同时给出了本体构建和本体映射方法，并给出相应的查询算法。其次在软件领域设计中引进 Agent 技术对类模型进行优化，并使用扩充的 KQML 原语对资源请求过程进行描述。最后给出基于本体的 TAR 数据协同管理软件的典型技术及典型应用实现。

第一章：绪论。阐述研究领域相关的背景及发展概况。结合本课题给出课题的研究内容与研究方法，列出研究的主要工作，并对各章节的内容进行安排。

第二章：TAR 协同数据管理软件的关键技术。首先探讨了 SOA 体系结构与 Web Service 技术，给出了基于 SOA/Web 服务的数据协同管理软件层次结构。然后阐述了面向 Agent 的软件开发框架和 KQML 表示方法，并且结合应用背景对 KQML 原语进行扩充，给出扩充后的 KQML 模板。接下来介绍了工作流建模方法和协同表示方法，给出“任务-代理-资源”的工作流元模型。最后分析了资源交互和共享技术，介绍了本体的描述方法并且结合应用给出了 OWL 的描述模板，数据交互采用 Mediator/Wrapper 数据集成框架。

第三章：TAR 协同数据管理软件的领域分析。首先建立了 TAR 协同数据管理软件的功能模型，对其功能需求进行了描述。其次采用 UML 用例图建立 TAR 协同数据管理软件相关用例模

型，并给出功能模型的 UML 活动图表示以及 BPEL 形式的活动模型描述。最后给出全局本体和局部本体的构建，提出一种综合的相似度计算的本体映射方法，并给出相应的查询转换算法。

第四章：TAR 协同数据管理软件的领域设计。通过对系统进一步分析，给出初始类模型，并对其采用 Agent 技术进行优化，并使用扩充的 KQML 原语对资源请求过程进行描述。然后给出软件的行为模型，顺序图来描述类之间的交互关系。最后给出关键数据库设计和界面设计。

第五章：TAR 协同数据管理软件的典型应用实现。阐述了系统所用的开发平台、软件运行环境的配置。给出了系统的构件图，使用 ACME 对系统中的相关构件给出可实现性的描述。最后列出系统典型应用代码实现部分。

第六章：总结和展望。阐述本研究工作的理论价值及其实际应用价值。总结本研究课题的收益点，并指出下一步研究的工作方向。

1.3 课题研究的主要工作情况

本学位论文从论文开题、相关资料的收集、研究方向的学习、该项目需求分析、模型研究、软件构架设计与实现、软件代码编写及软件测试工作到毕业学位论文的撰写与修改，总共历时一年多。完成该课题所做的主要包括以下几个部分。

（1）研究任务代理资源的数据协同管理相关背景、技术及相应的部分解决方案，学习并掌握了本体理论以及其在计算机支持的协同工作中的应用、Web 语义网技术、设计模式、Agent 技术、面向服务的 SOA 体系结构和 Web Service 实现技术、工作流的相应技术以及工作流描述语言 BPEL，知识查询和处理语言 KQML 和本体描述语言 OWL。

（2）研究软件形成的体系结构以及面向过程、面向对象、面向服务的区别，着重研究 SOA 和 Web Service 技术，结合本论文的具体应用给出软件架构模型。

（3）针对本课题研究相关数据集成技术及框架实现，运用 Mediator/Wrapper 数据集成框架和本体模型给出本课题对应的数据交互共享框架，在此基础上给出相关的数据共享策略，给出相应的本体映射方法和查询重写的算法描述及实现。

（4）分析了基于本体的 TAR 数据协同的领域功能模型，应用 UML 技术的用例模型和活动模型进行系统建模，形成系统需求分析报告。

（5）结合领域的需求报告，给出静态模型初始类架构，运用面向对象和 Agent 相关技术对类模型进行优化，设计系统的顺序图，形成基于本体的 TAR 协同数据管理的领域设计文档。

（6）学习了软件实现的相关技术，包括 J2EE、JSP、Ajax、JADE 和 eclipse 等。

通过本课题的研究，参与软件整体开发流程，能将软件架构相关领域知识很好的应用到实际的工程项目中。相关的理论知识能够更为深入的理解，为以后的工作奠定了良好的基础。通过论文的撰写，论文的写作能力也有所提高，收获颇多。

第二章 TAR 协同数据管理软件建模的关键技术

本课题的研究目的是通过在 TAR 协同数据管理框架中设计中引入 SOA 技术、Agent 技术、Mediator/Wrapper 编程模式和本体模型等来解决协同数据管理的问题。因此本章首先分析 SOA 技术及其实现技术，并给出了基于 SOA 体系结构的 TAR 协同数据管理框架。其次给出了面向 Agent 的软件开发框架，并结合多 Agent 之间的协作对 KQML 表示方法进行了扩充。随后探讨了工作流技术和系统表示方法，着重研究了信息协同技术。最后研究了资源交互与共享技术，包括本体描述方法、OWL 描述模板和数据交互共享方式，为本课题的后期研究奠定技术基础。

2.1 SOA 体系结构与实现技术

什么是软件体系结构？迄今为止没有一个公认的定义，不同软件体系结构学者提出了自己的概念和定义,但一般认为软件体系结构由如下实体构成:构件、构件间接口关系、限制、构件和连接件构成的拓扑结构、设计原则与指导方针[11]。

2.1.1 SOA 体系结构与设计理念

SOA 的英文全称为“Service Oriented Architecture”，中文翻译为“面向服务的体系结构”。W3C 将 SOA 定义为：“一种应用程序体系结构，在这种体系结构中，所有功能都定义为独立的服务，这些服务带有定义明确的可调用接口，可以以定义好的顺序调用这些服务来形成业务流程” [12]。由于 SOA 的两个领域，即业务领域与技术领域存在重叠，因此造成根据自身不同的需求对 SOA 进行不同的解释。SOA 不仅是一种现成的实现技术，而且是一种在计算环境中设计、开发、部署和管理离散逻辑单元模型的方法。理解 SOA，关键是要理解里面的“S”，即 Service 服务。服务可以说是一种既超越具体技术，又包含具体实现技术的业务功能。这些服务包括企业内部和外部的每一个业务细节，各个服务之间是可操作、独立、模块化、位置明确、松耦合且可以相互调用，不同其他的系统产生依赖关系[13]。

SOA 的优点主要体现在两个方面。第一，从技术开发角度，SOA 提供更加灵活的企业开发架构模式，屏蔽了业务逻辑组件的复杂性，具有跨平台和重用性，易于维护和良好的伸缩性，开发角色更加明确化，支持更多的客户端类型。第二，从资源整合的角度，SOA 体系结构可以依据现有的系统来发展，无需对系统重新创建[14]。

SOA 的基本特征包括三个部分。(1) SOA 的服务是一种可重用的组件，封装了企业的业务流程。一般分接口部分和实现部分。接口部分主要是服务提供者与访问者间进行程序访问的契约。实现部分则包括服务的 ID、服务的输入输出数据及这些数据在服务中的作用等。服务有五种类型：数据访问服务、组件服务、业务服务、复合服务、共享或企业基础结构服务。其中

数据访问服务是SOA架构中应用最广泛的服务，随着资源被广泛访问分享，数据层与应用层分离成为实现服务首先要解决的问题。数据访问服务使得用户对企业的各种关系型或者非关系型的数据源进行访问、集成、或者是转换，隐藏对数据源的直接访问转换和操作，同时也隐藏复杂的基本格式。这也是本文首要实现的服务，正是结合本文的应用背景以及任务代理资源这样一个方式，所以数据访问服务是我们使用SOA服务架构的依据，正是SOA这种特有的数据访问服务方式能极好的与本课题的应用相结合[15，16]。

（2）SOA 还有松耦合的特点。直观的说服务提供者和服务访问者能够借助定义良好的接口独立开发自身的应用。松耦合的优点就是灵活、当组成整个应用程序的一个服务内部结构和实现发生变化时，比如服务提供者更改服务的数据、接口或消息版本，但是不影响它的继续存在。服务的请求者也不知道服务的提供服务所使用的技术细节[17]。

（3）SOA 是一种粗粒度的服务，细粒度和粗粒度的区别是：细粒度是提供少量商业流程可用性服务，细粒度可用Web服务实现，也可用分布式对象实现；粗粒度显然比细粒度提供更多的服务实现功能，消耗的结构化数据或消息的数量不同，粗粒度不仅返回类似的消息或数据，还有内嵌的上下文。粗粒度服务不需要通过网络多次调用提供有意义的业务服务[18]。

2.1.2 基于 Web Service 的 SOA 实现技术

Web Service 技术是SOA的具体实现技术，SOA不等同于Web Service技术，Web Service技术只是其中一种技术实现形式。Web Service 组件不同于其他组件的根本之处在于 Web Service 采用一种标准的传输协议SOAP。开发服务组件的目的就是能够为客户端提供服务调用的功能， Web Service 实际上就是对调用的过程进行标准化的定义[19]：

(1)WSDL：即服务器端对所提供的服务的相关内容通过标准的方法向外界给予描述；

(2)SOAP：客户端调用服务器端的服务的协议是需要完全标准化的；

(3)UDDI：提供服务者通过一个公共的地址放置服务的内容，以便服务调用者查询。

顾名思义，Web 服务就是一个运行在 Web 上的服务程序。这个服务通过网络为程序提供远程调用方法。Web 服务实现了将软件作为服务的理念。Web Service 结构由三个角色和三个操作构建的。三个角色指的是服务提供者、服务中介、服务请求者。三个操作指的是发布、绑定和发现。Web Service 体系是一套协议栈构成的层次化的体系结构[20]。从下至上分别是网络协议、基于XML的消息、服务描述、服务发现和发布、服务流。右侧是各个协议层都要使用的公共的机制，包括安全管理和服务质量。底层是因特网传输标准协议。中间层是Web Service的相关标准协议，包括简单访问协议 SOAP，UDDI 统一描述、发现、集成协议以及 Web Service 描述语言WSDL。最上层是支持复杂的工作流和业务逻辑的工作流建模语言，如图2.1所示。

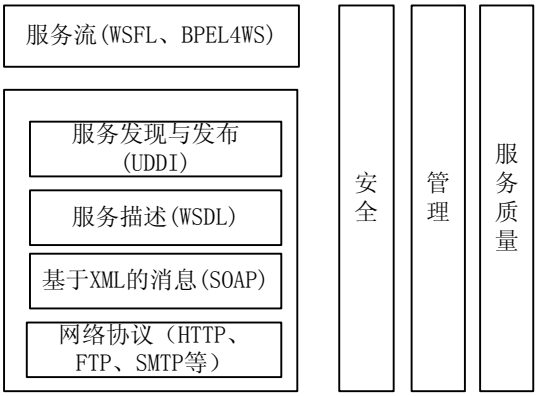

图 2.1 SOA 实现平台基本组件

Web Service 是 SOA 的一种技术实现，其中一个是 Web Service 和 SOA 的根本联系，即 WSDL，它是 Web Service 与 SOA 配套的接口定义的标准。由此引出基于 Web Service 技术的 SOA 三角模型[21]，如图 2.2 所示。

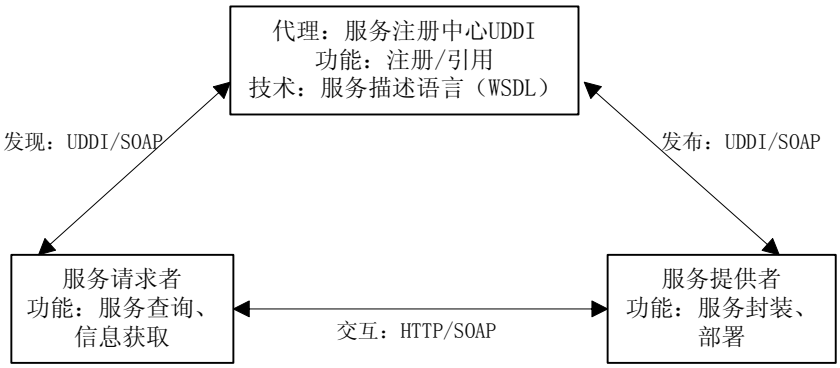

图 2.2 Web Service/SOA 三角型模型

下面给出三个角色之间进行交互操作的流程说明。服务提供者将服务的一些应用进行封装部署后，在服务注册中心进行注册，使用服务注册中心的服务描述语言 WSDL 对服务进行描述；服务请求者向服务注册中心发送信息获取的请求，服务注册中心将 WSDL 资源引用返回给服务请求者；服务请求者使用 HTTP 传输协议以 SOAP 消息的格式发出信息获取请求；服务提供者接收到请求后，响应请求，同样使用 HTTP 传输协议返回信息请求的结果给请求者[22]。针对交互中使用的协议给出如下的解释。

（1）WSDL（Web Service Description Language）

WSDL 是一种基于 XML 格式的关于 Web 服务的描述语言，主要用于 Web 服务的提供者将 Web 服务的相关内容通过发布提供给使用者，主要包括服务传输的方式、服务传输方式的接口、服务传输方式的接口的相关的参数以及服务传输的路径等，然后再对这些内容以一个完整的文档的形式生成。服务的使用者基于描述服务内容的文档，通过 HTTP 的协议将创建的 SOAP 的消息请求传递给 Web 服务的提供者。请求完成后，请求者接受到 SOAP 的消息返回，再依据之前的 WSDL 的文档将返回回来的 SOAP 的消息解析成自身可以理解的内容。

（2）SOAP（Simple Object Application Propotol）

SOAP 是 Web 服务的一个标准的通信协议，其传输消息的 XML 消息格式是标准化的。SOAP 的请求消息是将客户端的服务请求消息发给服务器，SOAP 答复消息是服务器返回给客户端的消息。

（3）UDDI（Universal Description、Discovery and Integeration）

UDDI 是一种创建注册表服务的规范，以便大家把自己的 Web Service 进行注册发布供使用者查找。

2.1.3 基于 SOA 体系结构的 TAR 协同数据管理框架

根据 SOA 技术在资源调度和信息共享方面的特性，在 TAR 数据协同管理框架中采用 SOA 技术和 Meidator/Wrapper 技术将系统中的异构数据源中的数据抽象封装，通过使用 SOAP 格式所定义的消息在 Web Service 之间传递，从而使系统能够及时、有效的获取资源，本文构建了基于 SOA 的 TAR 协同数据管理框架，如图 2.3 所示。

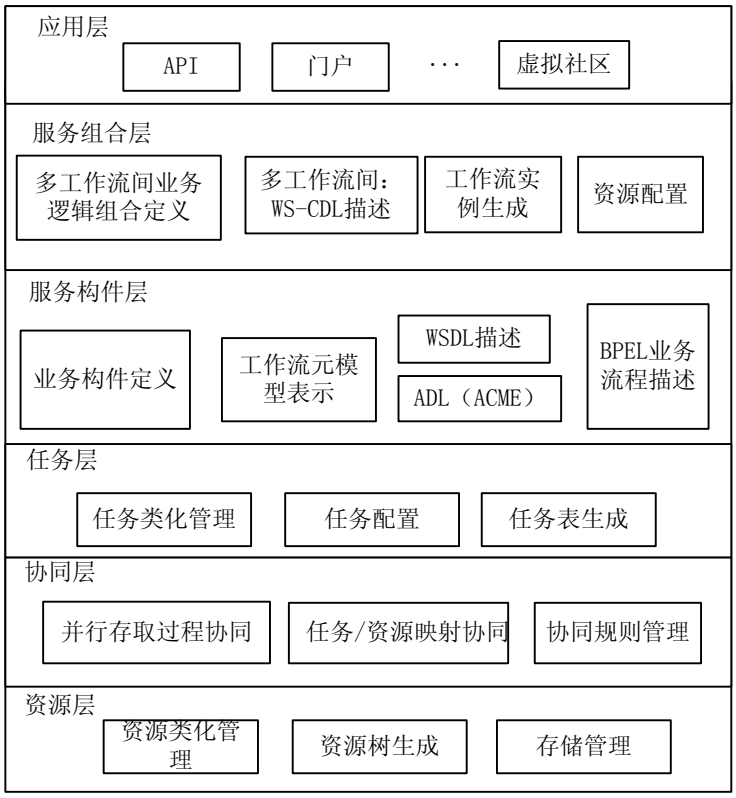

图 2.3 基于 SOA 的 TAR 协同数据管理框架

根据上图所示的框架图，下面将按照由下而上的顺序逐层进行详细分析。

（1）资源层：它是将物理层和数据层相结合而构建出的整个 TAR 协同数据管理的信息基础。通过利用协同层相关技术将企业内部的分布式数据虚拟化，从而使得企业内各种分布式异构信息得到充分的共享。主要包括：资源类化的管理、资源树的生成和资源的存储管理。资源

一般分为关系模式数据源和 XML 文件的数据源。使用本体对数据进行描述，生成针对不同数据源的资源本体树。

（2）协同层：协同层的主要任务是将资源层的相关数据能传递到任务层，包括并行存取过程协同、任务/资源映射协同和协同规则的管理。本文主要采用 Mediator/Wrapper 和代理域相结合的方法实现数据的协同，Wrapper 将底层数据源通过本体相关技术抽取成局部本体的形式，通过将数据源信息注册到 UDDI 注册中心，通过资源代理对资源进行监控，以便系统能够对资源变化做出及时的响应，Mediator 主要是承担任务层的资源请求并将查询进行分解和查询后的数据合成，以完整的查询结果返回给任务层。

（3）任务层：任务层包括任务类化管理、任务的配置和任务表的生成。通过对任务进行分类，再对相应的分类任务进行配置，最后生成对应的任务表。

（4）服务构件层：该层主要是提取具体应用系统的业务构件并对构件进行定义，采用工作流的机制对业务构件进行元模型表示，通过 ACME 描述各个构件以及之间的联系，通过 WSDL 对其接口进行描述，并使用 BPEL 对业务构件进行组合。

（5）服务组合层：该层主要是根据具体的应用业务逻辑需要，将下层提供的工作流进行组合，使用 Web 服务编排描述语言（WS-CDL）对组合后的工作流进行编排，通过编排生成实例，再通过资源配置完成资源匹配。

（6）应用层：应用层有多种方式对外界提供服务，主要包括 API、门户和虚拟社区等形式。本文主要采用门户的形式对外界提供服务。门户是一种应用框架，将各种应用系统、数据资源和互联网资源集成到一个信息管理平台，统一的用户界面提供给用户，能够快速建立企业与客户、内部员工、以及其他企业的信息通道，释放企业内外部各种信息。

2.2 面向 Agent 的软件开发框架

2.2.1 Agent 定义及其特性

为了提高 TAR 协同数据管理架构的智能性引入了 Agent 技术。Agent 技术来源于人工智能领域，一般人们认为 Agent 是将互联网技术与人工智能技术相结合的产物。Agent 技术目前广泛应用于各个领域，研究领域的不同导致对 Agent 概念的定义各不相同。来自 Agent 技术的标准化组织认为“Agent 是一种实体，这种实体是在环境中驻留着的，当环境中发生某个相关的事件时，它可以获取与之对应的数据，然后执行能够影响环境的相关行为[23]。”而研究软件 Agent 的人认为“具备智能性的软件 Agent 应具备一定程度的智能由此能够对部分的任务进行自主的执行，能执行用户的特定的任务，而且采用的与环境交互的方式是合适且可靠的[24]”。综合各方说法 Agent 即是一类能够在特定环境下感知环境信息，并能够和环境进行交互（作用或受作用于环境），为实现一系列既定的目标而灵活、自主地运行，并能够和环境中的其他

Agent 进行交互并协同工作的软件或硬件实体。

根据 Agent 的定义，智能 Agent 具有以下四个方面的特性[25, 26]。

（1）反应性：Agent 能感知自身所处的环境信息并能根据相关事件做出灵活反应。

（2）自治性，Agent 具备属于其本身的计算资源和局部于自身的行为控制机制，根据内部状态和感知到的环境信息，决定和控制自身的行为。

（3）协同性：在多 Agent 环境中，Agent 能与其他 Agent 通过规定消息格式进行通信，交互协同地完成共同目标，实现与其他 Agent 的合作协调。

（4）自适应性：区别于一般的传统应用程序，Agent 的行为是主动的。Agent 根据周围环境变化，主动产生目标，并且采取主动的行为。

通过前面对 Agent 定义解释以及 Agent 的一些特性的分析，下面针对 Agent 和对象的概念加以区别，主要分三点[27]。第一，对象同 Agent 最大的区别在于对象对环境的感应是被动的，一般通过消息的形式来触发对象活动，而 Agent 是一种主动感知外部变化并采取相应措施机制的实现，即 Agent 具备主动性。第二，对象通过外界驱动执行相应的动作，对于自己行为所产生的结果不承担任何责任，而相反 Agent 是具备自治性的，决定做与不做需要根据自身的利益或者目标决定，并且针对自己的行为需要承担相应的责任与代价。第三，Agent 具备对象所没有的理性、预动性、社会性等等。

针对上面对 Agent 与对象之间的区别的解析，在软件模型类架构时，可以将 Agent 用以辅佐类架构优化，将具备主动行为能力的实体抽象为 Agent（如资源注册等），将没有主动行为能力的实体抽象为对象。软件建模时，通过建立实体之间的相互作用关系，实现类架构设计。这种面向 Agent 的软件建模技术是基于一组以 Agent 为核心的概念体系，采用了一种新的方式来理解和认识一个系统，并对软件系统的建模提供一系列的机制、原则、技术和方法支持，面向 Agent 的软件建模将成为一种新颖的软件建模泛型。

Agent 关键技术问题主要分两方面：单个 Agent 的研究和多 Agent 系统的研究。单个 Agent 的技术主要是单个 Agent 的结构和如何构造单个 Agent。构造智能 Agent 方法有两种，一种是基于人工智能的方法，该方法是用符号表达系统中的对系统的环境和期望的相关行为，并且对这种表示使用句法规则加以处理以产生智能性的行为。实用推理 Agent 和慎思型 Agent 就是用的这种构造的方法。由于本课题不使用这两种 Agent，因此接下来主要介绍基于行为、情景的反应式的方法，这种 Agent 通过相关的事件驱动行为和对应的规则以适应对环境的响应。下面给出反应式 Agent 的单元结构模型，该模型是结合面向对象和构件化的思想，结合智能建模的方法学构建的。如图 2.4 所示。

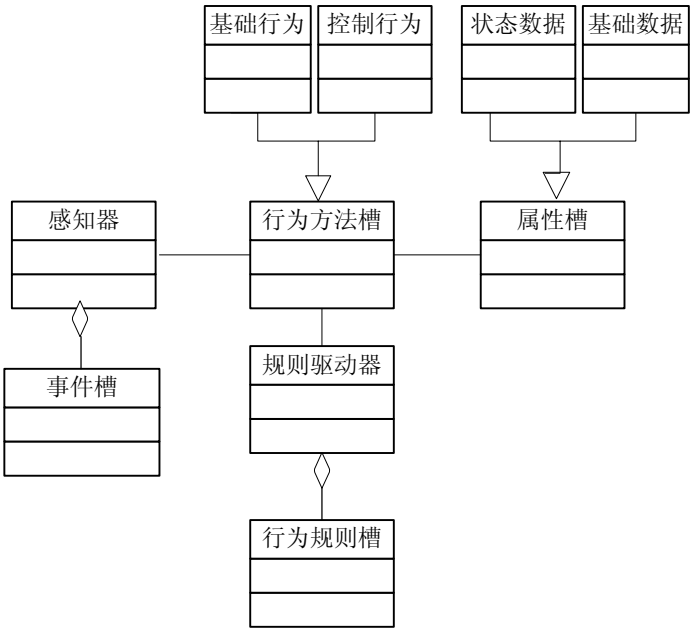

图 2.4 反应式 Agent 单元结构

Agent 的有四个单元部件组成：感知器单元、规则驱动器单元、行为方法槽单元、属性槽单元。其中每一个单元部件都由一个或者多个对象组成。单个 Agent 内部的运行机制是当有外界环境的输入动作时，感知器感知到后产生事先已经注册的某个事件，这些事件都装载在事件槽中。该被感知事件激发相应的规则驱动器，规则驱动器选择预先设定好的行为规则，对行为规则进行组合，并驱动行为方法槽中的具体行为执行动作，在执行动作时所使用的数据来自属性槽，属性槽中的状态数据是 Agent 自身状态所产生的，基础的数据是 Agent 执行任务时的数据。在使用单元 Agent 时，要详细的对每个 Agent 单元结构内部设计相应的实现方式。

2.2.2 KQML 表示方法及其扩充原语

由于单个 Agent 功能具有一定的局限性，它只能在一定范围内对环境的变化给予响应，当系统需求发生整体改变时，单个 Agent 缺乏相应的应变能力，仅仅对单个 Agent 的设计不能满足本课题的需求，无论是在传统的 SOA 框架中还是在主动服务的框架中，必须通过多 Agent 之间的合作来完成某个任务，因此多 Agent 技术是必不可少的。多 Agent 系统指的是具备不同目标的多个 Agent 相互协作，对其目标、资源等进行合理的安排，协调每个 Agent 的行为，能够在最大程度上对各自的目标得以实现[28]。由于多 Agent 系统中每个 Agent 本身具备一定的自治性，其本身具备能力和相应的知识且带有目标性，因而多 Agent 系统具备单 Agent 系统所不具备的能力，从某种意义上说是比单 Agent 系统优秀的一种系统。在多 Agent 系统中，个体 Agent 独立执行自己的任务，因此它们具备不同设计方法，而且也是使用不同的语言来实现。另一方面，每个 Agent 与其他 Agent 通信获取信息，相互协作解决整个问题。

多 Agent 之间的通信包括三个方面，交互协议、通信语言和传输协议。下面着重对通信语

言进行介绍。

1993 年 DARPA 的 KSE 外部接口小组 Finin 等人提出 KQML（Knowledge Query and Manipulation Language），KQML 是目前主要的 Agent 通信语言之一。KQML 规定了 Agent 之间消息传递的格式和处理消息的协议，支持 Agent 之间协作时信息的共享，从而达成异构系统间信息交互和集成。KQML 提供了一组协议用来识别、建立连接和交换信息，为多 Agent 系统通信和交互提供通用的一种通信框架。KQML 语言采用了“协议栈”的思想[29]，主要有如下三个层次。如图 2.5 所示。

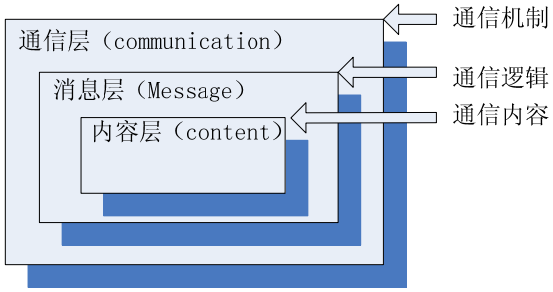

图 2.5 KQML 消息模型

内容层：描述 Agent 所传递的消息的内容，并使用表示语言表示出来。KQML 不关心内容中的具体的含义，可用任何可表示语言，正是这种内容语言的无关性实现 Agent 间的互操作。

消息层：消息层是核心层，用以明确传递消息的协议。由消息的发送方给出与内容有关的行为原语，确定具体的原语类型，同时还包括描述内容所使用的可选参数，比如使用的语言或者使用的本体等等。规定这些属性可以在内容透明的情况下，能够正确地传递消息。

通信层：描述一组与通信双方通信时的属性参数，并且对通信底层传输的信息进行编解码，该层位于基本的通信传输协议之上，且独立于传输协议。

表 2.1 KQML 保留参数及含义表

关键字	含义
: sender	原语发送者
: receive	原语接收者
: from	forword 中消息的传递的起源
: to	forword 中消息的传递的目的地
: in-reply-to	对前条消息的响应标识
: reply-with	对本条消息的响应标识
: language	原语内容所使用的语言
: ontology	消息所使用的术语的定义集
: content	原语传递消息的内容

一条 KQML 消息一般由一个语义动作、内容表达式和消息描述参数表示，KQML 定义了一些

行为原语，这些行为原语并不是 KQML 所有的原语表示，根据具体应用的需要可以添加新的原语，下面表 2.1 给出 KQML 的保留原语[30]。

在实际的应用中只有以上的消息原语不足以满足应用需求，根据应用需要对原语进行语义扩展，为了能够在任务代理资源的协同数据管理框架中对多 Agent 之间交互任务的完成，这里需要在原有原语的基础上进行一些扩充，使用特殊谓词对语义进行解析，下面对特殊谓词的含义进行解释[31]。BEL（Ag，St）表示对 Ag 而言，表达式 Pe 为真；KNOW（Ag，Pe）表示状态 St 对于 Ag 来说是可知的；WANT（Ag，St）表示 Ag 期望 St 所表示的动作或状态会发生；INT（Ag，St）表示 Ag 有意图使 St 所表示的动作或状态实现。下面给出扩充原语及其语义描述。

（1）request（A1，A2，P）A1 想要知道 A2 有没有意图 P

a.形式化表示，Want(A1,Know(A1,P))，P 表示 INT（A2，P）或￢INT（A2，P）

b.前提条件，Pre(A1)：Want(A1,Know(A1,P))

c.后继状态，Post(A1)：KNOW(A1, Know (A1, P1))，Post(A2)：Know(A2, Know(A2,P1))

d.已实现条件，Completion：Know(A1,P1)

（2）accept（A1，A2，P）A1 接受 A2 执行动作 P 的请求

a.形式化表示，INT(A1,P)

b.前提条件，Pre(A1)：INT（A1,P）∧KNOW(A1,WANT(A2,KNOW(A2,Q)))，Q 表示 INT（A1,P）或￢INT（A1,P）Pre（A2)：WANT（A2,KNOW（A1,Q))

c.后继状态，Post(A1)：KNOW(A1, Know (A2,INT（A1,P）))，Post(A2)：Know(A2, INT(A1,P))

d.已实现条件，Completion：Know(A2,INT（A1，P））

（3）reject（A1，A2，P）A1 告诉 A2 它没有意图执行 P

a.形式化表示：￢INT(A1,P)

b.前提条件，Pre(A1)：￢INT（A1,P）∧KNOW(A1,WANT(A2,KNOW(A2,Q)))，Q 表示 accept 的 Q，Pre（A2)：WANT（A2,KNOW（A1,Q))

c.后继状态，Post(A1)：KNOW(A1, Know (A2,INT（A1,P））)，Post(A2)：Know(A2, INT(A1,P))

d.已实现条件，Completion：Know(A2,INT（A1，P））

（4）trigger（A1，A2，P） A1 要求 A2 执行任务 P

a.形式化表示：WANT（A1,KNOW（A2,INT（A2,P)))

b.前提条件，Pre(A1)：WANT（A1,KNOW（A2,INT（A2，P)))，Pre（A2)：￢INT（A2,P）∧KNOW(A2,KNOW(A2,WANT(A1,Q)))

c. 后继状态，Post(A1)：KNOW(A1, INT(A1,P)),Post(A2)：INT(A2, P)) ∧KNOW(A2,KNOW(A2,WANT(A1,P)))

d.已实现条件，Completion：Know(A1,INT（A2，P）)

（5）comfirm（A1，A2，P） A1 告知 A2 已收到 P

a.形式化表示：WANT（A1,KNOW（A2,BEL（A1,P)))

b.前提条件，Pre(A1)：BEL（A1,P)) ∧WANT(A1,KNOW(BEL（A1,P)))，Pre（A2)：BEL（A2,P) ∧WANT(A2,KNOW(A2,Q))

c.后继状态，Post(A1)：KNOW(A1, KNOW（A2,BEL(A1,P)）），Post(A2)：KNOW（A2,BEL（A2,P))

d.已实现条件，Completion：Know(A2,BEL（A1，P）)

在以上给出的扩充的原语语义中描述了各条原语的逻辑顺序以及结束时的状态改变，通过对扩充原语的定义可以用以描述“任务-代理-资源”过程中各个 Agent 之间交互行为。

2.3 工作流建模方法与协同表示方法

2.3.1 工作流建模方法

工作流技术是通过将工作分解成定义良好的任务、角色，按照一定的规则和过程来执行这些任务并对它们进行监控，有效降低复杂业务过程系统的开发难度，提高企业生产经营管理水平和竞争力[32]。工作流技术具有在分布式异构环境下对复杂业务流程的处理能力，它可以管理、协调和控制一个虚拟组织的业务活动。由于工作流技术可以管理、组织和控制一个虚拟组织的业务活动，结合本课题任务-代理-资源的模式，可使用工作流对课题所发生的业务过程进行描述。工作流管理联盟组织（WfMC）提出的工作流元过程模型主要是用来对整个系统中业务过程进行描述[33]，如图 2.6 所示。

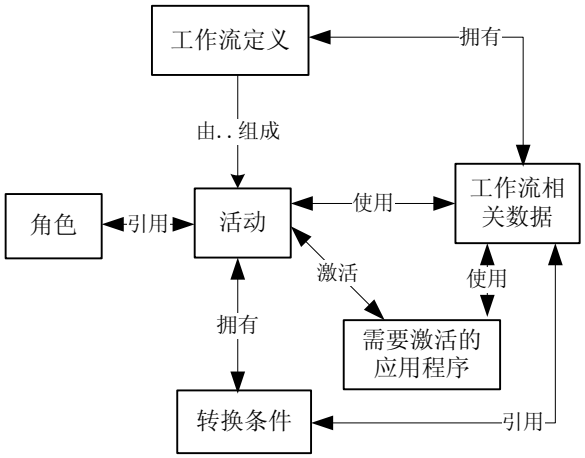

图 2.6 工作流元过程模型

由图可知活动是过程定义元模型的组成核心，工作流元模型包括以下几个实体[34, 35]。

（1）工作流过程类定义：定义活动过程的模型，一般包含如工作流模型名称、版本号、过程启动和终止的条件、系统安全、监控和控制信息等一系列基本属性。

（2）活动：它是工作流的重要组成成分，主要包括活动的名称、活动的类型（原子活动、子工作流等）、活动的前置/后置条件、活动的一些调度约束条件等。

（3）角色：一般是由参与者充当并操纵相应的活动，它主要属性有角色名称、角色的能力等。

（4）转移条件：主要为工作流中某个过程实例提供推进依据，主要包括的参数有过程推进条件、活动执行条件和通知发送条件等。

（5）被调用的应用：表示对所要完成的工作所采用的相关工具和手段，它的属性主要有应用程序的类型、名称、路径和相关参数等。

（6）工作流相关数据：工作流机可以根据相关数据来决定相关工作流实例的状态的转移变化，它主要具有的属性有数据的名称、数据的类型和数据的取值等。

根据工作流元模型的设计思想，结合协同数据管理框架实际业务流程相融合，提出 TAR 协同数据管理工作流元模型。如图 2.7 所示。

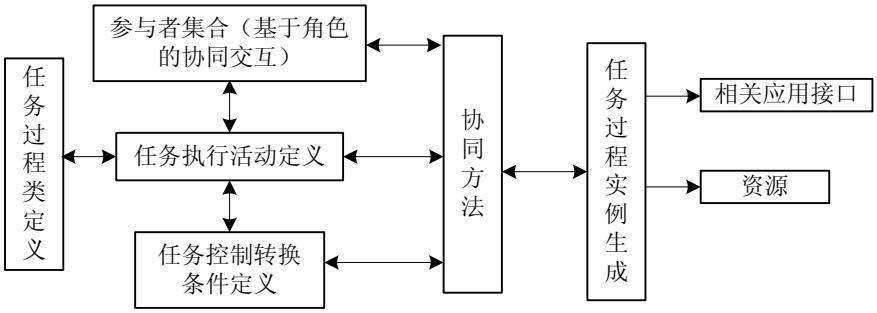

图 2.7 TAR 协同数据管理工作流元模型

在上面的框架中，从单个任务的角度出发，是将一个任务定义成过程类的形式，一个过程类有多个活动组成，通过参与者驱动任务执行的活动，并且通过控制转换条件在活动之间进行转换，形成的过程实例调用相关应用接口和相关资源完成一个任务的执行过程。从多个任务角度出发，任务形成多个过程类，形成多个任务过程实例，完成一个任务访问资源的请求过程，最终达到其目标状态。在实际的执行业务过程中，每一个企业的应用扮演着一定的角色、 行使一定的职责、执行具体的业务活动。一般情况下，每个任务过程包含着多个活动，其中这些活动处于多个异构的企业的运行平台和多个异构的数据库系统中，这些活动间的协同包括多个事务的处理，其中每一个相关事务的处理实际上就是一次互操作的行为，通过一定的协同方法解决互操作过程中的异构问题，最终完成整个业务流程的执行任务。

2.3.2 协同表示方法

协同计算是以群体协作为背景、以计算机和通信技术的发展和融合为基础、以应用领域广

泛为前提条件而形成的[36,37]。协同应用建模方法是实现协同的关键技术，从协作的方式和程度上讲，分为角色协同、信息协同、流程协同和计算协同[38，39]。结合本文 TAR 协同数据管理框架，主要涉及到了角色协同和信息协同，下面对两种协同方法给予阐述。

（1）角色协同：采用对象感知、访问控制和安全认证等方法，实现协同的主体之间的协作关系抽象和视图共享。其中角色是描述一个活动的参与者在群组中所拥有的权限和职责，表示人对协作环境的交互试图模式。角色协同的过程首先是按协作任务分解进行角色定义，按协作活动的规约进行角色协商，为每个用户分配 1-m 个可达成的一致角色，实现角色所扮演的工作任务即执行相关活动。按角色的协同属性描述，可以把角色定义为一个三元组形式，即

角色::={角色名，权限，职责，成员属性}

权限::={系统管理级，用户级，文件级，数据级}

职责::={角色所承担的特定任务}

成员属性::={成员标识，成员名，成员类型，扩展属性表}

成员类型::={人员，部门，系统，交互资源}

在本课题中，任务执行过程采用角色协同方法，任务权限控制采用基于角色的协同方法，把对资源和数据的访问权限授予角色。使用角色绑定，不直接对用户授权，明确角色的义务和权力，这样任务对资源的访问不依赖于具体的用户，在逻辑层将用户和访问权限分开，用户可以通过角色来动态改变自身所拥有的权限。而对于角色来说，角色之间也可以通过互相之间的交互对角色进行继承或者重用，以及在角色之间产生冲突的时候进行相应的处理，一般情况下考虑访问控制的安全性，避免角色之间的冲突。一个对资源的请求者不仅仅只拥有一个角色，而一个角色可以包含多个对资源的请求者，他们之间是多对多的关系，当资源请求者发出对资源的请求需求时，就激活该请求者被授予的角色或者是角色的集合，这样每个请求者就拥有所对应角色的访问权限，完成相应的请求操作。

（2）信息协同：根据信息交换，信息发布，信息检索，信息共享等处理内容的要求，提供内容与格式定义的规范，统一和协同的信息表示，以实现多数据源之间的数据互操作，达到数据资源共享和工作空间共享等目的。

信息协同的主要技术有表——树的映射框架，Map/Reduce 映射方法，数字中心方法。在本课题中，主要是采用表——树的映射框架。协同数据的管理就是使用了信息协同的方法，任务请求对数据资源的访问使用到的元模型表示、结构化数据表示方法、基于本体的描述语言的概念抽象与领域数据模型表示等都是信息协同的关键技术，采用这些技术完成任务对资源的请求。表到树的映射是指将底层以表的形式存储的数据抽取成本体树的形式供任务请求调用，本课题将底层的数据源抽取出局部本体，形成局部本体树，再将各个局部本体通过映射形成统一的全局本体，从用户访问层面得到的是一个统一的虚拟视图，使得用户更简便地访问资源。

2.4 资源的交互和共享技术

根据课题的应用研究，数据的协同不是单一的数据管理，而是指在分布式环境中异构数据进行交换与集合，形成一个虚拟的统一视图，从而进行数据的统一管理。比如在交易-物流业务协作的过程中，产品数据的数据源分布在不同的应用系统中，而这些数据源在数据的存储格式、结构和表达方式都有不同，导致数据的异构性。数据的交互共享有以下几个关键问题需要解决[40, 41]。

(1)统一的信息表示。实现不同系统之间交换数据的透明性，由于目前不同的系统之间，存储和表示数据的方式不一致，所以需要采用统一的表示方式。

(2)数据的异构性。数据异构一般分两种：①系统异构，主要是不同软硬件环境或者是不同的操作系统存在数据差异性。②数据语法异构，是指不同结构的数据源或者是不同的数据模型存在差异性。比如半结构化或者是关系数据库中的数据。③数据语义异构。语义异构是因为不同数据源中的数据在语义表示上有差异性。

(3)传输格式。数据协同采用的数据格式能满足各个系统之间同步或异步信息交换的要求，并能兼容网络系统和通信协议。

(4)数据交互安全性。要建立从数据格式、数据内容到网络传输等不同层面的安本防护机制，对传递的文档要基于某种规则进行验证。由于商业数据较敏感，一旦暴露对企业和个人会带来不必要的损失，因此需要对数据源的访问需要认证身份。

针对信息统一的表示和语法异构的问题，由于 XML 作为数据表示和交互的事实标准，具备结构性、扩展性、自描述性和平台的独立性，因此统一采用 XML 表示和交互不同应用系统的协作信息，与此同时采用公共的数据模型 XML Schema 解决数据源语法的异构问题。

针对语义异构的问题，采用本体对企业的异构资源建模，通过语义的映射保持分布的数据源语义的一致性，进一步解决异构的应用系统之间语义异构的问题。

针对系统异构和传输格式的问题，Web Service 技术具备强大的互操作性，它将服务请求者和服务提供者间的交互设计成完全语言和平台独立的，从而屏蔽两者在软件平台和操作系统的异构。简单对象访问协议（SOAP）和 HTTP 传输协议保证传输格式的一致性。SOAP 是由 W3C 公布推荐的一个基于 XML 的、在分布式的环境中交换信息的、简单的协议，在前面章节已经做详细叙述。

针对数据交互安全性的问题，WS-Security 协议为数据的安全提供了一定的保障。协议中规定了规约用来在 Web 服务消息中确保数据机密性和完整性。同时协议中描述怎么在 SOAP 消息中放入加密头和签名，应用层在对其进行处理，实现端到端的安全性，同时可以使用 HTTP 协议，屏蔽防火墙，适用性较好。CXF 对 WS-Security 有较好的支持，实现时定义 CallbackHandler 中的回调的方法且实现之，然后在 handle 方法中队身份进行鉴别，确保数

据交互的安全性。

2.4.1 本体描述方法

近年来，本体的研究日趋成熟。本体和具体的应用领域相关性高，所以各种文献从不同的问题领域和研究角度出发，给出不同的本体定义。综合起来，本体是在一定领域内的可以被用来共享、具有概念化和形式化特征的一种规格性的说明。本体的共享性是在本体中所体现的知识是能够被公共认可的和确定的。本体的概念化是指一个具体的概念系统中所包含的语义结构，是一组约束规则，这种约束规则是对某一种事实性结构的约束，且是非正式性的，可理解或者表达成是一组概念以及定义和概念间的关系。本体的形式化指的是计算机能够对其进行处理的一种表示方法。

1．本体的描述模型

本体建模核心是明确本体领域中的相关概念、概念属性、约束条件、概念间的层次关系等。一般具备以下五个要素：类（Classes）、关系（Relations）、函数（Functions）、公理（Axioms）和实例（Instances）[42]。

（1）类（Classes），又称概念。类是相似术语所表达的概念的集合体。类的含义广泛，如行为、推理过程、策略描述等。

（2）关系（Relations），关系在本体中非常重要，正是由于本体的概念之间存在复杂的语义关系，才将本体中的概念组织起来，本体中的关系表示领域中概念之间的交互作用。

（3）函数（Functions），是关系的特定的表达形式。函数中规定的映射关系，可以使得推理从一个概念指向另一个概念。

（4）公理（Axioms），公理通常都是一阶谓词逻辑的表达式。公理表示为永真子句。

（5）实例（Instances），代表元素，是概念的具体化，从语义上来讲实例表示的就是对象。

表 2.2 概念的关系表示

关系名	关系描述
part-of	表示概念整体和部分的关系。
kind-of	表示概念的继承关系。
instance-of	表示概念的实例与概念之间的关系
attribute-of	表示某个概念是另一个概念的属性。

只有满足以上条件的知识表示体系才可称为“知识本体”。如果在一个知识本体中没有有关函数和公理要素的相关说明，则只是一个简单的词汇表，或者称之为轻量本体。

其中，从语义角度，概念的基本关系有四种。如表 2.2 所示[43]。

在实际课题应用中，可以参考上面五元组进行本体的构造，不一定严格匹配，概念之间的关系也不局限于以上四类。

目前一般采用领域专家对行业内的信息进行归纳提取，手工方式构建。本体的构建主要遵循六个规则：（1）明确性和客观性，本体应该用自然语言对术语给出明确、客观的语义定义。（2）完整性，所给出的定义能表达特定术语的含义。（3）一致性，使用知识推理的方法产生的结果和该术语本身所拥有的含义不会引起歧义。（4）最大的非双向扩展性，增加专用的相关术语时，不会对已经存在的内容产生二次修改的行为。（5）最少约束，尽量少的对建模的对象规定相关的约束条件。（6）在构建本体过程中，需要领域专家的参与与协作。

常用的构建方法有很多，包括 TOVE 法、骨架法、KACTUS 工程法、SENSUS 法、IDEF5 法、七步法等，在此不一一赘述。本文主要结合七步法和李景教授在《本体理论在文献检索系统中的应用研究》中构建本体的步骤，给出以下六步：（1）确定构建本体的领域和目的。（2）确定本体中的概念。（3）确定本体中的概念间相互关系，得到概念层次结构。（4）构建相关的函数集。（5）构建相关的公理集。（6）利用 Protégé 构建本体。

2.4.2 OWL 描述模板

OWL（web ontology language）是 W3C 提出的语义互联网的本体描述语言标准，在 DAML+OIL 本体语言的基础上形成的，并且对 RDF（S）进行了扩充，避免了 RDF（S）存在的表达能力有限，存在语义冲突等缺点，相对于 XML、RDF 和 RDFSchema 具有更强的表达能力。例如，类型之间不相交性、基数、等价性，属性特征等。结合本课题 TAR 数据协同管理建模给出应用分析,OWL 在 TAR 数据协同管理软件本体建模过程中的优势主要体现在：

（1）XML 作为一种标准化的元数据语法规范，OWL 语法的格式依然采用 XML 的语法格式，解决了本体数据模型服务注册接口上的实现。

（2）通过在 Header Information 中引入 import，可实现本体的复用，避免其他本体子模型重复定义，OWL 的这一特性为模型扩展提供了可能性。

（3）对象类的语义信息通过 OWL 能够得到充分的表达。OWL 可实现对象类在逻辑上的运算（交、并、补等），更进一步地对其他相关的类进行必要的约束。

（4）对于对象类之间的关系，通过 OWL 能够得到充分的表达。OWL 对于关系的表示使用的是 Object 类型的属性，规定了每种约束的方法和规则。对象或者是实例为相应的属性值。

综上所述，正是由于 OWL 灵活的语法形式，对于课题中协同数据的本体建模起到了至关重要的作用，为了在建模过程中保证本体模型的一致性和标准化，将数据模型描述形式用模板的形式加以固定，便于数据管理软件本体模型的分工构建与维护，更好的实现交互性，因此给出本体建模的 OWL 模板。

OWL 采用元数据语义描述规范，数据协同软件本体建模按概念、属性、关系划分，描述模

板分为三类描述模板。

（1）概念描述模板

数据协同本体模型中类的描述主要从以下几个方面加以考虑：类的名称、当前类的父类、类的相关属性及属性之间的基本约束、类与类之间的关系等方面。下面给出概念描述的模板。

```
<owl:Class rdf:ID="ClassName">
    <rdfs:subClassOf rdf:resource="#FatheClassURI"/> //父类 URI
   属性引用声明和约束包
   关系描述包
</owl:Class>
```

（2）属性描述模板

数据协同本体模型中对象类的属性主要使用 Datatype 型对属性进行描述。Object 属性在对数据协同本体模型描述时只是当作表示类之间的关系的方法。主要从两个方面描述属性。

①描述属性一些基本的情况，如：所属的类的集合、数据类型、文字性的解释（中文含义、单位、备注等）。如下描述所示。

```
<owl:DatatypeProperty rdf:ID="DatatypeProperty">
    <rdf:type rdf:resource="&owl:functionProperty"/>//类的属性本身具备函数性
    <rdfs:domain rdf:resource="ClassURI"/>//用以说明属性属于某一个单独的类时，属性所属类的 URI
    <rdfs:domain> //属性如果不属于某一单独的类，有多个类都包括该属性，则应当逐一声明
        <owl:Class>
          <owl:Unionof rdf:parseType="Set">
          <owl:Class rdf:about="NameofClass1">
          <owl:Class rdf:about=" NameofClass2">
          ......
          </owl:UnionOf>
        </owl:Class>
     </rdfs:domain>
    <rdfs:range rdf:resource="DataType">//声明数据类型
    <rdfs:comment rdf:datatype="&xsd:string">相应的文字解释</rdfs:comment>
</owl:DatatypeProperty>
```

②当类引用属性时，类的模板中应该包括相关的引用的声明和约束，主要包括属性的取值

范围、读权限、写权限、属性的缺省值、强制性约束要求等等。属性的约束包和引用的声明里不仅要对 Datatype 型的属性加以约束，还要对 Object 型的属性加以约束和声明。如下所示。

```
<rdfs:subClassOf>
    <owl:Restriction>
     <owl:hasDatatypeProperty rdf:resource="DatatypeProperty">
       <owl:DefaultValue rdf:datatype==Datatype>value</owl:DefaultValue> //
       <owl:ConstraintRequirement       rdf:datatype="&xsd:string">value</owl:
owl:ConstraintRequirement>
       <owl:ReadableWirtable
           rdf:datatype="&xsd:string">value<owl:ReadableWirtable> //
      <owl:Upperlimited rdf:datatype=Datatype>value</owl:Upperlimited> //
       <owl:Lowerlimited rdf:datatype=Datatype>value</owl:Lowerlimited> //
    </owl:hasDatatypeProperty>
    <owl:hasObjectProperty rdf:resource=ObjectProperty/>
   </owl:Restriction>
</rdfs:subClassof>
```

（3）关系描述模板

基于 OWL 的数据协同本体模型中，使用 Object 型属性描述对象类之间的关系。对 Object 型的属性描述主要包括：①在类描述模板中的属性约束及引用声明处要对 Object 型的属性进行引用的声明，在上一个模板中有详细描述，在此不赘述。②在类的 OWL 关系描述模板中，需要描述 Object 型的属性的值约束和基数约束。如下所示。

```
<rdfs:subClassOf>
    <owl:Restriction>
     <owl:onProperty rdf:resource=ObjectPropertyURI/>
       <owl:minCardinality
rdf:datatype=="&xsd;Inte0067er">M<owl:mimCardinallty>
    <owl:Restriction>
<rdfs:subClassof>
<rdfs:subClassOf>
    <owl:Restriction>//min 约束
       <owl:onProperty rdf:resource="ObjectPropertyURI"/>
       <owl:allValuesFrom rdf:resource=="ClassURI"/>
```

```
    </owl:Restriction>
</rdfs:subClassof>
```

③描述 Object 型属性时，需要描述属性的函数性、属性所属的类、属性的传递性、属性的反函数和当前属性的逆属性。

```
<owl:ObjectProperty rdf:ID="ObjectPropertyName"
    <rdf:type rdf:resource="&owl;FunctionalProperty"/> // 表示属性的函数性
    <rdfs:domain rdf:resource="ClassURI"/>//属性的从属声明（单个类）
     <rdfs:domain> //属性的从属声明（多个类）
       <owl:Class>
           <owl:unionOf rdf:parseType="Collection">
              <owl:Class rdf:about="ClassURI_1"/>
              <owl:Class rdf:about="ClassURI_2"/>
              ......
           <owl:unionOf>
        </owl:Class>
      <rdfs:domain>
      <owl:inverseOf rdf:resource="#property_1"/> //声明属性的逆属性
      <rdfs:type rdf:resource="&owl;TransitiveProperty"/>// 声明属性具有的传递
    性
</owl:ObjectProperty>
```

2.4.3 业务间协作异构问题的解决方案

数据交互共享是数据协同不可或缺的部分。TAR 协同数据管理框架的交互共享数据方式主要是多数据源之间数据共享。数据交互共享主要关注以下三个方面。

1. Mediator/Wrapper 中间件技术作为数据交互共享的框架

使用 Mediator/Wrapper 中间件技术作为信息集成的方式是为了给用户提供一个统一个查询接口访问多种异构数据源，一般由一个中间件和多个包装器组成。一个包装器对应一个数据源，其中数据源可以是关系数据库、Web 数据源、XML 数据源等等。由于本课题主要的数据源是以关系数据库的形式存在的，所以底层主要的数据源是关系数据库。包装器是将每个子数据模型转换成公共的数据模型，对于应用层来说，是透明一致的访问机制。在中间件层，是以全局的视图形式提供给用户，因此用户发出请求时是针对全局模式的。中间件接受用户的请求并对其进行处理，根据事先规定好的资源映射规则，将查询进行分解成子查询，这些子查询对应到每个数据源，然后由包装器接受子查询并与封装的数据源进行交互。各个子查询由各个数据

源自身匹配的查询方式进行查询，并将查询产生的结果返回给中间件，中间件对各个查询的结果进行合并后提交给用户。采用这种技术由于包装器的封装功能，使得系统能支持大量的异构数据源，并且这些数据源是局部自治的，因此数据源的更新具有独立性。结合本课题的应用背景，给出结构图如图 2.8 所示。

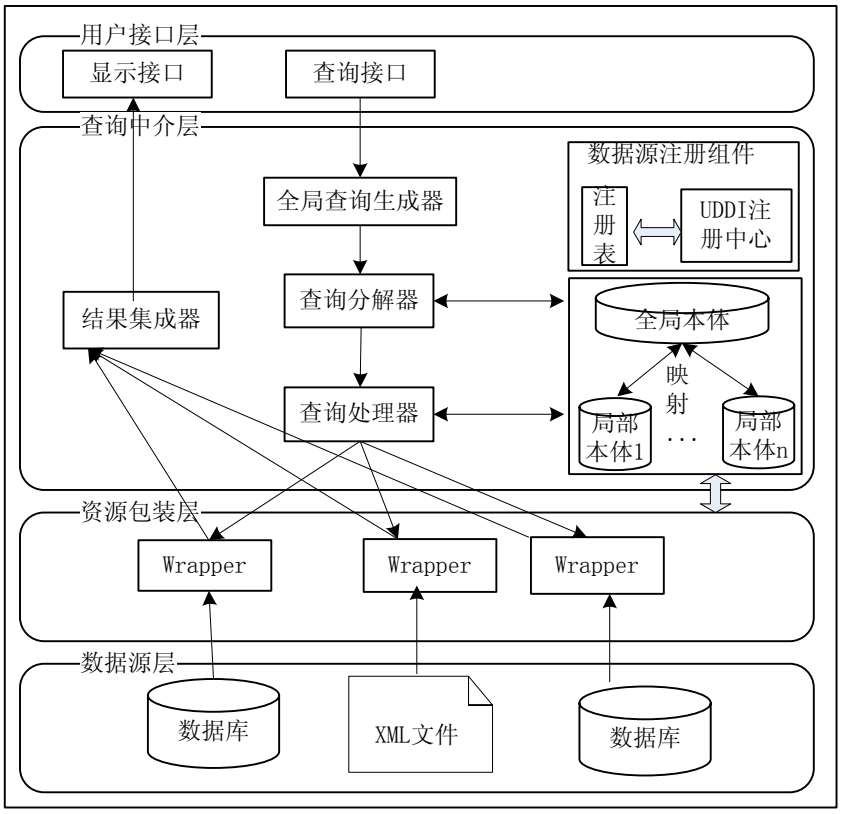

图 2.8 Mediator/Wrapper 结构

数据源层主要有关系数据源和 XML 文件形式的数据源，通过资源包装层的 Wrapper 对其进行包装后形成局部本体，再和领域的全局本体建立映射关系，用户接口层通过查询接口，生成全局查询，再通过两者之间的映射关系对查询分解后，交给查询处理器，查询处理器将交给 Wrapper 执行具体的子查询，Wrapper 向数据源发出 Web Service 请求，待子查询结果返回后，再交给结果集成器，通过查询中介层返回给用户。

2.采用一种综合相似度计算的本体映射方法

在 Mediator/Wrapper 框架中，查询分解器需要通过全局本体和局部本体之间的映射关系才能对查询进行分解。由于数据源信息资源语义描述的差异性，因此会产生异构数据间语义冲突。语义的冲突按照冲突的结构分成模式层和数据层。模式层的冲突是因为概念在不同数据源内用不同逻辑结构造成的冲突；数据层的冲突是由于对同一概念不同感知，在命名、标识符、精度、表示方式上面造成的冲突。因此需要通过本体映射来解决这一冲突。

处理本体映射的基本体系结构有三种：单本体方法、多本体方法、混合本体方法。单本体

只是一个全局本体，底层数据源都关联到这个全局本体。多本体结构是指每个数据源都有对应的局部本体，这些局部本体之间存在松散的联系。混合本体是指集成的系统有一个全局本体，每个数据源有自己的局部本体，这些局部本体同全局本体通过本体映射保证之间语义的一致性。综合三种方式，单本体较简单，结构也相对清晰，但是在实际应用中难以实现，一个信息源发生变化，全局本体都需要改变，不易于维护。而多本体方法因为没有全局本体的一致性，所以可以独立建立不用顾及其他局部本体，但是不同本体系统之间没有显示联系，完全独立建立，语义关系不方便建立，很难有效集成。混合本体方法的优势显而易见，容易添加新数据源，相关局部本体及其映射关系，共享词汇表使得源本体易于兼容，由于结合本课题的实际应用，本文采用混合本体的方法对异构数据源进行集成。

结合混合本体提出一种综合的相似度计算的本体映射方法，该方法首先抽取局部本体的特征，再在此基础上对其进行三种相似度的计算，包括：概念的相似度计算、属性的相似度计算和实例的相似度计算。通过相似度综合后生产映射，再对所生成的映射进行优化与修正，映射输出后匹配到全局本体。本文在 3.5.2 节给予详细的叙述。

3.查询转换方法

在对局部本体和全局本体建立好映射之后，需要解决如何在查询时将全局本体的查询转换成对各个数据源的查询。合理的查询分解转化能优化查询，并能对查询结果转化为统一格式起到关键性的作用。本课题采用的是 SPARQL 本体查询语言，具体的 SPARQL-SQL 之间的转换算法在 3.5.3 节给出。在此不赘述。

2.5 本章小结

本章研究了 TAR 协同数据管理软件建模所用到的关键技术和方法。首先探讨了 SOA 体系结构与 Web Service 技术，给出了基于 SOA/Web 服务的数据协同管理软件层次结构。然后阐述了面向 Agent 的软件开发框架和 KQML 表示方法，并且结合应用背景对 KQML 原语进行扩充，给出扩充后的 KQML 模板。接下来介绍了工作流建模方法和协同表示方法，给出“任务-代理-资源”的工作流元模型。最后分析了资源交互和共享技术，介绍了本体的描述方法并且结合应用给出了 OWL 的描述模板，数据交互采用 Mediator/Wrapper 数据集成框架以及提出采用综合相似度计算的本体映射方法。本章探讨的技术、方法与策略是后面探讨数据协同管理软件设计与实现的基础。

第三章 TAR 协同数据管理系统的领域分析

本章将通过对 TAR 协同数据管理系统领域需求的分析给出领域模型，将领域模型同软件建模方法和工作流技术进行系统模型的建立，构建领域的用例模型、选取典型的活动给出其活动模型，并结合工作流描述语言对系统的核心活动的业务流程给出 BPEL 形式化描述。结合本体建模方法构建局部本体和全局本体，给出本体映射方法和查询转换算法。实现了从软件的需求域到实际软件域、从需求的概念模型到实际的软件构架模型的过程。

3.1 基于领域工程的 TAR 协同数据管理框架的需求模型

领域功能模型是在分析系统需求的基础上，结合领域内的应用系统的特征，从而抽取需求构建功能模型。本节首先对协同数据管理的需求进行分析，然后构建系统的功能模型。

3.1.1 托管式物流的领域知识

电子商务是指在全球各地商业贸易活动中，在因特网开放的环境下，买卖双方不谋面地进行各种商贸活动，实现消费者网上购物、商户之间的网上交易和在线电子支付以及各种商务活动、交易活动、金融活动和相关的综合服务活动的一种商业运营模式。现代电子商务企业不仅是提供交易的平台，更是推出一种全新的物流模式，即一站式的仓储物流解决方案，通过这种方式，商家接到客户的订单后，只需将货物发往电子商务企业的指定仓库，货物的后续安排均有电子商务企业根据商家提供的货物申报信息及客户的指令完成。

使用这种模式主要优势在于：（1）操作一步到位，新模式之前商家需要花时间了解各个地区不同的物流费用，新模式下，不用再计算复杂的国际物流费用，只需设置国内运费和商品销售价格。（2）减少资金滞压，中国商家将客户所要的货物发往电子商务企业的仓储中心，剩下的由电子商务企业同客户进行沟通，资金结算。减少商家的资金压力。（3）安全的货运模式。商品通关、发货较以往更方便，避免了传统物流模式常发生的丢货、发错货现象。

托管式物流最核心的部分有两部分，一是面向客户发货和仓储功能，二是面向商家发货和仓储功能。本文主要是实现面向商家发货和仓储的功能，通过面向商家发货的方式，通过集中货量与供应商谈折扣的方式降低商家的物流成本，同时提高物流服务的质量，减少纠纷率。

面向商家发货和仓储的功能模块包括：货物订单管理、物流订单管理、交易评价管理、业务过程协同管理、资金管理、账户管理。如图 3.1 所示。

结合托管式物流的功能模型，下面对各个模块的功能以及之间的关系进行逐一介绍。

（1）货物订单管理：该模块主要是商家对自身拥有的产品在电子商务平台上发布展示给客户，以供客户进行选购，客户下单后，形成货物订单，商家将该订单进行处理后以预告的形

式传递给仓储中心，以便仓储中心随后形成物流订单，同时在后台填写相应的预告订单信息。后台工作人员对商家的订单进行业务过程协同跟踪。订单类型包括：所有订单、进行中的订单、已结束的订单、冻结中的订单、退款和纠纷订单。

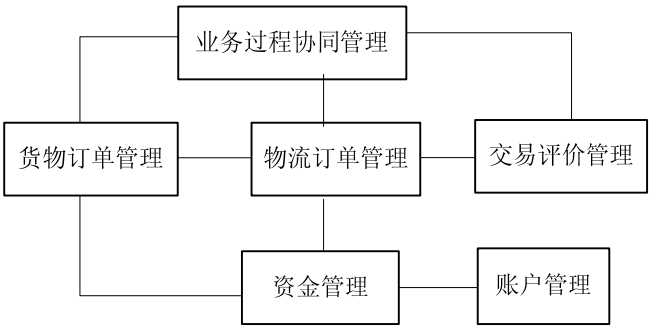

图 3.1 面向商家托管式物流功能图

（2）物流订单管理：商家发货到仓储中心，如果商家已经填写了货物的预告信息，且货物完好无损，货物将放入仓储中心的货架，待商家确认货物已被仓储中心接收确认后，仓储中心安排物流发货，形成对应的物流订单。商家、仓储中心、平台后台人员、客户、物流公司同时跟踪物流订单信息。

（3）交易评价管理：客户同商家达成交易时，可对商家的货物、信用等作出评价，商家通过后台可对交易评价进行管理。

（4）业务过程协同管理：主要是电子商务平台的后台人员跟踪交易情况，从客户下单、商家发货、货物进入仓储中心、商家确认货物进库信息、仓储中心发货、仓储中心确认物流信息、商家反馈物流信息等。在商家、客户、仓储中心和平台之间协调处理交易中出现的问题。

（5）资金管理：商家用来同客户进行交易支付和仓储中心进行物流资金交易的模块。

（6）账户管理：收款账户管理，商家接收货款时设置的账户，对账户可进行增添、删除、修改和查询的操作。运费账户管理，商家通过托管式物流方式需要支付给电子商务平台相应的物流费用，运费账户可进行添加、删除、修改和查询的操作。

3.1.2 基于 SOA 的领域功能模型

通过 3.1.1 对面向商家的托管式物流的业务描述，结合“任务—代理—资源”的数据协同管理思想，在本节给出基于 SOA 的协同数据管理功能模型，以 SOA 三角形模型为系统的理论模型，以 Web Service 为其具体实现技术，将电子商务相关数据信息以 web 服务的方式对外发布，使用 WSDL 对数据服务的应用功能与调用方法加以描述，并将其注册在 UDDI 注册中心以供任务进行调用。如图 3.2 所示。

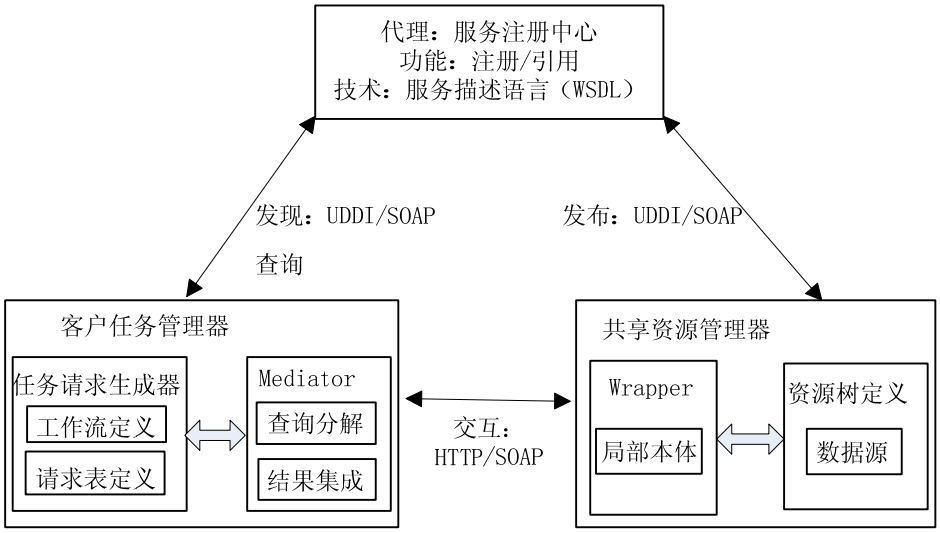

图 3.2 基于 SOA 的 TAR 协同数据管理模型

使用该模型的优势在于：（1）符合 SOA 松耦合的特点。任务请求方首先通过间接寻址找到数据源的信息，再通过直接寻址找到对应的数据源。当组成整个应用程序的一个服务内部结构和实现发生变化时，服务的请求者也不知道服务的提供服务所使用的技术细节。（2）采用了 TAR 框架结构。资源是服务提供者，任务是服务请求者，通过代理实现任务到资源的存取匹配。（3）将 SOA 的体系结构较好的运用在电子商务领域。商家和客户都是通过“任务-代理-资源”的方式完成信息交互。（4）将 Mediator/Wrapper 编程模式融入 SOA 体系结构。Wrapper 包装局部数据源，Mediator 负责对查询进行分解，并且合成查询结果。

在共享资源管理构件中，托管式物流信息如货物订单信息、物流订单信息等数据被抽取成局部本体封装，并将数据源的信息注册到服务注册中心，注册中心还存放全局本体以及全局本体同局部本体之间的映射关系，以提供全局本体的统一视图供用户查询。在服务注册中心，提供数据源的注册以及数据源的查找功能，为了更方便用户准备定位到需要的资源，注册中心按照不同的应用对资源进行分类管理，以便于不同的应用准确地匹配到对应的资源，同时不同权限的用户（全局本体管理者，局部本体管理者）可以对不同的资源层级进行管理。在任务请求管理器中，针对不同的应用经过 Mediator 的处理，将具体的应用匹配到具体的数据资源，同时也合成各个数据源返回的局部数据，以统一的数据结果返回给任务方。结合托管式物流的基于本体的 TAR 协同数据管理软件架构共有六个功能模块。领域需求的功能如下：

（1）资源封装：将货物订单信息、物流信息、资金信息等相关资源封装为 Web 服务。

Wrapper:：=(资源发布，XML 局部本体，数据库局部本体，映射表)

（2）资源注册：对资源进行注册，包括 WSDL 描述，资源调用的权限，全局本体和局部本体之间的映射规则，提供注册数据源和查询数据源界面。

资源注册表:：=（WSDL 描述，资源分类，请求分类，本体映射规则，资源相关描述）

WSDL 描述::=（端口类型，操作，输入，输出，目的地，端口号）

资源分类::=（按货物的所属的行业类目分类法）

服务映射规则::=（按语法相似度，按语义相似度，按结构相似度，按属性相似度，综合规则）

（3）资源请求处理：对简单数据的查询，根据注册信息，直接提供相应服务；对存在语义形式的查询，要根据全局与局部本体的映射表对查询进行处理，分解后再对相应的资源进行查询访问。对异常请求拒绝访问。

访问请求::=（简单数据请求，语义数据请求，更新数据请求，追加数据请求，删除数据请求，异常请求处理）

（4）数据分析：对各节点的信息进行综合统计分析，分析结果以服务形式进行发布。

分析分类::=（产品数据，物流数据，预报数据，资金信息，账户信息，交易评价信息）

（5）客户任务管理::=（货物订单管理，物流订单管理，资金管理，账户管理，交易评价管理，业务过程协同管理）

（6）业务流程管理：通过业务流程执行语言（BPEL）对托管式物流管理过程进行编排。

3.2 TAR 协同数据管理的用例模型

根据前面几节对 TAR 协同数据管理系统的领域分析及其功能模型的构建，本节将通过使用统一建模语言（The Unified Modeling Language，UML）来对 TAR 协同数据管理系统进行用例模型的构建。用例模型的构建是 UML 建模方法的一个基础部分，它主要是用来对系统或子系统的行为进行建模，描述系统执行者所理解的系统总体功能模块及各模块之间的交互关系[44]。用例建模通过描述系统中各元素的外部视图来展示各元素在系统中的上下文语境，使系统或子系统的抽象和划分更容易被开发人员理解，所以用例图一般都用在需求分析阶段，使需求分析更加清晰从而使后续软件开发目标也更加明确，确保在软件开发过程中对系统所具有的功能有准确的定位。

对于一个用例模型可以分为用例图和用例描述两大部分。用例图是用来描述多个用例之间的关系，它是通过画图的方式来完成；用例描述是用来详细描述用例图中每一个用例的详细信息并用文档的方式来表示。对于一个用例图一般包括以下六种基本元语[45]：

（1）主题（Subject）：它是使用一组用例以及它们之间的关系来进行描述的一个模块，通常是指一个系统或者子系统。

（2）用例（Use Cases）：它是用来描述一个系统（子系统、类或接口）的外部视图，即系统的功能需求和系统内典型事物的处理流程，但是用例并不指明其内部的具体执行过程。

（3）执行者（Actor）：它是指外部用户在与系统中用例进行交互或使用系统时所要扮演的角色。

（4）包含（Include）：主要是通过使用被包含的用例来对多个用例都含有的动作或行为片段进行封装，从而实现复用。

（5）关联（Association）：主要是用来描述用例之间或用例与执行者之间的双向事物流关系。

（6）扩展（Extend）：通过使用扩展用例来对基用例中的可选择的动作或行为片段进行封装，再将扩展用例进行扩展。

系统的用例图可以根据不同的抽象层次进行分割，对于一个系统来说一般它具有三个层次级的用例，它们分别为系统级用例（系统-子系统）、子系统级用例（子系统-功能）、功能级用例（功能-子功能）[46,47]。对于同一张用例图来说，它所包含的所有用例的抽象层次必须相同，否则将会导致用例粒度大小不均。通过使用抽象层次的概念，使得在相同层次的用例可以通过进一步的细分成子系统用例，从而能够清晰的反映整个系统的层次结构。

用例图一般采用分层建模，顶层用例还可以进一步划分为更详细的子用例，图 3.3 给出了数据管理用例模型，主要包括：①任务请求。②查询中介。③资源包装器。④数据源。⑤资源发现。⑥资源发布。⑦资源注册代理。

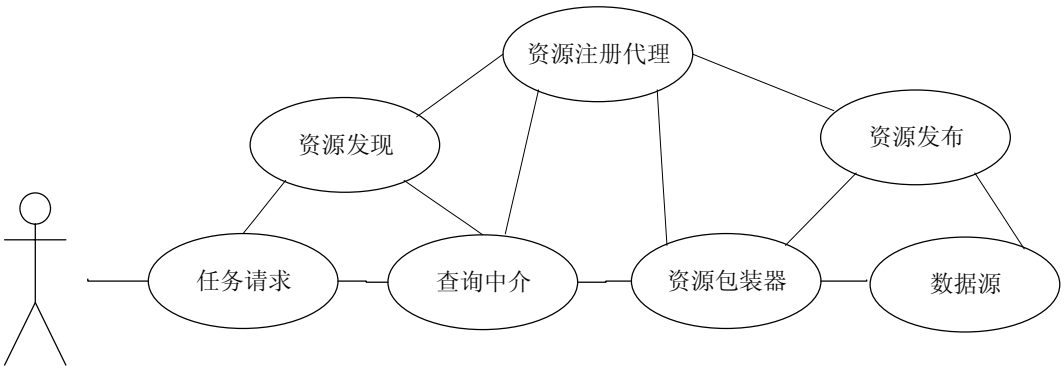

图 3.3 基于 SOA 的数据管理用例模型

图中的用例是对数据管理用例图的图符表示，下面将通过用例图描述并结合工作流技术对上图的用例模型做进一步的说明，用例图描述主要包括用例说明（描述用例所具备的功能/目标）、用例间的相互关系描述以及一些相关的非功能性的说明等信息。

（1）用例说明（用例/子用例的分割说明）

①任务请求：主要包括请求表的定义和工作流相关内容定义。

②查询中介：主要是负责全局查询的生成、查询分解、查询执行以及最后子查询结果合成。全局查询的生成根据用户的选择生成全局的统一查询模式，获取数据源的信息，根据映射规则将查询进行分解，将分解后的子查询传递给数据源以便针对各个数据源进行查询。

③资源包装器：是对底层数据源进行封装的容器，主要包括两类数据，一是数据库数据，二是 XML 文件格式的数据。资源包装器内的数据是通过将底层数据库的数据或者是 XML 文件格

式的数据通过本体映射的形式以 OWL 的语言进行描述的本体资源树，即局部本体。

④数据源：底层数据源，主要包括两类数据，第一类是数据库数据，第二类是 XML 形式的文件数据。

⑤资源发现：用户提出资源请求时，从资源注册中心查找相应数据源信息。

⑥资源发布：资源包装器将包装好的局部本体、全局本体及相应的映射表发布到注册中心，供用户请求数据使用。

⑦资源注册代理：主要负责数据源的注册和数据源的查找功能。

（2）用例间关系描述（工作流描述）

①数据源-资源包装器-资源发布-资源注册代理

数据源分为关系数据源和 XML 文件形式的数据源，两种形式的数据源分别包装成两种不同的局部本体通过包装器包装后发布到资源注册中心，同时注册的全局本体和局部本体之间的映射规则，资源注册中心对发布的数据源进行注册、分类和维护。

②任务请求-查询中介-资源注册代理

用户通过在注册中心查找所需要的资源完成相应的资源请求，比如商家需要查看货物订单的信息以及货物的物流信息，货物的订单信息和物流信息事先已经注册在注册中心，用户直接查找即可得到想要的信息。

③任务请求-查询中介-资源包装器-数据源

用户按角色分为商家、仓储中心驻场人员、平台后台客服人员、客户，每个用户对资源拥有不同的访问权限，通过各自的资源请求入口进入系统完成相应的资源请求任务，查询中介通过查询重写将全局查询传递给资源包装器，分解为相应的局部子查询，局部子查询再通过转换对应到具体的数据源，数据源完成查询请求将结果传回给资源包装器，资源包装器再将结果返回给查询中介，查询中介对结果进行整合后以统一的视图交给用户，完成整个资源请求过程。

（3）相关说明

本系统通过使用 UDDI 来注册和发布服务信息。将数据资源包装成有状态的资源，通过将包装好的资源在 UDDI 上进行注册和查找符合要求的资源实现对资源的有效管理。

基于上一个用例图是用以说明数据管理的模型，针对托管式物流管理应用，图 3.4 给出了相应的用例图。

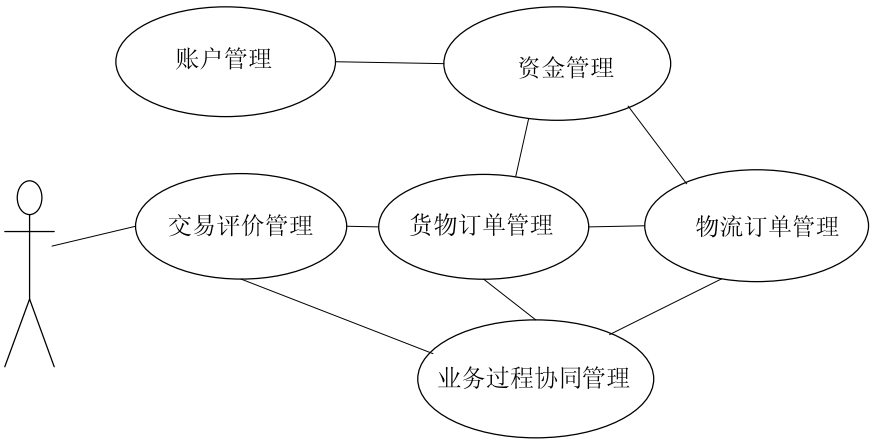

图 3.4 面向商家的托管式物流用例图

（1）用例说明（用例/子用例的分割说明）

①货物订单管理：货物订单根据交易情况分为所有订单、进行中的订单、已结束的订单、冻结中的订单以及含退款和纠纷的订单。在订单管理中设置采用托管式物流方式可以降低物流费用，商家具有设置相应折扣的权限，通过设置相应的折扣结算货物订单的费用。客户选择托管式物流方式，由商家将货物发往仓储中心统一物流。

②物流订单管理：商家在同客户交易过程中确定具体的物流方式（UPS、EMS、DHL 等）并设置相应的折扣，商家将货物发往仓储中并入库后，仓储中心会反馈给商家货物入库信息，商家在后台能查看到货物物流具体信息，确认无误后支付相应的物流费用。

③资金管理：商家在电子商务平台上交易的资金状况，用于商家在交易时查询资金、转移资金和存入资金使用。

④账户管理：收款账户管理，商家和客户交易时，客户应付给商家的交易金额，存入收款账户中，商家对收款账户具备管理的权限，包括增添账户、查询账户、删除账户和修改账户；运费账户管理，商家通过托管式物流方式运送货物，由电子商务平台的仓储中心统一发货，商家只需交给仓储中心相应的费用，该账户用于管理托管式物流方式所产生的物流费用，商家可查看相应的运费情况。

⑤业务过程协同管理：电子商务平台的客服人员需要对买卖双方进行交易的情况进行跟踪，对买卖双方在交易过程中发生的纠纷或者是异常给予仲裁等。

⑥交易评价管理：客户对商家的货物、服务、信用等的评价。商家对交易相关的评价具备知情权，可同客户进行沟通。

（2）用例间关系描述（工作流描述）

从图 3.4 可以看出“货物订单管理-收款账户管理-业务过程协同管理-物流订单管理-运费账户管理-交易评价管理”这一主业务工作流线。首先客户下单产生货物订单，商家通过后台查看相关货物订单情况、收款账户情况和资金情况，然后填写预告信息到仓储中心并发货，仓

储中心收到货后需要核实情况，如果未填写需要由驻场人员同商家沟通确认是否是托管式物流，如果是则产生物流订单号，传递给商家，商家通过运费账户交付相应的物流费用，仓储中心通过物流公司发货，货物到达买方后，买方对货物做相应的交易评价，整个业务流程完成。

3.3 基于工作流表示的活动模型

UML 的活动模型主要是用来描述系统中多个用例或对象如何协同完成一项工作任务以及它们之间的控制流程，也可以描述单个用例中的多个活动或多个对象之间的协同工作及它们之间的控流程，由此可见活动模型侧重描述活动到活动之间的控制流程。

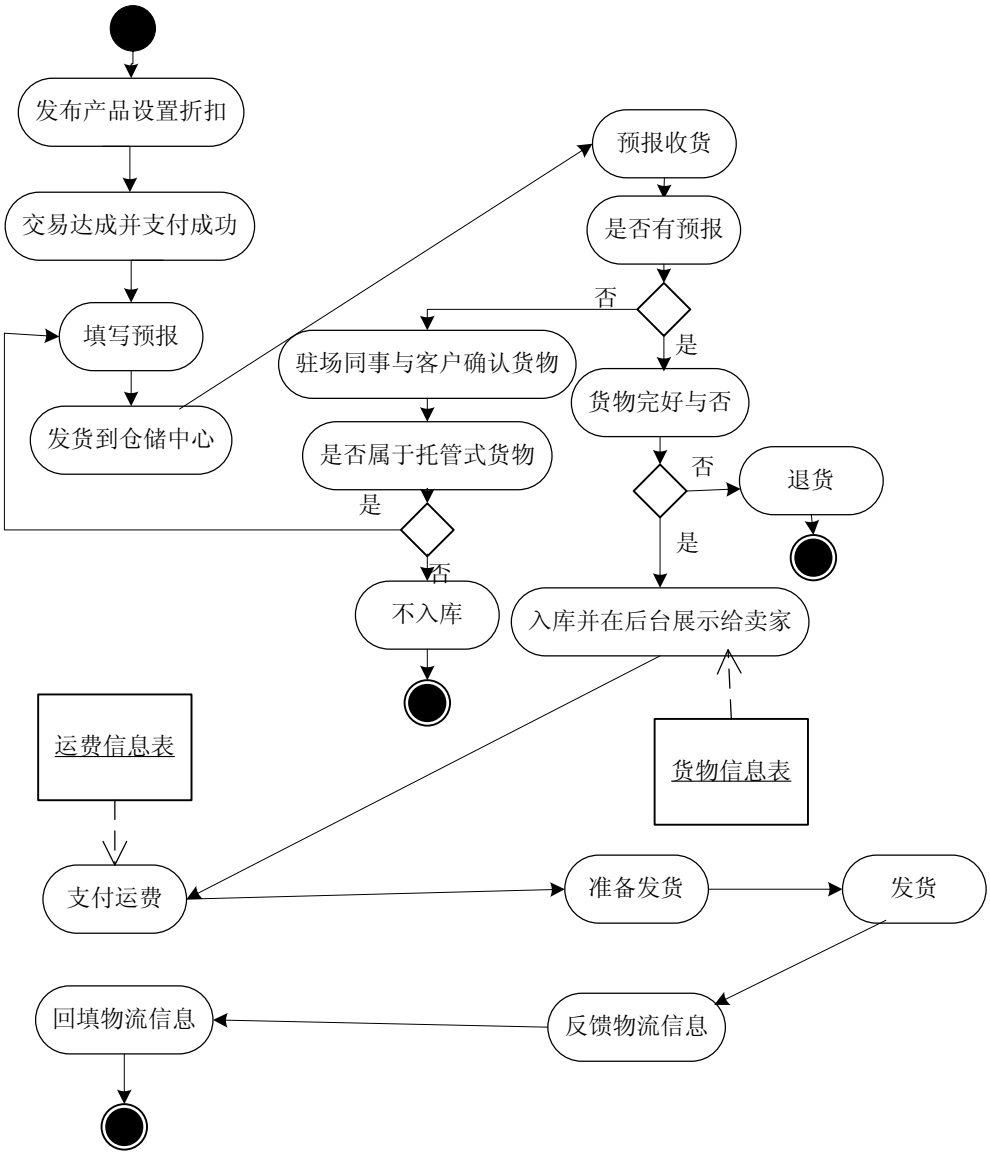

图 3.5 面向商家的托管式物流活动图

对于一个活动图主要包括五种元语[48]，它们分别是：（1）活动（Action）：主要是用来描述用例间或者对象间完成某项任务所要进行的处理步骤，而且在这些处理步骤中部分可以进一

步细化为一些更小活动的集合。（2）对象（Object）：它是一个活动中数据的输入输出源，是通过数据流相互连接的 I/O 实体。（3）同步条(Synch bar)：描述了业务流在此处开始或结束多路并发执行操作。（4）控制流（Control flow）：表示一个动作与其参与者和后继动作之间的关系。（5）数据流（Data flow）：描述动作和其输入和输出对象之间的关系。

本课题在软件过程模型的构建中，使用活动图来对软件底层用例的业务流程进行建模，从而形成了基于活动的业务流，将功能分解到活动级。通过其来描述系统要实现一些功能所要进行的一组活动以及活动之间的相互关系。图 3.5 给出了面向商家的托管式物流货物物流订单管理用例的活动模型，该图主要描述了实现货物物流订单管理用例功能系统要进行的活动与活动之间相应的关系。

3.4 物流订单管理的 BPEL 描述

业务流程执行语言（BPEL）主要作用是通过使用比传统编程语言更加抽象更接近于用户思维特征的表达方式来对系统的业务流程进行建模。BPEL 中的活动主要包括七大基本元素：(1)Receive(BPEL 的起点，客户端发送请求消息后，BPEL 引擎接收到请求后启动一个 BPEL 业务流程)；（2）Reply（BPEL 的终点，BPEL 业务流程执行到该点时将响应的结果返回给服务请求者）；（3）Invoke（向交互对象发送请求，并调用交互对象所提供的一些服务）；（4）Flow（定义多个并行的活动）；（5）Link（表示活动间的依赖关系）；（6）Source（活动链接的源）；（7）Target（活动链接的目标）。

根据上述对 BPEL 中活动（Activity）的定义结合上一节的物流订单管理模块的活动图，给出 BPEL 业务流程表示如下：

```
<Process name = “Warehouse&LogisticsService” >
   <Partners>
         <partner name  =  “noticeFill”
         serviceLinkType  =  “lns：noticeFillLinkType”
         myRole = “noticeFillRequester”
         partnerRole = “noticeFillService” />
         <partner name = “goodsDelivery”
         serviceLinkType = “lns：goodsDeliveryLinkType”
         myRole = “goodsDeliveryRequester”
         partnerRole = “goodsDeliveryService” />
         <partner name = “receiveForecast”
         serviceLinkType = “lns：receiveForecastLinkType”
         myRole = “receiveForecastRequester”
```

```
        partnerRole = “receiveForecastService” />
        <partner name = “goodsConfirm”
        serviceLinkType = “lns: goodsConfirmLinkType”
        myRole = “goodsConfirmRequester”
        partnerRole = “goodsConfirmService” />
        <partner name = “goodsStorageandDisplay”
        serviceLinkType = “lns: goodsStorageandDisplayLinkType”
        myRole = “goodsStorageandDisplayRequester”
        partnerRole = “goodsStorageandDisplayService” />
        <partner name = “freightPayment”
        serviceLinkType = “lns: freightPaymentLinkType”
        myRole = “freightPaymentRequester”
        partnerRole = “freightPaymentService” />
        <partner name = “goodsSend”
        serviceLinkType = “lns: goodsSendLinkType”
        myRole = “goodsSendRequester”
        partnerRole = “goodsSendService” />
        <partner name = “logisticsFeedback”
        serviceLinkType = “lns: logisticsFeedbackLinkType”
        myRole = “logisticsFeedbackRequester”
        partnerRole = “logisticsFeedbackService” />
        <partner name = “logisticsBackfill”
        serviceLinkType = “lns: logisticsBackfillLinkType”
        myRole = “logisticsBackfillRequester”
        partnerRole = “logisticsBackfillService” />
    </Partners>
    <Containers>
       <container name = “goodsInfo” messageType = “goodsInfoID”/>
       <container name = “paymentInfo” messageType = “paymentInfoMessage”/>
    </Containers>
    <links>
       <link name = “discountSet-to-dealReached” />
```

```
        <link name = “goodsDelivery-to-goodsSorage” />
        <link name = “freightPayment-to-goodsSend” />
        …
    </links>
    <receive partner = “noticeFill”
      portType = “noticeFillPT”
      operation = “noticeFill”
      <source linkName “discountSet-to-dealReached” />
    />
    <sequence>
      <invoke partner = “goodsStorageandDisplay”
        portType = “goodsStorageandDisplayPT”
        operation = “StorageandDisplaygoods”
        inputContainer = “goooodsInfo”
        <target linkName “freightPayment-to-goodsSend” />
        <source linkName “goodsDelivery-to-goodsStorage” />
      />
      <invoke partner = “freightPayment”
        portType = “freightPaymentPT”
        operation = “Payfreight”
        inputContainer = “paymentInfo”
        <target linkName “freightPayment-to-goodsSend” />
        <source linkName “goodsDelivery-to-goodsStorage” /> />
    </sequence>…//其他略
  </Process>
```

3.5 TAR 协同数据管理框架的本体建立、映射与查询

在基于本体的 TAR 数据协同管理框架中，本体的构建是实现数据协同的基础性工作。本文采用混合本体模型，在该模型中包括一个业务领域协作本体和多个应用系统的局部本体，如图 3.6 所示。

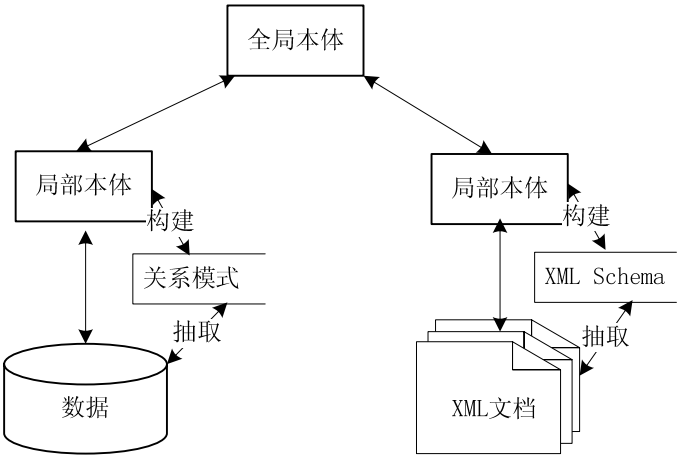

图 3.6 改进的混合本体模型

在该模型中，数据源主要分为关系型数据库和 XML 文件，分别进行局部本体的构建，然后再在顶端建立领域本体。局部本体主要是描述局部的数据源的语义，对其概念模型进行明确，一般情况下，数据源的数据模型之间存在差异，即使相同语义上也可能有差异，局部本体对避免这种数据源的语义或者是模型差异有较好的表现。与此同时，全局本体与局部本体分离符合松耦合的标准。在此基础上引入本体映射算法，建立领域本体和局部本体间的映射关系，为解决异构应用系统的语义异构奠定基础。

3.5.1 本体的构建

1. 关系数据库（RDB）局部本体的构建

大量的结构化数据存储在 RDB 中。RDB 采用的是关系模型。从 RDB 构建局部本体，主要是对 RDB 中所包含的关系信息进行分析，并且用本体的形式表达出来。从 RDB 中抽取本体往往是依据 E-R 图来构建，因此依据特征进行分类，并对每一种类别制定对应的规则。本课题主要依据以下规则构建本体。

表 3.1 关系数据源局部本体转换表

关系模式	局部本体
关系名	OWL：Class
普通属性的域	XSD：xsdDataType
普通属性	OWL:DatatypeProperty
外键	OWL：ObjectProperty
关系间多对多的关系	两个 OWL：ObjectProperty OWL：inverserOf 定义互相逆向 定义 OWL：ObjectProperty 的 rdfs：domain 定义 OWL：ObjectProperty 的 rdfs：range

表中给出了基本的一些映射关系，其中普通属性还需要对 OWL:DatatypeProperty 的 rdfs：domain 和 rdfs：range 进行取值的约束表示。以上规则对简单的构建方式十分有效，如果出现复杂的情况仍需要手动进行转换。

本课题以行业类目为服装行业和通讯行业的数据为例给予说明，其他的行业表类似，例如数据源 1 中以关系型数据库存放类目信息，数据库表示如下：

类目表：Categories（id，cname，quantity）

服装行业：apparel（Quantity，Sweaters，Jeans，Sportswear，Pants）

通讯行业：Communication（Quantity，Cellphone，Computer）

下面给出使用上面规则的得到的局部本体如图 3.7 所示。

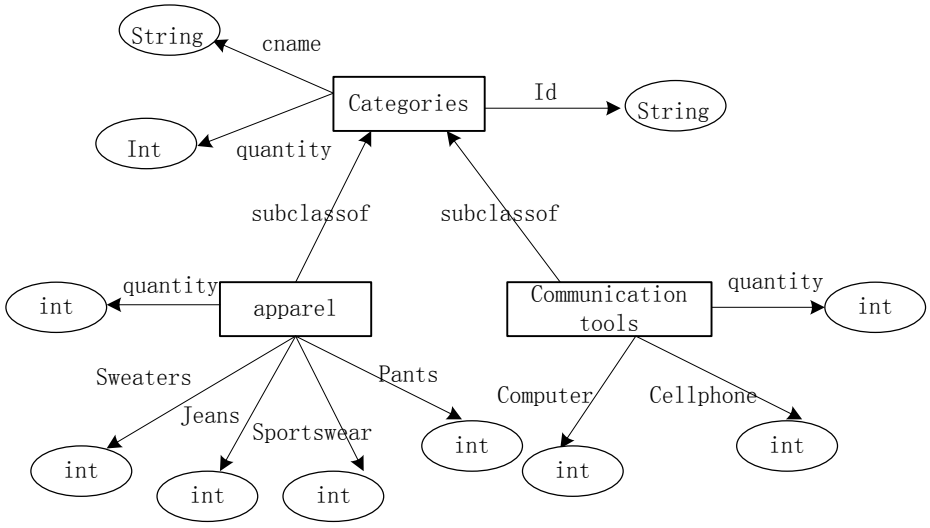

图 3.7 关系数据源局部本体图

2.XML 数据源的局部本体的构建

局部数据源是以 XML 形式的半结构化文档存在的按照 XML Schema 做转化。转化的步骤是先按照给定的映射规则进行转化，对于复杂的情况需要借助人工进行转化。

本课题将 XML 形式构建本体的方法分为两种，元素级和结构级。

元素级的本体构建定义了基本类和类的属性。具体转化方式见表 3.2 所示。

表 3.2 XML 数据源局部本体转换表

XML	局部本体
ComplexType	OWL：Class
SimpleSchema	OWL:DatatypeProperty
Attribute	OWL:DatatypeProperty

结构上面是将 XML 的层次结构转化到 OWL 形式中，主要考虑元素与属性之间的关系和元素与子元素之间的关系。前者定义为 OWL：Class—OWL：DatatypeProperty；后者定义为 OWL：

Class—rdfs:subClassOf。

下面给出数据源 2 的部分 XML 形式的文档：

```
<xs:schema xmlns:xs=http://www.w3.org/2011/XMLSchema
elementFormDefault="qualified">
    <xs:element name="Apparel">
      <xs:complexType>
        <xs:element ref="quantity"/>
        <xs:element ref="Sweaters"/>
        <xs:element ref="Jeans"/>
        <xs:element ref="Basketballclothes"/>
        <xs:element ref="Yoga"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Communication tools">
      <xs:complexType>
        <xs:element ref="quantity"/>
        <xs:element ref="Computer"/>
        <xs:element ref="phone"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="Categories">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="cname"/>
          <xs:element ref="id"/>
          <xs:element ref="quantity"/>
          <xs:element ref="Apparel"/>
          <xs:element ref=" Communication tools"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Sweaters"type="xs:int"/>
```

```
    <xs:element name="Jeans"type="xs:int"/>
    <xs:element name="Basketball clothes"type="xs:int"/>
    <xs:element name="Yoga"type="xs:int"/>
    <xs:element name="quantity"type="xs:float"/>
    <xs:element name="Computer"type="xs:int"/>
    <xs:element name="phone"type="xs:string"/>
    <xs:element name="cname"type="xs:string"/>
    <xs:element name="id"type="xs:int"/>
</xs:schema>
```

对应的局部本体如图 3.8 所示。

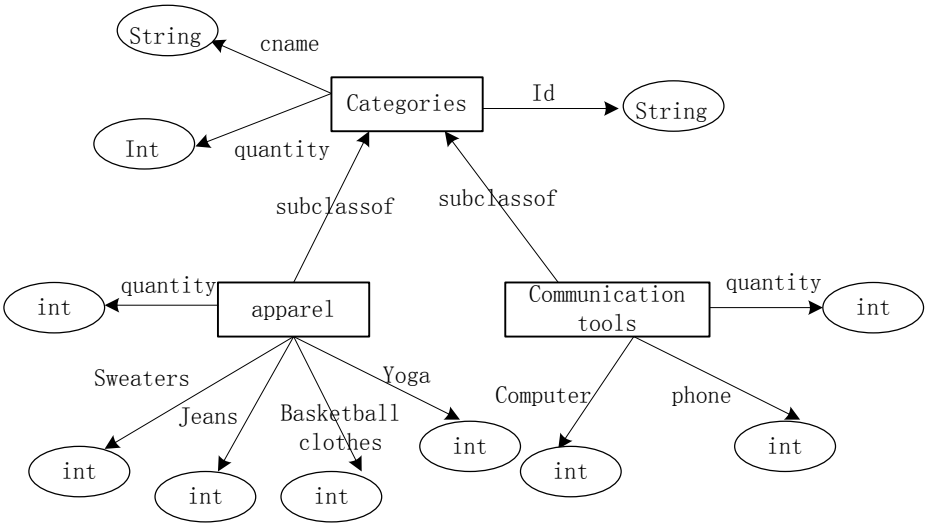

图 3.8 XML 数据源局部本体图

3.全局本体的构建

全局本体是系统为用户提供的统一的全局查询视图的语义化描述。通过建立全局本体，使领域概念间的关系在语义的层次上能够得到统一，另外全局本体的推理能力较好的确定局部本体概念之间的关系。本课题考虑实际应用中的要求，将协作业务本体分为两种，企业本体和领域本体。企业本体是描述企业的元模型，将企业描述规范和系统化，保证其一致性和完整性；领域本体是领域元模型，用于定义在特定的领域的企业描述的约束，且给予领域的可复用的概念和关联。本文使用这种方法进行领域本体的构建。

针对具体的领域本体的构建，主要是在相关领域专家的共同参与下，构建针对企业协作的业务的概念、概念的属性、概念属性之间的关系和概念的实例。构建的步骤包括以下四个：确定领域本体的应用的范围、提取相关的术语（原语），定义领域的全局本体，对本体进行编码。其中定义全局本体包括：定义类的概念及其关系，一般是 subClassOf 关系；定义每个类

的对象特征属性（ObjectProperty）和数据类型属性（DataTypeProperty）。对每个属性添加约束条件。本体编码采用 OWL 语言作为描述语言。通过上述方法给出领域本体的部分片段，如图 3.9 所示。

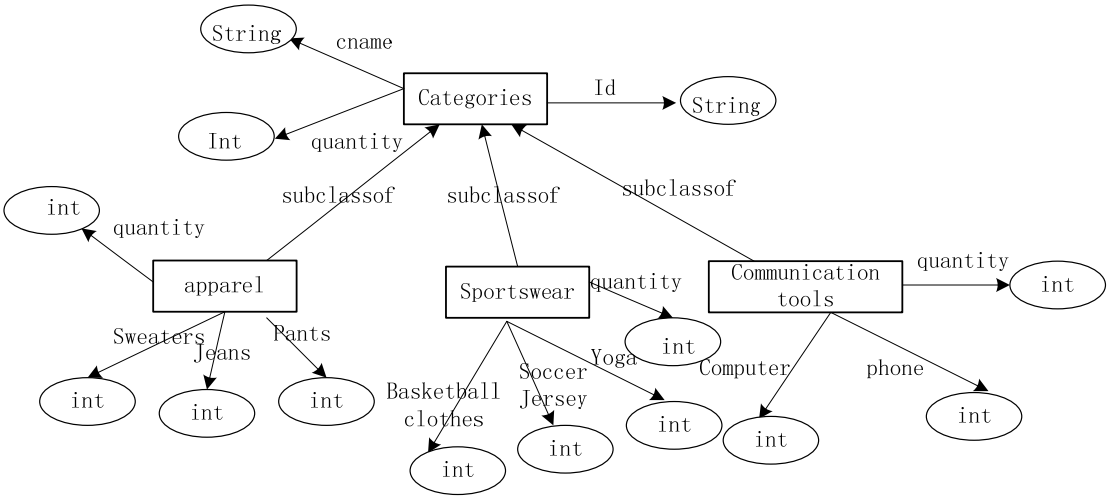

图 3.9 全局本体图

本课题使用 protégé本体建模工具进行建模，采用第二章中给出的 OWL 模板表示方法，由于篇幅所限给出部分 OWL 描述：

```
<owl:Class rdf:ID="Communication tools">
    <rdfs:subClassOf>
        <owl:Class rdf:ID="Categories"/>
    </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Apparel">
    <rdfs:subClassOf rdf:resource="# Categories "/>
</owl:Class>
<owl:Class rdf:ID="Sportswear">
    <rdfs:subClassOf rdf:resource="# Categories "/>
</owl:Class>
<owl:DatatypeProperty rdf:ID="Yoga">
    <rdfs:domain rdf:resource="# Sportswear "/>
    <rdfs:range rdf:resource="http://www.w3.org/2011/XMLSchema#int"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="quantity">
    <rdfs:range rdf:resource="http://www.w3.org/2011/XMLSchema#int"/>
```

```
    <rdfs:domain rdf:resource="# Categories "/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="Basketball clothes">
    <rdfs:range rdf:resource="http://www.w3.org/2011/XMLSchema#int"/>
    <rdfs:domain rdf:resource="# Sportswear "/>
</owl:DatatypeProperty>
      .......
```

一般情况下，领域专家需要花大量的时间和精力建立和维护领域本体，但是在实际的应用中，复杂的领域知识和多种的建模方法导致在短时间内建立普遍认可的领域本体是不切实际的，因此本文采用自组织的本体构建和维护机制：由企业系统的设计人员承担构建本体的任务，使用知识库的过程中建设和维护本体，并且在本体的使用过程中对其进行跟踪和反馈，提高可用性和准确性。自组织的本体建设和维护机制如图 3.10 所示。

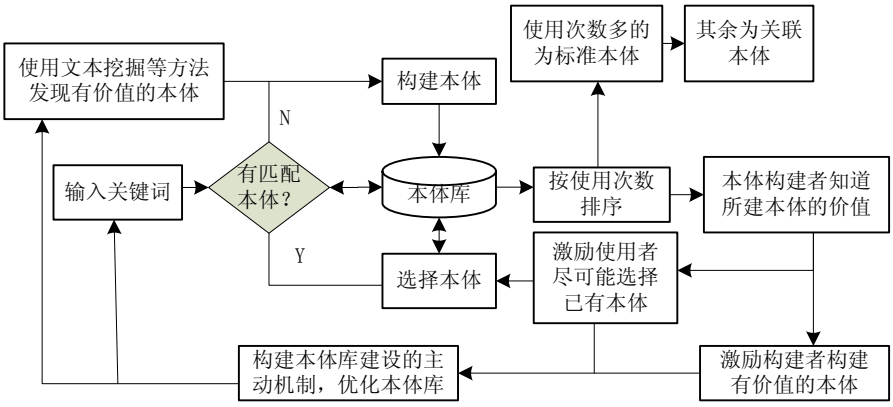

图 3.10 自组织本体构建与维护机制

如上图所示，先是领域专家建立初始的领域本体，导入数据库。其次，本体间的组织关系通过使用 Java Applet 技术给予动态的显示，本体的操作和维护通过使用 Jena API 实现。使用自组织的机制优势在于：（1）优化本体的准确性和覆盖率。（2）本体构建过程的有序化和持续性。在构建本体的同时主动关心本体库，优化本体库，持续改进本体的建立。（3）本体库的动态性和开放性。企业可依据具体的需要增添本体。首选已有本体，如果没有再进行新本体的添加。（4）本体的自动排序。根据使用的次数自动对本体进行排序。（5）使用文本挖掘的方法发现有价值的本体。

3.5.2 本体映射方法

参与企业业务协作的信息系统拥有各自语义的模型，而语义上的不一致给系统的互操作带来了困难。通过 3.5.1 构建了全局本体和局部本体，由于局部本体是依据单个应用局部的数据源构建的，因此局部本体之间存在语义上的异构，而全局本体是参照领域概念建立的，因此只

单独建立全局本体和局部本体是不完备的，为了实现企业的信息系统之间在语义上能够统一，需要通过建立本体之间的映射来消除语义异构，满足用户查询时共享数据源的要求。图 3.11 给出具体的映射图。

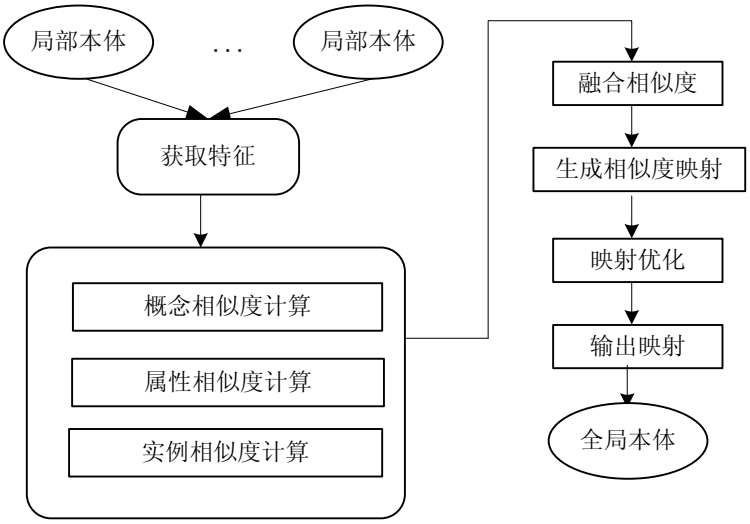

图 3.11 全局本体和局部本体映射图

本体的映射是通过输入两个以上的不同本体，依照相互之间的语义联系关系构建这些本体中元素相应的语义关系。本体映射类型一般主要分为概念-概念、属性-属性、属性-概念等。映射过程即是确定本体间的元素对应关系。本体映射粒度主要有以下六种类型，1:1，1：n，n:1，1：null，null:1，n:m。本文的映射类型主要集中研究 1:1 的映射模式。

下面对图 3.11 给出阐述。（1）抽取特征：这是本体进行映射最基本的步骤，主要是从本体中抽取本体的概念和属性的集合，构建本体的模型。（2）相似度的计算：采取多种相似度的计算方法进行概念间的相似度计算，本文主要考虑概念与概念之间和属性与属性之间的映射。从语法、语义和结构三个方面以及属性的约束进行相似度的计算。再对映射结果进行综合，得到综合相似度。（3）映射生成：选取合适的阈值方法进行比较，生成映射的结果。（4）映射的优化：对已经生成的映射需要进行检查，通过一定的规则进行优化，然后再添加入本体映射文件中。另外一旦数据源发生变化，需要对映射关系进行维护。下面逐一介绍各个相似度计算算法思想。（5）迭代：重复（1）～（4）的步骤，直到获取满意的结果。

在领域本体和局部本体之间建立语义互联关键在于确定本体间的相似或者相同元素的映射关系，其中相似度的计算是发现本体映射关系重要的方法。在企业间业务协作的领域，建立本体间的映射关系的语义相似度的计算是难点。本文采用一种综合相似度的计算方法。该方法包括基于语言学的相似度和基于结构的相似度。

1．概念（类）的相似度

概念相似度的计算包括概念名称的相似度和概念属性的相似度。

（1）基于语法距离的概念名称的相似度计算

该方法是对本体的概念相似度计算，因为名称由字符串组成，因此采用编辑距离的方法。“编辑距离”是指为两个字符串之间转换所需要的最小数目的单元编辑操作，包括字符的插入、删除、替换以及相邻字符的互换。基于该编辑距离的方法，结合动态规划算法计算距离，给出概念字符串的相似度如下：

$$Sim_{dist}(strA, strB) = \max(0, \frac{\min(|len(strA)|, |len(strB|) - ED(strA, strB)}{\min(|len(strA)|, |len(strB)|)}) \quad (3\text{-}1)$$

其中|len(strA)|，|len(strB)|是字符串长度，ED（strA，strB）是编辑距离。这种相似度计算方法对于仅仅只有下划线、单复数形式变化等区别的数据找出其中的相似度作用很大，例如概念“tree”和“three”,它们之间的编辑距离 ED 为 1，则 $Sim_{dist}(tree, three) = 0.75$，因此这种方法比较简单，但是 $Sim_{dist}(father, feather) = 0.83$，很明显这两个词之间的概念相似度远没有这么大，因此需要结合其他的相似度计算方法。

（2）基于语义的概念名称的相似度计算

这种相似度的计算一般需要借助语义词典，WordNet 是比较典型的语义词典。它是一种在线词汇的参考系统，由 Princeton 大学认知科学实验室从 1985 年起开始开发，目前仍在不断发展。它的特点在于是依据词义组织词汇的信息，采用同义词集表示概念，词汇关系体现在词语之间，语义的关系体现在概念之间。WordNet 构造的关键在于对词汇的概念节点如何进行表示，以及在这些节点间建立各种语义关系。WordNet 将词汇组织成同义词的集合，每个同义词的集合注明一个词汇概念，其中每一个节点表示一个词义，一个词义包含了多个同义词语。词汇之间的关系有上位、下位、同义、反义、蕴含等。使用该方法可以解决异名同义的词汇。

基于 WordNet 的相似度计算有多种算法，文献[49]使用精确度、F 度量和召回率综合指标评价这些算法，实验的结果证明 Jiang-Conrath 算法相对于其他算法效果更好，因此本课题采用 Jiang-Conrath 的方法，该方法是在确定共同祖先后，使用子节点的条件概率对语义距离进行计算，且语义距离和语义的相似度是成反比的关系。将语义距离定义为 $Dist(strA, strB)$，其计算公式是：

$$Dist(strA, strB) = 2\log(p(lso(strA, strB))) - (\log(p(strA)) + \log(p(strB))) \quad (3\text{-}2)$$

其中 $lso(strA, strB)$ 是 $strA$，$strB$ 最近的共同祖先，$p(str) = \frac{wordcount(str)}{totalwords}$，$wordcount(str)$ 为节点包括其子节点的所拥有的所有词汇数量。$totalwords$ 是 WordNet 的词汇数量。

基于语义词典的概念间的相似度：

$$Sim_{wordnet}(strA, strB) = \frac{1}{Dist(strA, strB)} \quad (3\text{-}3)$$

通过上述两种相似度计算方法，可以得到概念名称的相似度为：

$$Sim_{name}(strA,strB)=\alpha * Sim_{dist}(strA,strB)+(1-\alpha)Sim_{wordnet}(strA,strB) \quad (3\text{-}4)$$

（3）基于属性的概念相似度计算

基于属性的概念相似度分为名称相似度 $Sim_{propname}$ 和数量相似度 $Sim_{propnum}$，其公式如下：

$$Sim_{propname}=\frac{1}{N}\sum_{i=1}^{N_1}\max(sim_{dist}(p_{1i},p_{2j})) \quad (3\text{-}5)$$

$$Sim_{propnum}=\begin{cases}\dfrac{|N_1-sub|}{4},(N_1\le 4,sub\le 3)\\ \dfrac{|N_1-sub|}{N_1},(N_1>4,sub\le 3)\\ 0,other\end{cases} \quad (3\text{-}6)$$

结合属性名称和数量的相似度，基于属性总相似度为：

$$Sim_{prop}(strA,strB)=\beta * Sim_{propname}(strA,strB)+(1-\beta)Sim_{propnum}(strA,strB) \quad (3\text{-}7)$$

概念总的相似度综合概念的名称和属性的相似度，得出：

$$Sim_{concept}(strA,strB)=\gamma * Sim_{name}(strA,strB)+(1-\gamma)Sim_{prop}(strA,strB) \quad (3\text{-}8)$$

其中，α,β,γ 是权重因子，取值范围从 0-1，根据相似度计算结果对其进行调整，以取得比较好的效果。N_1,N_2 分别是 strA,strB 的属性个数；P_{c1},P_{c2} 表示概念 strA，strB 的属性集，且 $p_{1i}\in P_{C1},p_{2j}\in P_{C2},sim_{dist}(p_{1i},p_{2j})$ 表示基于编辑距离的 p_{1i},p_{2j} 的名称的相似度；sub 表示属性数量差的绝对值，N 表示数量的最小值。

2．属性的相似度计算

属性相似度的计算需要考虑属性名称相似度、属性定义域（Domain）和属性值域（Range）相似度的影响。综合以上三种相似度得出属性相似度的公式为：

$$Sim_{property}(prop_1,prop_2)=\alpha * Sim_{pname}+\beta Sim_{domain}+\gamma Sim_{range} \quad (3\text{-}9)$$

其中，α,β,γ 是权重因子，取值范围从 0-1，且 $\alpha+\beta+\gamma=1$。Sim_{pname} 表示属性名的相似度，该相似度的计算同概念名相似度计算的方法相同，在此不赘述。Sim_{range} 和 Sim_{domain} 表示属性的值域和属性定义域相似度。对于 Object 属性，Sim_{range} 和 Sim_{domain} 是两个对象类型的属性值域类和定义域类之间的相似度；Datatype 属性的 Sim_{domain} 是两个数据类型的定义域类间的相似度，Sim_{range} 可通过匹配数据类型获取。

3．实例相似度的计算

实例相似度的计算主要是名称相似度和基于属性值的相似度的综合，考虑到实例所属的概念相似度值，将其作为影响因子，计算的公式如下所示：

$$Sim_{\text{inst}}(inst1,inst2)=Sim_b * (\alpha * Sim_{instname}+(1-\alpha)Sim_{propvalue}) \quad (3\text{-}10)$$

$$Sim_{propvalue} = \frac{1}{N}\sum_{i=1}^{N_1} \max(sim_{dist}(inst1.p_{1i}, inst2.p_{2j})) \quad (3\text{-}11)$$

$Sim_{propvalue}$ 表示基于属性值的相似度，$inst1.p_{1i}$ 和 $inst2.p_{2j}$ 分别表示 inst1 和 inst2 所对应的属性 p_{1i} 和 p_{2j} 的值；$Sim_{instname}$ 表示实例名之间的相似度，计算同概念名之间相似度计算方法相同；Sim_b 表示实例所属于的类的相似度。

4．选择阈值

阈值的作用是比较计算相似度得出的结果。大于阈值的可认为是相似的。阈值的选取方法很多。包括百分比法、固定值、差值。在本课题中，为了防止映射结果有偏差，不唯一确定使用哪种方法，需要通过训练样本得到一个经验值，在实际操作过程中，根据反馈的结果进行动态的调整。

5．本体映射关系的发现

通过相似度的计算以及阈值的选取，可以得出初始的两个本体中待匹配的实体间的相似度。得到初始的相似度后接下来就要发现本体之间映射的关系，对于映射关系的发现有两个规则：

（1）每个待匹配的本体实体之间计算出的相似度值必须大于设定的阈值。

（2）每个实体只能存在于一个映射中，对计算过程中产生的“一对多”的映射，只保留一组最大的相似度映射。

通过以上两个规则，候选的映射关系可以确定，且作为迭代环节计算相似度的基础。迭代时充分使用本体之间的结构关系，如：计算本体中相关概念相似度的过程中，属性的相似度可利用前一次计算的属性相似度。循环经过多次后，结果趋于一个稳定的状态。

6. 映射的优化和决策

接下来要对映射进行优化，优化的方法是去除相似度值高但不合理的映射，添加相似度值低但合理的映射。下面给出优化的规则：

（1）C_1 映射到 C_2，若 C_1^f, C_1^s 同时映射到 C_2^f，则至少有一个映射错误，同时错误映射为 C_1^s 到 C_2^f 的映射关系。（C_1^f, C_2^f 为 C_1 和 C_2 的父概念，C_1^s, C_2^s 为 C_1 和 C_2 的子概念）

（2）对 $Onto_1$ 中的 C_1，若 C_1^f, C_1^s 分别映射到 $Onto_2$ 中的 C_2 的 C_2^f, C_2^s，且通过筛选映射得到 C_1 的最佳映射为 $Onto_2$ 中的 C_3，那么应将最佳映射调整为 C_1 和 C_2 的映射，去除之前 C_1 和 C_3 的映射。

（3）对属性映射关系，检查定义域和值域对应的元素的映射关系。如果两个属性的定义域都是概念，要检查概念是否之间存在映射关系、值域的类型相同与否。

在进行语义相似度计算的企业应用的领域本体和局部本体的任意的两个类之间、属性之间和实例之间会得出相似度值，首先将最大的相似度的实体确定为最终结果，接着将最终的结果和阈值进行比较，大于阈值的认为是相似的，最后进行保存，根据结果建立两个本体之间的映

射。如果所建立的映射关系不能满足用户的需求，则提供交互的接口供用户进行局部本体语义的添加，在本系统中，因为类的属性是描述概念类之间区别的特性，因此主要是添加本体中类的属性，通过添加后的语义再次进行技术，最后完成本体之间的映射文件的生成。

3.5.3 查询转换方法

在对局部本体和全局本体建立好映射之后，需要解决如何在查询时将全局本体的查询转换成对各个数据源的查询。合理的查询分解转化能优化查询，并能对查询结果转化为统一格式起到关键性的作用。本课题采用 SPARQL 本体查询语言，下面给出 SPARQL-SQL 查询转换算法。

Input：SPARQL

Output：sqlVec（保存 SQL 语句）

Step1：解析 SPARQL 语句，并保存到临时表中，临时表 1：InfoWhere（SID，property，value），用以保存 where 子句的绑定变量和属性标识符。临时表 2：InfoSelect（SID，property），用以保存 Select 子句中的类绑定的变量和属性标识符。临时表 3：InfoClass（SID，class），对 Where 子句进行解析，根据属性的标识获得类绑定变量以及在全局本体中的类集合信息。临时表 4：InfoCondition（SID，property，value），保存 where 子句中属性为常量的信息。

Step2：Select 子句的生成

```
Vector selectV;
For(int i=0;i<InfoSelect.rowCount();i++){
   string classBind = InfoSelect.getValueAt(i,1);
   string proSign = InfoSelect.getValueAt(i,2);
   vector ClassSet = InfoClass.find(classBind);
   vector proSet = Onto.ClassOfProperty(proSign);//Onto 为全局本体
   string nameClass = ClassSet.lowest();
   string namePro = proSet.lowest();//解析全局本体，找到 proSign、ClassSet 和 proSet 在局部本体中相应的类与属性，并解析各自的描述本体，获得对应数据源 DS 中表和列：DSj. Prosing、DSj. namePro 和 DSi. nameClass
   selectV.add(“select * from DSi.nameClass”);
   selectV.add(“select DSj.proSign from DSj.namePro”);
}
```

Step3:where 子句的生成

```
Vector whereV;
For(int i=0;i<InfoCondition.getValueAt(i,1)){
```

```
        string classBind = InfoCondition.getValueAt(i,1);
        string proSign = InfoCondition.getValueAt(i,2);
        whereV.add(“DSi.nameClass. proSign = VpSign”);//VpSign 为 proSign 对应的常量值
        whereV.add(“DSj.proSign = VpSign”);
    }
    return sqlVec;
```

3.6 本章小结

本章首先介绍了基于领域工程的 TAR 协同数据管理框架的需求模型，包括托管式物流的领域知识，给出面向商家的托管式物流功能模型，其次给出基于 SOA 的 TAR 协同数据管理软件功能模型，分析了 TAR 协同数据管理软件的各个模块的具体功能进行了描述。其次使用面向对象模型方法结合工作流技术，分割了软件功能，给出顶级用例模型，并对关键用例进行了展开讨论，对系统中物流订单管理过程给出了基于元过程表示的活动模型，并用 BPEL 对活动模型进行了描述，最后对局部和全局本体的建立、映射和查询做了详细的阐述，提出一种综合的相似度计算的本体映射算法。通过本章研究工作，建立了 TAR 协同数据管理软件采用 UML 表示的用例模型、关键用例的活动模型，并给出了物流订单管理过程的活动模型的 BPEL 描述。通过本章的研究，完成了从领域需求模型到软件功能模型的转换。

第四章 TAR 协同数据管理系统的领域设计

本章将采用面向对象的软件建模方法，根据前一章对 TAR 协同数据管理软件的领域分析与功能描述，对 TAR 协同数据管理系统的对象体系结构模型进行设计。首先通过构建 TAR 协同数据管理系统的相关类图，确定系统的静态结构，并结合设计模式与面向 Agent 技术对软件类构架进行了优化。然后对系统的行为模型进行设计。最后对系统中典型应用的数据库表和界面进行了设计，从而实现需求域到软件域的设计过程，从概念模型到软件模型的进化过程。

4.1 TAR 协同数据管理软件的类架构

软件构架是用来解决领域应用问题的一种可重构的软件体系结构，软件构架设计的关键就是软件类构架的设计。软件类构架的设计主要分为两个阶段，它们分别是：初始类模型设计阶段和类模型优化设计阶段。本节将根据领域分析过程中所构建的用例图和活动图首先构建系统的初始化类模型；然后结合系统中所用到的信息载体并通过对它们的分析结合设计模式、细化类模型对系统初始类模型进行了优化。

4.1.1 初始类模型设计

UML 的类图是用来对软件静态系统结构进行描述，通过对软件系统中所使用到的各种实体类进行定义，一方面描述类的属性和操作等内部结构信息，同时对软件系统中各类之间的交互关系进行描述。类模型是面向对象建模的核心，它是通过将在领域需求分析中有意义的概念提取出来并进行相应的抽象处理，结合交互关系将它们进行连接而构建的。

初始化类模型是基于概念的类模型，不具体涉及设计细节，通过类图来进行描述。类通常是指一组对象的集合，在这个集合中的对象拥有着很多共性特征，类所提供的接口主要是用来指明该类可能向外部提供的一组操作的集合，类模型所要表达和描述的是软件系统中各类之间的相互关系，类之间的关系一般主要有以下五种[50]：

（1）关联：它可以用来描述系统中各模型元素之间的一种语义联系，可以是单向的也可以是双向的，但它所描述的是一种很弱的联系。

（2）聚集：一种特殊的关联关系，表述的是系统中具体对象之间的整体与部分的关系。

（3）组合：它也是一种特殊的关联关系，所表述的是整体与部分的关系，但是该部分与整体是具有统一的生存期，即整体消亡部分也随之消亡。

（4）依赖：它主要是描述相关类之间的使用和提供关系，用来表示相互依赖的类之间的非结构性关系。

（5）泛化：它也可以表示为类之间的继承关系，主要描述了一般元素和特殊元素之间的

一种分类关系。

初始类构架是对业务处理逻辑的描述，在设计中要遵循一条主线的思想，其定义了系统类之间的基本语义关系，体现类分组的思想，因此初始类模型是属于逻辑级的结构模型。本文的初始类构架图采用 UML 类图描述。初始类构架图的建立关键步骤与分析重点主要分为：①分析系统的领域知识确定系统需求，确定建模的目标，通过对软件系统底层用例、相关活动的活动图以及活动中所传递的信息载体进行分析从而提炼出系统中的用到的所有初始类；②通过分析用例图中各用例之间的关系、活动图中的各活动之间的信息流和数据流来抽象软件系统中各个类之间的相互关系，将系统的业务流主线映射到初始类图的设计中；③对类之间的各种交互类型要能够准确识别，并应用到系统类图设计中；④要能够准确定义软件系统中各类提供的接口，并作相应的说明，使每个类都能够准确的表达自身能提供给外界用户的所有服务或操作，并对每个对象进行命名和编号，构建系统的初始化对象表；⑤通过使用代理对象提供访问控制来优化对象间的相互联系，通过使用动作对象提供一组预处理活动，简化类之间的交互。通过上述步骤，提取系统中的所有类并识别各类之间的联系，给出初始化类模型如图 4.1 所示。

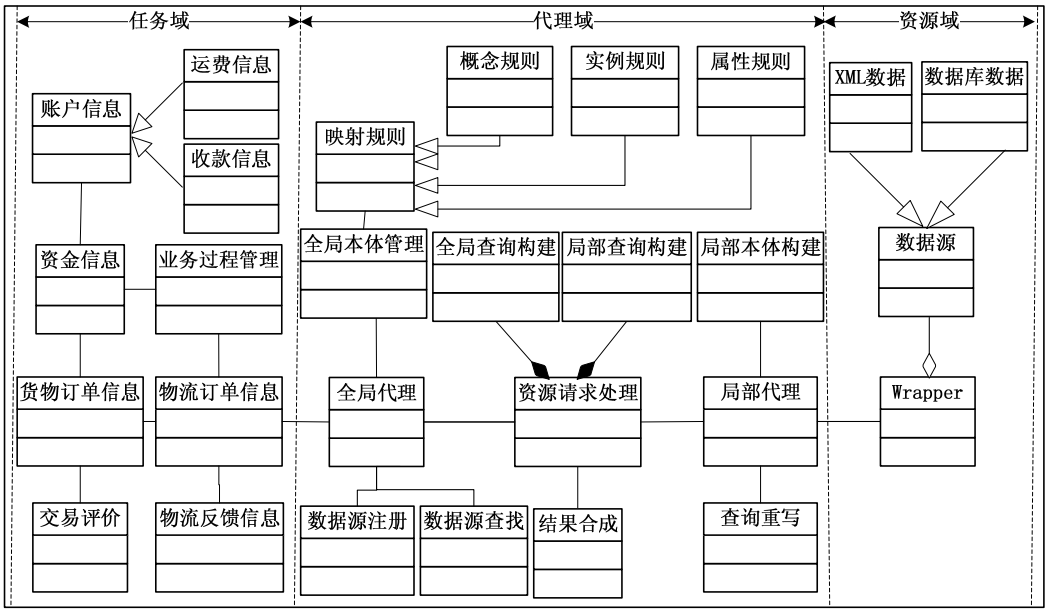

图 4.1 初始类模型

下面对初始类模型进行说明，包括：业务工作流说明、类分组说明和关键类的相关说明。

1. 业务工作流说明

（1）物流订单信息-全局代理-资源请求处理-局部代理-查询重写-Wrapper-数据源

这是“任务-代理-资源”框架的主流程，这里物流订单信息只是任务中的一个子任务，其他任务域内涉及到请求资源的类都可以替换之，提出资源请求后交给全局代理。全局代理查找数据源，将数据源信息交给资源请求处理类。资源请求处理类根据数据源信息进行全局查询的构建，并且结合映射规则进行查询分解，也即局部查询的构建，交给局部代理。局部代理对子

查询进行查询重写，交给 Wrapper 对应的数据源。

（2）数据源-Wrapper-局部代理-资源请求处理-结果合成-全局代理-物流订单信息

数据源通过 Wrapper 构建局部本体，通过局部代理切换到资源请求处理类。资源请求处理类通过对子查询结果进行合成，交给全局代理。全局代理再以统一的视图返回给任务域里的资源请求类。

（3）货物订单信息-收款信息-业务过程管理-物流订单信息-运费信息-物流反馈信息-交易评价

在任务域客户下单后形成对应的货物订单信息，商家可以在收款信息里查询对应的收款记录。填写预报信息后，后台人员通过业务过程管理类对货物进行核实，确认是否是托管式物流货物，如是则形成物流订单，商家在后台可以查询到对应的物流订单信息。商家付运费后可以查看到对应的运费信息。仓储中心发货后，形成物流反馈信息传递给商家，同时客户会做出交易评价。

（4）全局代理-全局本体管理-映射规则-数据源注册-数据源查找

全局代理类包括数据源注册和数据源查找功能，并提供全局本体和局部本体之间的映射规则。

（5）全局代理-数据源查找-映射规则-资源请求处理-局部代理-查询重写-Wrapper-数据源-结果合成

全局代理类传递给资源请求处理类相应的数据源信息。资源请求类构建全局查询，再通过全局代理类传递给映射规则，资源请求类构建对应的局部查询，再查询重写到各个数据源，数据源将查询结果返回给局部代理进行结果合成。

2．类分组说明

（1）任务域类分组:: ={货物订单信息，物流订单信息，资金信息，账户信息，运费信息，收款信息，业务过程管理，物流反馈信息，交易评价}

（2）资源访问协同域类分组:: ={全局代理，资源请求处理，局部代理，数据源注册，数据源查找，全局本体管理，映射规则，全局查询构建，局部查询构建，结果合成，查询重写，局部本体管理}

（3）资源域类分组:：={Wrapper，数据源，关系数据源，XML 数据源}

3．关键类说明

（1）全局代理类：全局代理类主要是用来对数据源进行注册和查找，同时对全局本体进行管理，也包括映射规则管理。数据源通过全局代理类注册到数据注册中心，在用户提出查询请求时，全局代理类返回给资源请求处理类数据源的信息。全局本体和局部本体之间的映射规则也通过全局代理类发布到注册中心。全局本体通过全局代理类注册到注册中心。

（2）资源请求类：资源请求类主要是对用户提出的查询进行分析后形成全局查询，并且根据映射规则形成相应的局部查询，然后执行查询，传递给局部代理，接下来由局部代理完成剩下工作。对局部代理传递过来的子查询进行整合。

（3）局部代理类：接收资源请求类的局部子查询，通过查询重写，将任务分派到各个数据源，在任务分派类中为每个局部查询创建一个线程，多线程并发执行可以提高查询的性能，由各个数据源返回子查询结果，传递给资源请求类。

4.1.2 面向 Agent 的类模型优化

类模型的优化方法主要是指对类的结构进行优化和类的行为进行优化[51]。本课题运用 Agent 的智能性对类模型进行行为优化。Agent 的行为优化方法将 Agent 作为主动对象在面向对象方法的基础上对模型进行优化。

本课题采用的是反应式 Agent 的方法，将类模型中的部分具备反应特性、协同特性和自治特性的实体转换为 Agent。结合图 4.1 给出的类模型，可以从模型中抽取三个实体作为 Agent：全局 Agent、资源请求处理 Agent 和局部 Agent。下面对各个 Agent 给出详细的说明。

全局 Agent 主要负责数据源的注册、数据源的查找功能，接收用户输入的字符串。数据源的注册主要是将封装好的数据源模块在注册中心注册并发布，绑定 Web Service 和数据源，为用户提供统一的访问视图；数据源查找功能主要是在注册中心找到与用户请求相匹配的数据源，也即数据源所在的位置和局部本体信息。数据源查找功能模块接收来自资源请求处理 Agent 的查询请求，将查找的结果返回给资源请求处理 Agent。

资源请求处理 Agent：主要负责生成全局的查询，分解全局的查询，执行查询，结果合成。全局查询生成主要是将用户选择的查询方式组合成全局的查询；查询分解功能将初始的全局查询同本体管理器进行通信，再对其进行相应的语义分解转换，再通过查找对应的数据源信息，生成子查询；查询的执行功能是将子查询发送给局部 Agent，由局部 Agent 对其进行处理。结果合成是指查询的子结果完成后组合来自局部 Agent 的中间结果，再对结果进行整合。

局部 Agent：主要负责将资源请求处理 Agent 的全局查询转换为对各个相关数据源的查询，并且在任务转发类中使用多线程并发执行，为每个局部查询创建一个线程，可以同时执行多个线程，提高执行的性能，维护封装好的数据源，将子查询结果封装后返回给查询处理 Agent，是用户访问局部数据的接口，先确定数据源是什么类型，再针对不同的数据源类型执行对应的功能。对外提供局部本体的相关信息，接受外部查询的请求。

Agent 的单元结构的设计主要是针对 Agent 的单元结构模型进行可实现性的设计，采用表格的形式对各个主要的部件进行说明。资源请求处理 Agent 同时和全局 Agent 与局部 Agent 关联，第二章中给出的 Agent 单元结构设计的四个部分包括属性槽设计、事件槽设计、行为规则槽的设计和行为方法槽的设计。下面给出资源请求处理 Agent 的行为优化实现方式。

（1）事件槽的设计:是感知器主要部分，定义触发 Agent 的行为条件。如表 4.1 所示。

表 4.1 资源请求处理 Agent 的主要事件表

事件	事件的描述
TaskRequestEvent	任务数据请求事件
GlobalFeedbackEvent	全局Agent返回数据源信息事件
TaskErrorEvent	任务未完成的出错事件
GlobalQueryEvent	全局查询生成事件
QueryDecomposeEvent	查询分解事件
ExecuteQueryEvent	执行查询传递给局部Agent
Task distributionEvent	任务分发事件

（2）行为方法槽：下表给出了资源请求处理 Agent 的重要的行为方法，行为方法的类型包括控制和基础两个部分。如表 4.2 所示。

表 4.2 资源请求处理 Agent 的主要行为方法表

行为方法	类型	行为方法描述
InitialAgent(String QueryAgentID)	控制行为	资源请求处理Agent初始化
GetDataSrc(int UserId)	基础行为	获取相关数据源注册信息
ClearAgent(String AgentId)	控制行为	资源请求处理Agent清除

（3）行为规则槽事件：Agent 的每条行为规则都包括判断条件和动作部分，判断条件是以表达式的形式给出。

表 4.3 资源请求处理 Agent 的主要行为规则表

事件(Event)	判断条件(Condition)	动作(Action)	优先级
TaskRequestEvent	RequestValid==True	TaskRequest()	4
GlobalFeedbackEvent	TaskRequestCome==True	GlobalFeedback()	3
GlobalQueryEvent	GlobalFeedback==True	GlobalQueryEx()	2
QueryDecomposeEvent	GlobalQueryFinish==True	QueryDecompose()	1
ExecuteQueryEvent	TaskQueryEnd==True	TakeDown()	0

常数值、变量、函数和运算符等组成以表示表达式，将最终结果（逻辑真或假）代入变量后，得到的结果表示规则在当前条件下是否可以执行，动作是当判断条件判断规则可以执行时 Agent 该如何执行动作，还有些辅助属性，如适用的范围等，这里未涉及到。优先级从“0”到“4”依此递减，“0”的优先级最高。资源请求处理 Agent 的行为规则表如表 4.3 所示。

（4）属性槽的设计：属性槽中包括 Agent 的状态数据和基础数据。下面的表给出资源请

求处理 Agent 的属性槽表。如表 4.4 所示。

表 4.4 资源请求处理 Agent 的主要属性表

属性名称	属性类型	属性的描述
QueryAgentID	状态数据	资源请求处理Agent的标识符，不能为空
QueryAgentName	状态数据	资源请求处理Agent的名称，不为空
QueryAgentSender	基础数据	与资源请求处理Agent交互的Agent
QueryAgentOntology	基础数据	资源请求处理Agent的通信本体
DataReceive	基础数据	全局Agent返回的数据源信息
DataMerge	基础数据	中间数据合成

4.1.3 资源请求过程的 KQML 描述

在框架中参与资源请求任务的的共有三个 Agent 节点：全局 Agent、资源请求 Agent、局部 Agent。结合第二章扩充的 KQML 原语可以将整个资源请求过程给出如下描述。

（1）资源请求处理Agent向全局Agent请求执行查找数据源的任务

```
(request
:sender QueryAgent
:receiver GlobalAgent
:language KQML
:ontology TAR_ontology
:reply-with req001
:content (getDatasrcInfo(Dataid, datasrcName, datasrcScale,
datasrcCategory, datasrcAddr)))
```

（2）全局Agent接受资源请求处理Agent的请求

```
(accept
:sender GlobalAgent
:receiver QueryAgent
:language KQML
:ontology TAR_ontology
:reply-with acc001
:in-reply-to req001
:content (accept sendDataSrcInfo(acceptid))
```

（3）或全局Agent接受资源请求处理Agent的请求

(reject

:sender GlobalAgent

:receiver QueryAgent

:language KQML

:ontology TAR_ontology

:reply-with rej001

:in-reply-to req001

:content (reject sendDataSrcInfo(rejectid))

（4）资源请求处理Agent启用全局Agent执行查找数据源的任务

(trigger

:sender QueryAgent

:receiver GlobalAgent

:language KQML

:ontology TAR_ontology

:reply-with tri001

:content (sendDataSrcInfo(Dataid, datasrcName, datasrcScale, datasrcCategory, datasrcAddr)))

（5）全局Agent告知资源请求处理Agent相关数据源信息

(tell

:sender GlobalAgent

:receiver QueryAgent

:language KQML

:ontology TAR_ontology

:reply-with tel001

:in-reply-to tri001

:content (sendDataSrcInfo(Dataid, datasrcName, datasrcScale, datasrcCategory, datasrcAddr)))

（6）资源请求处理Agent确认收到全局Agent的数据源信息

(confirm

:sender QueryAgent

:receiver GlobalAgent

:language KQML

:ontology TAR_ontology

:in-reply-to tel001

:content (confirm getDataSrcInfo))

（7）资源请求处理Agent收到数据源信息后分解全局查询，执行查询处理，触发局部Agent执行局部子查询

(trigger

:sender QueryAgent

:receiver LocalAgent

:language KQML

:ontology TAR_ontology

:reply-with tri002

:content (execute localQuery)

（8）局部Agent执行完子查询后发送消息给资源请求Agent确认完成

(tell

:sender LocalAgent

:receiver QueryAgent

:language KQML

:ontology TAR_ontology

reply-with tel002

:in-reply-to tri002

:content (execute localQuery Successful))

（9）资源请求Agent收到局部Agent消息后整合查询结果告之全局Agent全局查询结果

(tell

:sender QueryAgent

:receiver GlobalAgent

:language KQML

:ontology TAR_ontology

:reply-with tel003

:content (task successful)

4.1.4 细化类模型设计

初始类模型是系统构架的静态模型，简单刻画系统中类之间的关系和类的分组模型。细化类模型是详细描述类的属性和操作。细化类模型的对象属性表是在框架实现时设计数据库的重

要依据，对象操作表是框架实现时对象所需要执行的操作[52]。以下是细化类模型的步骤[53]：①细分对象粒度；②用例图的一个独立的用例表示为一个类分组；③确定各对象的准确含义和任务职责，用以确定属性表和操作表。下面以托管货物单和资源类为例，表 4.5 和 4.6 给出对象属性表与对象操作表。

表 4.5 对象属性表

托管货物单对象属性表	
货物单号	wl_inlistid
货架号	wl_Shelfid
货物入库日期	wl_indate
商家 ID	wl_sellerid
商家店铺名	wl_sellername
货物编号	wl_goodsid
货物价格	wl_goodprice
运费	wl_sendprice
驻场员 ID	wl_receiverid
备注	wl_remarks

资源对象属性表	
数据源名称	datasrcName
数据源级别	datasrcScale
数据源类目名	datasrcCategory
数据源地址	datasrcAddr
数据源描述	datasrcDzescp
数据访问权限	accessPermit
数据源类目号	datasrcclassID
数据源所有者	datasrcOwner

表 4.6 对象操作表

托管货物单对象操作表	
《关键操作》	
货物单跟踪（）	GoodsOrdTrack（）
货物单查询（）	GoodsOrdQuery（）
《基本操作》	
货物单提交（）	GoodsOrdSubmit（）
货物单修改（）	GoodsOrdModify（）
货物单删除（）	GoodsOrdDelete（）
货物单指派（）	GoodsOrdAssign（）
货物物流跟踪（）	GoodswlTrack（）

资源对象操作表	
《关键操作》	
资源注册（）	DataSrcReg（）
资源查找（）	RegisterSeek（）
《基本操作》	
资源注销（）	DataSrcLout（）
资源格式转换（）	DataSrcTypeCov（）
更新资源请求（）	updaReq()
删除资源请求（）	deleteReq()
出错处理（）	errProce()

4.1.5 精化类模型设计

在对类模型细化后，更进一步的划分类模型后得到的精化类模型，步骤是[54]：（1）进行类分组，将初始类模型的一个用例图对应的对象集划分逻辑结构；（2）将初始类模型中的类划分为层次结构，依据该层次结构分割对象；（3）保留初始类模型的事务类之间的关联关系。结合上述步骤对系统中资源访问协同域的类模型进行精化，其精化类模型如图 4.2 所示。

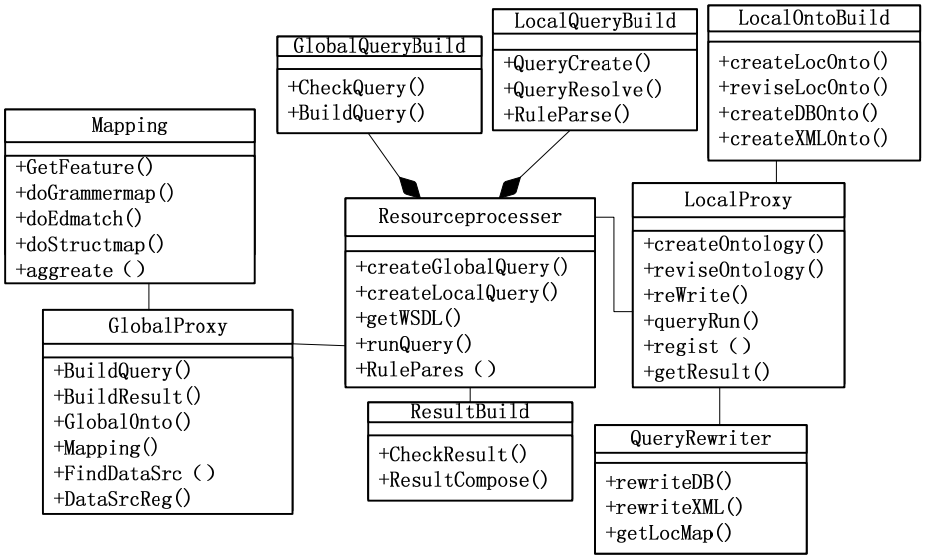

图 4.2 资源访问协同域精化类模型

4.2 对象行为模型设计

对象的行为模型主要由对象之间的交互关系和对象所包含的状态以及活动组成，因此对象行为模型可以由对象的交互模型和对象的状态模型来描述[55]。对象的交互模型主要是用来描述单个用例中相互协作的对象之间所进行的动态交互行为以及在交互过程中所进行的消息传递关系，因此可以使用顺序图来描述对象的交互模型；对象的状态模型主要侧重于描述单个对象在其生命周期内为响应某个事件所经历的状态和控制的转变顺序，状态模型是一种使用状态机来表示的行为模型，它可以体现对象从一个状态到另一个状态的控制流。

对象的顺序图（Sequence Diagram）是用来描述多个参与交互的对象之间在消息传递上的时间顺序以及它们行为的执行顺序，是交互关系的二维图表示（横向轴表示在协作中各独立对象的类的角色；纵轴是时间轴，时间沿竖线向下延伸，该竖线也被称为生命线），一般包含四种基本元素：对象（Object）、消息（Message）、生命线（Lifeline）和激活（Activation）[56]。一个顺序图描述对象间的六种基本关系：顺序、重复、分支、条件、同步和异步关系。

在进行顺序图设计时需要遵循以下的步骤：从活动图以及初始类图中抽取出“信息载体、角色与资源”形成顺序图中的对象；把活动图中的活动转换成相应的对象间的操作；在对象生成周期中，明确对象间的消息连接、消息传递以及操作中所触发的事件。图 4.3 给出了数据查询功能的顺序图，并给出相关辅助说明。

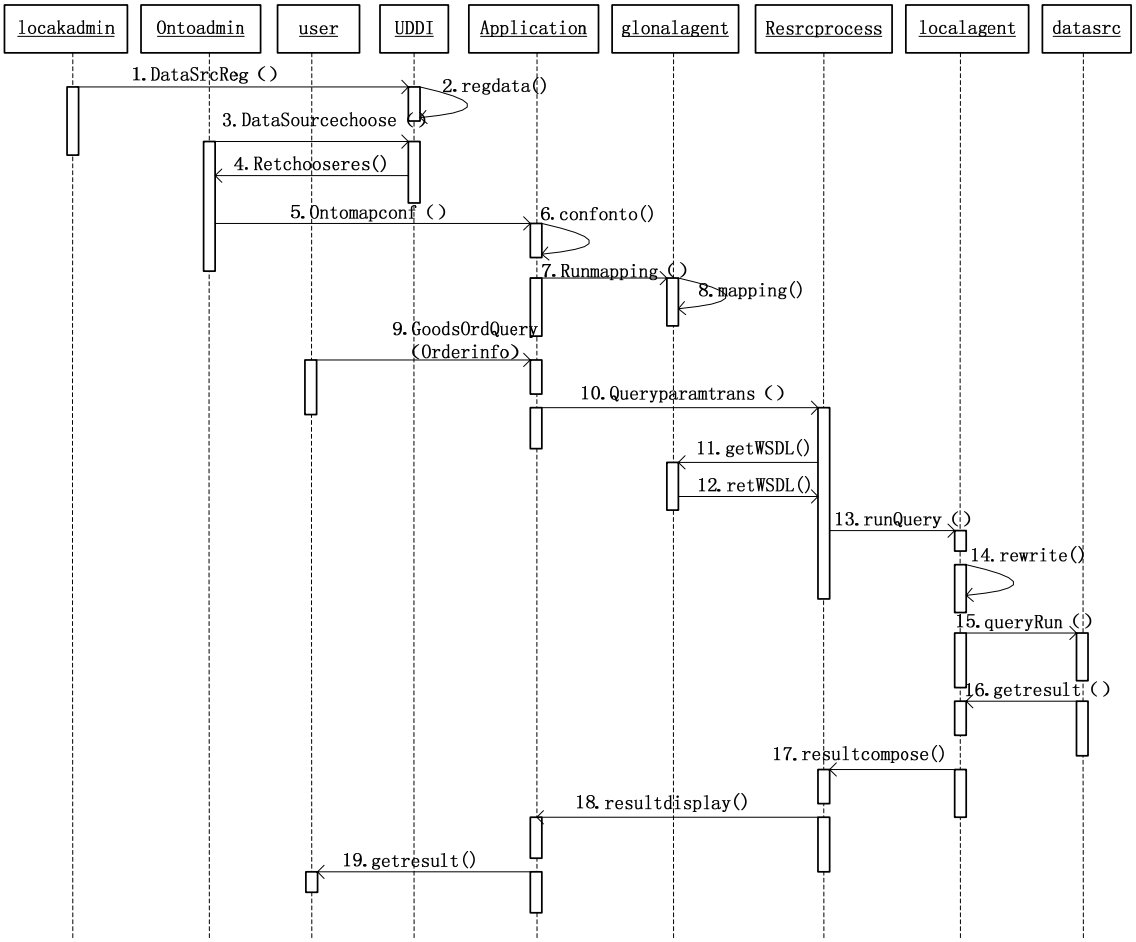

图 4.3 资源请求顺序图

下面给出资源请求顺序图说明文档：

(1) 初始化：

数据注册到 UDDI：DataSrcReg()，regdata（）

配置数据源，从 UDDI 选择需要的数据： DataSourcechoose（），Retchooseres（）

配置映射选项：Ontomapconf（），confonto（）

计算本体间相似度：Runmapping（）

(2)用户查询数据：

提出查询请求：GoodsOrdQuery（）

应用层接到查询请求后交给资源请求 Agent：Queryparamtrans（）

获取数据源注册信息：getWSDL（）

根据用户选择生成全局查询：createGlobalQuery()

对查询进行分解：rewrite（）

相应的数据源执行子查询 queryRun（）

子查询的结果返回到局部 Agent：getResult（）

资源请求 Agent 对子查询结果进行组装：resultCompose（）

查询结果在应用层显示：resultdisplay（）

用户获取结果：getresult()

4.3 典型应用数据库设计

根据类模型以及系统中各个对象的属性表和操作表进行系统的数据库设计。数据库设计的好坏直接影响到整个系统效率和质量，因此在数据库设计包括以下几个要点[57]：

（1）统一数据库基表命名规范：对于数据库基表的命名采用小写系统名+大写基表名，如果基表名中有多个单词则每个单词的首写字母要大写，如 riskProductName。

（2）建立系统基本文件清单：根据系统初始类模型提取系统中需要构建的所有基表名称，为了提高系统的性能可以采用视图或临时表的方法减少系统的基表数量。

（3）设计基表的数据结构：根据初始类模型的细化模型即对象属性表设计基表的数据结构。使用概念趋同的方法对基表中的数据项进行定义使其具有一定的通用性。

（4）设计数据代码：根据运作流程对特定数据代码进行定义，使其更贴近于实际应用。

数据库设计还应注意以下事项：

（1）数据的唯一性应该建立在整个系统角度，切忌出现重复性。

（2）数据类型和数据长度在同一数据项的不能基表中应该保持一致。

（3）数据列表项的值在不同基表中保持一致。

（4）数据表不可同模块简单的组织，之间不是一一对应的关系。

（5）数据存取权限根据不同的应用、不同的角色设置不同的权限，切忌非合法角色用户或者是应用读取。

结合上述设计过程，表 4.7 给出 TAR 协同数据管理系统中所要用到的主要基表文件清单。

表 4.7 主要基表清单

序号	基表名（英文）	基表名（中文）	相关说明
LogisticsTable001	wl_payment_Info	支付信息表	存储产品的支付信息
LogisticsTable002	wl_logistics_Info	物流信息表	存放货物物流信息
LogisticsTable003	wl_good_Info	货物信息表	存放商家货物信息
LogisticsTable004	wl_ prior_Info	预告信息表	存放商家预告发货信息
LogisticsTable005	wl_fare_Info	运费信息表	存放所发货物运费信息
LogisticsTable006	wl_discount_Info	产品折扣信息表	存储货物运输物流折扣信息
LogisticsTable007	wl_Reg_Info	服务注册表	存放服务相关的注册信息
LogisticsTable008	wl_dataSrc_Info	数据源信息表	存放数据源具体信息表
LogisticsTable009	wl_map_Info	映射信息表	存放局部与全局本体间映射关系

表4.8 注册资源信息表

序号	列名	数据类型	长度	IsNull	键值
1	ID	varchar	50	N	Primary key
2	datasrcName	varchar	50	N	
3	datasrcScale	Int	10	Y	
4	datasrcCategory	varchar	20	Y	
5	datasrcAddr	varchar	50	Y	
6	datasrcDzescp	varchar	200	Y	
7	accessPermit	varchar	50	Y	
8	datasrcclassID	Int	10	Y	
9	datasrcOwner	varchar	50	Y	

表 4.9 托管货物信息表

序号	列名	数据类型	字段长度	IsNull	键值
1	ID	varchar	50	N	Primary key
2	wl _inlistid	varchar	50	Y	
3	wl _Shelfid	Varchar	50	Y	
4	wl_indate	Datetime	8	Y	
5	wl_sellerid	Varchar	50	Y	
6	wl_sellername	Varchar	50	Y	
7	wl_goodsid	Varchar	20	Y	
8	wl_goodprice	Varchar	20	Y	
9	wl_goodsendprice	Int	10	Y	
10	wl_receiverid	Int	10	Y	
11	wl_receivername	Varchar	20	Y	
12	wl_remarks	Varchar	50	Y	

表 4.10 映射规则表

序号	列名	数据类型	字段长度	IsNull	键值
1	ID	Varchar	20	N	Primary key
2	ontomapruleID	Varchar	20	Y	
3	ontomapruleName	Varchar	50	Y	
4	datasrcId	Int	8	Y	

此外还要特别注意，不能一味的追求数据库设计的简要，因为有些时候适当的冗余反而可以提高系统的运行效率。数据库的设计不是一次完成的，而是不断往返迭代的，需要根据系统需求进行逐步完善。

4.4 系统界面设计

系统界面是一个软件系统的入口，是软件系统和用户之间的一种可使直观的联系，这就要求软件不仅要具有良好的使用性能还要具有一个友好、易于操作的用户界面[58]。软件构图主要原则包括：（1）易操作性：主要包括操作快捷键设置、控件功能的相关提示信息、操作步骤的简单性和合理性以及相关操作参数在缺省情况下的默认值设置。（2）布局合理性：注意一个窗口内所有控件的布局和组织艺术性，使界面美观、合理。（3）输入的简易性：要尽量减少用户不必要的输入，并给出一些必要的输入提示，实时监测用户的输入信息，及时提醒输入的错

误信息提高用户的工作效率。（4）信息的反馈性：能够对一些操作的执行提供相应的反馈信息，使用户可以及时了解到软件的运行情况。（5）操作的“屏蔽”性：在特定状态下，要对在该状态下无意义的操作进行屏蔽。

在对本系统菜单进行设计时，按照功能分为两级菜单：系统级菜单和子系统功能级菜单，系统级菜单以横栏的显示，依此列出各个系统级菜单，子系统功能级菜单以树形列表的方式显示，选择每个系统级中的一个子系统级菜单，左侧相应显示该子系统所包含的功能列表，在功能列表里又对各个部分进行细分，主页面显示与选择的功能列表一一对应，如图 4.4 所示。

图 4.4 系统界面图

4.5 本章小结

本章重点探讨了 TAR 协同数据管理软件模型的设计，通过使用面向对象的软件建模方法和软件体系结构设计理论，分别对系统的对象结构模型、对象行为模型和系统数据库表以及系统的界面进行了分析设计。首先根据第三章对 TAR 协同数据管理软件的领域分析以及模型优化，抽象出系统的初始类模型，通过使用设计模式和面向 Agent 技术对其进行优化。随后对用例图中的各对象之间的交互过程进行分析，结合 TAR 协同数据管理软件的业务流程及其所含对象的状态变化，使用 UML 构建了系统的顺序图和状态图即对象的行为模型，并在上述分析的基础上结合数据库设计要点，列出了 TAR 协同数据管理软件中的关键基表文件清单，给出了部分基表的详细内容。最后结合界面设计相关要点，给出了系统的部分界面设计实例，为软件的最终实现奠定了软件模型基础。

第五章 TAR 协同数据管理系统的典型应用实现

本章将在领域分析和设计的基础上，给出了 TAR 协同数据管理系统的典型应用的实现。首先给出了系统开发环境的选择及相关配置情况，随后结合面向构件的软件设计思想构建了系统的构件图，并结合 ACME 描述语言对系统的构件进行可实现性描述。最后给出了基于本体的 TAR 协同数据管理系统的典型实现代码。

5.1 开发环境的选择与配置

本软件开发使用 WindowsXP 操作系统平台，数据库是 MySql。采用 Eclipse3.4 开发环境，使用 JDK1.6 作为 JAVA 开发工具，Agent 开发采用 JADE3.6 开发。Web 服务器是 Tomcat6.0。

（1）JDK1.6 的安装配置

安装目录为 D:\Program Files\Java\jdk1.6，环境变量配置如下：

JAVA_HOME = D:\Program Files\Java\jdk1.6;

CLASSPATH = .;%JAVA_HOME%\lib\dt.jar; %JAVA_HOME%\lib\tools.jar;

PATH = %JAVA_HOME%\bin;%JAVA_HOME%\lib;

（2）Tomcat6.0 的安装与配置

安装目录为 D:\Program Files\Tomcat6.0

环境变量配置如下：

CATALINA_HOME = D:\Program Files \Tomcat6.0

启动 Tomcat，输入 http：//localhost:8080，出现 Tomcat 欢迎界面则表示安装成功。

（3）AXIS 的安装与配置

解压 axis-bin-1_3.zip，将解压出来的 axis 目录复制到 Tomcat/webapps 目录下，并将 axis/WEB-INF/lib 目录下的文件复制到 Tomcat/common/lib 下。此时重新启动 Tomcat 并在地址栏中输入 http://localhost:8080/axis/happyaxis.jsp,如果可以访问表示设置成功。

(4) Eclipse 的安装与配置

Eclipse 只需将安装解压即可使用，eclipse.exe 主运行文件。eclipse 在第一次运行时会自动查找系统中安装的 JDK 并完成相应的配置。

（5）JADE 的安装与配置

新建目录 D：\JADE 并将 jade3.7 解压到该目录下，然后在环境变量中添加 JADE_HOME = D:\JADE ， 并 在 CLASSPATH 中 添 加 %JADE_HOME%\lib\jade.jar; %JADE_HOME%\lib \jadeTools.jar; %JADE_HOME%\lib\http.jar; %JADE_HOME%\lib\iiop.jar; %JADE_HOME%

\lib\commons-codec\commons-codec-1.3.jar;%JADE_HOME%\classes。

（6）本体编辑工具 Protégé

Protégé是本体编辑工具，基于 Java 环境开发，具有图形化界面，支持可视化编辑，其支持多重集成具有很强的扩展性，并能对数据进行一致性检查，可定制输出的格式，采用多种表示文本的格式对本体进行转换，本文版本为 Protégé3.3.1，其操作界面如图 5.1 所示。

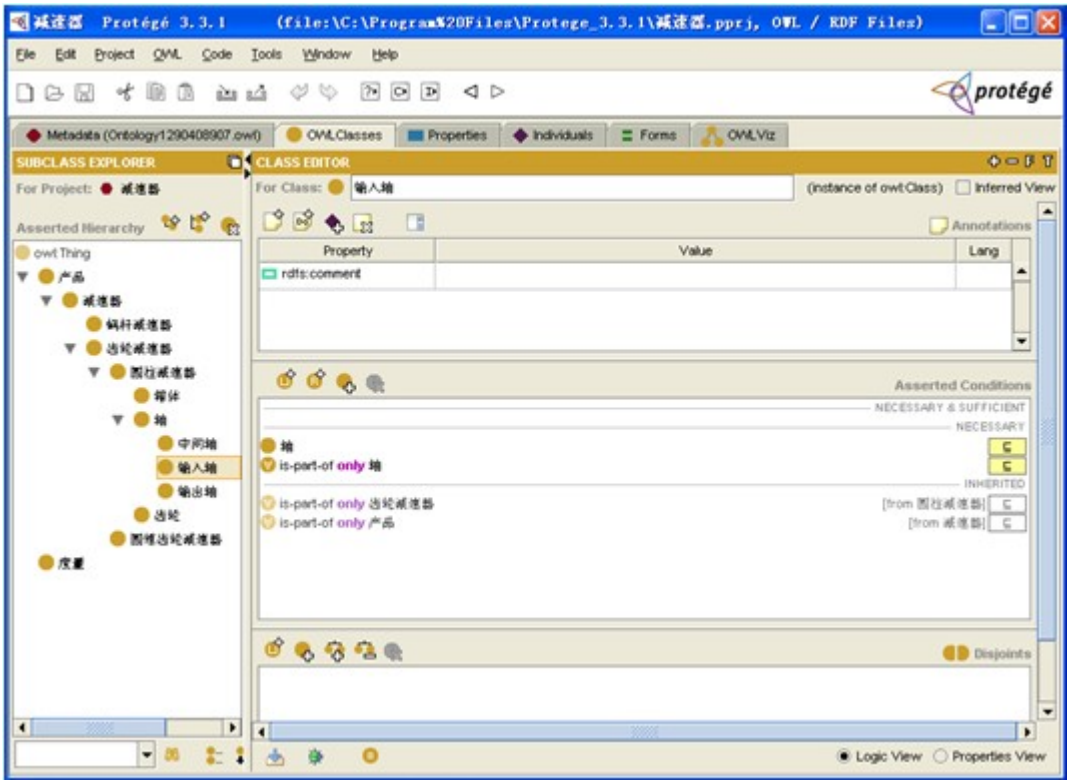

图 5.1 本体建模工具

（7）Jena

Jena 是语义 Web 应用的一种框架，是使用 Java 语言描述且源码开放，为 OWL 和 RDF 提供推理引擎和编程的环境，推理引擎是基于规则的，通过 Jean 应用程序可以对本体模型进行解析、构建或者是查询。Jena 的 API 可对使用采用 OWL 描述形式的本体数据进行处理，本体的 API 和 Jena 推理子系统共同可以提取特定本体中的数据，且提供对导入的本体进行文档管理的文档管理器。Jena 接受数据通过硬盘、OWL 文件和 SQL 进行存取。Jena 可实现 SPARQL，对模型的查询也支持。使用 Jena 的 API 函数可对 OWL 类型的文档解析，提取类和属性等。

在 Eclipse 环境下，右健单击工程名字，选择 Properties→Java Build Path→Libraries→Add Extenal JARs”，找到 Jena2.4 lib 目录下的所有 jar 文件并将其添加到工程即可。

（8）JWNL

本文使用的 WordNet 词典进行语义相似度的计算，Java 程序通过 JWNL 这一接口访问词典

的应用程序。除此之外，还能发现词汇间的关系，并且对其进行语义上的处理。

5.2 TAR 协同数据管理软件的可实现性描述

构架即指软件的体系结构，是可测量、可演化和可重构的软件框架，是可模式化、模型化进而进行模块化的问题解决框架[59]。软件构架是具有整体抽象层次的软件骨架结构，主要由一组构件、构件之间的相互连接关系以及构件之间的相关约束的一种物理视图。

构件（Component）是一个可替换的系统组成部分，通过它来对系统中的一些功能实现进行封装并向外界提供一组可调用接口。构件表示的是物理抽象的概念，一个构件必须具备（1）是一个可交付的物理单元；（2）是一组功能的集合，它的设计必须符合一定的设计标准；（3）能够通过的接口向外界提供各种服务。

连接（Connection）它是构件间进行交互或者接口调用时的中介体，它提供了基于角色的接口说明机制，并且通过它可以描述构件间的通信的路径、机制和预期结果。

约束（Constraint）它是描述构件和连接件的结构组成关系及连接约束，体现了系统构架中的交互接口、连接关系以及构造元素的定义，并给出了为外部提供服务的公共接口。

1.协同数据管理构件图

从构件图来说，分为三种组成部分：处理类构件、数据类构件及连接器[60]。对 TAR 协同数据管理系统的物理系统进行分析，根据其信息处理过程进行建模，获得具有映射关系的软件系统的构架框架，构架中包括个构件模块，其中 logisticsManager、GlobalManager、InteractionMatch、LocalManager、RegisterInfo 属于处理类构件，DataSource 构件属于数据类构件。如图 5.2 所示。

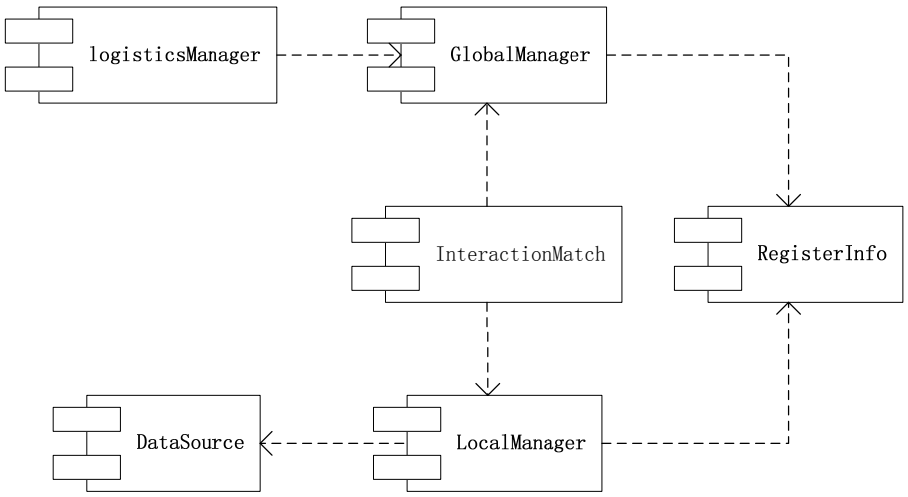

图 5.2 系统构件图

2. 协同数据管理典型构件的 ACME 描述

软件体系结构描述语言（ADL）很多，其中主要包括关注结构化特性的 UniCon、基于事件

驱动的 Rapide、用于描述体系机构的形式化推理的 Wright、基于 XML 的 xArch、以 xArch 为基础的基于 XML 的 xADL2.0 以及为多种体系结构描述语言提供一种可互换形式的 ACME。结合系统的构件图我们选择通用的体系结构描述语言：ACME。

ACME 提供了在不同 ADL 体系结构规范描述之间变换的机制，可作为体系结构设计工具间的通用交换格式，具有很强的中介表达与集成交互能力，是描述体系结构或结构族的一种通用 ADL 工具。ACME 的七种基本的体系结构设计元素实体：构件、连接件、系统、端口、角色、表示和表示图，构件、连接件和系统是三个核心构件。下面对这七个设计元素实体进行说明。

(1)构件（Component）：用来表示各种软硬件资源或它们的一些抽象映射，是组成系统的基本单元。例如外部存储设备资源、数据库、服务器资源或文件系统等都是典型的构件。一般来说，系统计算单元都可以抽象成构件。

(2)连接器（Connector）：用来连接构件之间的元素，为各构件间的通信和相互协作搭建桥梁，在它上面可以进行交互，如管道、远程过程调用、Http 协议和数据库连接等。

(3)系统（System）：由多个构件和多个连接器相结合构成的，描述了系统的各组成元素之间的各种关系，是构件和连接件的配置。

(4)端口（Port）：它是构件与外界进行交互的接口。构件可以通过其所提供的多个端口向外界提供功能和数据源，外界可以通过这些端口调用构件所提供的各种服务。

(5)角色（Role）：表示构件之间进行交互过程中的参与者，对于一些复杂的角色还需要提供构件间交互作用的一些描述信息。

(6)表示（Representation）：主要是对构件或连接件的内部描述，例如在 ACME 中可以根据不同的观点和结构分解方法对同一个元素做多种形式描述。

(7)表示图（Representation Map）：通过表格的形式来表达和展示系统各实体层之间的交互关系。

```
Family AritectureFamliy={  //定义 ACME 体系结构的风格
   Property Type AttributesTemplate = Set{String};
   Property Type MethodsTemplate = Set{String};
   Property Type ConstrainsTemplate = Set{String};
   Property Type PatternsTemplate = Set{<ClassT1, ClassT2, Pattern>};
   Property Type ClassTemplate={
   Property Attributes= AttributeTemplate;
   Property Methods=MethodsTemplate;
   Property Constraints=ConstrainsTemplate;}
   Component Type CompnentTemplate={
```

```
    Property ClassGroups=ClassTemplate;
    Property Patterns=PatternsTemplate;
    Property Ports=PortsTemplate;};
    Connector Type ConnectorTemplate={Roles{caller;callee};};
};
Component GlobalManager:ComponentTemplate=new ComponentTemplate extended with{
        ClassGroup={GlobalAgent,GolbalOntoMag,TraitAnalyze,GrammarDistSimilarity
        ,SemanticDictionarySimilarity,StructureSimilarity,AttributeSimilarity,Ma
        pBld,}
        Patterns={<GlobalAgent, GolbalOntoMag,Association>,
        < GolbalOntoMag,TraitAnalyze,Association>,
        < TraitAnalyze,GrammarDistSimilarity,Aggregation>,
        < TraitAnalyze,SemanticDictionarySimilarity,Aggregation>,
        < TraitAnalyze,StructureSimilarity,Aggregation>,
        < TraitAnalyze,AttributeSimilarity,Aggregation>,
        < TraitAnalyze,MapBld,Aggregation>,}
        Ports={Port1,Port2}
}
Component LocalManager:ComponentTemplate=new ComponentTemplate extended with{
        ClassGroup={LocalOntologyAgent,LocalOntologyBld,InqueryRewri}
        Patterns={<LocalOntologyAgent,LocalOntologyBld,Association>,
                    <LocalOntologyAgent,InqueryRewri,Association>,
                    <LocalOntologyBld,InqueryRewri,Association>,
                    Ports={Port1,Port2}
}
Component DataSource:ComponentTemplate=new ComponentTemplate extended with{
        ClassGroup={DataRequest,DataWrap,DataTemplate}
        Patterns={<DataRequest,DataWrap,Generalization>,
                  <DataTemplate,DataTemplate,Association>}
        Ports={Port1}
}
//其他略
```

```
……
Connector LogisticsManager-GlobalManager:ConnectorTemplate;
Connector GlobalManager-RegisterInfo: ConnectorTemplate;
Connector LocalManager-RegisterManage: ConnectorTemplate;
Connector InteractionMatch-GlobalManager: ConnectorTemplate;
Connector InteractionMatch-LocalManager: ConnectorTemplate;
Connector DataSource-LocalManager: ConnectorTemplate;
Attachments{
  LogisticsManager.Port1 to LogisticsManager-GlobalManager.caller;
  GlobalManager.Port2 to LogisticsManager-GlobalManager.callee;
  GlobalManager.Port1 to GlobalManager-RegisterInfo.caller;
  RegisterInfo.Port2 to GlobalManager-RegisterInfo.callee;
  InteractionMatch.Port1 to InteractionMatch-GlobalManager.caller;
  GlobalManager.Port2 to InteractionMatch-GlobalManager.callee;
  InteractionMatch.Port2 to InteractionMatch-LocalManager.caller;
  LocalManager.Port2 to InteractionMatch-LocalManager.callee;
  LocalManager.Port1 to LocalManager-RegisterInfo.caller;
  RegisterInfo.Port1 to LocalManager-RegisterInfo.callee;
  DataSource.Port1 to DataSource-LocalManager.callee;
  LocalManager.Port1 to DataSource-LocalManager.caller;
}}
```

5.3 TAR 协同数据管理软件的典型应用实现

由于篇幅有限，本节将针对第三章给出的综合相似度计算的本体映射算法给出部分实现代码，在此给出关键的两个算法（语法距离和语义相似度）

```
import java.util.HashMap;
public class AllSimilarMeans{//抽象类，封装常用方法和相似度计算
    private HashMap hm = new HashMap();
    public static String getTag(Entity ent)throw Exception{ //提取元素的字符串信息
       String outcome;
       outcome = ent.getTag(VocabularyAdaptor.INSTANCE.getLanguageURI("entity"));
       if(outcome == null){
       outcome = ent.getURI();
```

```
        while(outcome.indexOf('#')!= -1){
                outcome = outcome.substring(outcome.indexOf('#')+1);}}
    return outcome;}
    public double getSimilarresult(TupelofEntity tent)throws Exception
    {//计算相似系数
        Double outcome = (Double)hm.get(tent);
        if(outcome == null){
            outcome = new Double(getSimilarresult(tent.ent1,tent.ent2));
            hm.put(tent, outcome);}
        return outcome.doubleValue();
    }
    protected abstract double getSimilarresult(Entity ent1,Entity ent2)throws
Exception;//算法方法实现接口
}
public class GrammarDistSimilarity extends AllSimilarMeans{//语法距离相似度计算
    protected double getSimilarresult(Entity ent1,Entity ent2)throws Exception{
        String str1,str2;
        double gdNum,distrubutor;
        str1 = getTag(ent1);
        str2 = getTag(ent2);
        gdNum = (double)GrammerDistcalculate(str1,str2);
        int strlen1,strlen2;
        strlen1 = str1.length();
        strlen2 = str2.length();
        distrubutor = strlen1>strlen2?strlen1:strlen2;
        return 1-gdNum/distrubutor;}
    private int GrammerDistcalculate(String str1,String str2){
        int strlen1,strlen2,i,j;
        strlen1 = str1.length();
        strlen2 = str2.length();
        int dist[][] = new int[strlen1+1][len2+1];
        dist[0][0]=0;
```

```
        for(i=1;i<=strlen1;i++)    dis[i][0]=i;
        for(j=1;j<=strlen2;j++)    dis[0][j]=j;
        for(i=1;i<strlen1;i++){
            for(j=1;j<=strlen2;j++){
            dist[][]=Math.min(str1.charAt(i-1)==      str2.charAt(j-1)?dist[i-1][j-
1]:dist[i-1][j-1]+1,Math.min(dist[i-1][j+1],dist[i][j-1]+1));}
            return dist[strlen1][strlen2];}}
   }
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Set;
import sun.text.CompactShortArray.Iterator;
public class SemanticDictionarySimilarity extends AllSimilarMeans{//语义相似度计算
    private boolean isSyn(Set setUri1,Set setUri2){//检查两个词是否是同义词
       Iterator its;
       String suri;
       Set synSet = new HashSet();
       its = setUri1.iterator();
       while(its.hasNext()){
              suri=(String)its.next();
              synSet.addAll(getSynonymURI(suri));}
              its = setUri2.iterator();
              while(its.hasNext()){
                     suri=(String)its.next();
                     if(synSet.contains(suri)) return true;
              }
              return false;
       }
    protected double getSimilarresult(Entity ent1,Entity ent2) throws Exception{
       final int SIZE=2;
       String sstr =new String[];
       String SURI;
```

```
    double distrubutor;
    Set setUri[] =new HashSet[];
    Iterator its;
    sstr[0] = getTag(ent1);
    sstr[1] = getTag(ent2);
    if(sstr[0].equalsIgnoreCase(sstr[1]))return 1;
    try{
           for(int i=0;i<sstr.length;i++){
           setUri[i]=getLogicalURI(sstr[i]);}
           }
           catch(SQLException ex){
                  throw new Exception("error");
           }
           if(this.isSyn(setUri[0], setUri[1])) return 1;
           else return getHyperSimilar(setUri[0],setUri[1]);
    }
protected double getHyperSimilar(Set setUri1,Set setUri2)
    throws Exception{//上义词间的相关程度
    Set ctpy1 =getHyper(setUri1);
    double sizeCtpy1=ctpy1.size();
    Set ctpy2 = getHyper(setUri2);
    ctpy1.retainAll(ctpy2);
    double intersect = ctpy1.size();
    double union = sizeCtpy1 + ctpy2.size()-intersect;
    return(intersect/union);
    }
private Set getHyper(Set ent)throws Exception{//检索词的所有上义词
    Set toRen = new HashSet();
    Iterator itrs = ent.iterator();
    while(itrs.hasNext()){
          Concept temp = (Concept)wordNetModel.getConcept((String)itrs.next());
           Set father = temp.getSuperConcepts();
```

```
            Iterator it = father.iterator();
            while(it.hasNext()){
                    Concept curt =(Concept)it.next();
                    toRen.add(curt);
                    toRen.addAll(curt.getAllSuperConcepts());
                    }
            }
            return toRen;
      }
   }
```

5.4 本章小结

本章主要是对 TAR 协同数据管理软件构架的部分典型功能进行了实现。首先阐述了开发环境的选择和运行环境的配置，展示了 TAR 协同数据管理软件的实际开发过程。随后使用面向构件的软件建模方法构建了系统的构件图，并使用 ACME 对系统的部分构件进行了可实现性描述。同时给出了系统综合相似度计算的本体映射算法部分实现代码。

第六章 总结与展望

在撰写论文过程中，本人阅读了大量与课题相关的的文献和书籍，其中包括协同数据管理、电子商务技术、协同计算技术、语义 Web 等研究领域的知识；面向对象和面向 Agent 的软件开发方法、工作流技术和设计模式等方法论知识；UML 面向对象建模、工作流执行语言 BPEL、知识查询与操作语言 KQML、软件体系结构描述语言 ACME 和本体描述语言 OWL 等建模和表示方法；多 Agent 开发工具 JADE、Eclipse 等软件开发工具。将所学的理论知识较好地运用到了实际的软件开发流程中，获得了一定的成果。

本文的总体设计思路是，首先在 SOA 及其实现技术的指引之下，给出了基于 SOA 的 TAR 数据协同管理的软件体系结构图。通过对电子商务应用的领域知识的分析构建了基于 SOA 的 TAR 数据协同的功能模型。采用 UML 建模工具对系统的对象体系结构进行分析设计，通过面向 Agent 的技术对模型进行优化，使用可视化 / 形式化描述（BPEL、ACME、KQML）对软件模型进行可实现性描述。最后基于开发平台对软件模型进行实现。在整个开发流程中运用了面向对象方法、软件架构设计方法并结合软件模型优化方法对数据协同管理框架进行分析、设计和开发。现对本课题的主要工作收效及创新点做如下总结。

（1）通过对 SOA 及其实现技术的分析，构建出基于 SOA 的 TAR 协同数据管理框架，实现了 SOA 技术在资源调度和信息共享方面的松耦合性，将 TAR 框架较好地融入到协同数据管理中。

（2）通过对数据交互共享技术的分析，构建了如图 2.8 所示的 Mediator/Wrapper 框架，采用该框架能较好地解决异构数据源交互共享的难题，为用户提供统一的查询视图。

（3）通过对进行面向商家的电子商务托管式物流管理领域分析，构建了如图 3.1 所示的功能模型和图 3.2 所示的数据管理模型，并采用业务流程描述语言 BPEL 对物流订单管理模块的业务流程进行了形式化描述。

（4）通过对 TAR 数据协同管理模型进行分析，结合 UML 建模工具构建了如图 3.3 所示的数据管理用例模型。该模型使得 TAR 框架较好地运用了 SOA 的松耦合的特性，通过资源发布/发现实现间接寻址和直接寻址的衔接，并将 Mediator/Wrapper 框架加入其中解决异构数据源访问问题。

（5）通过分析本体建模技术，在 2.4.2 节给出了 OWL 描述模板，使得对资源的描述可以用模板的形式固定下来，具有较好地通用性。通过对应用领域的分析结合图 3.6 的本体模型，构建了如图 3.7 和图 3.8 所示局部本体树和图 3.9 所示的全局本体树。针对图 3.9 的全局本体树运用 OWL 描述模板给出了部分 OWL 描述。

（6）分析了单一本体映射方法的不足并对其进行了改进，构建了一种综合的相似度计算的本体映射方法。该方法充分考虑资源的语法异构、语义异构和结构异构的情况，对不同的本体具有较好地适应性，使得相似度计算的结果准确性更准确，能较好捕捉本体实体之间的映射关系。

（7）采用面向对象的软件建模方法对协同数据管理框架进行分析和设计，结合 UML 建模工具构建了初始类模型如图 4.1 所示，并结合 Agent 技术对该对象体系结构进行优化，从各实体对象中提取具有 Agent 特性的实体，进一步提高 TAR 协同数据管理的智能性和协同性，并通过扩充 KQML 原语对资源请求过程进行了描述。

随着软件设计技术的不断发展，研究工作需要不断深入，对下一步要进行的研究有了如下设想。

（1）通过组合多种设计模式对软件体系结构做进一步优化，从而降低系统各个部分之间的耦合度，提高软件组件的重用性。

（2）可将主动服务的思想加入到 SOA 模型中，提取用户的兴趣模型为用户提供个性化的资源请求服务。

（3）本课题没有对系统的安全性进行充分考虑，如果系统需要投入实际的运营，就要对系统安全做更多的研究。

（4）改进全局本体的生成机制，实现本体间的动态生成与维护，进一步提高本体生成的自动化程度，对本体间的映射模型作进一步研究，不断优化本体映射方法。

经过一年多时间的研究学习，对软件系统的开发过程有较深入的理解，通过从领域需求分析到系统软件框架设计、系统对象体系结构设计及优化、构件设计等软件开发过程的实践，进一步提升了软件设计理论知识。通过编写软件实现代码，进一步提高了编程能力。通过阅读各种软件开发方面的书籍，能够更全面地理解问题、分析问题，不断地弥补自身的不足。回顾整个系统设计开发的过程，感触颇多，这些日后都将成为学习和工作的宝贵财富。

参考文献

[1]John H. Connolly and Ernest A. Edmonds (Eds.).CSCW and artificial intelligence. 北京:世界图书出版公司,1998.

[2]Dan Diaper and Colston Sanger (eds.) CSCW in practice:an introduction and case studies. 北京:世界图书出版公司,1999.

[3]汤庸,冀高峰,朱君等.协同软件技术及应用.北京:机械工业出版社,2007.

[4]高西(S.P. Ghosh)著,文卿译.数据管理的数据库组织/(美).北京:国防工业出版社,1982.

[5]塞勒姆(Salemi,Joe)著,石祥生译.客户/服务器实用技术指南:Sybase sql 服务器/(美).北京:电子工业出版社,1994.10.

[6]王云鹏,郭学旭,潘翔等.计算机辅助协同设计中的数据管理研究.计算机辅助设计与图形学学报,2003,15(11)：1415~1421.

[7]吴丹,王先逵,魏志强等. 基于协同服务平台的分布式产品数据管理. 清华大学学报（自然科学版）,2002,42（6)：791~794.

[8]谢建平,于科,万立等. 基于 Web 的协同设计环境下产品数据管理系统模型研究. 计算机工程,2001,27（4)：34~37

[9]邱福生,刘文剑. 基于多智能体的网络协同产品数据管理技术研究. 计算机集成制造系统,2006,12(5)：702~794.

[10]卢止鼎,李兵,肖卫军等.基于 XML 的文件系统与多数据库系统的集成[J].小型微型计算机系统,2002,23(5)：588~591.

[11]Togar M.Simatupang,R.Sridharan. The collaborative supply chain.International Journal of Logistics Management,2002,13(1):15~30.

[12]Michael P.Papazoglou 著,龚玲,张云涛译.Web 服务原理和技术.机械工业出版社,2010.

[13]丁兆青,董传良等.基于 SOA 的分布式应用集成研究,计算机工程,2007,33(10),246~248.

[14]Anne James. Computer Supported Cooperative Work: Some Perspectives and the Hidden Web.Proceedings of the 2007 11th International Conference on Computer Supported Cooperative Work in Design.2007:3~6.

[15]Sunderam Vaidy,Hirsch Michael,Gray Paul,et al.CFF:Collaborative Computing Frameworks.IEEE Internet Computing,2000,4(1):16~24.

[16]Bayardo R,Bohrer W,Brice R,et al.Infosleuth：Agent-based Semantic Integration of Information in Open and Dynamic Enivironments[C].In Proceedings of the ACM

SIGMOD International Conference on Management of Data(ACM'97),2000：195~206.

[17]陈启祥,杨军. 面向服务的软件架构 SOA 以及其支撑技术的研究[J]. 湖北工业大学学报,2005,20(4)：38~41.

[18]Stojanovic Z . A Method for Component-based and Service-oriented Software Systems Engineering[J]. Delft University of Technology,2005,(5)：457~460.

[19]Zhu Q,Sun Y,Motheramgari S.Developing Cost Models with Qualitative Variables for Dynamic Multi-Database Environments[R].In Proceedings of IEEE International Conference On Data Engineering(ICDE 2000),San Diego,2000：413~424.

[20]Fausto Ibarra.The Enterprise Service Bus:Building Enterprise SOA[M].US.BEA systems,2004:37~103.

[21]Liangzhao Zeng,Boualern Benatallah etc.QoS-Aware Middleware for Web Services Composition[J].IEEE Transactions on Software Engineering,2004,30(5):311~327.

[22]Srinevasan N,Paolncci M,sycara K.Semantic Web service discovery in the OWL-s IDE[A]Proceeding of the 39th Annual Hawaii International Conference on system Sciences,Kauai Hawaii,USA:IEEE Computer Society,2006:1~10.

[23]M.Wooldrige 著,石纯一等译. 多 Agent 系统引论. 电子工业出版社,2002.

[24]张健,曾广周.面向 Agent 的基本思想研究.计算机工程与应用,2007,43（4）：9~11.

[25]Christina B.Vilakazi and Tshilidzi Marwala.Agents and Multi-agent Systems and Application to Condition Monitoring. IEEE, 2007:644~649.

[26]Marek Woda, Tomas Walkowiak. Agent based approach to events monitoring in Complex Information Systems. The Second International Conference on Emerging Security Information, Systems and Technologies, 2008:397~402.

[27]Yue Zhang,Mark Panahi and Kwei-Jay Lin.Deployment of Accountability Monitoring Agents in Service-Oriented Architectures.CEC-EEE,2007.

[28]Tobias Wittmann.Agent-based models of energy investment decisions. Heidelberg:Physica-Verlag Heidelberg, c2008.

[29]唐小燕,李斌,许有志.一种 Agent 协同规范及其应用研究.计算机应用研究,2005,(01):28~48.

[30]N.R.Jennings,P.Faratin. Autonomous Agents For Business Process Management. Applied Artificial Intelligence,2000, 14 :421~463.

[31]Michael Wooldridge,Nicholas R.Jennings.Agent theories,architecture,and languages:A survey.In Intelligent Agents,Springer,1994.

[32]赵文,胡文蕙,张世昆. 工作流元模型的研究与应用.软件学报,2003,14(6):1052~1058.
[33]王莉,刘厚泉,吴雪峰. 基于 BPEL 的业务流程管理系统架构的研究与应用.计算机工程与设计,2006,27（18）：3507~3510.
[34]李红臣,史美林. 工作流模型及其形式化描述. 计算机学报,2003,26(11)：7541~13641.
[35]史美林.计算机支持的协同工作理论与应用.电子工业出版社,2000.
[36]张颖,施海虎,柳军飞. 一种以活动为中心的软件过程元模型.计算机工程与设计,2004，25（4）：612~615.
[37]李长云,邬惠峰,应晶等.支持领域复用的过程元模型.小型微型计算机系统,2006,(6)：1083~1087.
[38]范玉顺.工作流管理技术基础—实现企业业务过程从组、过程管理与过程自动化的核心技术. 清华大学出版社,2007.
[39]孙瑞志,史美林. 支持工作流动态变化的过程元模型.软件学报, 2003,14(01):62~67.
[40]王晓东.基于 ontology 知识库系统建模与应用研究,华东师范大学,博士学位论文,2003.
[41]刘升平.XML 的模型论语义及其应用,北京大学,博士学位论文,2005.
[42]邓世鸿,含世渭,杨冬青等.基于 xML 的本体表示和检索技术的研究,计算机工程与应用,2003,3:14~15
[43]王湖南.基于 RDF 的知识表示机制的研究,电子科技大学,硕士学位论文,2002.
[44]邓勇,丁峰,沈钧毅. 基于 UML 的软件体系结构建模方法的研究. 小型微型计算机系统,2001, 22 (10): 1206~1209.
[45]Grady Booch, James Rumbaugh, Ivar Jacobson 著.UML 用户指南.机械工业出版社,2006.
[46]陈琴,朱正强. UML 在设计模式描述中的应用. 计算机工程与设计,2003,24（4）：81~84.
[47]E.Gamma 等著,李英军等译.设计模式.机械工业出版社,2000.
[48]Mary Shaw,David Garlan. 软件体系结构（牛振东,江鹏,金福生等译）. 清华大学出版社,2007.
[49]Alexander Budanitsky,Graeme Hirst.Sematic distance in WordNet:An experimental,application-oriented evaluation of five measures[A]. //In:Proceedings of the NAACL 2001 Workshop on WordNet and other lexical resources[C],2001.
[50]毛新军.面向主体的软件开发.北京: 清华大学出版社,2004.
[51]万麟瑞,胡宏,孙红星. 面向构件的软件开发方法研究. 小型微型计算机系统,2003，24(3)：365~370.
[52] 胡晓辉,周兴社,党建武.一种面向 Agent 的分布监控系统行为建模和设计方法. 计算机应

用,2005,25：284~286.
[53]王志坚.软件构件技术及其应用.科学出版社,2005.
[54]杨芙清,梅宏,李克勤. 软件复用与软件构件技术. 电子学报,1999,27(2).
[55]张世琨,张文娟,常欣等.基于软件体系结构的可复用构件制作和组装.软件学报,12(9):1351~1359.
[56]曹建福,周理琴. 基于构件的软件开发模型及其实现. 小型微型计算机系统,2002,23(6)：99~102.
[57]阎宏.Java 与设计模式.北京.电子工业出版社.2002:621-635.
[58]Grady Booch, James Rumbaugh, Ivar Jacobson 著.UML 用户指南.机械工业出版社,2006.
[59]Sung, CS,Park SJ. A component-based product data management system[J]. International Journal of Advanced Manufacturing Technology, 2007, 33(5):614~626.
[60]Teng Zhang.YuShun Fan.Eleanor T.loiacono.A practical scheduling method based on workflow management technology. Int J Adv Manuf Techmol,2004:919~924.

致 谢

两年半的研究生生涯即将结束，回首往昔，在这两年半的研究学习中，我经历了太多的困惑与艰辛，但是乐观向上的生活态度帮助我克服了种种困难，在解决问题的过程中我也收获了很多。

首先，我要向我的导师万麟瑞老师表示深深的尊敬和感激。两年多来，，他的言行品德、严谨的治学态度为我树立了榜样，万老师不管在学习还是思想上都给了我很多有用的意见和建议，通过他渊博的学识及时纠正我的研究误区。尤其是在我硕士毕业论文的撰写上，他细致耐心地和我讨论研究中遇到的种种问题，给了我大量的指导意见，使我最终顺利完成了论文的撰写并迎接毕业答辩。这份恩情我会永远记在心里，并衷心的祝愿万老师身体健康，工作顺利，生活愉快！

其次要感谢我的父母，感谢你们在我的成长道路上一直陪伴我，给予我最大的精神鼓励和默默地支持，感谢你们为我竭尽全力地提供更好地学习和生活条件，你们的爱我唯有用更加努力的工作和学习予以回报。

还要感谢 508 教研室所有兄弟姐妹们的支持与鼓励。在学习上，我们经常围坐在一起探讨我们所遇到的学术问题，共同进步，营造了一个良好的学习氛围；在生活上，我们相互帮助，闲暇时携手踏青郊游，劳逸结合，不仅提高了我的学习效率也在我的脑海里留下了很多美好的回忆；感谢我的室友们，是你们在我遇到困难时及时与我分担，在我收获成功时同我一起分享；感谢南航，为我提供了一个良好的学习条件和氛围，让我能够在校园里安静的生活学习和思考。

最后向在百忙之中抽出时间审阅论文和参加答辩的各位老师表示由衷的感谢！衷心地感谢理解、支持和帮助过我的所有人！

在学期间的研究成果及发表的学术论文

[1] 王月．基于本体的航材采购数据管理软件模型研究．中国通信学会第八届学术年会．（已录用）