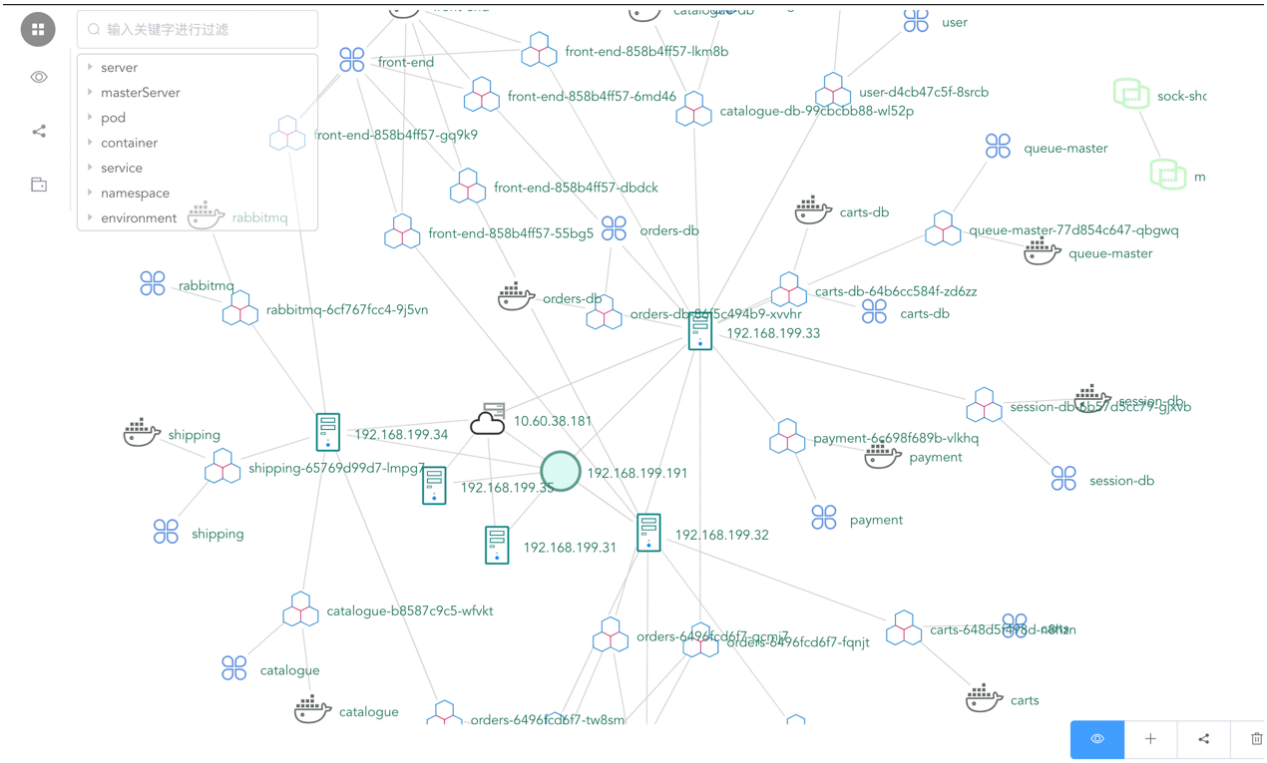
Requirement Specification

Project Introduction

Background

- **Knowledge Graph** : a kind of structured data processing method, which involves a series of technologies such as knowledge extraction, representation, storage and retrieval. From the origin, it is the fusion of knowledge representation and reasoning, database, information retrieval, natural language processing and other technologies.



- **Operation and Maintenance**: generally refers to the maintenance of network hardware and software established by large organizations, among which the traditional Operation and Maintenance refers to IT Operation and Maintenance. Operations and maintenance are important and extensive responsibilities for the entire software development life cycle, failures can occur at any time, and the value of operations and maintenance is becoming increasingly important because of the frequent iteration of business applications.

Combined operations and knowledge graph, let the knowledge graph offline mining operations history data, establishing all kinds of portraits, tease out all kinds of high level of knowledge, operations staff have already been good operations using data sets and knowledge graph to make decisions, not only reduce the operations staff workload, and improve the operational efficiency and accuracy.

Due to the complexity of the distributed system, once a fault occurs at a certain location, a series of service failures will result. However, simply repairing the affected service cannot fundamentally solve the problem, and the **root cause** of the fault needs to be located and repaired. In the operation and maintenance knowledge graph, knowledge reasoning can deduce the key factors that affect the abnormal service, and then quickly locate the root cause of the fault, so as to quickly recover the affected service and reduce the loss caused by the service failure.

This project focuses on knowledge reasoning, involving knowledge representation, storage and retrieval, not event extraction or relational extraction. Knowledge Reasoning is the process of obtaining new Knowledge from existing Knowledge based on specific rules and constraints.

Prupose of the project

1. Construct the ontology layer of the knowledge graph, so as to improve the structure of the knowledge graph.
2. Transmit self-defined rules, obtain reasoning results based on the rules, and write the newly discovered knowledge into the database.
3. Perform ontology reasoning on knowledge graphs and write newly discovered knowledge into the database.
4. Make deep learning-based reasoning on knowledge graph and write newly discovered knowledge into the database.
5. Import data from fuseki into neo4j and establish an intermediate layer to eliminate the influence of different knowledge repositories.

Target Users

- Operation and Maintenance Engineer
- Operations staff
- Members in x-lab

Boundary of the project

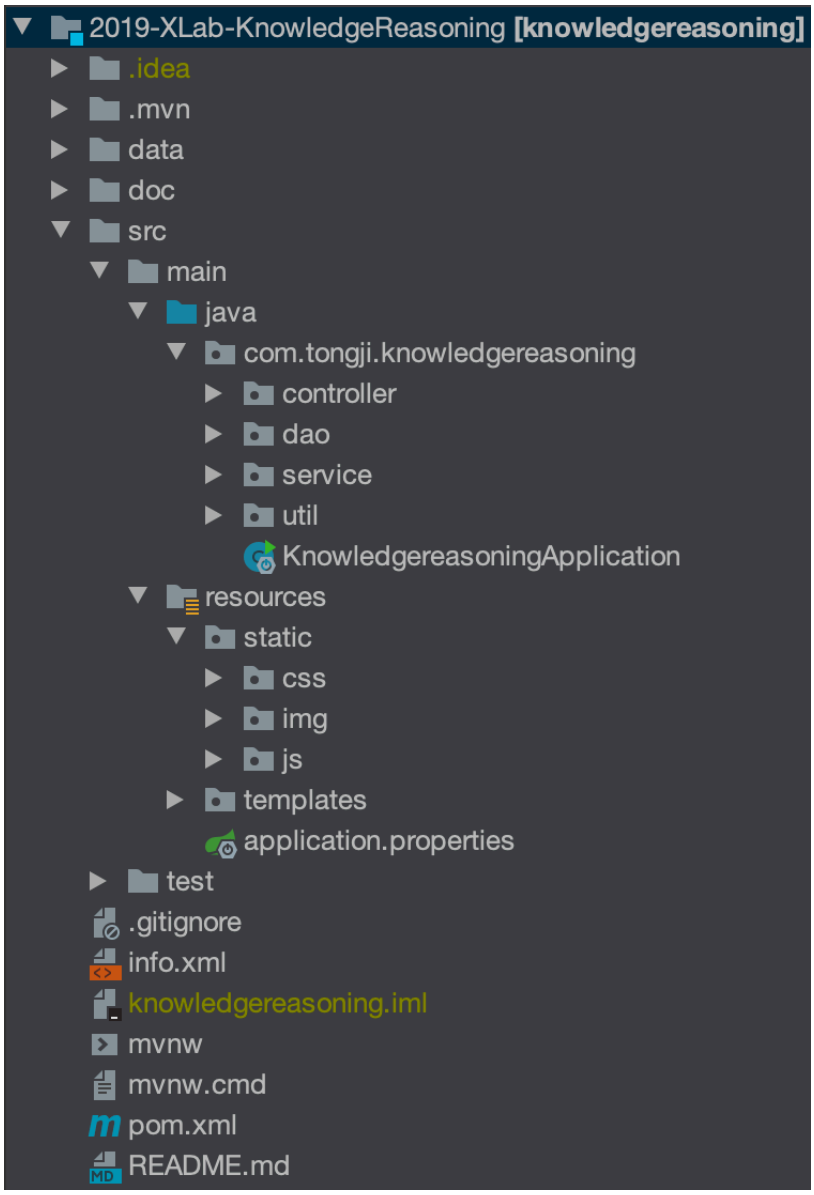
- **This project will do:**
    - Build the metadata layer.
    - Rule reasoning.
    - Ontology reasoning
    - Deep learning-based reasoning.
    - Import data from fuseki into neo4j.
- **This project won't do:**
    - Origin data built in knowledge graph.
    - Whether the failure is reasonable.

Involves tools

Framework

- **SpringBoot**
  - **Basic Introduction**

    SpringBoot is a new open source lightweight framework developed by the Pivotal team in 2013 and released its first release in April 2014. Based on the Spring4.0 design, it not only inherits some of the best features of the Spring framework, but also further simplifies the setup and development of Spring applications by simplifying configuration. In addition, by integrating a large number of frameworks, SpringBoot is able to solve the problems of conflicting versions of dependent packages and unstable references.
  - **Features**
    - Create stand-alone Spring applications
    - Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
    - Provide opinionated 'starter' dependencies to simplify your build configuration
    - Automatically configure Spring and 3rd party libraries whenever possible
    - Provide production-ready features such as metrics, health checks and externalized configuration
    - Absolutely no code generation and no requirement for XML configuration
  - **Project Architecture**



Database

- **Graph database: Neo4j**
  - **Basic Introduction**

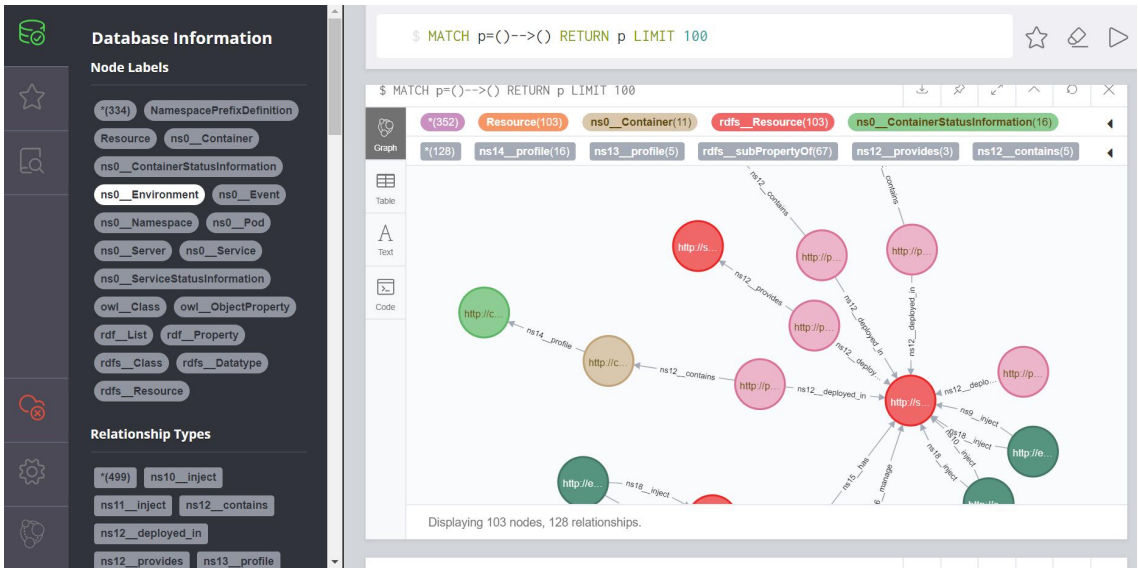
    Neo4j is a high-performance,NOSQL graphics database that stores structured data on the network rather than in tables. It is an embedded, disk-based Java persistence engine with full transactional characteristics, but it stores structured data on a network (mathematically called a graph) rather than in a table. Neo4j can also be seen as a high-performance graph engine with all the features of a mature database. Programmers work under an object-oriented, flexible network architecture rather than rigid, static tables -- but they can enjoy all the benefits of a fully transactional, enterprise-class database.

  - **Features**

    1. The mismatch of object relations makes it so difficult and laborious to squeeze the object-oriented "round objects" into the relational "square table" that it can be avoided.

2. The static, rigid, and inflexible nature of relational models makes it very difficult to change schemas to meet changing business needs. For the same reason, databases often lag behind when development teams want to apply agile software development.
3. Relational models are poorly suited to express semi-structured data -- and industry analysts and researchers agree that semi-structured data is the next big thing in information management.
4. Network is a very efficient data storage structure. It is no coincidence that the human brain is a vast network, and that the world wide web is similarly wired. Relational models can express networkoriented data, but they are weak in their ability to traverse the network and extract information.

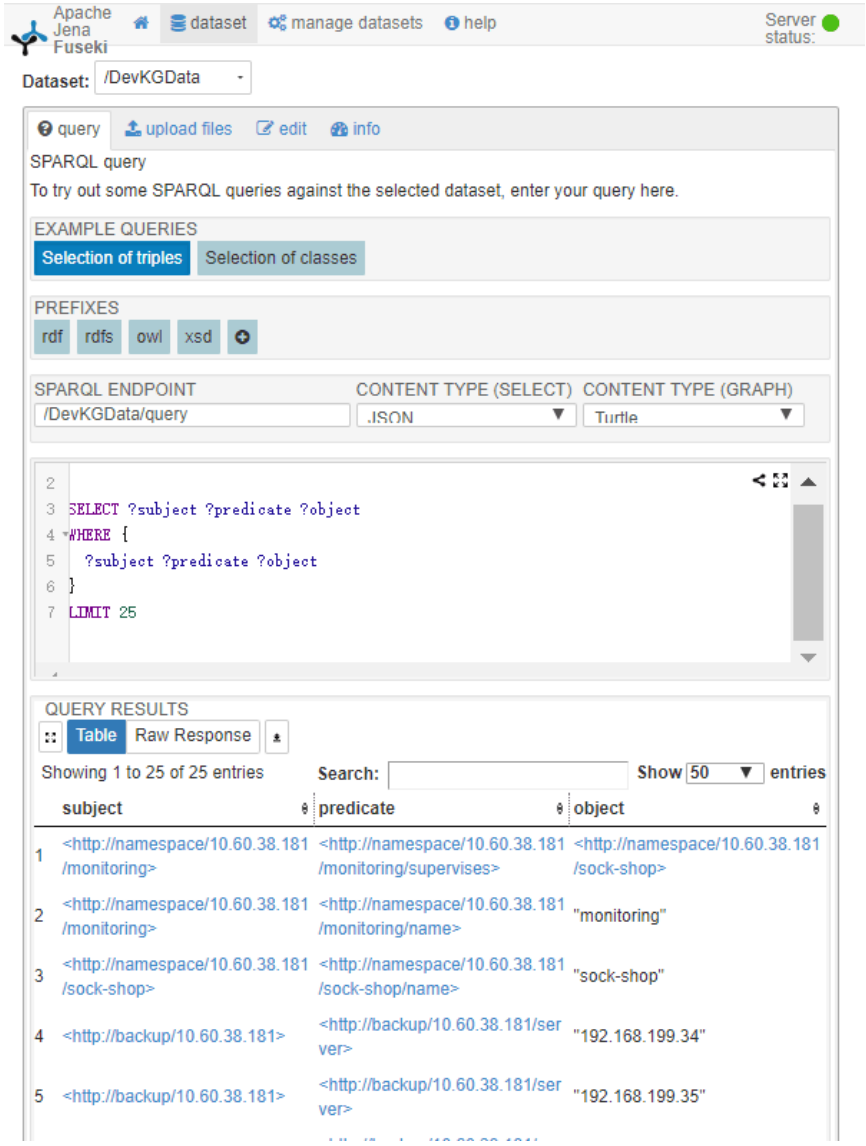
- **Graph Database for Knowledge Reasoning**



- **Knowledge Graph database: Fuseki**
  - **Basic Introduction**

    Apache Jena Fuseki is a SPARQL server. It can run as a operating system service, as a Java web application (WAR file), and as a standalone server. It provides security (using Apache Shiro) and has a user interface for server monitoring and administration.

  - **Knowledge Graph database for Knowledge Reasoning**



- **Query language: SPARQL**
  - **Basic Introduction**

    SPARQL(SPARQL Protocol and RDF Query Language), a Query Language and data acquisition Protocol developed for RDF, is defined for the RDF data model developed by W3C, but can be used for any information resource that can be expressed in RDF. The SPARQL protocol and RDF query language (SPARQL) officially became a W3C recommendation on January 15, 2008. SPARQL builds on previous RDF query languages, such as rdfDB, RDQL, and SeRQL, and has some valuable new features.
  - **Query Steps**

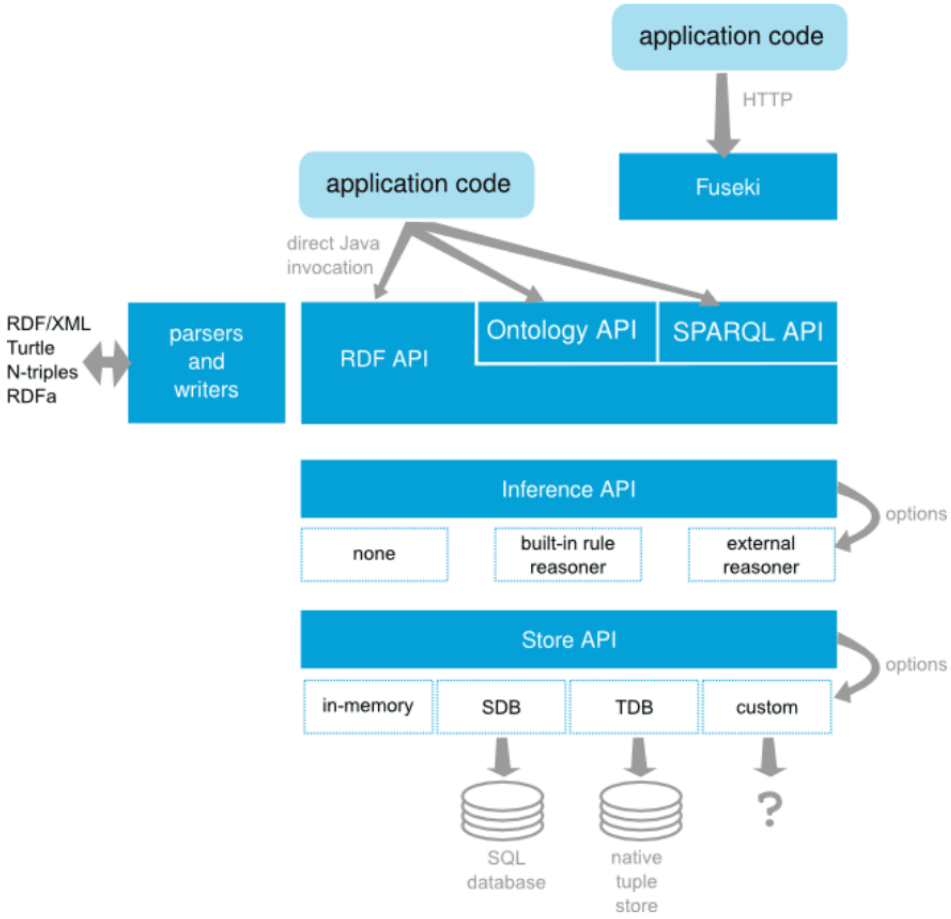
    1. Construct a query graph pattern, expressed as RDF with variables.
    2. Match to a subgraph that matches the specified graph pattern.
    3. Bind, bind the result to the variable corresponding to the query graph pattern.
  - **Example in Knowledge Reasoning**

```
PREFIX pods_rel: <http://pods/10.60.38.181/>
SELECT *
{
  <http://services/10.60.38.181/sock-shop/orders>  ^pods_rel:provides
/ (pods_rel:deployed_in* | pods_rel:contains* ) ?o .
}
```

Reasoning

- **jena**
  - **Basic Introduction**

    Apache Jena is an open source Java semantic web framework for linking data and building a semantic web. It can store RDF and RDFS type data. Apache Jena provides TDB, Rule Reasoner, and Fuseki components, of which TDB is the component used by Jena to store RDF type data, which belongs to the storage-level technology; Rule Reasoner can perform simple rule inference and support users to customize inference rules; Fuseki is provided by Jena SPARQL server, support SPARQL language for retrieval, can run efficiently on stand-alone and server side.
  - **Example**



- **prolog**

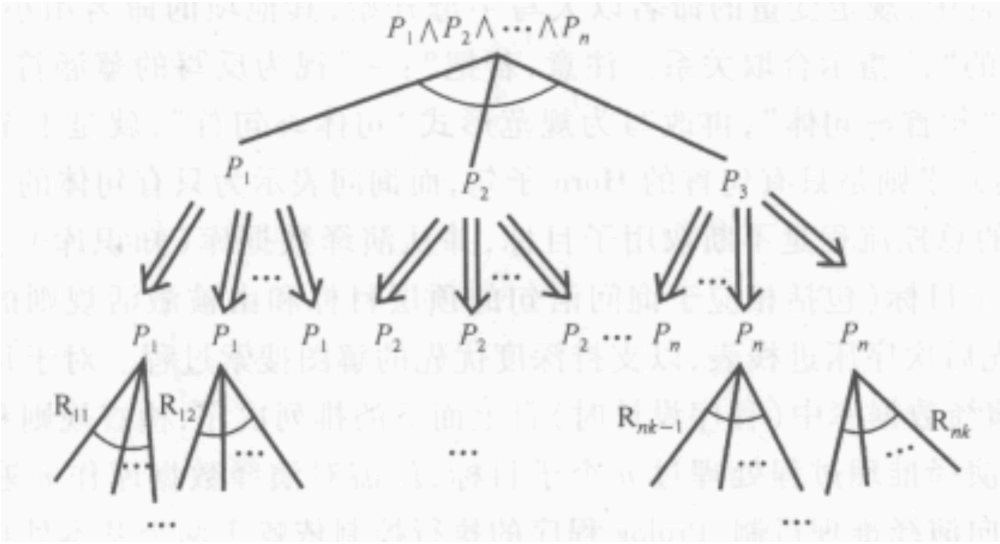
- **Basic Introduction**

  Prolog(Programming in logic) is a logical Programming language oriented to deductive reasoning

- **Features**

  1. The syntactic structure of the Prolog language is fairly simple, but descriptive.
  2. Prolog program has no specific running order, and the running order of the program is completely in accordance with mathematical logic derivation (digestion method). It's not up to the programmer.
  3. Prolog is a descriptive language that describes a problem in a specific way, and the computer automatically finds the answer to the problem.
  4. Prolog programs do not have control flow statements such as if, case, or for. In general, programmers don't need to know how a program is running, just how comprehensive the description of the program is. However, Prolog also provides some ways to control the flow of a program that are quite different from methods in other languages.

- **Example**



- **datalog**

  - **Basic Introduction**

    Datalog is a data query language designed to interact with large relational databases and has a syntax similar to that of Prolog. Just as SQL is only a specification, transact-sql and pl-sql are its concrete implementations; Datalog is also a specification, BDDBDDB, DES, OverLog, Deals, etc. All implement their own language according to the syntax of Datalog, so Datalog has no specific execution environment (e.g. Java for Java virtual machine, Prolog for SWI-Prolog).

  - **Similarities and differences with Prolog**

    The syntax for Datalog is a subset of the Prolog; But the semantics of Datalog are different from those of Prolog. The order in which facts and rules appear in a Prolog program determines the outcome of the execution. It is likely that the order of occurrence of the two rules will be reversed and the program will fall into an endless loop. The Datalog program does not require facts and rules to appear in the same

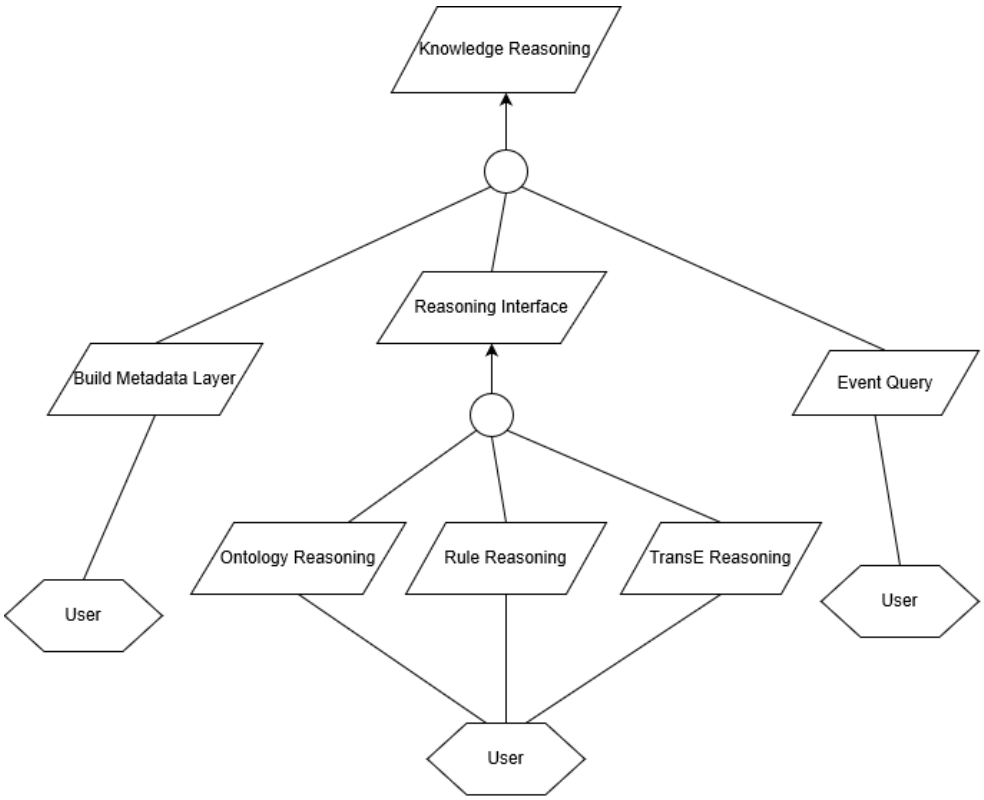
order.

- **Example**

  `Relationship`: Person(Name,Age,Sex) Parent(Father,Mother) `Father (X, Y) : - the Person X, _, m), the Parent (X, Y)` Where Person and Parent belong to the basis DB(Extensional Datenbank:EDB), and father belongs to intentional Datenbank(IDB, that is, the view pushed out).

Rquirement Analysis

System KAOS Diagram



Use Case Analysis

Use Case: *Building the Metadata Layer*

Description

Build the metadata layer to improve the structure of knowledge graph. The metadata layer contains data layer, pattern layer and logic layer. And refactoring of the metadata is also needed in building the metadata layer use case.

Participants

First of all, users who use our knowledge reasoning system should building the metadata layer.

Main system of our knowledge reasoning project will attend building the metadata layer use case.

Pre-condition

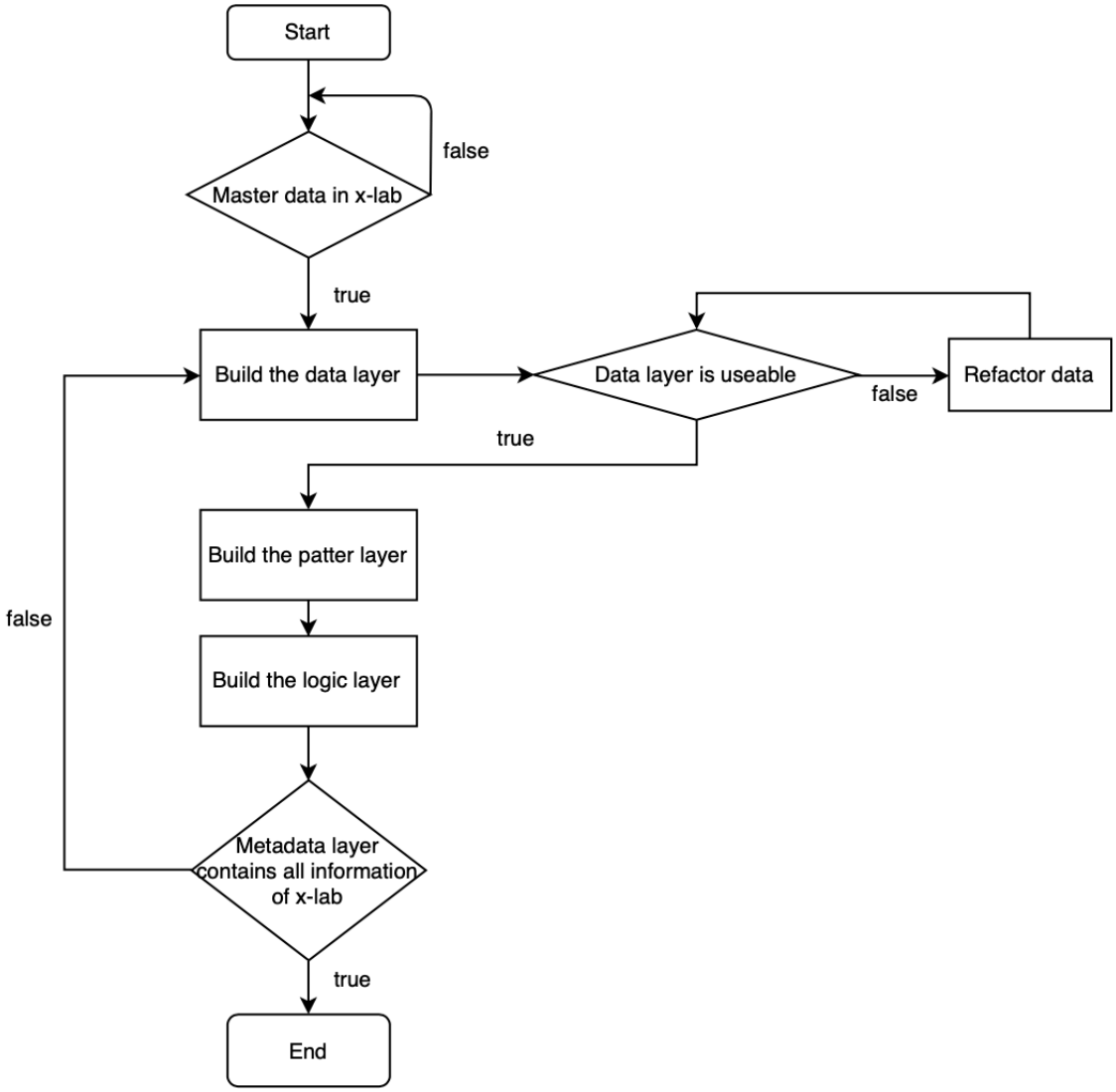
- Building knowledge graph project is finished.
- All network hardware and software in x-lab are clear.

Post-condition

- The metadata layer will complete.
- Knowledge graph will be more structured.
- Use the metadata layer to do reasoning.

Process Flow

1. Master all the hardware and software in x-lab.
2. Building the data layer.
    - Refactor of the data built in the data layer
3. Building the pattern layer.
4. Building the logic layer.
5. Repeated iteration of 2~4.
6. Reviewing throughout the process.



Exceptions

- We should find some way to confirm metadata layer is completed which means that it contain all information of x-lab.

> Solution: By randomly sampling the database and knowledge map, our ontology layer contains samples whose reliability meets the project's general criteria.

- The raw data layer is not always useable.

> According to the structure of the data, we use several ways to refactor the raw data in order to satisfy the data requirement in the knowledge reasoning project.

Use Case: *Ontology Reasoning*

Description

Ontology reasoning is carried out on the knowledge graph and the newly discovered knowledge is written into the database. This use case plays a significant role in perfect knowledge in x-lab because we cannot find all knowledge artificially.

Participants

Users who use our knowledge reasoning system can use ontology reasoning to perfect his or her own knowledge database.

Pre-condition

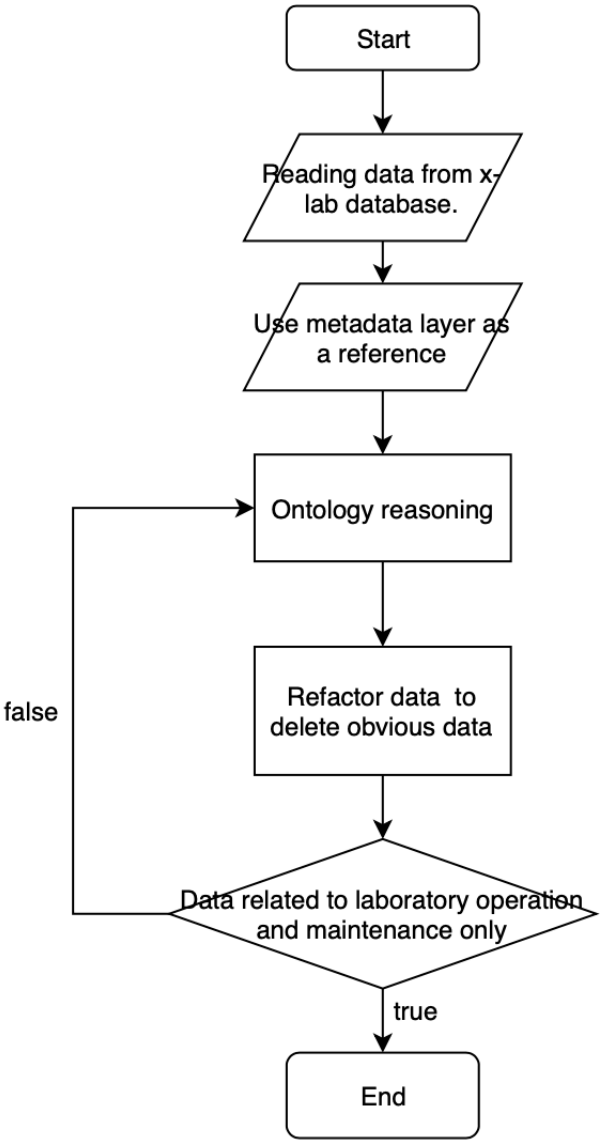
- The metadata layer is built successfully.
- The knowledge is imported from database in x-lab.

Post-condition

- Newly discovered knowledge will be written into the database.

Process Flow

1. Reading data from x-lab database.
2. Use metadata layer as a reference.
3. Ontological reasoning of the operation and maintenance data.
4. Refactor to the data to delete obvious data and retain data related to laboratory operation and maintenance only.



Exceptions

- Ontology reasoning yields a lot of new data that is obvious and not useful for later use.

  > Refactor all the data after ontology reasoning. Only data related to laboratory operation and maintenance will be retained.

Use Case: *Rule Reasoning*

Description

Rule reasoning is carried out on the knowledge graph and the reasoning result based on rules will return to users. This use case plays a significant role in perfect knowledge in x-lab because we can use it to find or write new knowledge to the database.

Participants

Users who use our knowledge reasoning system.

Pre-condition

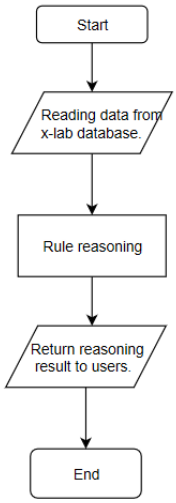
- The knowledge is imported from database in x-lab.
- User enters the rules.

Post-condition

- Konwledge based on rules will be find and return to users.

Process Flow

1. User enters rules.
2. Reading data from x-lab database.
3. Start rule reasoning based on the rules user entered.
4. Write reasoning result into database.
5. Return rule reasoning result to users.



Exceptions

- Rule reasoning won't get any knowledge if the rules user entered are incorrectly.

Use Case: *Deep Learning Reasoning——TransE Reasoning*

Description

TransE reasoning is carried out on the knowledge graph and the newly discovered knowledge is written into the database.This usecase plays a significant role in perfect knowledge in x-lab because we cannot find all knowledge artificially.

Participants

Users who use our knowledge reasoning system.

Pre-condition

- The knowledge is imported from database in x-lab.

Post-condition

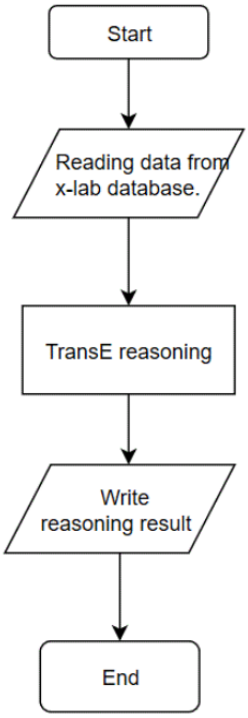
- Newly discovered knowledge will be written into the database.

Process Flow

1. Reading data from x-lab database.
2. Start transe reasoning to get new kowledge.
3. Write the new data to the database.

- TransE reasoning yields some new data that is incorrectly.

> Modify train parameters to get different models, valid and test these models to get the best model.



Use Case: *Query Event*

Description

Query Event is carried out on the knowledge graph and the query result will return to users. This use case plays a significant role in perfect knowledge in x-lab because we can use it to find failure components.

Participants

Users who use our knowledge reasoning system can use the query usecase to get the failure components of the knowledge database.

Pre-condition

- The knowledge is imported from database in x-lab.

Post-condition

- Failure component will be find and return to users.

Process Flow

1. User enters query statement.
2. Reading data from x-lab database.
3. Start query operation.
4. Return query result to users.