密级：　　保密期限：

北京郵電大學

硕士学位论文

题目：物联网系统的知识管理子系统的设计与实现

学　　号：_____

姓　　名：_____

专　　业：计算机技术

导　　师：_____

学　　院：网络技术研究院

2016 年 12 月 15 日

独创性（或创新性）声明

本人声明所呈交的论文是本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，也不包含为获得北京邮电大学或其他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作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：_____ 日期：_____

关于论文使用授权的说明

学位论文作者完全了解北京邮电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北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许学位论文被查阅和借阅；学校可以公布学位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、汇编学位论文。（保密的学位论文在解密后遵守此规定）

非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：_____ 日期：_____

导师签名：_____ 日期：_____

基于物联网平台的知识子系统的设计与实现

摘　要

近几年来，物联网发展极其迅速。伴随其发展，物联网设备产生的信息量也越来越大。物联网传感设备本身的异构性和多样性导致产生的信息具有异构性、不稳定性、演化性等特点。因此基于本体的物联网资源模型和资源框架已经成为解决设备接入、控制、管理问题的基础。同时，在物联网中，信息往往都是以事件到来的形式。如何对事件进行有效的处理和响应成为物联网平台的重要问题。已有的基于规则的处理方案对于规则过于细化，在实际使用中十分繁琐。而且，过于依赖人工定义规则会十分容易出错，从而无法保证正确性。

因此，在已有的基于模型的框架基础上，为了更有效的利用物联网资源和已有的知识，本论文提出了一个管理已有物联网知识模型的软件系统—知识管理子系统。

本论文主要从三个方面对知识管理子系统进行研究。首先，在深入研究物联网知识建模，描述逻辑与一阶逻辑的转化，调研 Jena 框架的基础上，借鉴发布订阅的事件推送机制，提出并设计了一个基于事件推送机制的物联网平台知识管理子系统。

然后，本论文分别设计与实现了知识子系统的各个模块。知识库管理模块，包括添加，删除，备份，语义 SPARQL 查询等管理功能。其次事件接入和转化模块，基于实验室已有的发布订阅系统实现系统所需的事件转发等功能。推理模块，设计并实现了包括 OWL 加载、OWL 解析、一阶逻辑表达式转化在内的整个分析转化流程，并着重阐述了一阶逻辑转化规则和基于发布订阅的实时推理功能的设计与实现。通过将用户的需求转化成一阶逻辑表达方式，结合已有的存储在 Web 后台 TDB 数据库里的物联网资源模型，利用一阶逻辑推理器快速的验证是否成立，从而推出结果。

最后，通过对系统进行功能和性能测试，测试结果表明，本论文所设计的基于物联网平台的知识管理子系统能够方便的对模型管理和查询，及时的处理事件信息，并能结合用户需求迅速推理出结果。该系统的实现对于物联网知识管理问题具有一定的参考价值和意义。

关键词　物联网　知识管理 本体 描述逻辑 一阶逻辑推理

DESIGN AND IMPLEMENTATION OF LOGISTICS MATCHING TRANSACTION SYSTEM BASED ON ANDROID

ABSTRACT

In recent years, The development of Internet of Things is extremely rapid. With its development, Internet of things the amount of information is also growing. The heterogeneity and diversity of the sensing devices in the Internet of Things cause heterogeneity, instability and evolution of the information. Therefore, the ontology-based Internet of Things model and resource framework have become the basis for solving the problem of access, control and management of equipment. At the same time, in the Internet of Things, the information is often related to the pattern of the arrival of the event. How to effectively deal with the incident has become an important issue in the Internet of Things platform. The existing rule-based processing scheme is too detailed for the rules, which is very complicated in practice. Moreover, overly dependent on the definition of artificial rules is very error-prone, can not guarantee correctness.

Therefore, on the basis of the existing model-based framework, in order to make more effective use of the IOT resources and existing knowledge, this paper proposes a knowledge management subsystem, which is a software system for managing the existing knowledge model.

In this paper, the knowledge management subsystem is studied from three aspects. First of all, on the basis of deeply researching the modeling of IOT knowledge, transformation of description logic and first-order logic, and Jena framework, this paper proposes an event-push mechanism based on event-push mechanism. Subsystems.

Then, we design and implement the modules of knowledge subsystem respectively. Knowledge management module, including add, delete, backup, semantic SPARQL query management functions. Second,

publish and subscribe module, based on our laboratory's existing publish and subscribe system to achieve the required functions such as event forwarding. Design and implementation of the first-order logic transformation rules and the real-time reasoning function based on publish-subscribe, the paper designs and implements the whole analysis transformation process including OWL loading, OWL parsing and first-order logical expression transformation, and emphatically elaborates the design and implementation. The first-order logical reasoning is used to verify the first-order logical expression which transformed from the user's definition and the existing object network resource model stored in the TDB database.

Finally, through the function and performance test of the system, the test results show that the knowledge management subsystem based on Internet of Things platform can conveniently manage and query the model, and process the event information in time, and combine with the user's requirement quickly reasoning out the results. The realization of this system has some reference value and significance for the knowledge management of Internet of Things.

KEY WORDS IoT Knowledge Management Ontology DL FOL

目录

第一章 绪论

1.1 研究背景

物联网(Internet of Things)，指的是将通过各种信息传感设备，如传感器、激光扫描器，射频识别器，定位系统，红外感应器等装置与互联网结合而形成的一个巨大网络。其目的是让所有的物品都与网络连接在一起。物联网服务系统可以自动的，实时的对物体进行识别，定位，追踪，监控并触发相应的事件。

随着物联网技术的迅速发展，其应用领域已经覆盖人们生活的各个方面。目前物联网的应用领域主要包括射频识别、传感器网络、M2M 和两化融合等（信息化与工业化的高层次深度结合）等。与此对应的，出现了非常多的物联网应用系统，包括监控系统、智能家居、锅炉房监控系统等。如此众多的监控设备，导致了物理传感数据的差异。系统需能够接收多种不同的实时异构数据，并对这些数据进行处理，从而实现对设备的监测和控制。因为物联网的数据(事件)都是由各种传感器上传的，因此具有非常强的实时性。同时在很多应用领域需要物联网在接收事件之后做出迅速的处理并及时的反应。在事件处理方面，论文[1]提出了一个基于规则的物联网事件处理方法，但此方法主要基于非常详细的规则来实现，对于一个新的类型事件需要添加许多详细的规则，在实际应用操作上对于操作人员来说难度非常大，而且基于如多的人为定义规则很容易产生错误，所以方法需要改进。

对于大量的物联网异构数据，人们更多的是希望处理并整合其中的有价值和高聚合的信息。从大量信息中分析过滤出有用的信息。因此，为了更有效的利用资源和已有的知识，我们需要有效的管理。应对数据的异构性，可能通过建模的方式，用模型来屏蔽差异。在物联网应用领域，国内许多学者和专家都采用本体表示的方法来对物联网资源进行建模。本体起源于哲学概念，概括来说是概念与概念间关系的形式化表示。本体描述语言能描述复杂的关系，目前在语义网、医学信息共享、知识管理、地理信息共享以及数字图书馆等领域有广泛的应用与研究。本体是构建在描述逻辑的基础上的，可以作为一种能在知识和语义层次上的描述物联网概念模型的建模工具。而且因为其与描述逻辑的强关系，因此具有一定推理能力。描述逻辑是一阶逻辑的可判定子集。同时因为描述逻辑拥有清晰的语义描述，而且能进行形式化的表示，所以本体描述语言大都是以描述逻辑为基础。对物联网中的资源进行建模包括对分领域描述物联网中的资源和实体、信息存储、构建实例模板。

在物联网整个大的环境中，物联网中的数据通常是通过事件的方式到来。因此事件同样需要建模。论文[1]提出根据不同领域的事件分别进行建模，通过描述事件的领域信

息，沿用本体对资源进行标准化的描述，上层将物联网的信息分层次分类别进行描述，生成资源，实体，属性以及概念的实例化数据等元素提出的资源建模工具可以有效的对物理资源进行建模，绑定和存储。但是此工具缺乏相应的管理和查询功能。另外对于本体的查询，W3C 定义了与之相关的 SPARQL 语义查询标准。而且，众多的语义库都对 SPARQL 有支持，因此，可以利用 SPARQL 对已有的本体模型进行查询，例如，当事件来到之时，我们可以根据事件所有携带的信息进行 SPARQL 查询，从而得到事件相关的领域信息，然后做进一步的处理因为模型反映的是事件的描述性信息。因此，以知识库的形式将模型组织管理起来能够带来非常大的便利。

另一方面，利用已有的事件模型和领域模型之间的关系，结合本体与描述逻辑之间的关系，我们可以通过推理的方式来验证信息和目标。例如，我们可以定义目标。并且将目标的定义通过一阶逻辑描述，已有的模型和事件关系同样处理成一阶逻辑的形式，然后通过推理和证明来验证目标是否成立。从而挖掘出事件中隐含的信息，并决定如何处理事件。描述逻辑是一阶逻辑的可判定子集，而模型的描述语言 OWL 也是以描述逻辑为基础的，在这方面，论文在描述逻辑和一阶逻辑之前的转化规则方面有非常详细的概述。目标是结合已有的模型定义的，比如，可以结合出租车的 GPS 数据、道路信息等进行建模，然后定义拥堵的目标，将目标处理成一阶逻辑表达式。随着事件的到来，将事件携带的领域信息（通过查询知识库）同样处理成一阶逻辑表达式。这可看作是一阶逻辑中的 p，将目标看作 q，利用一阶逻辑推理库的推理验证能否从 p 推出 q，并根据能否推出 q 分别对事件作出不同的响应。同时，因为事件是实时更新的，因此在不同的时刻，p 也会同步更新。从而导致推理结果的实时更新，从而根据推理结果进行相应的反馈。

1.2 研究内容

本课题的主要研究内容是设计并实现一个物联网知识管理子系统。需要运用面向领域的设计思想，通过建模过程屏蔽底层的本体信息，把业务相关的概念向用户呈现。提供一个可视化的资源建模环境。需要支持资源和实体的绑定功能，以满足感知信息到情景信息的转化。提供一个用户友好的 Web 管理平台，提供 SPARQL 查询功能。结合发布订阅事件系统，对事件进行演绎推理新的知识的功能。需要支持物联网实时数据处理功能，提供并行处理能力。

根据以上要解决的技术难点。本设计要进行以下几个工作：

1）数据的建模：本课题利用已有的资源框架和建模思想、方法、工具对领域信息进行建模。采用本体 OWL 对资源进行标准化的描述, 通过借鉴业内优秀的的建模

方法和工具，将物联网中的信息分领域进行描述和构建资源模板和资源实例，生成类、属性、实例等元素，

2）资源模型本体的管理：已有的众多模型本体需要一个系统的知识的管理模块。方便查询和管理使用。本课题结合已有的用 owl 表示知识的条件，采用 Jena 的文件数据库 TDB 作为内部数据存储系统。提供基于语义 Web 的 SPARQL 查询。同时开发一个 web 管理界面，使用户方便的管理和查看知识库。

3）目标和规则的定义：以服务的形式提供给用户。本课题结合目前已有的深圳出租车的运行数据集，通过对数据进行建模和事件建模，绑定具体数据到已有模型。例如通过定义区域在某一时刻是否拥堵为目标进行推理。事件到来之时，触发相应的操作，包括根据知识的领域进行相应分类，然后结合所属领域或者其他领域的知识进行处理。

4）推理和并行：本课题通过定义好的目标条件，结合已有的数据模型，通过利用 SMT 理论求解器的一阶逻辑推理功能，快速验证已有一阶逻辑的可满足性，从而推出条件是否可满足，进而推知目标条件是否成立并做出相应的处理。

本课题采用的数据来自深圳市出租车的实时 GPS 数据，每条数据均看作一个实时事件。事件均以发布订阅的形式到来。事件的处理流程如下：每当事件来临，通过事件所属的领域信息，通过 SPARQL 查询语言从已有的知识库中查询相对应的模型，如果没有则不处理。如果有相对应的模型，则根据模型对事件进行处理并实例化，此实例化过程可以利用 Jena API 进行处理，通过解析模型，同时结合事件的属性，编写解析工具处理成 Z3 的处理格式。在此过程中，历史生成实例均按其所属领域和时间线存储在持久库中，同时，对于最近的时间段缓存一部分在内存之中。本课题采用了深圳出租车实时状态数据，数据大概每 30 秒记录一次，数据包括采集时间，经度，纬度，空车状态，行车方向等字段。在缓存之中除了已有的按照时间顺序组织的队列之外，还有采用队列加多个哈希表的维护方式，对于每一辆车维护一个最新的状态，新的实例来临，根据哈希表可快速的判断状态是否变化，若无变化则可不更新，从而节省时间和空间。如果状态发生变化，则分别对应每个对每一个并根据当前的状态进行分类，取出时间顺序队列里的数据交给推理模块执行。同时，新的实例会加入到时间顺序队列，并在一定的周期内写入到持久库之中。

本课题在前人基础上，结合已有的资源模型和建模工具，同时结合不同业务领域，不同资源之间的相关性，开发知识管理子原型系统。在已有的建模工具的基础上，为了方便用户的管理，以服务的形式提供用户便捷查看和管理知识。提供基于语义 Web SPARQL 查询功能。以及根据事件与已有模型以及定义的目标进行推理的功能。

本论文的主要研究内容包括：

(1) 物联网知识子系统功能需求的研究与总结。调研当前资源建模和管理常用的技术，如本体在物联网建模的利用，事件处理，描述逻辑和一阶逻辑的关系。一阶逻辑推理器的使用等。从而结合并总结出本课题所设计的系统需要的功能。

(2) 物联网知识子系统实现所需的关键技术的研究。

1) OWL 本体存储，查询，推理的研究和设计。调研 Jena 语义框架的使用，调研 Hermit 的 OWL 解析机制和源码分析，获取 Hermit 的中间处理结果。设计出基于 Jena TDB 的 OWL 持久化和管理方案，结合已有需求并实现关键代码。

2) 基于 Web 的研究与设计。调研前端框架 VUE，Angular，调研后台开发框架 Jetty，结合需求实现关键部分代码。

3) 推理功能的研究与设计。调研微软开源推理器 Z3 等推理软件的推理功能的设计和使用。通过功能界面推导出后台的实现，进而设计出推理模块的功能，并完成关键部分的编码。

(3) 物联网知识子系统设计和实现。利用一阶逻辑理论求解器开发技术，利用 Hermit 的 OWL 解析和中间结果，采用 Jetty 作为后台服务器，同时结合发布订阅推送机制和已有的 Fuseki 本体存储服务器，实现一个较为完整和成熟的原型系统。

1.3 论文结构

本论文共包括六章，每个章节的主要内容安排如下：

第一章是绪论，首先介绍了本论文的研究背景，然后明确了本文的研究内容，最后对本论文的组织结构进行了说明。

第二章是相关技术，主要是对本课题中涉及到的相关技术进行介绍，主要包括物联网网络本体建模、描述逻辑到一阶逻辑的转化、Jena 框架、理论求解器 Z3 等关键技术。

第三章是需求分析和概要设计，首先对本论文中设计的系统进行了需求分析，然后描述了本论文中设计的系统的功能结构和整体架构。

第四章是详细设计及实现，针对第三章提出的物联网知识管理子系统中的重点模块进行详细设计和实现的说明，主要包括知识管理模块、事件接入模块，基于模型的推理功能模块。

第五章为系统测试及验证，主要介绍针对物联网知识管理子系统的功能及性能测试，并给出测试结果分析。

第六章为总结与展望，主要对本文的研究工作进行了总结，并且对后续的研究工作进行了展望。

第二章 相关技术

本章内容是介绍本论文中所涉及到的相关概念和技术。其中包括：本体领域建模、描述逻辑和一阶逻辑、Jena 语义框架以及一阶逻辑推理器 Z3。

2.1 本体建模

在设计和实现物联网知识子系统的过程中，为了更方便的存储领域知识，我们采用具有丰富表达能力的本体来作为领域知识的载体。为了解析目标和资源模型 schema，需要掌握网络本体语言库的使用。

2.1.1 本体的定义和构建方法

本体是一种知识的表示方法，本体的定义在不同的也不尽相同。Gruber[2]在 1993 年对本体进行了定义：本体是概念的显示化表示。之后，Stuber[3]在这些基础上对本体进行了扩展。他认为本体是共享概念模型的、明确的、形式化规范说明。具体含义包括四个概念：模型、明确性、形式化、共享。本体可获得公众对所描述领域相关知识共同理解，从而确定该领域的知识及公认词汇。能够在不同层次上对这些词汇及关系进行明确的定义，并实现一定程度上的知识重用和共享。

本体的构建方法当前还没有统一的标准方法，主要是基于本体来对信息进行集成，是实现知识共享、信息交换以及解决语义冲突的基础，通过构建统一术语和概念，使得异构系统以及设备之间的互操作成可能。其中 Perez[4]等人提出了 5 个本体建模原语。原语的使用，使得本体能够顺利地表达知识。5 个分别为公理、概念、函数、关系和实例。下面对这些原语进行详细介绍[5]。

（1）公理：是永真命题，指的是定义在概念或关系上的规则。

（2）概念：是对某一类对象集合的描述，可以指代任何事务。概念是对客观世界个体的抽象，能够表示个体的集合。包括概念的名称及其描述。

（3）函数：是一种特殊的关系，在这个关系中，前 n-1 个元素可唯一确定第 n 个元素。

（4）关系：指的是概念间的相互关系，与对象元组的集合相对应。关系包括定义域和值域。在本体中，定义域是指一个概念，而值域既可以是概念也可以是具体的值 。本体包含以下基本关系：part-of、kind-of、instance-of 和 attribute-of。

（5）实例：是对概念的具体化，代表了满足这个概念属性的元素。

当前相关研究文献中的本体构建方法[6]共有 8 种，分别是 Uschold 和 Gruninger 提出的骨架法，Polytecnic 大学研究人员提出 METHONTOLOGY 方法，KBSI 公司提出的 IDEF5 方法，斯坦福大学提出的七步法，Doulas Lenat 提出的 CYC 方法，TOVE 方，KACTUS 方法，SENSUS 方法。其中斯坦福大学的七步法[7]应用最广,该方法将构建本体分为七个步骤，分别是确定本体领域范畴、考察已有的本体的复用性、列出本体中的重要术语、定义类之间的等级体系结构、定义类的属性信息、定义属性的方面、创建实例。本文主要采用此方法进行本体建模。

2.1.2 本体的描述语言

由于本体是知识表示的一种方式，所以需要用明确并且无歧义的语言来描述。因此自然语言无法胜任此项任务。在本体的发展过程中，提出的描述语言多种多样。在物联网和语义 Web 领域应用的最多的是是 OWL[8]语言，而 OWL 是建立在 RDF/RDFS 基础之上的，这里先介绍一下 RDF/RDFS 。

如图 2-1 所示：

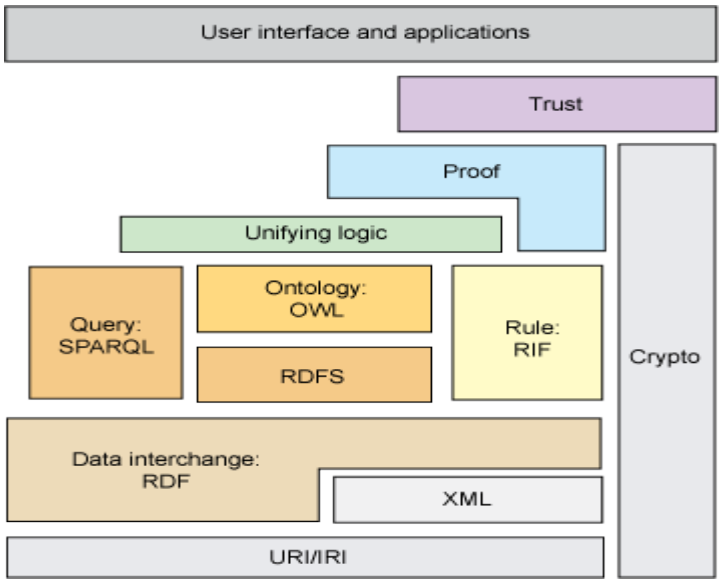

图 2-1 本体描述语言结构图

1) RDF/RDFS，RDF 是资源描述框架[9]，是用于表示互联网资信息的框架。主要用来描述资源与资源之间关系的语言规范。RDF 是 W3C 在 XML 的基础上制定一套标准，其含义包括以下几个概念：资源、描述、框架。RDF 本质上是定义了资源-属性-属性值，用来描述 Web 上的资源。 同时在实现方面几乎所有的语义库都支持多种方式的本体表示形式。

2) OWL：Web Ontology Language 的简称，是当前主流的本体描述语言，也是 W3C

提出的语义网本体描述语言的标准。与 RDF 相比，提供了更多的形式语义的词汇，表达能力强于 RDF。OWL 在表达能力上可依次分 OWL Lite、OWL DL 和 OWL Full[10-12]三种子语言，其中 OWL Lite 的表达能力最弱，词汇也最少，所以在应用方面比较少。OWL Full 则有最少的限制，因此有着最强的表达能力，但是没有可计算性保证。到目前为止，仍没有支持 OWL Full 的推理机。相比较而言，OWL DL 在词汇上有一些额外限制，例如类、个体、数据属性、对象属性等在 OWL DL 中是两两不同的，而在 OWL Full 中这些都可看作类来表示。因此，在 OWL DL 中对称、传递和反转关系是不能用于数据属性的，同样基数限制也不能应用于可传递的属性。这都是为了保证 OWL DL 是可被推理计算的。

2.1.3 本体的构建工具

本体构建[13]工具主要是辅助本体编辑、构建、开发和维护的软件。主要有基于特定语言如 Ontolingua，OntoSaurus，WebOnto 和 WebODE 等软件[14]，独立于特定语言的工具 Potege、OntoEdit、OilEd，它们支持导入导出多种本体描述语言，比如 OWL、XML、RDF 等。斯坦福大学医学中心开发的 Protégé 是目前应最广的本体构建工具。本论文也采用 Protégé 作为本体构建的工具。Protégé[15]是一个跨平台、可扩展、独立于领域的本体编辑软件，本论文使用最新的 5.1.0 版本。Protégé 开发界面非常友好，允许用户在概念层次上设计实领域模型。文件的导入导出也支持几乎 XML、OWL、RDF/RDFS 等所有格式。

2.2 逻辑推理

2.2.1 描述逻辑

描述逻辑[16](DescriptionLogic)是基于对象的知识表示的形式化，它吸取了 KL-ONE 的主要思想，是一阶谓词逻辑的一个可判定子集。除了知识表示以外，描述逻辑还用在其它许多领域，它被认为是以对象为中心的表示语言的最为重要的归一形式。描述逻辑的重要特征是很强的表达能力和可判定性，它能保证推理算法总能停止，并返回正确的结果。在众多知识表示的形式化方法中，描述逻辑在十多年来受到人们的特别关注，主要原因在于：它们有清晰的模型-理论机制；很适合于通过概念分类学来表示应用领域；并提供了很多有用的推理[17-18]服务。基于这些特点，所以很多本体描述语言都是以描述逻辑为基础的。基于描述逻辑的本体，可以用如下公式进行表示：O=(T,A)。其中 T 表示概念和关系集合，A 表示断言集合。描述逻辑的语义是通过模型论来定义的，模型论形象的说明了语法和领域模型的关系。

描述逻辑作为一阶逻辑的可判定子集，与一阶逻辑的对应关系如表所示：

表 2-1 DL 与 FOL 关系表

描述逻辑	一阶逻辑
概念	一元谓词
角色	二元谓词
实例	常量
构子	逻辑连接词、函数

根据表 2-1 的对应关系以及一阶逻辑公式的定义，Borgida 提出了描述逻辑到一阶逻辑的翻译σ(a)，为了简便，在一阶逻辑下我们依旧写成 a。又如概念 C，到一阶逻辑下我们应该表示σ(C(x))，省略翻译符号σ，这样表示对推理并无影响。

表 2-2 DL 与 FOL 关系表

概念/角色/构子	描述逻辑语言	一阶逻辑语言
原子概念	C	$C(x)$
角色	R	$R(x,y)$
个体	o	a
合取概念	$C \cap D$	$C(x) \wedge D(x)$
析取概念	$C \cup D$	$C(x) \vee D(x)$
否定概念	$\neg C$	$\neg C(x)$
存在概念	$\exists R.C$	$\exists y(R(x,y) \wedge C(y))$
任意概念	$\forall R.C$	$\forall y(R(x,y) \rightarrow C(y))$
定性数量值概念	$\geq nR.C$ $\leq nR.C$	$\exists y1 \cdots \exists yn(yi \neq yj \wedge R(x,y) \wedge C(y))$ $\forall y1 \cdots \forall yn((yi \neq yj \rightarrow \neg R(x,y)) \wedge C(y))$
概念包括、等价	$C \subseteq D$ $C = D$	$\forall x(C(x) \rightarrow D(x))$ $\forall x(C(x) \leftrightarrow D(x))$
角色包括、等价	$R \subseteq S$ $R = S$	$\forall x \forall y(R(x,y) \rightarrow S(x,y))$ $\forall x \forall y(R(x,y) \leftrightarrow S(x,y))$
概念断言	C(a)	C(a)
角色断言	R(a,b)	R(a,b)

本论文利用上表的转换规则来将描述逻辑翻译成一阶逻辑子句。

2.2.2 本体推理器 Hermit

Hermit 本体推理器是一个开源的 Java 本体推理库，给定一个 OWL 本体文件，通过 Hermit 可以快速得得本体文件是否一致，除些之外，还有根据已有的类关系推出间

接的类关系等功能。本体推理主要基于描述逻辑，描述逻辑是一阶逻辑的可判定子集。Hermit 实现基于著名 Hypertableu 算法，提供了比以往任务算法更快的推理功能。把本体的分类时间从小时级提高到秒级。同时也是第一个能轻松分类以往一些被证明是对任何推理系统都非常复杂的本体的推理器。Hermit 采用 OWL API 编写，也是 Web 语义本体编辑软件 Protégé 的默认推理器。Hermit 提供了多种对本体的加载接口，包括文件和输入流，其过和是先通过 OWL API 的接口加载本体文件为一个本体表示对象，然后基于本体表示对象生成一个推理器，在推理器的初始化中，会对本体表示对象的属性，关系进行翻译成 DLClause 对象，每一个 DLClause 都对应一个关系或者属性，表示的是基于描述逻辑的属性限制。本论文通过分析 Hermit 的开源源码，得到 Hermit 在处理 OWL 文件时的中间结果 DLClauses，这样避免了手工加载解析和翻译 OWL 的任务。方便处理成一阶逻辑的形式。

2.2.3 理论求解器 Z3

OWL API 是 Z3 是微软公司开源的一阶逻辑理库，是目前最好的 SMT 求解器之一，它支持多种理论，主要的用途是软件验证和软件分析。Z3 的原型工具参加了 2007 年的 SMT 竞赛，获得了 4 个理论的冠军和 7 个理论的亚军，之后在陆续参加的 SMT 竞赛中获得大多数理论的冠军。目前 Z3 已经被用于很多项目，比如 Pex、HAVOC、Vigilante、Yogi 和 SLAM/SDV 等。

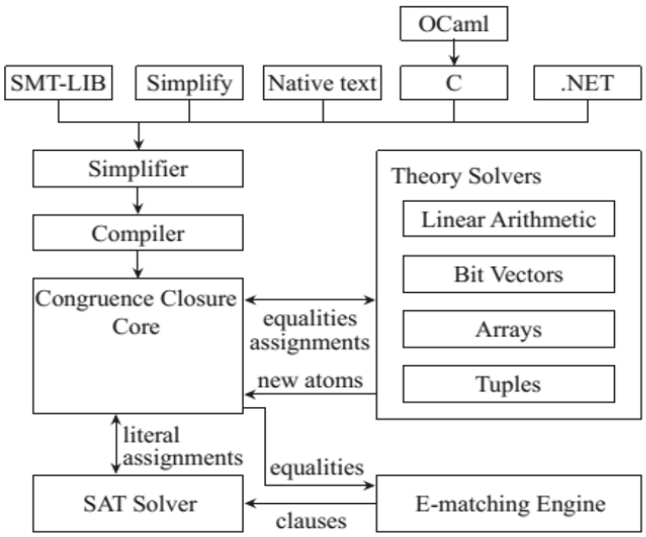

图 2-2 Z3 结构图

Z3 的 Simplifier 采用了一个不完全，但是高效的简化策略。它的 Compiler 是将输入转换成内部的数据结构和 congruence-closure 节点。CongruenceClosure Core 接受来自 SAT solver 的赋值，然后处理 EUF 和相关组合理论，它采用的方式称作 E-matching。

SAT Solver 是对公式的命题框架进行求解，并将结果交给 Congruence Closure Core 处理。Theory Solver 主要包含 4 种：Linear arithmetic、BV、Arrays 和 Tuples。理论求解器是建立在 congruence-closure 算法上的，这也是目前大多数 SMT 求解器使用的方式，也就是说， congruence-closure 可以看作核心求解器，各个理论求解器是外围求解器。

在 Z3 提供的 Java API 中，维护一个 Context 的概念。Context 是一个类似封闭空间的概念。可以通过 Context 定义 Solver 对象和创建 BoolExpr 对象，所有 BoolExpr 对象表示的是一个布尔概念，只有两种取值：真和假，同时 BoolExpr 可以由多个 Expr 对象通过与、或、非等方式组合起来表示复杂的表达式。Solver 对象是用来推理和证明满足性的主类，所有生成的 BoolExpr 都需要添加到 Solver 中，并且所有添加到 Solver 中的 BoolExpr 是以并的概念进行推理的，调用 Solver 的 check 接口可以得到当前整个 Solver 中所有的 BoolExpr 的并是否可以满足。

Solver 有一个栈的概念，可以维护 BoolExpr 加入的顺序。后面加入的 BoolExpr 在栈的上层。在任何时刻，Solver 可以调用 push()方法，在此后，如果 Solver 调用 pop 接口，则会将上一次调用 push()时到当前所有加入的 BoolExpr 移除。此方法可以让我们更加高效的利用 Solver 的推理功能，因为我们可能需要频繁的添加和移除 BooExpr。利用 Z3 的一阶逻辑推理功能，可以快速验证已有一阶逻辑的可满足性，从而推出条件是否可满足。进而推知目标条件是否成立，从而做出相应的反馈。

2.3 语义库 Jena

Apache Jena 是 Apache 的一个开源的用于开发语义 Web 和 Linked Data 的 Java 框架。此框架包括许多用于处理 RDF 数据的 API。Jena 提供丰富的本体处理 API，方便使用者加载，解析，生成本体文件，同时支持多种格式。OWL 是知识表示形式，但是文件载体可以是多种格式，在 Protégé 的 OWL 本体生成保存格式中就包括 RDF、OWL 等多种格式。Jena 全面支持各种类型的文件类型。

整个框架的的结构如图 2-3 下所示：

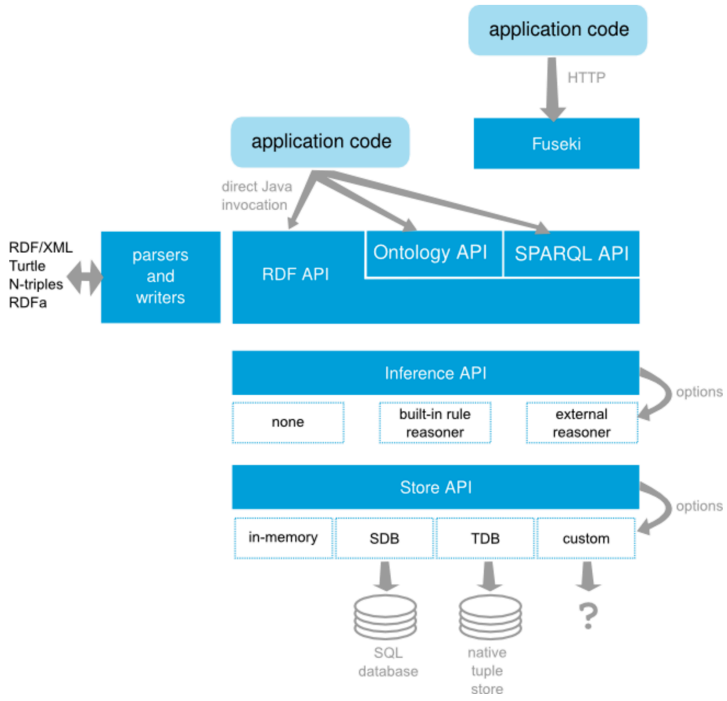

图 2-3 Apache Jena 语义框架

2.3.1 文件数据库 TDB

TDB 是 Jena 的一个组件，是一个基于文件系统的存储库。支持各种形式的 Jena API 查询，提供非常高的性能。在本项目中，我们采用 TDB 作为持久化层，已有的本体文件通过 Jean 提供的 API 存入 TDB 文件中。TDB 同时结合结合了 Jena 的 ARQ SPARQL 引擎，可直接支持 SPARQL 查询。在 TDB 中，数据是以文件的形式持久化的。数据以数据集的形式组织，每个数据集对应一个文件目录，这样我们可以以数据集的形式来实现知识库。

2.3.2 SPARQL 查询语言

SPARQL[19-20]是 W3C 制定的一个语义网标准协议，主要用于 RDF 数据的查询。它是在以前的比如 rdfDB，RDQL 以及 SeRQL 的基础上开发的。W3C 的官方文档对 SPARQL 有非常详细的介绍。SPARQL 包括 IRI、literal、lexical form、plain literal、language tag、typed literal、datatype IRI、blank node 等概念。SPARQL 将 RDF 看作一个数据图的形式。数据之间关系用 Triple 三元组的形式表示。在 RDF 模型中，基本对象是资源、属性和陈述。一个特定的资源加上该资源—命名的属性及属性值构成一个资源陈述.有向标记图(directed label graph)是 RDF 的基本数据模型.其最基本的单位是 statement.statement 陈述是由主体(subject)，谓词(predicate)，客体(object)组的。SPARQL 查询语言是基于图模式匹配实现的的，最简单的图模式是三元组模式，三元组模式被写

作为一列主语、谓语，宾语。复杂图模式能被简单图模式组合而成， 常见的复杂图模式是基本图模式、组合图模式、可选择图模式、联合图模式、RDF 数据集图模式、值约束条件六种模式中的一种.

本论文采用 SPARQL 作为知识的查询语言，利用 Jena 的 SPARQL 引擎实现对知识库的查询。

2.4 发布订阅系统

发布订阅是一种信息发布模式，其中有两种角色，发布方和订阅方。消息的发送者(publisher)不需要知道消息的接收者，只要在发布消息之时指明消息的标记类别，并将消息发布到统一消息空间。相应的，消息的接收都只需指明消息的类别，当此类别的消息被发布时，统一消息空间会将其发送给所有的此类消息的订阅者。发布订阅技术是一种一对多的分布式消息中间件。其基于主题的形式有效的将发布和订阅两方解耦。

发布订阅系统的核心技术是事件分发机制，具有高吞吐率、同时支持离线数据和实时处理等特点。消息是通信的基本单位。由消息生产者（producer）发布关于某话题（topic）的消息，此消息以一种物理方式被发送给了作为代理（broker）的服务器（可能是另外一台机器）。若干的消息使用者（consumer）订阅（subscribe）某个话题，然后生产者所发布的每条消息都会被发送给所有的使用者。发布订阅是一个显式的分布式系统——生产者、使用者和代理都可以运行在作为一个逻辑单位的、进行相互协作的集群中不同的机器上。

2.5 本章小结

本章主要对课题中涉及到的相关技术进行了介绍，为后续设计和实现部分打下基础。首先介绍了语义 Web 相关知识，包括本体的定义，本体的描述语言和本体的构建工具。其次介绍了本体的基础描述逻辑以及描述逻辑和一阶逻辑的关系和转换规则、基于 Hypertableu 算法的 Hermit 本体推理器对本体的加载和分类机制、一阶逻辑推理库也即微软公司的理论求解器 Z3 的相关信息及其主要功能和原理。随后讨论了语义 Web 框架 Jena，讨论了 Jena 的文件数据库 TDB 和 RDF 查询语义 SPARQL。最后介绍了物联网系统的知识管理子系统中使用的发布订阅消息转发系统的机制。

第三章 系统需求分析

从第一章可以看出，在物联网平台中，各种不同的传感器实时的将探测的数据信息上传到系统之中，物联网系统需要接收大量的实时数据，并从这些数据中处理出有效的信息加以利用。结合数据的异构性等特点，为了屏蔽具体数据的差异，需要将事件相关的领域信息抽象出来，基于本体建模并保存到已有的知识库中。模型存储之后需要有效的管理起来，需要提供方便的查询功能以便新的事件到来时可以查询其相关领域信息。同时，对事件的推理处理也是本论文的主要功能模块，事件本身也携带有领域信息，与已有知识库中的领域知识存在着各种联系，利用本体模型基于描述逻辑同时描述逻辑也是一阶逻辑子集的事实，我们可以利用一阶逻辑推理器来实现基于模型的事件推理功能。

因此，对于在本论文研究背景中提到的基于物联网系统的知识管理子系统，本小节将从系统的功能性需求和非功能性需求共两个方面进行需求分析。

3.1 系统功能性需求分析

本课题的重点是以模型为基础，结合事件驱动模式和逻辑推理，设计和实现一个处理能力强大，低耦合、高复用的知识管理子系统。用户可以利用此系统进行知识库的基本管理，SPARQL 查询以及基于目标的推理等操作。首先，要在本体的基础上对领域知识进行建模，实现基于本体的模型表示。其次，将模型按照知识库的形式组织起来，需要根据模型的领域知识进行分类，不同的领域知识存储在不同的知识库中。此外，本系统通过发布订阅系统来订阅获取消息事件，需要系统与发布订阅互连。最后，我们需要实现基于目标的推理功能，实现对事件的实时处理和响应。

3.1.1 知识建模

知识库的基础是模型，因此模型的建立是本课题的基础需求。本体的构建主要是基于本体来对信息进行集成，是实现信息交换、共享及解决语义冲突的基础。通过构建统一的概念和术语，为同一领域的异构事件的互操作和集成提供了可能。领域模型的构建需要把领域相关知识转化为本体描述语言来表示。因此，知识建模包括两个子需求，本体的构建方法和本体的构建工具的选择。本体的构建方法有许多种，但目前还没有一个标准的方法。Gruber 在 1995 年提出的 5 条准则可以作为我们构造本体的基本思路。这 5 条准则分别致性、完全性、明确性和客观性、最大单调可扩展性和最小承诺。但这五条准则通常很难都满足，需要我们在实现构造本体的过程中权衡。当前领域相关文献中

提出的方法共有八种：骨架法、METHONTOLOGY 法、IDEF5 方法、七步法、CYC 方法、TOVE 方法、KACTUS 方法、SENSUS 方法。同时建模工具也多种多样，有斯坦福大学的 Protégé、OntoEdit、OilEd 等。模型的构建首先要对现实的需求说明，包括确定本体和范围，获取领域知识。然后要将现实知识抽象化，包括定义类及其关系、实例化。最后是利用工具将描述好的模型建立。用例图如下图 3-1 所示：

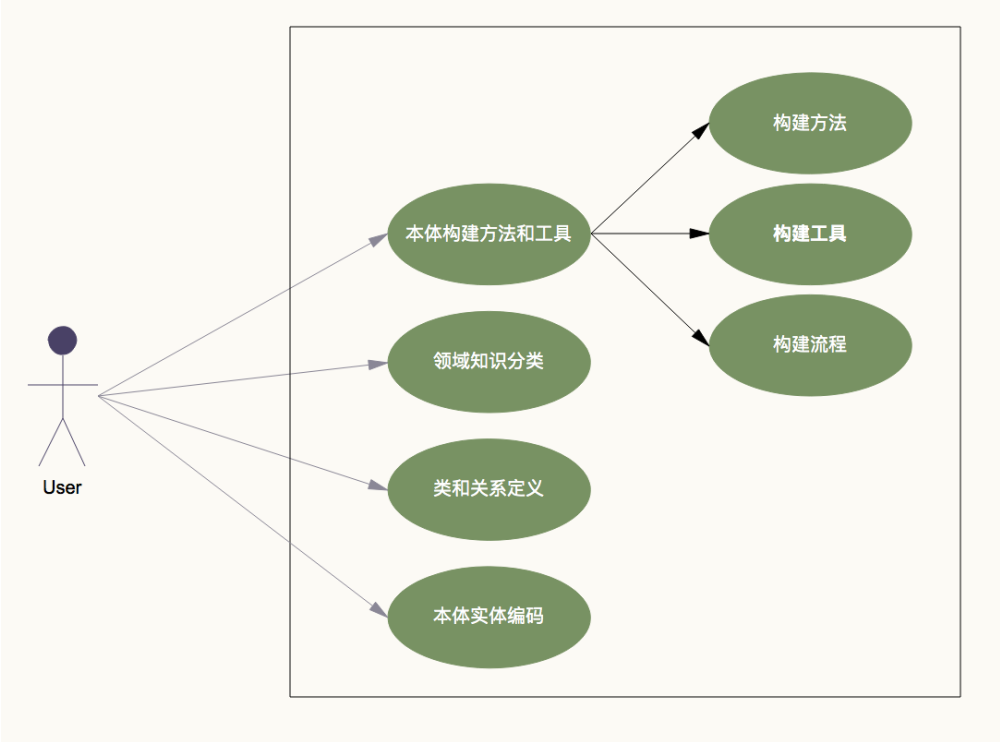

图 3-1 知识建模用例图

3.1.2 知识库的组织管理

在实现资源和事件的建模之后，我们需要依据领域信息将已有的模型进行分类，管理，便于查询和使用。因为事件的多样性、异构性等特点，因此，我们不能把所有的模型放在一个知识库中。需要根据特定的领域进行分类，同时要兼顾知识领域的大小，不能划分过大，这样会导致一个知识库里的知识过于庞大，从而导致管理和查询时的系统运行负载过大而使效率降低，因此在领域划分的同时要兼顾领域的大小，做一些权衡。每个知识库包括特定领域内的信息功能需求用例图如下图 3-2 所示：

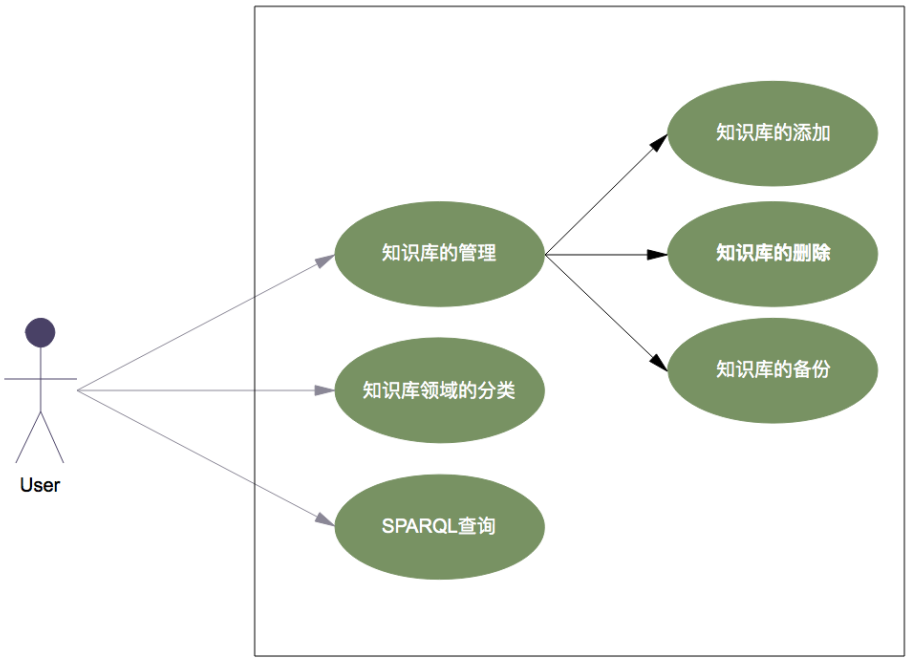

图 3-2 知识库管理用例图

如图所示，对包括知识库添加功能、知识库删除功能、知识库备份功能、SPARQL查询功能。具体功能描述如下

2）功能描述

A. 知识库添加功能：

该功能主要包含添加内存模式和添加持久化模式数据集两个子功能。内存模式：用户输入要新增的知识库名，选择内存模式进行新建。持久化模式：用户输入要新增的知识库名，选择持久化模式进行新建。

B. 知识库删除功能：

该功能主要涉及知识库的删除功能，在知识库管理界面，用户选择所要删除的的知识库并确认，从而将选中知识库删除。

C. 知识库的备份：

此功能主要涉及知识库的备份功能，在知识库管理界面，用户选择所要备份的的知识库并确认，从而将选中数据集备份。

D. SPARQL 查询功能：

该功能主要涉及对知识库的 SPARQL 查询功能，针对某一知识库，通过编写 SPARQL 查询语句，返回对应的知识信息，完成查询功能。

3.1.3 与发布订阅相连

物理平台最重要的特点就是事件，在前面我们提出了基于模型事件推理机制，使得推理功能具有事件驱动的特性。但是推理本身是不具有我们所要订阅的一些事件和主题的。因此这就需要一个可以提供获取主题和事件的模块。已有的基于主题实现的发布订阅系统内部就已经定义了许多的主题。这些主题在发布系统之中都是以主题树的形式存储的。主题与主题之间是以父子关系的的，并且如果我们向发布订阅系统订阅了一个父主题时就会默认的不处理此父主题对应的所有子主题。发布订阅系统能够基于事件收发消息，有非常好的耦合特性，是大规模分布式系统建立的有效方式，所以我们需要将基于模型的事件推理模块与发布订阅相连。

在实现推理功能模块与发布订阅系统相连的时候，我们使用开发的事件接入模块来实现，也即先使推理功能模块与事件接入模块相连，然后事件接入模块将订阅消息、发布和取消订阅消息中转，通过消息发送给发布订阅系统；虽然目前推理功能模块中 Solver 的数量比较少，理论上他们直接与发布订阅系统相连实现，实现订阅、收发消息是可以的，但是当 Solver 数量很多，多个服务同时向发布订阅系统订阅或者发消息的时候就可能因为吞吐量不够，导致系统性能降低，所以事件接入模块来中转是有必要的，除此之外事件接入模块还可以管理订阅者的地址，端口号，这主要是为客户端查询订阅者地址提供方便，还可以实现对订阅主题进行管理，它可以从 WSN 发布订阅系统那边获取主题树，如果事件接入模块服务所订阅的主题在发布订阅中不存在，那么事件接入模块会创建相应的主题，并且发布订阅系统那边也会添加相应的主题满足订阅者的需求。图 3-3 为系统对推理功能模块与发布订阅系统相连的用例图。

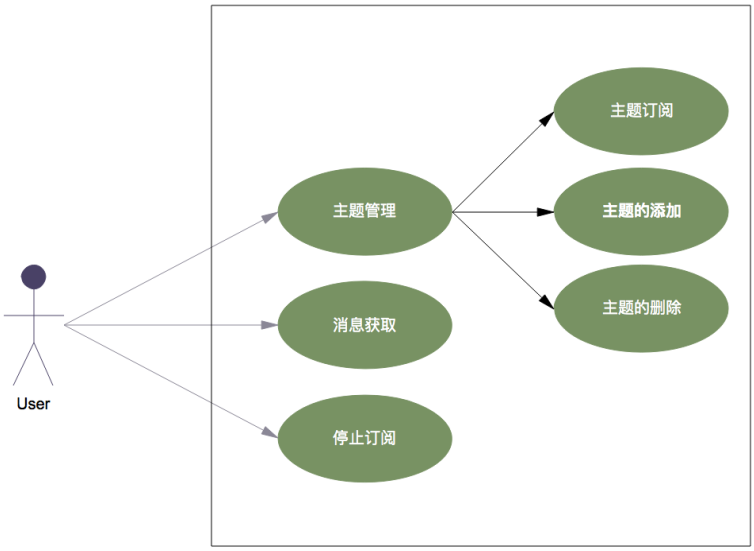

图 3-3 数据接入用例图

3.1.4 事件转化和处理

在实现了事件接入的功能之后，我们需要考虑事件的转化处理。结合前面的分析，对于每一个新来的事件，推理功能模块只知道事件的领域信息(在本课题中，领域信息即事件的主题)，为了拿到与事件相关的领域信息，我们需要分析事件自身携带的信息，根据此信息生成查询知识库的查询语句。为了统一利用知识库的查询接口，需要结合面向对象的多态概念。因此我们需要把事件生成包含领域信息的 Java 对象，结合本课题使用的深圳出租车 GPS 数据，对每一条有效数据可以生成一个包含各个字段的数据对象，然后交由推理模块进行处理。如何根据事件的领域信息将事件转化成 Java 是事件处理的前提，图 3-4 是事件转化用例图。

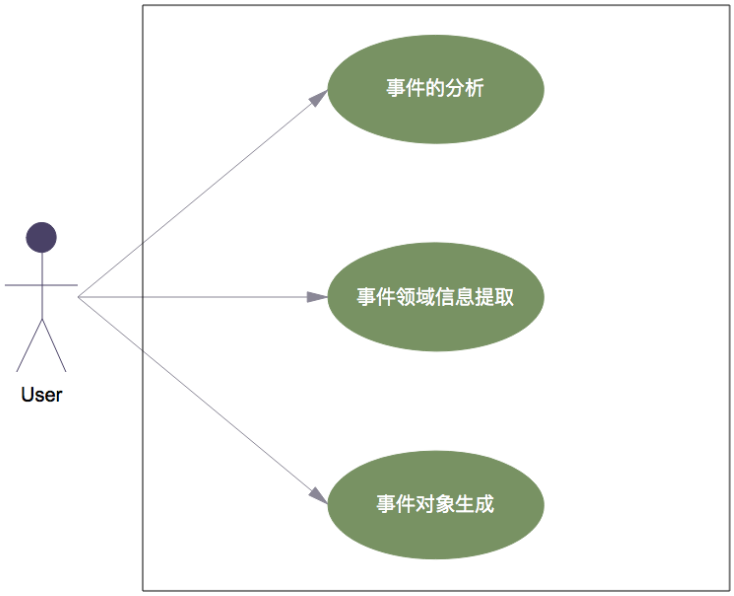

图 3-4 事件的转化用例图

3.1.5 逻辑推理

采用本课题使用的深圳出租车 GPS 数据，在出租车数据到来的时候，我们利用数据来实时推理出道路的拥堵情况。因此，出租车的速度和出租车的地理位置是我们最为关注的两个事件信息。当出租车的 GPS 数据到来时，我们首先要判断出租车具体在哪一条道路上，其次，我们需要根据出租车的速度来判断道路的拥堵情况，具体的拥堵程度需要由推理目标来定义。比如，可以认为 80%以上的车在这某一段时间内的速度低于 2 米每秒就可以认为此道路为拥堵状态。因此，在我们推理应用之中，首先要对道路和出租车进行建模。模型的建立采用建模模块的方法与工具完成。然后结合 Z3 的 Solver 的特性，一旦有新的数据到来就应该更新推理器 Solver 的内容，也就是添加 BoolExpr

对象到 Solver 中。但是在实际应用场景中，数据量会非常大，如果每一个数据都新建一个对象就会造成服务器非常大的负荷。因此，当数据量增大的时，我们要考虑先过滤无用数据。比如，对于某条道路而言，其他道路的数据没有什么影响因素，因此，我们可以基于模型把非相关的数据给过滤掉。结合描述逻辑，我们可以利用包含的概念来实现这一功能。位置和道路是一个包含概念，我们可以定义道路为一个矩形的概念，而出租车的位置为一个点的概念。给定个一条道路，我们的一个推理功能需求就是迅速的判断出租车的位置是否在此道路的范围内。

推理的目标功能需求应该结合模型来定义，模型是推理的基础，主要子功能包括包括推理目标定义、推理目标解析、结果处理。一阶逻辑推理器的推理的用例图如下所示：

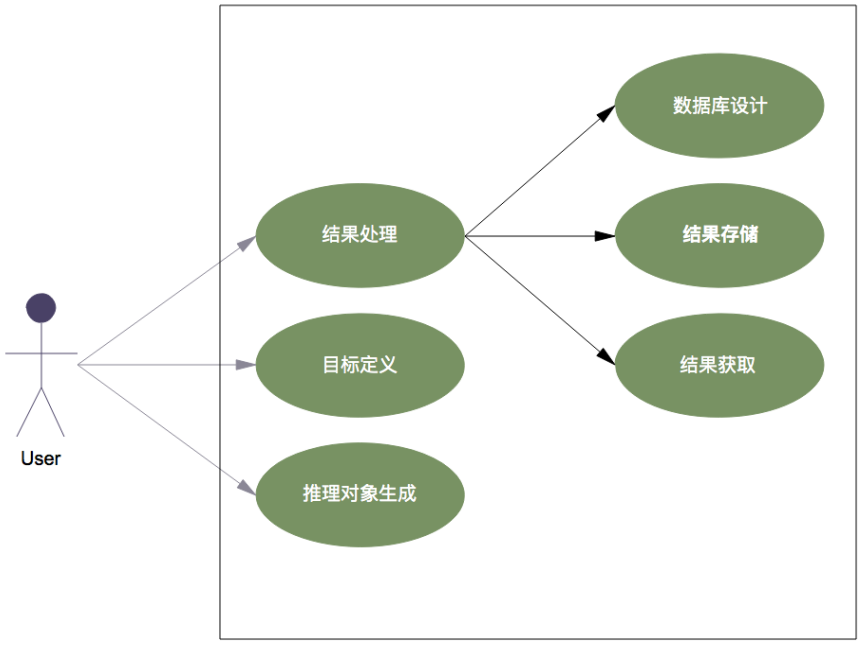

图 3-5 逻辑推理用例图

目标定义功能主要涉及推理目标定义，用户在前端定义目标的形式和参数，并选择相关的领域，选择推理后，系统会实时接收相关领域事件，推理结果会实时写入数据库中。获取推理结果主要用于推理结果的展示，在 Web 前端推理界面，在用户提交了推理请求之后，选择推理结果展示，通过后端实时的获取结果信息并以 Json 形式返回并展示给用户。

3.2 系统非功能性需求分析

1. 响应速度：系统要求能够在用户可以接受的时间内对用户提出的网络请求进行相应。

2. 可使用性：系统要求容易使用，界面友好。能够尽可能的符合用户的操作习惯。
3. 并发性：系统能够同时支持多个用户进行访问。

3.3 本章小结

本章在前两章研究背景和相关技术介绍的前提下，从以上几个方面介绍了基于物联网平台的知识管理子系统的需求分析。主要介绍了本体建模、知识库的管理和查询、与发布订阅的互连、基于模型的阶逻辑推理以及用户界面等需求。从功能性和非功能性两个角度，分析了系统的主要需求，为后续的设计和实现提供了明确的目标。

第四章 系统概要设计

4.1 系统架构设计

在前面章节介绍了相关背景以及相关技术基础下，我们提出了可行性的需求分析。根据需求分析可知，我们主要在 Java 开发环境 Intellij Idea 平台上开发，开发基于 Jena 的知识管理模块、基于消息中间件的事件接入模块以及推理功能模块，图 4-1 是系统架构图。

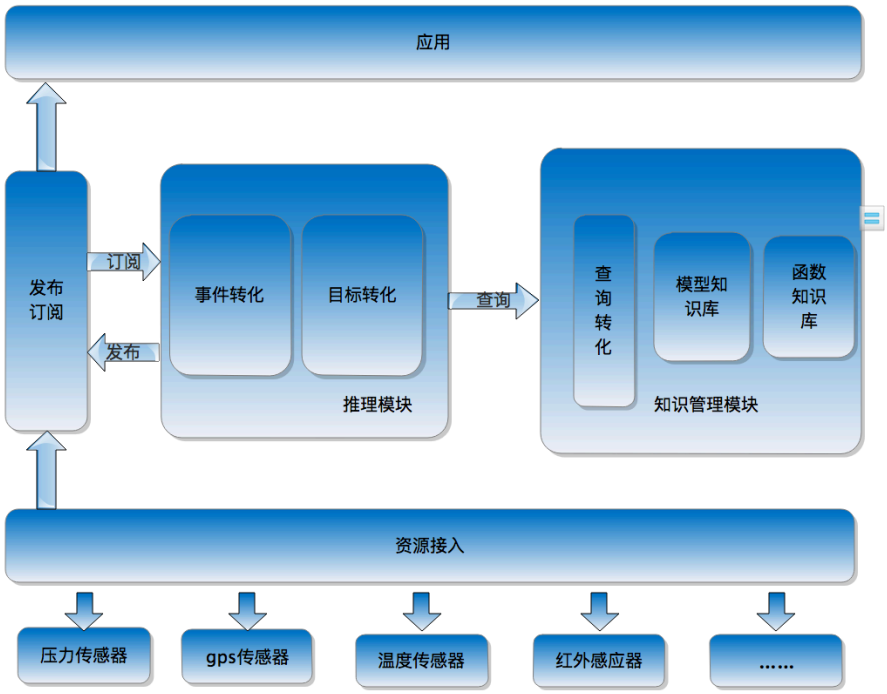

图 4-1 系统结构图

从功能角度分析，此系统各模块需要实现以下功能：1）知识管理模块，知识管理模块包括知识建模，知识库的管理两个子模块。首先我们需要利用 Protégé 等本体建模工具，分领域的建立对应的本体模型，要定义领域内的类和类之间的关系，包括类的数据属性、对象属性、属性的条件约束。然后，将已有的模型采用知识库的组织形式管理起来，实现知识库的基于 SPARQL 查询功能。2）事件接入模块，需要要实现数据的接入功能，在已有的发布订阅消息中间件的基础上实现事件的主题订阅、主题取消、事件转发、事件对象生成等功能。为了获取事件，我们需要和发布订阅系统连接，从 WSN 获取主题树信息，判断订阅者订阅主题是否存在，如果不存在可以及时建立以代订阅。获取主题之后根据订阅的主题列表获取事件。然后，根据事件的 schema 将事件转化生

成事件对象。3）推理模块，通过事件对象的领域信息生成知识库查询对象，利用查询对象获取知识库中相关的领域模型信息，将此信息交由一阶逻辑转化模块进行转化成为一阶逻辑表达式。同时，推理目标定义同样需要转化为一阶逻辑表达式才能进行推理。首先，目标的定义需要根据已有的知识进行，因此需要提供一个目标定义层将知识库中的知识获取并映射成为用户接口，并利用此接口将信息反应在用户界面，包括属性的选择，方法的选择等。然后用户可以在推理目标定义界面进行目标定义。定义完成之后的目标提交到后端由一阶逻辑转化模块将其处理成一阶逻辑表达式对象。并将一阶逻辑表达式交给推理引擎与推理目标一起进行推理，并根据推理结果进行反馈，从而实现事件的实时处理。

下面就整体流程中涉及的各个阶段分别进行各功能模块的结构设计。

4.2 知识建模模块设计

物联网平台需要实现实时的接收物联网事件信息，并对这些事件信息在已有的知识库中进行查询得到的的领域信息，然后通过基于推理的方式结合目标分析出问题事件，做出反应并提供平台展示。结合不同的任务之间的独立性将整个系统划分成不同的模块，不同的模块之间的数据传输的实时性需要实时的信息通信方法，本系统采用分布式发布订阅系统，

根据系统的需求，针对上节所述的系统提供的主要功能。首先需要对物理传感设备及其信息和已有的物理环境进行建模，以本体的形式表示并存储。然后，基于 Jena 开发一个知识库管理模块，将建立好的所有本体模型分领域存储到知识库中，在开发知识库管理模块时提供基于 SPARQL 语义查询接口，方便后续使用。其次，事件以发布订阅的形式到来，数据的接收和转化划分为专门的事件接入模块。事件的响应和处理展示则由推理模块负责，包括事件领域信息的查询，一阶逻辑转化，推理及结果的存储等部分。流程图如下所示：

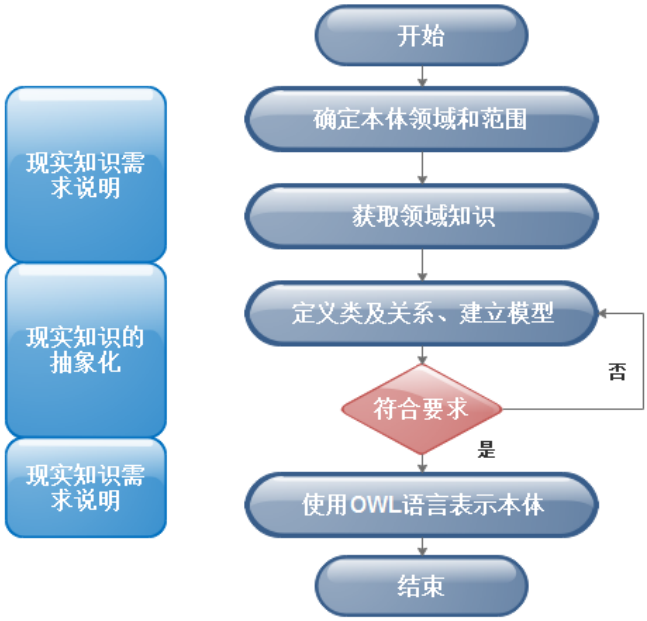

图 4-2 建模流程图

4.3 知识管理模块结构设计

知识管理模块主要涉及本体的建模和知识库的管理以及查询。本课题将知识库分为两类，一类是通过建模过程对物联网建模的本体，另一类是推理目标所需要的领域知识，包括一阶逻辑函数算子，事件关系属性信息等。本体的管理和查询可以直接基于 Jena 的 TDB 文件数据库对进行组织和管理，不同领域的信息存储在不同的知识库中，可方便的提供 知识库的新建，备份，删除以及本体信息存入等功能。而除了本体之外的知识则通过编码基于配置文件进行管理。同时结合 Jena 的 SPARQL 查询引擎 ARQ，在本体查询功能支持方面知识管理模块可非常方便的提供基于 SPARQL 查询的 Java 接口，方便其他模块的查询操作。在非基于本体的知识库管理方面，主要是对定义好的函数、算子的管理和使用，通过提供基于领域的不同工厂方法进行实现，同时可以结合数据库和文件映射的方式进行持久化保存。为此首先要对物理环境和事件进行资源建模，并将事件和物理环境的相关关系建立联系，映射为实际编码，在本课题中主要提供一个查询的抽象接口实现对。这可以利用 Java 的面向对象多态并结合设计模式来实现，提供高层的 Java 接口/抽象类，具体实现根据实际的事件实现。流程结构设计图如下图所示：

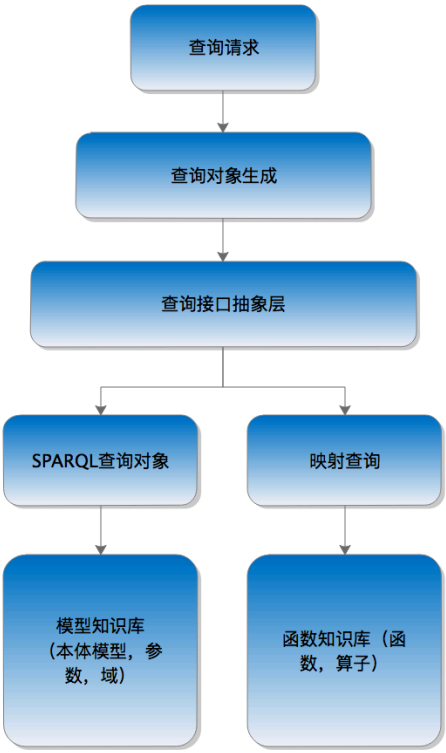

图 4-3 知识库管理模块设计图

如图所示，因为涉及两类知识类，基于模型的知识库和函数知识库。我们通过提供一个接口层来屏蔽具体的实现。具体的实现主要有两种，对于基于模型的知识库的查询，我们采用基于文件的 TDB 进行存储，利用 Jeana 的 SPARQL 查询引擎进行查询，对于一个查询，首先要根据查询对象生成具体的 SPARQL 查询对象，将 SPARQL 查询对象交由 TDB 进行处理，获取 ResultSet，然后遍历解析具体的 ResultSet 生成返回结果。对于函数知识库的查询，可以采用映射的形式来实现，每一类主题对应的函数库由主题到函数库的映射关系决定，映射关系存储在后台内存之路，可以实时的获取。并且可持久化在文件或者基于关系的非关系型数据库之中。本课题当前的实现是将映射关系维持在文件之路，系统启动时将其加载到内存中的一个 Map。这样可以迅速的获得一个主题对应的函数库信息。

4.4 事件接入模块结构设计

事件接入模块主要是解决系统中事件的转发问题。上图 3-2 系统结构图所示，发布订阅主要分为数据的发布和数据的订阅模块。数据的发布模块是数据的来源，所有的物联网事件都通过发布模块发布到消息中间件中。数据的订阅模块则负责数据的订阅和过滤功能。基于配置的订阅模块可以决定订阅哪些主题的信息，当消息中间件中有这些主

题的事件出现则交给推理模块进行处理。如果是向发布订阅系统订阅，就会解析出订阅主题并向发布订阅系统请求订阅，若是要退出订阅，就会向发布订阅系统请求退出订阅。发布订阅系统受到相应的请求后会做出响应并返回处理消息。事件接入模块存储了订阅用户的地址，订阅主题信息供客户端用户查询，同时判断订阅者订阅的主题是否存在，如果不存在会创建订阅主题。事件接入模块会向发布订阅系统获取主题树，事件主题是以主题树的形式存储的，在进行匹配时都是从根节点逐渐到子节点开始匹配，直到找到对应的节点，用户订阅主题、取消订阅和发布事件所用的主题都是从主题树获取，如果所订阅主题不在主题树中，可以请求建立主题，如果没有建立相关主题，就会订阅失败。

结构图如下所示。

图 4-4 事件接入模块结构图

4.5 推理模块结构设计

结合功能需求分析，我们要利用一阶逻辑推理器证明目标是否成立。首先，用户的目标推理请求需要通过处理成一阶逻辑表达式。同时，结合目标中携带的领域相关信息，通过事件接入模块订阅相关主题信息接收数据，也即事件。然后，对于每一到来的事件，首先我们要根据事件转化成为事件对象，这可以通过主题映射到事件结构的方式实现，每一个主题对应一个 schema，根据 schema 确定事件的携带信息结构，然后解析生成具体的事件对象。然后，根据事件对象生成知识库查询语句查询知识库，得到相关事件的相关领域信息，并将事件和相关领域信息进行处理得到一阶逻辑表达式对象。推理需要目标，用户定义的目标同样需要处理成一阶逻辑表达式，因为用户定义目标的时候，提供的界面参数和选择参数的域都对应后台的函数知识库，所以目标定义提交之后，由后台负责处理，解析获得相应的函数库，再进一步得到一阶逻辑表达式对象。从而与事件处理而来的一阶逻辑表达式一起进行推理。并根据推理结果进行响应和结果数据存储。

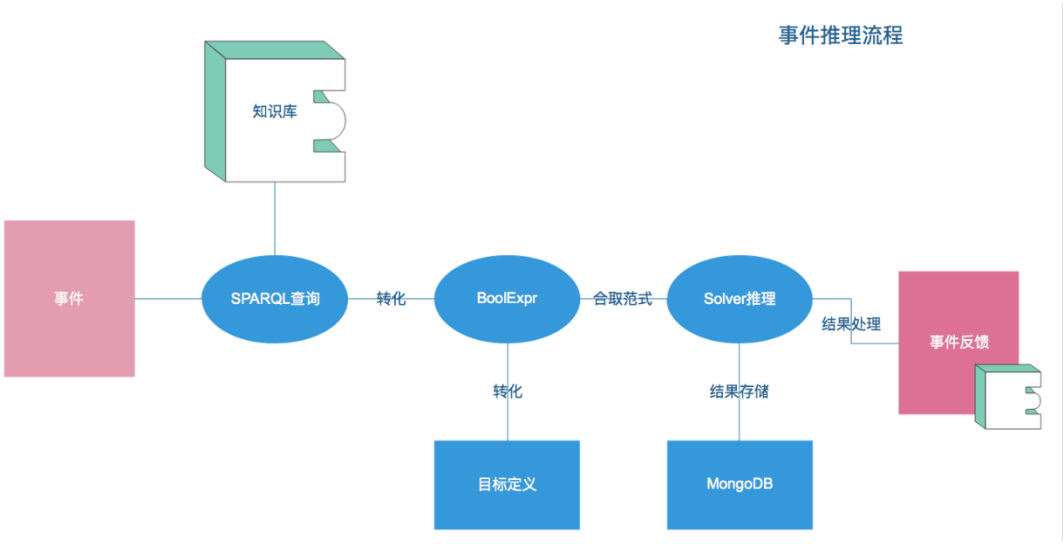

图 4-5 推理流程图

在本论文使用的理论求解器 Z3 中，一阶逻辑的推理证明主要基于以下规则来使用，基于已有的事实 P，若要推理出 Q 成立，则是要证明 Q 的取反不可满足。因为 P→Q == ¬P∨Q，其中 Q 是我们的目标，P 是已有的事实，即事件处理之后的数据。P→Q 的取反即是 P∧¬Q。因此，我们的目标是证明上式不可满足。因为上式为假，则可得 P→Q 为真，从而可证明 Q 成立。利用 Z3，我们维护一个 Solver，这一个主要的推理类，我们通过将用户的目标处理成 Q，已有的事件知识定义为 P，然后将 P 以及 Q 的逻辑反表达式加入到同一个 solver 中，调用 solver 的 check() 函数，如果结果是 Status.UNSATISFIABLE 则证明成立，反之则无法证明。

采用本课题使用的深圳出租车 GPS 数据，在出租车数据到来的时候，我们利用数据来实时推理出道路的拥堵情况。对于新来的事件，首先需要查询知识库取出相关的模型信息，利用 SPARQL 可以迅速有效的查询出来。因为查询出来的信息是 OWL 格式，因此首先需要将其解析转化为描述逻辑子句，在我们的实现中，描述逻辑子句是通过本体推理器的 API 来转化的，描述逻辑子句用 DLClause 对象来表示。然后再将 DLClause 对象转化为一阶逻辑表达式 BoolExpr 对象。同时，对于每一个目标，我们都维护一个 Solver 对象，转化成的 BoolExpr 对象都添加到对应的 Solver 里进行推理。

如下图所示：

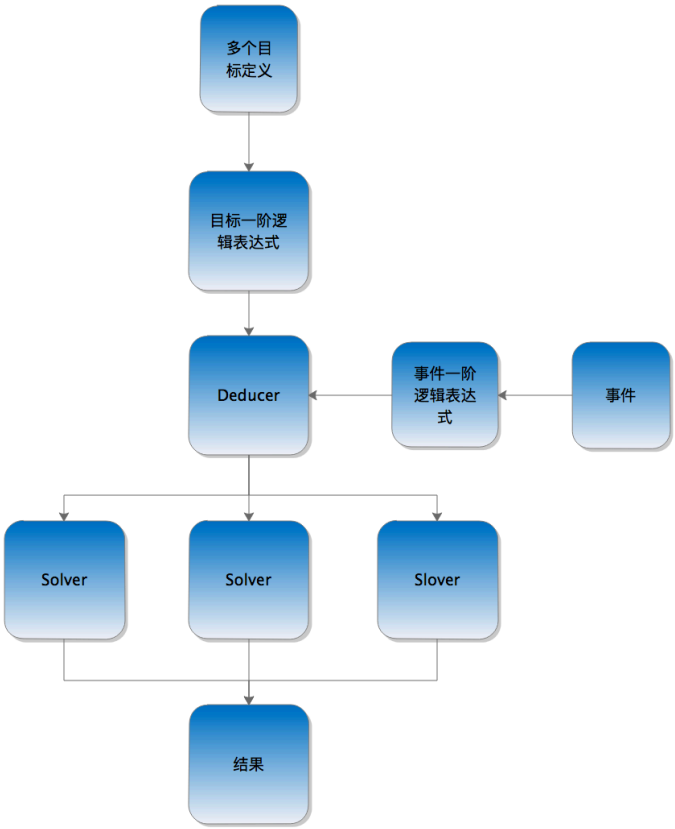

图 4-6 多个目标推理过程图

对于多个目标定义，经过处理会生成多个 Solver，每个 Solver 负责处理一个目标，事件处理之后生成的一阶逻辑表达式会依次添加到相关的 Solver 中，然后调用对应的 Solver 的 check()接口获取推理结果值，并将结果存储在数据库中持久化。

4.6 用户界面模块结构设计

用户界面模块主要提供用户一个直观的 Web 用户界面，定义推理目标、推理结果展示和简单的知识库查询管理。因此包括前端和和应用后端两部分。结合需求分析，前后端的的交互主要采用 Json 数据格式。同时由于本应用属于轻应用，所以后台实现选择 Jetty。用户首先定义推理目标，推理目标界面主要提供用户参数的可选输入项，这可通过后台渲染实现。用户需要选择领域信息，结合本实现采用的样例数据可以选择比如 traffic 领域。后台根据此信息返回相应领域查询函数知识库得到可选参数，函数名等信息，然后更新界面供用户选择目标定义参数。

定义好推理目标之后，用户提交推理请求，推理目标携带用户 id 提交到后台进行推理，处理结果存储在 MongoDB 数据库中，同时附上用户此次推理的唯一标志 id。处

理结果的呈现：对于处理结果的呈现，用户界面采用轮询的方式向平台后端发送请求，请求包括推理时提交的唯一 id，系统后台每一次接到数据请求时就访问利用此 id 从 MongoDB 数据库获取结果并返回给用户界面。用户界面将返回来的处理结果利用开源的图形库渲染在在界面中，过一段时间再次对后台提交请求结果。

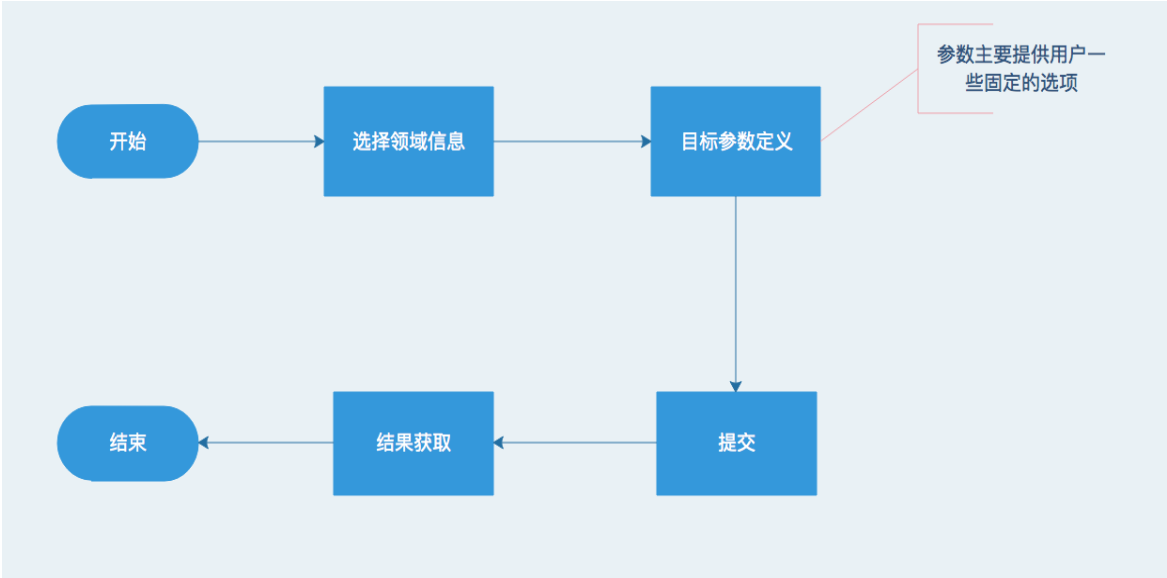

图 4-7 用户界面模块流程设计图

4.7 本章小结

本章主要内容是本论文提出的知识管理子系统的方案概要设计，开篇介绍了系统总体架构，接着是对各模块的整体设计，并介绍了各模块的功能和内部关系等。通过本章，我们可以详细的了解本系统的总体概要设计，形成一个清晰的概念。接下来的章节我们将对各模块的具体实现细节做详细讲解。

第五章 系统详细设计与实现

5.1 知识库管理模块的实现

知识库管理模块功能需求主要是主要包括本体的建模、数据集的生成、数据集删除、数据集的备份和 SPARQL 语言查询支持。

5.1.1 本体建模模块

结合的已有的物联网平台建模工具，通过将领域知识转化为 OWL 本体的过程就是建模流程。首先是要确定本体描述的领域和相关知识，列出领域相关的知识术语和概念。然后建立本体的架构。如果涉及到已有的本体，则选择重用操作，再定义领域中的概念关系。本课题通过选用深圳市出租车的实时 GPS 监控数据来进行对系统的验证。出租车 GPS 数据中包括多个字段，分别是车牌号，时间，经度，纬度，是否载客，速度，车的方向。其中车的方向包含东、东南、南、西南、西、西北、北、东北等 8 个方向，分别用 0—7 表示。

利用上图的建模流程，我们先定义好道路及相关的领域模型。每段道路可以看成是一个长方形的概念。长方形的概念可以用左上角和右上角的概念组合肤浅。因此，道路应包含两个子属性，左上角点，和右下角点。同样，点也是一种概念。点有两个属性，经度和纬度，在本体之中都用数据属性 double 来表示。本论文采用 Protégé 本体编辑软件作为主要建模工具，建模界面如下所示：

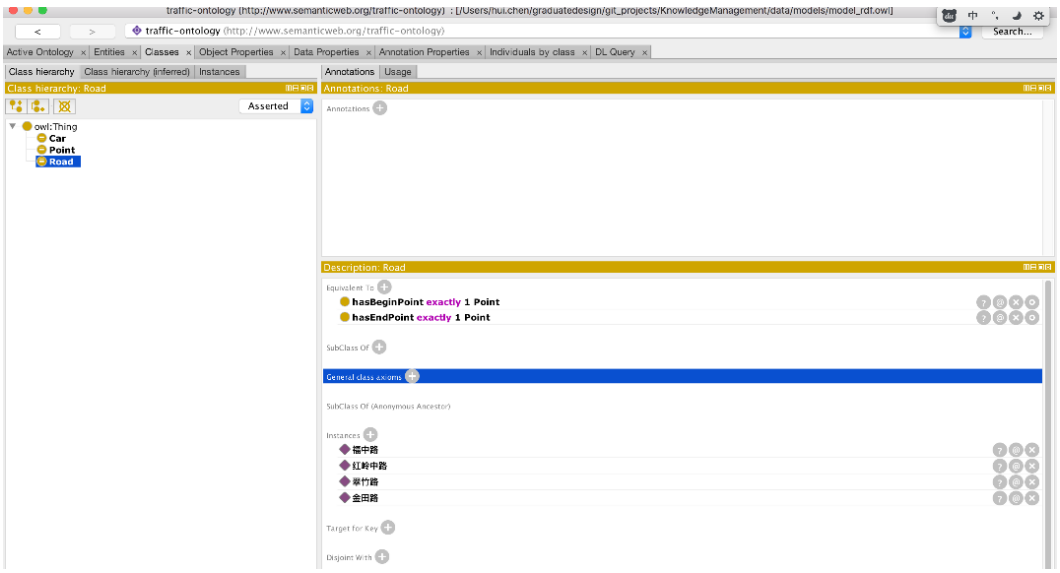

图 5-1 Protégé 建模图

在此本体中，共有路，点，车的概念，如前所述，一条路等价于两个点，有且仅有一个起点和一个终点。有且仅有在定义数据的时候用 exactly 约束条件来表示。点的概念是由两个浮点数据属性组成，同样用 exactly 约束条件，表示一个点有且仅有个个经度度数，一个纬度数据。车的概念对应一个类，类有一个关键数据属性速度，速度是一个浮点属性。例子中定义了路、点、车的实例。如下图所示，共有福中路、红岭中路、翠竹路、金田路等路的实例，同时有福中路起点、福中路终点、红岭中路起点、红岭中路终点、翠竹路起点、翠竹路终点、金田路起点、金田路终点等八个点的实例，对应四条中的点属性。

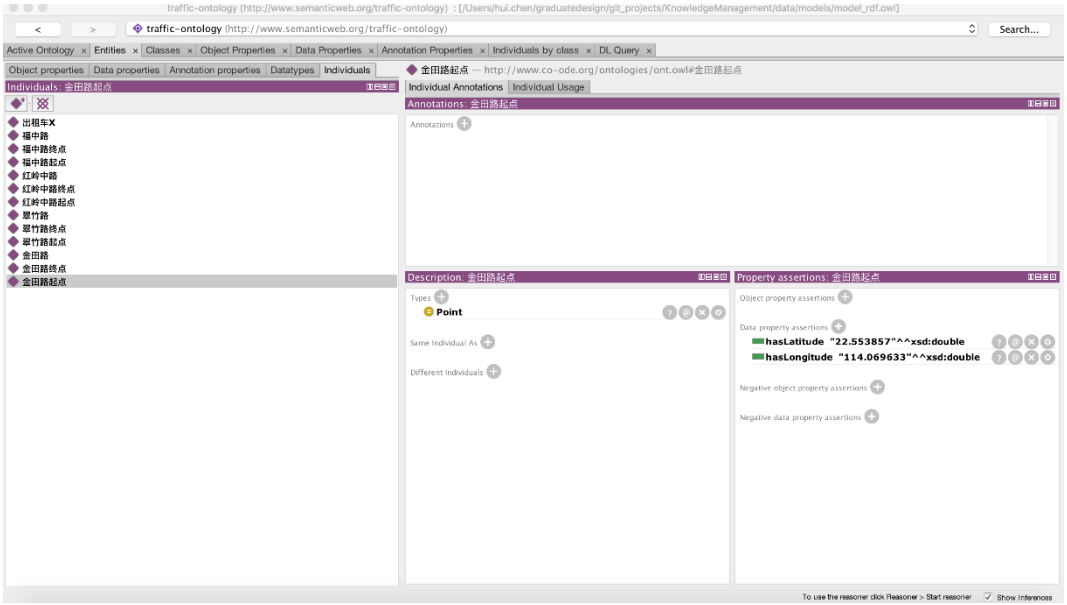

图 5-2 实例建模图

```
<!-- http://www.semanticweb.org/traffic-ontology#Point -->

<owl:Class rdf:about="http://www.semanticweb.org/traffic-ontology#Point">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/traffic-ontology#hasLatitude"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onDataRange rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/traffic-ontology#hasLongitude"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onDataRange rdf:resource="http://www.w3.org/2001/XMLSchema#double"/>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

图 5-3 点的模型部分截图

```
<!-- http://www.semanticweb.org/traffic-ontology#Road -->

<owl:Class rdf:about="http://www.semanticweb.org/traffic-ontology#Road">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/traffic-ontology#hasBeginPoint"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/traffic-ontology#Point"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="http://www.semanticweb.org/traffic-ontology#hasEndPoint"/>
            <owl:qualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:qualifiedCardinality>
            <owl:onClass rdf:resource="http://www.semanticweb.org/traffic-ontology#Point"/>
        </owl:Restriction>
    </owl:equivalentClass>
</owl:Class>
```

图 5-4 道路模型部分截图

将上述建立好的模型，通过知识管理的接口保存。因为涉及道路交通，所以本例将上述领域模型将此模型保存在名为 Traffic 的知识库中。

5.1.2 知识库管理模块的设计与实现

知识库管理利用 TDB 中的数据集的来实现，每一个知识库按照数据集来组织和管理。同时，因为每一个数据集是存储在文件系统中，利用 TDB 的 API 可以加载到内存。因此，结合事件的查询需求，我们需要将数据集按照领域进行分类，避免一个数据集过于庞大而内存不够加载使用。因此，在本模块实现时，我们需要将数据集按照特定领域分类组织，每上领域给予一个命名，数据集的名字就是领域名。同时，在配置文件中给给一个映射，每当系统启动即加载配置文件。

1）知识库管理模块的实现

知识库对应 TDB 数据集，TDB 数据集的生成通过工厂类 TDBFactory。TDBFactory 静态工厂方法用以创建和连接 TDB 数据集。用户在 Web 前端输入的数据集名，以及用户选择的内存模式还是持久化模式，内存模式在后台服务器停止运行后数据会销毁，下次运行时数据不可用。而持久化模式则会以文件的形式存储在文件之中，之后再次运行时数据可用。

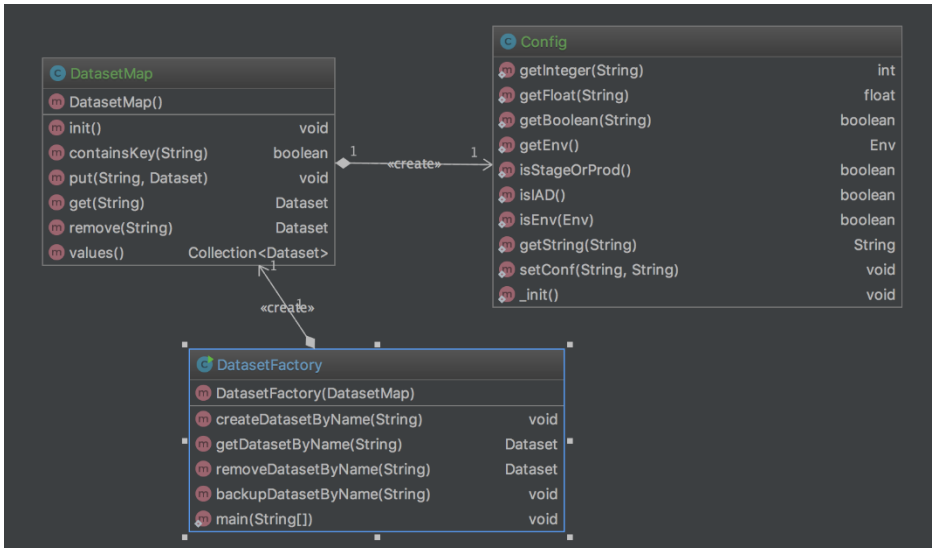

图 5-5 知识库管理类图

给定一个新的数据集名，新建一个数据集的详细实现具体涉及到的类有：DatasetFactory 类、Dataset 类，DeducerData、DatasetMap、Config 类。

表 5-1 对上述的这些类和对应的实现功能作了简要介绍：

表 5-1 知识库管理主要涉及类

类/接口	功能描述
DatasetFactory	知识库管理主类，默认是 resource 目录下的每个子目录是一个数据集。同时根据输入的名字生成、删除、办公数据集，利用 Map 进行名字和数据集之间的映射。
DeducerData	生成的数据对象，用于推理模块处理。
Dataset	数据集对象，表示一个数据集（知识库）。
DatasetMap	维护数据集名和数据集对象的映射。
Config	配置类，主要用于读取配置文件。

表 5-2 DataFactory 类 createDatasetByName 方法

方法	protected void createDatasetByName (String datasetName)
描述	根据知识库名生成对应的知识库，并且将名字和知识库对象的映射更新到 map 中，在适当的时候持久化到文件中。
参数	@Param datasetname 知识库名
返回	void

表 5-3 DataFactory 类 getDatasetByName 方法

方法	protected void getDatasetByName (String datasetName)
描述	根据知识库名获取知识库，并返回对应的知识库，若不存在则返回 null。
参数	@Param datasetname 知识库名
返回	Dataset 返回的知识库对象

表 5-4 DataFactory 类 removeDatasetByName 方法

方法	protected void removeDatasetByName (String datasetName)
描述	根据知识库名移除对应的知识库，如果不存在则返回。并且将名字和知识库对象的映射更新到 map 中，在适当的时候持久化到文件中。
参数	@Param datasetname 知识库名
返回	void

5.1.2 SPARQL 语句查询的设计与实现

知识库的管理功能最大的功能就是 SPARQL 查询，已有的本体模型存储在知识库中，新来的事件需要查询知识库事件的领域信息。利用 Jena 语义框架的 SPARQL 查询引擎 ARQ，SPARQL 查询需要生成 Query 对象，然后将对象传给知识库对象。

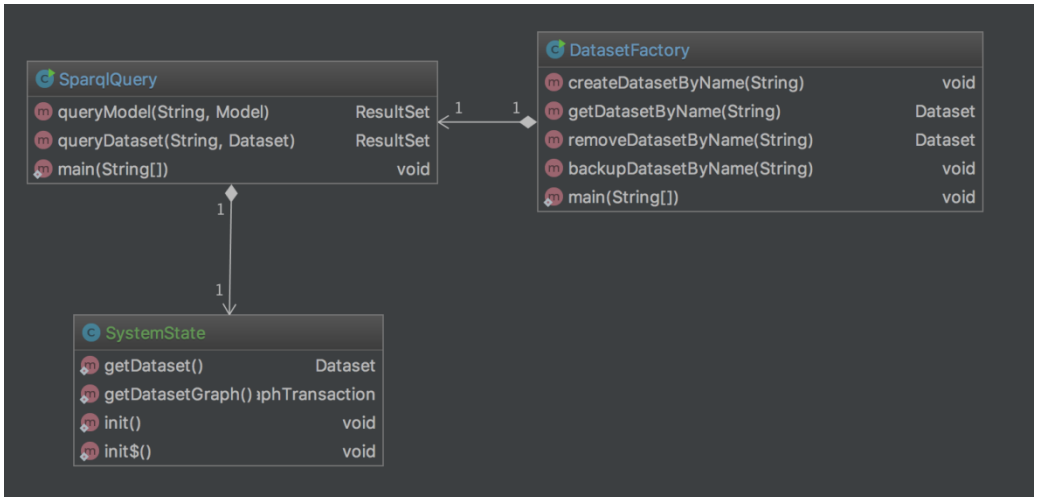

图 5-6 SPARQL 查询支持类图

知识管理模块利用 Jena 的 SPARQL 查询引擎实现基于知识库 Dataset 的 SPARQL 查询功能，SPARQL 查询语句首先需要转化生成为 Query 对象，然后调用静态工厂类 QueryExecutionFactory 的 create 接口获得 QueryExecution 对象，调用此对象的 exexSelect() 接口可以得用结果集对象 ResultSet，具体涉及到的类有：SparqlQuery,类 SystemState

类，DatasetFactory 类。

表 5-2 对上述的这些类和对应的实现功能作了简要介绍：

表 5-5 知识库 SPARQL 查询主要涉及类

类/接口	功能描述
SparqlQuery	Sparql 查询主类，提供查询 Jena Model 和 Jena Dataset 静态的接口，返回 ResultSet。
SystemState	初始化类，加载数据集和维护系统状态
ResultSet	SPARQL 查询结果集对象。
Dataset	Dataset 数据集类
Model	本体表示接口。
DatasetFactory	知识库管理主类，默认是 resource 目录下的每个子目录是一个数据集。同时根据输入的名字生成、删除、办公数据集，利用 Map 进行名字和数据集之间的映射。

表 5-6 Sparql 类 queryModel 方法

方法	protected ResultSet queryModel(String queryString, Model model)
描述	响应传入的 SPARQL 查询请求和提供的 Jena Model 对象获取结果集对象，结果集可提供迭代器遍历三元组结果。
参数	@Param queryString SPARQL 查询语句 @Param model Jena Model 对象，Jena 内部对本体的表示接口
返回	ResultSet 结果集

表 5-7 Sparql 类 queryModel 方法

方法	protected ResultSet queryModel(String queryString, Dataset model)
描述	响应传入的 SPARQL 查询请求和提供的数据集对象获取结果集，结果集可提供迭代器遍历三元组结果。
参数	@Param queryString SPARQL 查询语句 @Param dataset 数据集对象
返回	ResultSet 结果集

5.2 事件接入模块的实现

推理模块接收的数据是物联网平台上传的高速实时数据流。物理设备实时数据首先经过资源接入管理，通过相对应的协议进行处理和解析。将实际的物理数据转化为结构化的数据。并将这些结构的数据经由发布订阅消息间件发给推理模块进行处理。

事件接入模块的主要目的是提供一个数据转发功能。因此发布订阅消息空间中的主题众多，我们只需要订阅一部分主题相关的消息。本课题主要将其分为数据订阅和接收两个子模块。

5.2.1 数据订阅模块的设计与实现

数据源是经过发布订阅系统进入推理功能模块中的。本模块涉及相关事件的订阅。在本课题中，推理功能模块是消息的消费者。物联网平台数据接入平台是消息的生产者。结合面向对象的编程思想，数据订阅模块相关类图如图 5-3 所示，此部分主要涉及的类有：PubSub 类、Configuration 类、TopicEntry 类。

表 5-3 对上述的这些类和对应的实现功能作了简要介绍：

表 5-8 订阅模块主要涉及类

类/接口	功能描述
TopicEntry	发布订阅系统主题对象实体。
Configuration	动态信息配置类。可以在配置文件中对发布订阅系统相关 IP、端口等信息进行动态配置。
PubSub	实现了与发布订阅系统消息空间节点的交互。 主要方法： 获取所有的主题名称 allTopicNames(); 获取所有的主题树目录 allTopicContents(); 向主题树中新增主题 uploadTopic(String, String); 订阅主题 subscribe(String); 取消主题 unsubscribe(String); 得到消息主题的 schema 文件 getSchema(String)

表 5-9 PubSub 类 uploadTopic 方法

方法	protected void getSchema(String topic)；
描述	根据订阅主题名得到相应的 schema 文件。
参数	@Param topic 主题名 。
返回	File 文件对象。

表 5-3 PubSub 类 doPost 方法

方法	protected TopicEnty subscribe (String topic);
描述	根据主题名获取对应的主题。
参数	@Param topic 主题名字。
返回	TopicEntry 返回的主题对象。

5.2.2 数据接收模块的设计与实现

数据接收模块主要是用于接收数据订阅模块所有订阅主题的数据，只要此相关主题的消息被发布，推理功能模块就需要得到这些数据（事件）进行处理。如何接收并正确的预处理这些数据由数据接收模块来实现。

由于推理功能模块能接收处理的数据是带有领域信息的对象，所有数据接收模块需要将从发布订阅接收到的数据转化为 Java 对象，具体涉及到的类有：SchemaFunction, 类 NotificationProcessImpl 类，DeducerData 类。

表 5-4 对上述的这些类和对应的实现功能作了简要介绍：

表 5-10 数据接收模块主要涉及类

类/接口	功能描述
SchemaFunction	根据输入的 Schema 文件，解析事件属于哪个主题。
DeduceData	生成的数据对象，用于推理模块处理。
NotifictionProcessImpl	负责监听是否有数据到来，每当有新的数据到来时，首先解析出数据是属于个主题。然后根据相应的主题生成对应的数据对象。

表 5-11 SchemaFunction 类 getTopicNameBySchema 方法

方法	protected String getTopicNameBySchema(Document schema)
描述	根据输入的 schema 文件的内容判断数据属于哪个主题，返回值为类名。
参数	@Param schemat org.w3c.dom.Document 对象。
返回	String

5.3 推理模块的设计与实现

推理功能模块是核心模块，推理模块主要提供给用户能够基于目标进行即时推理的功能。用户通过前端定义目标，提交交给后台推理。本模块主要包括 OWL 解析，一阶逻辑子句转化，推理等三个子模块。

5.3.1 OWL 解析模块的详细设计与实现

OWL 本体的解析包括本体的加载和解析两个步骤。本体的格式包括文件或者输入流的格式。本质上文件的处理也是先通过加载文件为一个流对象再调用数据流的加载函数。通过将本体进行加载入内存，然后解析生成本体对象，以便后续通处理本体对象获取描述逻辑子句。本课题通过分析 Hermit 本体推理器的加载过程，采用与之同样的方

法，即利用 OWL API 进行进行加载并生成 Ontology 对象。OWL 本体解析模块主要实现本体的加载解析两个功能。

OntManager：使用单例模式，因为可能会出现多个线程同时调用的问题，所以采用多线程安全模式创建一个本体管理器单例。

ParseOWLToOWLOntology：解析 OWL 输入流或者本体，利用本体管理器生成 OWLOntology 对象。解析文件的流程实质是先加载文件为 InputStream 标准流的形式，然后再调用上述流的加载流程。

拿到 OWLOntology 对象之后，我们需要利用 Hermit 的 Reasoner API 分析 Ontology 的内部结构，生成 OWLClausifier 对象，然后根据 Ontology 调用 clausify 函数对 Ongoloty 对象进行子句化过程。通过子句化之后生成 OWLDLOntology 对象，此对象内部包含一个 DLClause 的集合对象，此对象集合包含所有的描述逻辑子句。

表 5-5 对上述的这些类和对应的实现功能作了简要介绍：

表 5-12 本体 OWL 解析模块主要涉及类

类/接口	功能描述
OntManager	采用单例模式实现的本体管理器管理器单例类。
ParOWLToOWLOntology	加载本体文件，利用 Hermit 的 Reasoner API 生成 OWLOntology。 parse(InputStream inputStream) 根据输入流解析成本体对象。 parse(File file) 根据输入的文件解析成本体对象。
OntologyManager	本体加载的主类，生成本体对象。

表 5-13 ParOWLToOWLOntology 类 parse 方法

方法	protected OWLOntology parse(InputStream inputStream)
描述	根据输入流解析成本体 OWLOntology 对象。
参数	@Param InputStream 输入流对象。
返回	OWLOntology 本体对象

表 5-14 ParOWLToOWLOntology 类 parse 方法

方法	protected OWLOntology parse(File file)
描述	根据输入文件解析成本体 OWLOntology 对象。
参数	@Param File 输入文件对象。
返回	OWLOntology 本体对象。

5.3.2 一阶逻辑子句转化模块的详细设计与实现

本模块的功能主要是将 DLClause 转化为对应的 Z3 的一阶逻辑量词形式。在 Z3 中量词是用 Quantifier 表示。因此，结合 DLClauser 的 String 格式，每一个 DLClause 转化为对应一个量词表达式 Quantifier 对象，然后将其强制转化一阶逻辑表达式 BoolExpr 对象。利用 Context 类的 mkAnd 函数将所有的 BoolExpr 对象合取成一个合取范式。模块具体涉及到的类有：OWLToZ3，Context 类。

表 5-6 对上述的这些类和对应的实现功能作了简要介绍：

表 5-15 一阶逻辑子句转化模块主要涉及类

类/接口	功能描述
QuantifierGenerate	一队逻辑表达式对象生成主类。 findDomainFormulaString(String) 从 DLClause 的 String 表示找出谓词的前提结论。 mkQuantifier(Context, DLCLause) 生成量词表达词的函数，谓词的前提和结论也可能谓词，分别解析前提和结论。
Context	生成 Solver、BoolExpr、Quantifier 的工厂对象。

表 5-16 QuantifierGenerate 类 findDomainFormulaString 方法

方法	protected String[] findDomainFormulaString(DLClaused dlclause)
描述	根据输入的 DLClause 对象获得谓词的前提和结论。
参数	@Param dlClause DLClause 对象。
返回	String[] 谓词的前提和结论。

表 5-17 QuantifierGenerate 类 mkQuantifier 方法

方法	protected Quantifier mkQuantifier (Context context, DLClaused dlclause)
描述	将 DLClause 转化为一阶逻辑表达式量词对象，首先根据输入的 DLClause 对象获得谓词的前提和结论。然后将前提和结论分别处理成一阶逻辑表达式对象。最后用 Context 的 mkImply 将两者组合成谓词表达式。
参数	@Param context Z3 Context 对象。 @Param dlClause DLClause 对象。
返回	Quantifier Z3 谓词表达式。

DLClause 的形式为 A→B，在一阶逻辑的涵义就是 A 推出 B，在系统的处理中间结果中，每一个 DLClause 的 A 和 B 对应以下形式，A 对应一个合取概念，B 对应一个析

取概念。

A：

<http://www.semanticweb.org/traffic-ontology#Point>(X),

<http://www.semanticweb.org/traffic-ontology#hasLongitude>(X,Y1),

<http://www.semanticweb.org/traffic-ontology#hasLongitude>(X,Y2)

B：

not(xsd:double)(Y1) v not(xsd:double)(Y2) v Y1 == Y2

我们定义了地图上点的概念，一条点有且仅有两个浮点属性，一个经度，一个纬度。A 推出 B 的含义是如果一个点有两个经度，那么可以推出这两个点要么其中一个不是浮点数，要么这两个相等（因为只能有一个经度）。转换成 Z3 一阶逻辑表达式的格式如下图所示：

```
(let ((a!1 (or (forall ((temp Real))
                 (! (not (= temp Y1)) :skolemid skid2 :qid Q2))
               (forall ((temp Real))
                 (! (not (= temp Y2)) :skolemid skid2 :qid Q2)))))
  (=> (and (= (|<http://www.semanticweb.org/traffic-ontology#Point>| X) true)
           (= (|<http://www.semanticweb.org/traffic-ontology#hasLatitude>| X)
              Y1)
           (= (|<http://www.semanticweb.org/traffic-ontology#hasLatitude>| X)
              Y2))
      a!1))
```

图 5-7 转化生成的一阶逻辑表达式（全称量词）图

5.3.1 推理的详细设计与实现

结合前面的设计分析，因为用户的推理目标同样需要处理成一阶逻辑表达式 BoolExpr。本实现采用一个 Deducer 类，此类维护一个 Solver 的列表，可以同时推理多个推理目标。每一个 Solver 保存一个推理目标表达式 BoolExpr 的逻辑反。即利用 Context 的 mkNot(BoolExpr)接口可获得一个布尔表达式的逻辑反。然后，每来一个事件，如前所述根据事件会经过处理最终会生成一个布尔表达式 BoolExpr，根据其与哪些目标相关添加到相应的 Solver 中。

然后调用 Solver 的 check()接口，如果结果是 Status.UNSATISFIABLE 则证明相应的目标成立。具体涉及的类有：Client 类，Deducer 类, DeducerData 类, FetchModel 类等。

表 5-7 对上述的这些类和对应的实现功能作了简要介绍：

表 5-18 一阶逻辑子句转化模块主要涉及类

类/接口	功能描述
Client	生于接收推理目标，包括参数的提取和转化。从发布订阅接收事件数据。调用查询知识库 API。过滤无用数据。 parseTarget(Target target)将目标对象处理成 BoolExpr。

（续上表）

Context	生成 Solver、BoolExpr、Quantifier 的工厂对象
Deducer	推理主类。 deduce(BoolExpr)将布尔表达式添加到对应 Solver 中，调用 Solver 的 check()得到推理结果。
FetchModel	获取知识库信息的代理类

表 5-19 Deducer 类 deduce 方法

方法	protected void deduce(BoolExpr boolexpr)
描述	将布尔表达式添加到对应 Solver 中，调用 Solver 的 check()得到推理结果。并将推理结果存储到数据库中。
参数	@Param boolexpr 一阶逻辑表达式对象。
返回	Void

表 5-20 Client 方法

方法	Protected void parseTarget(Target target)
描述	生成对应的 Solver 对象，根据目标对象生成 BoolExpr 对象，并将对其取逻辑反加入到 solver 中，最后将 Solver 加入到 Deducer 中。
参数	@Param target 目标请求对象。
返回	void

5.4 用户界面模块

界面模块主要是提供用户一个直观的 Web 管理平台。主要有两大功能，对知识库的直接管理功能，包括数据集的添加、删除、备份、SPARQL 查询。推理目标的定义功能，用户可以通过推理标界面，按照提供的选项输入领域信息，选择相关参数，然后提交给后台，后台完成后将数据数据到到数据库 MongoDB 中。然后，用户可以选择展示结果，同样请求会提交到 Web 后台，后台根据请求访问数据库获取数据，然后以 Json 的数据格式返回。

后台的实现采用 Jetty 实现，通过继承 AbstractHandler 实现 DataHandler，这是请求处理的主类，根据请求的不同执行不同的操作。具体涉及到的类有：DataHandler，类 Main 类，MongoToo 类、JettyService 类、DatasetFactory 类。

表 5-8 对上述的这些类和对应的实现功能作了简要介绍：

表 5-21 数据集删除功能

类/接口	功能描述
DataHandler	Web 请求处理类。根据接收到的 HttpRequest，判断其中的请求地址后缀，如果是/deduce 则将其余参数传给推理器，如果是/data 则获取结果并返回。
Main	后台服务入口，Web 后台的服务启动类。
MongoTool	MongoDB 数据库的类，采用单例模式实现。提供获取和存储数据的接口。
JettyService	Jetty 的启动类，提供启动函数 start。
DatasetFactory	数据集的管理调用类，根据用户的请求添加、删除、备份数据集（知识库）。

表 5-22 DataHandler 类 doPost 方法

方法	protected void handle(HttpServletRequest request, HttpServletResponse response)
描述	响应用户接口的请求，根据请求的内容生成进行处理，包括推理、结果获取以及模型知识库的增删改查等功能调用。
参数	@Param request http 请求。 @Param response http 响应对象。
返回	void

表 5-23 Jetty 类 start 方法

方法	protected void start(int port, AbstractHandlerhandler)
描述	根据传入的接口和 AbstractHandle 对象生成 Jetty 的 Server 对象，然后启动 Server 开始监听输入的端口。
参数	@Param port 要监听端口。 @Param handler 实际响应类对象。
返回	Void

5.5 本章小结

本章分别对设计的各个模块进行了的的实现介绍，包括实现技术、工具、具体实现方法。主要从以下几个模块给出了详细的设计流程图和实现类图、主要代码，包括本体建模模块、知识库管理模块、事件接入模块、推理模块和用户界面模块这五个大的模块。其中推理模块又细分了几个小的部分：OWL 解析模块的设计与实现、一阶逻辑子句转

化的设计与实现、推理的设计与实现，对这几个小部分分别进行了流程设计和类图实现以及相应的类图讲解，到本章为止我们的知识管理子系统也完成了我们的需求实现，接下来我们将对本系统进行测试和验证。

第六章 系统的测试及验证

6.1 测试目标及环境

6.1.1 测试目标

对于在本论文研究背景中提到的基于物联网平台的知识管理子系统，本章将从功能测试和性能测试两个方面对原型系统进行测试及验证。

（1） 功能测试。主要目标是测试系统的 Web 的前端和后台，包括数据集的管理功能：添加、删除、上传、备份，数据集查询查询，推理及结果展示等功能模块。并通过将平台应用到真实的样例来测整体的处理流程。并且保证功能正常运行。

（2） 性能测试。主要目标是测试基于物联网平台的知识管理子系统在处理用户需求时的性能。包括前端管理时的系统的响应时间、SPARQL 查询的响应时间、推理系统每秒处理的消息数量，以及 CPU 占用、内存占用等性能指标。

6.1.2 测试环境

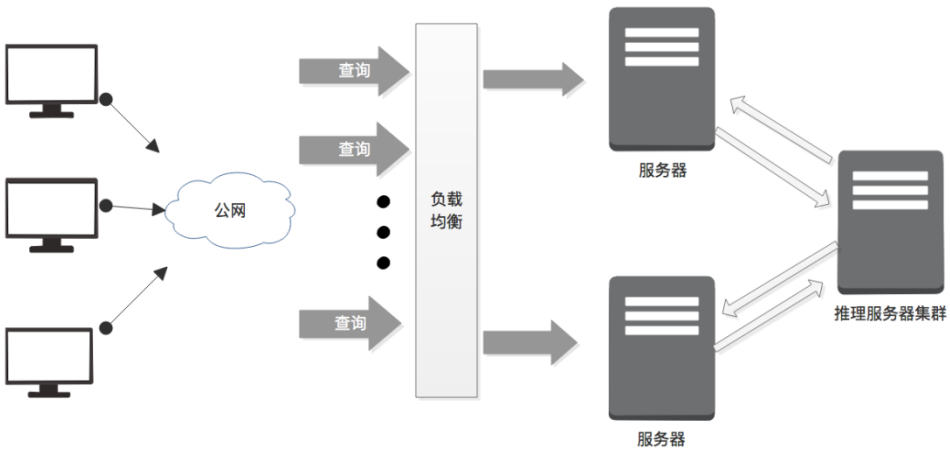

图 6-1 测试环境部署图

图 6-1 所示为测试环境的部署拓扑结构：

- Web 客户端：网页客户端，用于管理数据集和目标定义以及结果展示。
- Web 后端：Linux 系统下 Jetty 环境，运行后台程度。
- 推理服务器：Linux 系统下 Jetty 环境，运行推理服务器程序。
- 数据库服务器：安装 MongoDB，负责存储推理结果的相关数据信息。

表 6-1 虚拟服务软件配置

操作系统	Ubuntu Server 14.04 LTS
Java 虚拟机	Java Runtime Environment 1.8.0_112
Z3 版本	4.5.0
Hermit 版本	1.3.8
Jena 版本	3.1.1
MongoDB 版本	3.4.0

表 6-2 虚拟服务器硬件配置

处理器	Intel(R) Xeon(R) CPU E5640，主频 3.00GHz+2.99GHz，6 核
内存	4GB
网卡	百兆网卡
硬盘存储	600GB

表 6-3 客户端硬件配置

处理器	英特尔酷睿 I3，主频 2.5GHz，双核
内存	2GB
网卡	百兆网卡
硬盘存储	250GB

表 6-4 客户端软件配置

操作系统	Windows 7 旗舰
Java 虚拟机	Java Runtime Environment 6.0
网络浏览器	Chrome

6.2 系统功能测试

6.2.1 数据集管理功能测试

(一) 数据集模块

1. 添加数据集功能

1) 测试用例

表 6-5 数据集添加功能

用例目的	该用例用于测试“数据集添加”功能。
描述	该用例用于测试“数据集添加”功能，是系统的关键步骤。
前提条件	用户打开网页，进入管理数据集页面。
测试步骤	1） 点击加入新数据集。 2） 输入新的数据集名，选择数据集类型。 3） 点击“创建数据集”按钮。
预期结果	登录成功，用户进入数据集主页面。
测试结果	测试通过。

2) 测试结果

图 6-2 是数据集添加功能过程的截图。显示了创建数据集的过程。

图 6-2 数据集添加功能

2. 删除数据集功能

1) 测试用例

表 6-6 数据集删除功能

用例目的	该用例用于测试“数据集删除”功能。
描述	该用例用于测试“数据集删除”功能，是系统的关键步骤。
前提条件	用户打开网页，进入管理数据集页面。
测试步骤	1） 点击“已有数据集”按钮，进入已有数据集界面； 2） 选择一个数据集点击“移除”按钮； 3） 弹出界面，点击确认；
预期结果	删除成功，用户进入管理数据集页面。
测试结果	测试通过。

2) 测试结果

图 6-3 是 Web 端数据集移除功能中确认移除界面。显示了用户选择了 test 数据集并

点击移除按钮后弹出的界面，等待用户确认操作。

图 6-3 数据集删除界面

3. 备份数据集功能

1) 测试用例

表 6-7 数据集备份功能

用例目的	该用例用于测试“数据集备份”功能。
描述	该用例用于测试“数据集备份”功能，是系统的关键步骤。
前提条件	用户打开网页，进入管理数据集页面。
测试步骤	1）点击“已有数据集”按钮，进入已有数据集界面； 2）选择一个数据集点击“备份”按钮； 3）弹出界面，点击确认；
预期结果	备份成功，用户进入管理数据集页面。
测试结果	测试通过。

2) 测试结果

图 6-4 是 Web 端数据集备份功能中确认备份界面。显示了用户选择了 traffic 数据集并点击移除按钮后弹出的界面，等待用户确认操作。

图 6-4 数据集备份确认界面

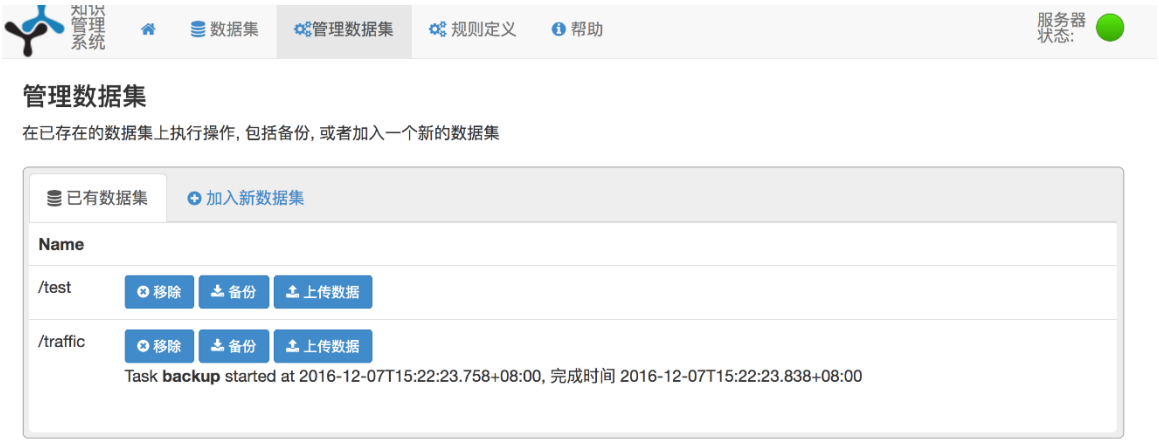

图 6-5 数据集备份完成界面

图 6-5 是 Web 端数据集备份功能中确认备份之后。显示了用户选择了 traffic 数据集并点击确认备份后的界面，数据集下方有备份的标注。

(二) SPARQL 模块

1. 单个文件上传功能

1) 测试用例

表 6-8 文件上传功能

用例目的	该用例用于测试“单个文件上传”的功能。
描述	该用例用于测试“单个文件上传”的功能，是系统的关键步骤。
前提条件	用户打开 Web，进入主页面。
测试步骤	1） 点击首页左上角的“点击数据集”，数据集列表； 2） 在数据集列表中，点击任意一个数据集数据； 3） 点击上传文件按钮，选择本地文件； 4） 点击上传按钮。
预期结果	测试成功。
测试结果	测试通过。

2) 测试结果

图 6-6 是对选定的数据集 traffic 选择单个文件后上传成功的界面。

6-6 数据集备份确认界面

2. 批量文件上传功能

1) 测试用例

表 6-9 文件上传功能

用例目的	该用例用于测试“文件批量上传”的功能。
描述	该用例用于测试“文件批量上传”的功能，是系统的关键步骤。
前提条件	用户打开 Web，进入主页面，。
测试步骤	1）点击首页左上角的“点击数据集”，数据集列表； 2）在数据集列表中，点击任意一个数据集数据； 3）点击上传文件按键，选择多个本地文件； 4）点击上传所有文件按钮。
预期结果	测试成功。
测试结果	测试通过。

2) 测试结果

图 6-7 是文件批量上传成功时的界面。

图 6-7 文件批量上传确认界面

3. SPARQ 查询功能

1) 测试用例

表 6-10 文件上传功能

用例目的	该用例用于测试“SPARQL 查询”的功能。
描述	该用例用于测试“SPARQL 查询”的功能，是系统的关键步骤。
前提条件	用户打开 Web，进入主页面，。
测试步骤	1）点击首页左上角的“点击数据集”，数据集列表； 2）在数据集列表中，点击任意一个数据集数据； 3）点击查询按键，编辑 SPARQL 查询语句； 4）点击查询。
预期结果	测试成功。
测试结果	测试通过。

2) 测试结果

图 6-8 是 SPARQL 查询语言输入界面，图 6-9 是 SPARQL 查询结果界面。

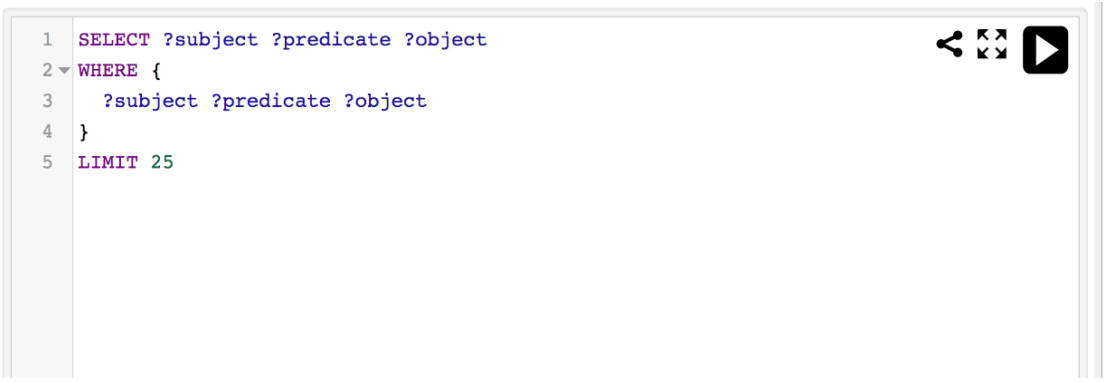

图 6-8 SPARQL 查询输入界面

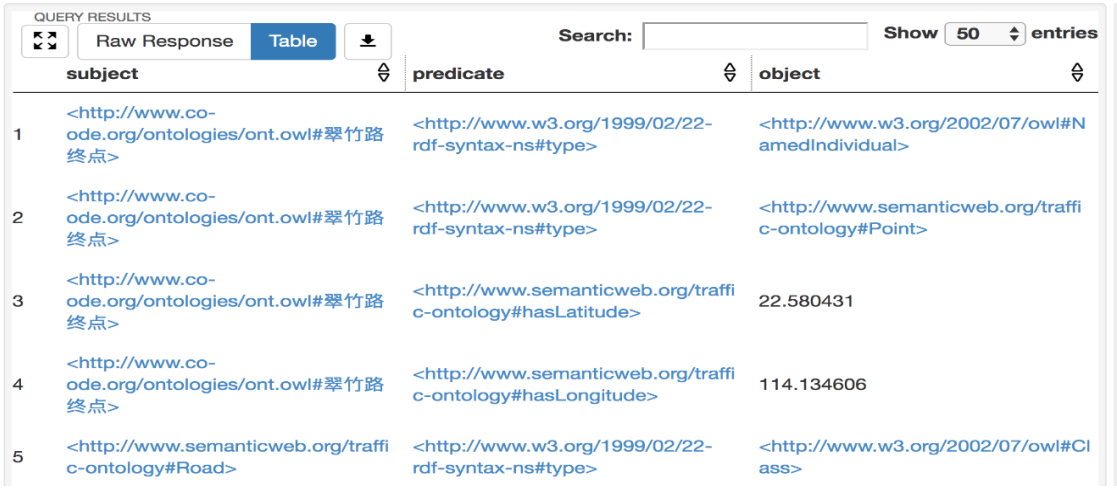

图 6-9 SPARQL 查询结果界面

（三）规则定义模块

1. 规则定义功能

1）测试用例

表 6-11 规则定义功能

用例目的	该用例用于测试“规则定义”功能。
描述	该用例用于测试“规则定义”功能，是系统的关键步骤。
前提条件	用户打开 Web 端，进入“规则定义”页面。
测试步骤	1） 用户选择道路。 2） 用户输入规则定义数据。 3） 点击“提交”按钮
预期结果	测试成功。
测试结果	测试通过。

2）测试结果

图 6-10 是推理规则定义界面。

图 6-10 推理规则定义界面

2. 推理结果展示功能。

1) 测试用例。

表 6-12 推理结果展功能

用例目的	该用例用于测试“推理结果展示”功能。
描述	该用例用于测试“推理结果展示”功能，是系统的关键步骤。
前提条件	用户打开 Web 端，进入“规则定义”页面。
测试步骤	点击“结果”按钮；
预期结果	测试成功。
测试结果	测试通过。

2) 测试结果。

图 6-11 是推理结果展示界面，其中 x 坐标表示时间，坐标表示拥堵程度，其中 10 表示畅通，30 表示轻微拥堵，50 表示拥堵，70 表示严重拥堵。

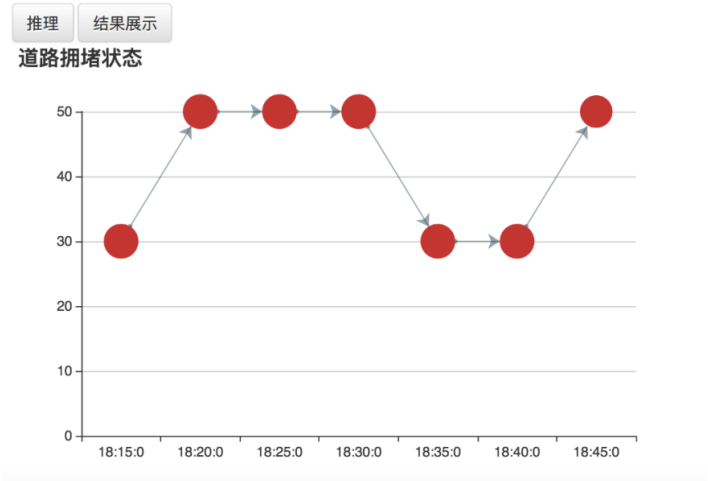

图 6-11 推理结果展示界面

6.3 性能测试

性能测试主要包括压力测试以及系统正确处理能力等情况。主要从两个方面进行性能测试，知识库的查询性能和推理功能模块的处理速度。

表 6-13 测试环境配置

操作系统	Ubuntu Server 14.04 LTS
Java 虚拟机	Java Runtime Environment 1.8.0_112
Z3 版本	4.5.0
Hermit 版本	1.3.8
Jena 版本	3.1.1
TDB 版本	3.1.1
MongoDB 版本	3.4.0

由于我们的测试环境的 PC 机，同时所有需要的软件都在同一台机器上部署，模拟数据发磅，发布订阅系统，MongoDB 数据库、推理模块。所以我们通过多线程发送数据来模拟压力测试，查看知识库的响应性能和推理模块的处理速度。

采用对知识库进行 SPARQL 查询时的测试结果如表 6-14 所示：

表 6-14 性能测试结果 1

线程数量	每个线程发送的查询请求	SPARQL 语句中 Where 子句中的条件个数	总耗时 ms
1	1000	1	3
1	1000	2	4
1	1000	3	6
1	3000	1	10
1	3000	2	14
1	3000	3	20
1	5000	1	14
1	5000	2	20
1	5000	3	31
1	10000	1	29
1	10000	2	41
1	10000	3	58
4	1000	1	15
4	1000	2	21
4	1000	3	29

（续上表）

4	3000	1	49
4	3000	2	71
4	3000	3	99
4	5000	1	68
4	5000	2	101
4	5000	3	152
4	10000	3	152
4	10000	1	220
4	10000	2	311

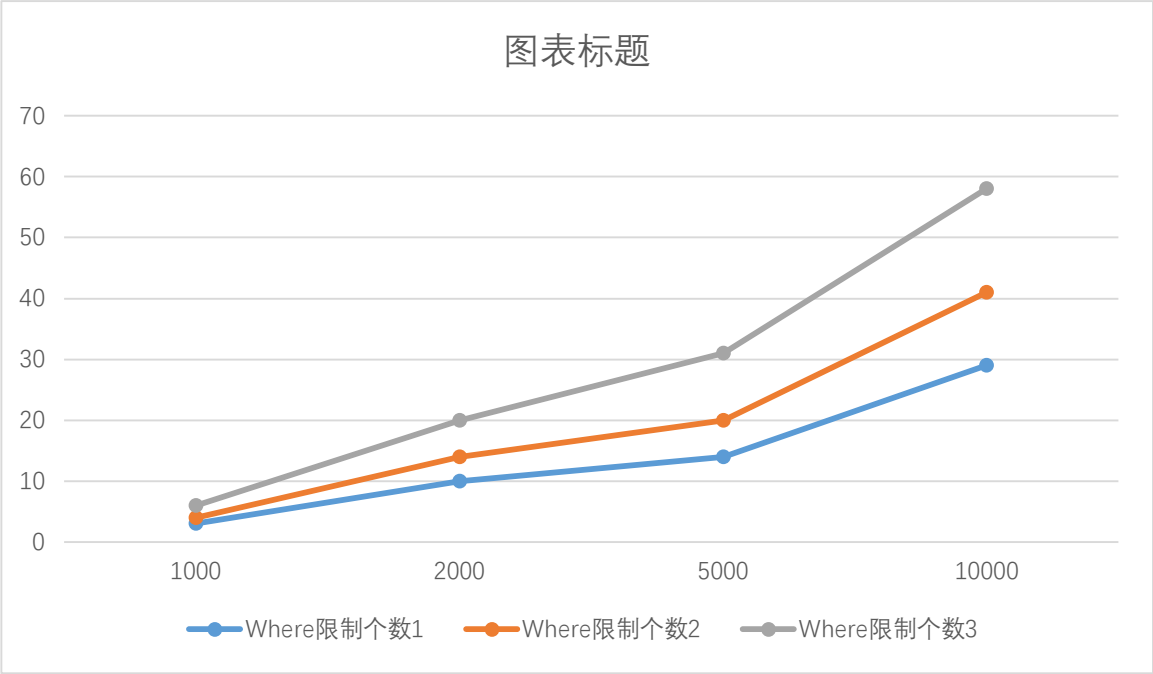

图 6-12 单线程测试界面

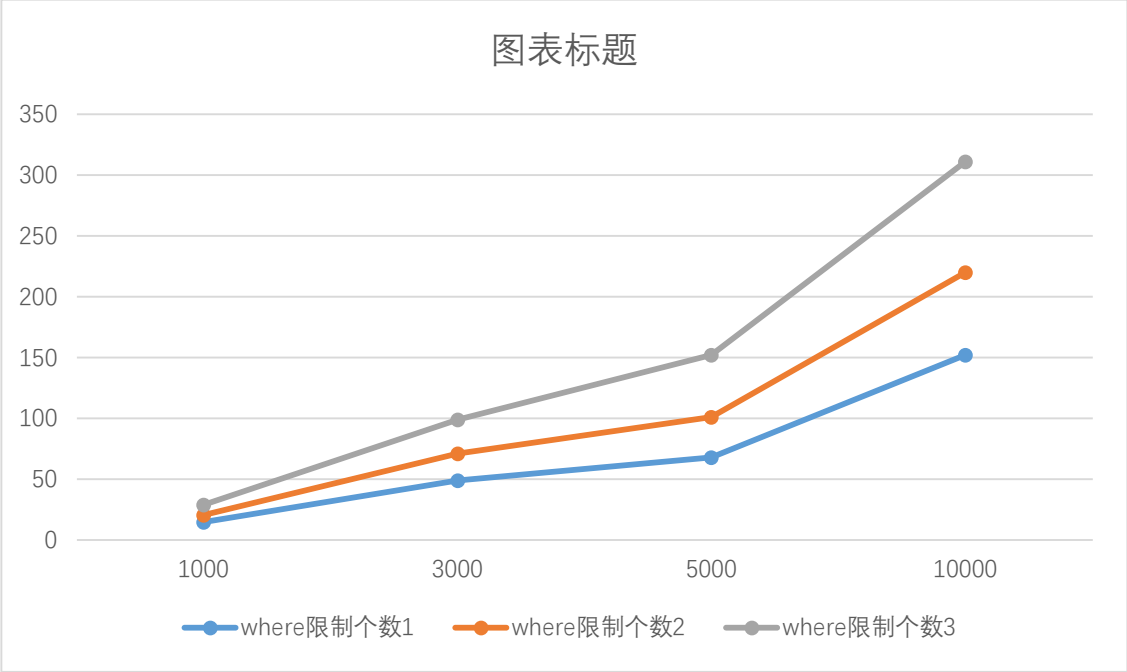

图 6-13 4 个线程测试结果图

从表 6-13 可以看出，在我们把所有的软件都部署在普通 PC 机上，分别采用 1、2、4 个线程并发的用 SPARQL 查询知识库，性能测试结果图表明，影响 SPARQL 的因素主要是网络时延，查询功能在压力测试下完全可以胜任。统计图如图 6-12 和 6-13 所示。

表 6-14 性能测试结果 2

线程数量	每个线程发送的数据量	处理时间(ms)	CPU 使用率
1	10000	2	20%
1	20000	2	40%
1	50000	3	70%
1	100000	4	93%
2	10000	2	50%
2	20000	2	70%
2	50000	4	95%
2	100000	8	940%
4	10000	3	50%
4	20000	4	80%
4	50000	8	93%
4	100000	20	94%

表 6-14 是我们对推理模块的压力测试结果表，同样把所有的软件都部署在普通 PC 机上，分别采用 1、2、4 个线程并发的发送数据（本例中主要是深圳出租车的 GPS 数据）可以看出，影响性能的主要因素是数据量的大小和并发数。

6.4 本章小结

本章从功能和性能两个方面对文中实现的系统做了详细的测试和验证。其中功能性测试主要针对平台提供的知识库管理、推理目标定义和结果展示等功能进行了测试，描述了各模块功能的测试用例，以截图的方式对测试结果进行了展示。性能测试方面则主要是针对知识库的查询和推理模块对事件的处理进行相应的测试，并从每秒处理事件数量、事件处理时间、CPU 等性能指标对测试结果进行了统计和展示。

第七章 总结与展望

7.1 论文总结

物联网平台中资源的异构性、多样性使的基于本体的模型。本论文在深入研究本体建模在物联网中的应用以及逻辑推理的基础上，借鉴面向对象的一些思想，设计并实现了基于模型的物联网知识管理子系。并将知识管理子系统的实现主要分为四个模块：领域模型的构建、知识库的管理和查询、数据的接入和推理功能模块。

本论文完成的工作主要包括：

（1）对基于 SCA 的物联网服务系统中的事件处理平台所使用到的相关技术进行了调研，包括本体领域建模，描述逻辑与一阶逻辑的转化、Jena Web 语义框架、一阶逻辑推理库 Z3、本体推理器 Hermit 等。

（2）从功能性和非功能性共两个方面对事件处理平台进行了需求分析。然后阐明了知识管理平台中各个模块的功能结构，最后给出了知识管理子系统的整体架构设计。

（3）对基于模型的的物联网知识子系统中的具体的模块详细设计和实现进行了阐述。包括领域模块的构建、知识库的管理、知识库的查询、事件接入模块的实现、推理功能及其子模块 OWL 的解析和加载、一阶逻辑逻辑子句的生成转化、推理的实现等进行了详细的设计与实现。在介绍过程中，主要通过对具体实现类的功能介绍及其核心代码进行了详细的说明。

（4）从功能和性能两个方面对文中实现的系统做了详细的测试和验证。其中功能性测试主要针对平台提供的知识库管理、推理目标定义和结果展示等功能进行了测试，描述了各模块功能的测试用例，以截图的方式对测试结果进行了展示。性能测试方面则主要是针对知识库的查询和推理模块对事件的处理进行相应的测试，并从 cpu 占用、内存占用、每秒处理事件数量、事件处理时间等性能指标对测试结果进行了统计和展示。

7.2 下一步研究工作

本论文参考了。本论文在深入研究物联网知识建模的基础上，借鉴事件驱动的一些思想，设计并实现了基于模型的物联网服务系统中的知识管理子系统，基本完成了知识管理系统的设计和实现和测试。但是还有很多的工作需要进一步的研究与完善：

1） 建模部分，目前建模涉及领域太小，信息量太少，下一步需要扩大领域建模范围，增大知识库，另一方面，知识库的查询接口目前来说还能胜任，但在应对大量数据的真实场景会比较吃力，在并行性和查询预处理上有待提高。下一步可考虑将知识库采

用分布式实现，得用负载均衡缓解单一结点的查询压力，从而提高系统效率。

2）目标定义部分，现有的目标定义阶段提供的算子（函数）太少，对于不同的数据源有比较固定的定义。下一步的计划是增大函数库，增大系统的应用范围和可用性。

3）在测试部分，我们对各个模块进行了功能性的测试并使用真实世界的数据进行了流程测试，但由于没有在真实的环境中的使用，所以我们仍然不知道真实情况下使用的结果。所以下一步的工作是采用真实环境对知识管理子系统的进行测试，根据使用的结果再对系统进行完善和进一步的修改。

4）在性能方面，由于推理模块目前还是单机模式，推理性能有所限制，在大量事件处理场景会导致性能下降，而且本论文所使用的计算环境还不复杂，所以在就对复杂的环境之下的处理还得进一步开发。

参考文献

[1] 吴思齐. 物联网资源建模工具的设计与实现[D]. 北京邮电大学, 2014. MLA

[2] ITU-T Y. Overview of ubiquitous networking and of its support in NGN[J]. ITU-T Recommendation, 2009.

[3] Open Geospatial Consortium. OpenGIS SWE Service Model Implementation Standard, 2011[J]. Open Geospatial Consoritum, 155.

[4] 王洪微.知识本体模型下的不一致推理机制研究[D].大连：大连海事大学，2013

[5] Boulos M N K, Resch B, Crowley D N, et al. Crowdsourcing, citizen sensing and sensor web technologies for public and environmental health surveillance and crisis management: trends, OGC standards and application examples[J]. International journal of health geographics, 2011, 10(1): 1.

[6] Borgida A. On the relative expressiveness of description logics and predicate logics[J]. Artificial intelligence, 1996, 82(1): 353-367.

[7] 周明. 物联网应用若干关键问题的研究[D]. 北京: 北京邮电大学, 2014.

[8] 赵帅. 物联网资源管理框架及服务提供平台[D]. 北京邮电大学, 2014.

[9] 谢桂芳. SPARQL-一种新型的 RDF 查询语言[D]. 湘南学院学报,2009.

[10] 张雪. 基于物联网业务平台的设备管理方法研究与实现[D]. 北京邮电大学, 2014.

[11] 刘佩云. 基于本体的物联网设备共享信息模型的研究与实现[D]. 北京邮电大学,2013.

[12] 王莉. 基于本体的知识检索系统研究与实现[D]. 中国海洋大学,2008.

[13] 李丽. 基于本体的网页文本分类的研究[D] . 北京交通大学,2008.

[14] 杨柳. 模糊本体建模方法及语义信息处理策略研究[D] . 中南大学,2011.

[15] 王翀. 基于 OWL 元模型的本体建模研究[D] . 武汉大学学报,2004.

[16] 秦小燕. 一阶逻辑系统的计量化研究[D] . 西南交通大学,2015.

[17] 徐贵红,张健. 语义网的一阶逻辑推理技术支持[J] . 软件学报,2008.

[18] 刘柱子. 基于描述逻辑的本体推理技术研究[D]. 天津理工大学,2015.

[19] 张宗仁. 基于自然语言理解的 SPARQL 本体查询[J] . 计算机应用,2010.

[20] 汪璟玢,方知立. 面向分布式的 SPARQL 查询优化算法[J]. 计算机科学, 2014.