DESIGN AND IMPLEMENTATION OF A FILTER ENGINE FOR SEMANTIC WEB DOCUMENTS

Takanori Kozuka and Annika Hinze
Department of Computer Science, University of Waikato
{tk27, a.hinze}@cs.waikato.ac.nz

Abstract

This report describes our project that addresses the challenge of changes in the semantic web. Some studies have already been done for the so-called adaptive semantic web, such as applying inferring rules. In this study, we apply the technology of Event Notification System (ENS). Treating changes as events, we developed a notification system for such events.

1. Introduction

The Semantic Web is a new concept for extracting information from the Web pages in computer-readable form. The Semantic Web is supported by three key technologies: RDF, RDF schema, and Ontologies. These technologies are used to describe the information for this purpose, but they mainly function in static situation.
The problem this paper focuses on is how to detect and reflect the dynamic changes of information in the Web world. In the first chapter, we introduce the brief outline of the Semantic Web (Section 1.1), and the example scenario (Section 1.2). Then we discuss about the problem of change management in the Semantic Web in detail.

1.1. The Semantic web

The Semantic Web is not something completely new, but is an application of the World Wide Web (WWW). In this section, we highlight the advantage of the Semantic Web over the WWW.

The World Wide Web
Internet opened the new era of information society. People have access to information from office/home with just turning on the computer. The new media changed people's life style, and opened the opportunity to publish documents to general public. Many businesses position the Web as vital PR media, and provide the information through the Web. Documents on the Web are designed for the purpose of each publisher, thus each Web page may have a unique appearance. Consequently, Web pages generally are suitable for human to read, but not suitable for computer to extract meaningful information automatically. Data on Web are typically unstructured, because they are not intended to use as a database. There are lots of valuable information hidden in the ocean of Web

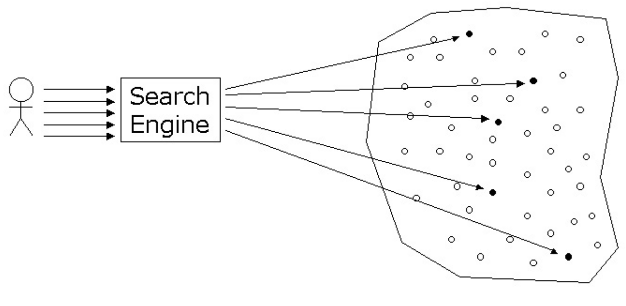


Figure 1 Search Engine Access in the WWW

pages. But, when people need to find a set of interconnected information, for example, information about a particular article (its abstract, full-text, the rating of the article, or information about author and publisher), users often have to visit each Web site individually to collect the necessary information. For this purpose of information seeking, people use search engines. (see Figure 1) The idea of the Semantic Web is based on people's demand to gain the inter-connected information in a meaningful way.

The Semantic Web

Instead of forcing users to access to Web sites one by one, the Semantic Web provides the single interface, called agent, enabling users to register details of requirements that users need. Then agent visits relevant Websites, and collects the necessary information for users. (see Figure 2)

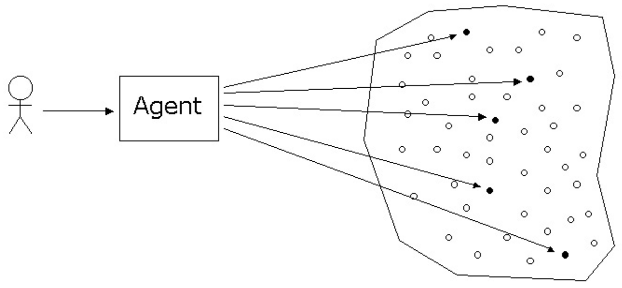


Figure 2 Data Access in the Semantic Web

The Semantic Web is defined formally as an "*extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation*" [12]. The intention of the Semantic Web is to interconnect documents on various Web sites, and to extract information from the documents. Terms used in the documents possibly have different meanings based on the publishers' purpose. Therefore they need to be interpreted and to be transformed into structured database, which would provide meaningful contents for individual needs. This frees users from the task visiting an enormous number of Web sites to find the information that they are looking for. In general, typical user does not visit more than one or two pages out of the search results extracted by search engine. Thus the potentially relevant information easily can be overlooked. In contrast, the Semantic Web collects all the relevant data as long as it is presented in a certain format. This feature enables users to discover information they need more easily and in scalable manner.

In the Semantic Web, information can be selected from sources and provided to the users by *agents* based on the users' interest or information need. (See Figure 2)

In a short summary, the Semantic Web;

- collects data from various different sources,
- interprets the meaning of the data,
- connects the data into a structured database,
- extracts knowledge (or useful information) from the database, and
- provides meaningful contents to users
- through a single interface (agent)

1.2. Scenario

This section describes a scenario that allows us to identify open problem in the Semantic Web.

One afternoon, John Smith received the phone call from his friend, Jane. She seemed to have the spy ware in her computer. The virus protection program warned the infection by the spy ware, but does not have the vaccine for that. She did not have any recognized problem yet, but she wanted to exterminate it and asked John how to do it.

1. *Search (Profiling)*

John wrote down the message by the virus protection program, and typed it into his terminal of the Semantic Web. Soon after the terminal returned the candidate spy wares that she experienced. There are so many candidates, as the warning message was too general to identify the particular target. John decided to focus on the latest ones, for which the vaccine is available. John changed the conditions, and typed it to the terminal again. Under the new conditions, the four spy wares were found, that were reported within the last two months. One of those is just found few days ago, and the vaccine is only available from one company. The virus protection program that Jane is using has not developed the vaccine for that.

John set the system to carry on further search to the system, and explained to Jane the situation. Jane downloaded three vaccine programmes, which are already available. However upon the connection to the Internet, her system still kept warning the existence of the spy ware.

2. *Notification*

Two days later, John found two notices from the system. One tells the vaccine became available for the last one spy ware. The other notice is about one vaccine that Jane has downloaded two days ago. It reported the applicable OS version by the vaccine. Unfortunately that tells the vaccine is not valid for Jane’s OS version. John explained to Jane, and she got the new vaccine, but the problem still remains, so they had to wait until the new version would be developed.

Few days later, John found another notice, which tells the development of the vaccine for Jane’s OS version. After downloading the new vaccine, the problem solved. So, John cancelled further notice about this spy ware from the Semantic Web.

In this scenario, John could not get the information that he needs in the first query. In such a case, the current Semantic Web requires users to access the system again and again until the information would become available. Or, realistically, users would give up, and approach to another channel to search the information.

The scenario also introduced the case of updating notification. Semantic Web so far has limitations in handling updating/deleting of information. Once information was delivered to users, users would not be aware of any updating/deletion until the users would perform the same query again. Even if the same search was performed, it would not be easy to find the deletion of particular information.

1.3. Problem Description

The idea of Semantic Web gave us the great opportunity to utilise the huge amount of data on the World Wide Web. The data are generated and added to Web pages every second extending the size of data sources. This extension of data sources is the backbone for Semantic Web, but it also takes the risk of the out-of-dating into the information gained from Web pages. What if a change would be made on the information on the page after the user accessed that page? What if the publisher would add new pages under the existing page? Not only the insertion of new data, existing data would possibly be updated or deleted from the pages that user referred to. These changes happen often in the Web world. And all these changes may affect users, as they may contain relevant information for them.

However, so far the Semantic Web extracts the information statically, and does not take this dynamic change into account after retrieval from the Web pages. Its main concern is to extract the data at the certain point of time. Once the data are extracted from Web, the system would not dynamically reflect any changes on those pages. Thus, the adaptability to such change is the next key issue for the Semantic Web. This dissertation proposes a first help towards the adaptability of the Semantic Web.

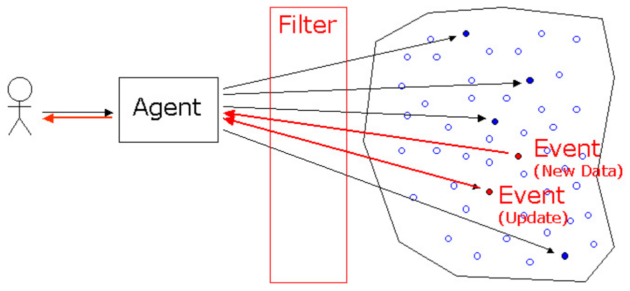


Figure 3 The Adaptive Semantic Web

As we will see in the next Section, RDF/DRFS technologies and their XML like syntax is one asset of the Semantic Web. These technologies describe resources and relationships between them, enabling users to find the relevant information. But the problem is, they work statically: existing query languages do not react to the dynamic change of RDF documents. In this project, we focus on how to identify and filter changes in the Semantic Web, which functions as namely ***The Adaptive Semantic Web***[1]. (Figure 3)

We propose to develop an Event Notification System (ENS) for the Adaptive Semantic Web. In this new function, users will receive notification about changes, which are relevant to the users' interests. The concept of Event Notification System will be discussed in detail in Section 2.2.
We will now first identify issues for event notification in the Semantic Web and then specify the focus of their work.

This concept of change management in the Semantic Web leads to the following research issues:

(1) Event type
An Event Notification System treats certain type of changes as events. There are many different types of changes. Changes can be the new occurrence of a document or the change of the document. Changes can refer to time. Or, changes can be the increase/decrease of value. This type of events is, for example, the body temperature. Changes can be primitive type (means, occur independently) or composite type (i.e. particular sequence of changes, for example, B occurs after A). For details on event types in our system, see Section 3.3. In this project, we only focus on the ***primitive*** types of events: the change of the contents in Web pages.

(2) Event Observation
Events in the Semantic Web are generated by Web publishers. The question is how to observe the events occurrence. It is not practical to check through Web pages in the world. Therefore a strategy of how to observe the events in the most efficient way needs to be developed in this project. We focus on events occurring in a given database of Semantic Web documents.

(4) Query Language and Filtering
The document format of the Semantic Web is the so-called ***RDF*** (Resource Description Framework). Currently no query language is available to catch changes of RDF documents. So, the initial issue here is "how to detect the changes".

[1] The name "Adaptive Semantic Web" has been initially introduced by Peter Dolog et al. [15]

Queries will be interpreted and filter the irrelevant events, leaving only the relevant ones for given queries. We need to find the most efficient approach for the filtering. The filter language is influenced by the database type used to store the RDF documents. In this project, we use a trigger approach to detect and filter events.

(3) Data Type / Storage
As the document format of the Semantic Web is RDF, which is the application of XML, it is one option to directly store the document in XML. Considering the Semantic Web handles the large amount of documents from World Wide Web, it is not scalable to store the documents in a file system. One option to solve the scalability issue is to convert documents into database entries, which is generally more efficient. If we transform documents into entries in a database, we have several options for the storage: relational database, or object relational database. In general, Object Relational Database is considered to be faster in processing due to its ability to handle data recursively. The selection of the database type needs to be considered together with the way to observe and filter events, as the storage system would influence these methods. For our project, we use a relational database (Oracle) and the RDF storage system ***Sesame***.

In this report, we focus on the development of a ***filter engine for the semantic web documents***, which detects events, filters out irrelevant events, and stores the relevant events for notification. Profiling of users, query parser, and notifier of events are remained for further research.

The remainder of this report is structured as follows: In Section 2, we discuss the technical background of the Semantic Web and this dissertation in more details, which are foundation of our implementation. We also briefly discuss the related work. In Section 3, we discuss the design of system. We then discuss our implementation in Section 4. And finally in Section 5, we summarize the achievement of this project, and review the limitation and future work to be done.

2. Background

In this section, we discuss the key technologies of the Semantic Web and our Event Notification System. The search engines and search agents in the Semantic Web have similar function; both accept the key queries such as word/sentences and return the relevant data for user. However their architectures are very different. The search Engine focuses on the keywords, and returns the Web pages, which contain keywords in metadata (header part) or in their contents (body part). It does not recognize the meaning of words, but return documents if they contain keywords.
The Semantic Web, on the other hand, more focuses on the meaning of data. It uses RDF documents, which describe the content and relation of Web pages. Structure of RDF documents is provided by the ***Resource Description Framework*** (RDF), ***RDF Schema*** (RDFS), and ***Ontologies***. Applying these technologies, RDF documents describes the relations of ***resources***. Because the documents are written using a certain schema, the documents are human-readable and can also be processed by a machine.
The Semantic Web so far offers static search of data. Our main interest is to design the filter engine of an Event Notification System for the Semantic Web data in order to extract the dynamic changes of Web documents which are relevant to users. In this Section, we first discuss the data handling issues in the Semantic Web introducing its key technologies, and then we present a general introduction of the Event Notification Systems.

2.1. Data Issues in the Semantic Web

Three layers of the Semantic Web are RDF, RDF Schema, and Ontologies. Web pages data are collected from the WWW, interpreted, and connected to extract the meaningful information. Then they are stored for future queries. Especially storage is important for our Event Notification System. In this section, we introduce the RDF, RDF Schema, and Ontologies, and then discuss storage and query issues.

2.1.1. Resource Description Framework and RDF Schema

RDF documents can be described using XML syntax, triples, or a graph representation. RDF documents are (of course) written using RDF and RDFS, but their structure is more expressively described in triples and graph. Hence we first focus on the triple and graph representation to study the concepts of RDF documents. And then we discuss XML representation to develop the discussion to the architectural issue.

2.1.1.1. Conceptual Framework of the Semantic Web

First, we introduce the components of triples and their relation, which are also used in the graph representation. Next, we show how to describe triples in graph format. Then we discuss the practical approach to build the relations of resources and extract the meaning from them.

(A) TRIPLE
The RDF documents are conceptually expressed in the form of ***sentences***. A *sentence* has three components: ***subject***, ***predicate***, and ***object***. All sentences contain these three components. As they are structured in three components, it is also called ***triples***. Dividing sentences into triples, it becomes possible to store RDF documents into database. Each component is identified by *Uniform Resource Identifier* (URI). [13]
Consider the following example: There is a person whose name is John. He is represented by *Uniform Resource Identifier* (URI) http://www.example1.org.nz/, which is his own Website. He wrote the report with the title of 'The Semantic Web'. This document is uploaded in the Website whose URI is http://www.example2.co.nz/. The relation between 'John' and 'The Semantic Web' is stated in the web page whose URI is http://www.example3.co.nz/. This relation can be described by the following sentence:

	Subject	Predicate	Object
Sentence	John	hasWritten	'The Semantic Web'
URI	http://www.example1.org.nz	http://www.example3.co.nz	http://www.example2.co.nz

Suppose again John is the student of the University of Waikato, whose Web address is http://www.example4.ac.nz/. This relation is also stated in the web page whose address is http://www.example3.co.nz/. Then the relation can be written as follows:

	Subject	Predicate	Object
Sentence	The University of Waikato	hasStudent	John
URI	http://www.example4.ac.nz	http://www.example3.co.nz	http://www.example1.org.nz

Predicate is like 'verb' used in common English sentence. Predicate bridges any two entities, which have URIs, and makes a sentence. Two entities, which are bridged by predicate, are called

subject and object. The meanings of subject and object are much like English grammar term. They are not necessary to be Web pages. Any entities, which have URI, can be subject/object. Subject and object are also called ***resource***. There are two types of resources: ***property*** and ***class***. Predicate has to be a property, but subject and object can be either property or class.
It is important to give the URI to the resources. In the Internet world, everybody can write anything. Two different people may use same term in different meanings. Or two different terms can be used to express same meaning. For example, consider the term "create". The religious people use this term to express the creation of the world by the God, while the artists mean drawing the picture. Or, the office administrator would use that term for generating the business report. Connecting the resources to a particular URI, we can avoid such confusion, and define the meaning of the resource clearly.

(B) GRAPH

Triples can also be expressed in the ***graph*** format. In the graph, classes (subjects and objects) are represented by oval shape and property by arrows. (See Figure 4) Attributes of classes are expressed as ***literal***, which are represented by rectangles. Literals are the descriptive details of classes, such as the title of the document, creation date, number of pages, name of the person, etc.

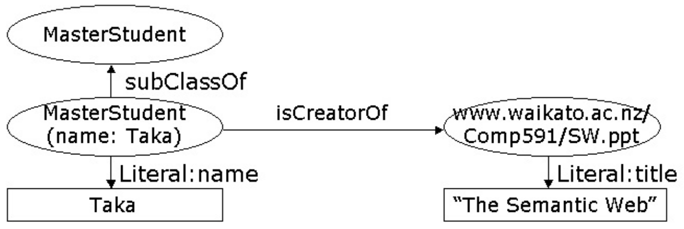


Figure 4 Graph

Applying this simple grammar, we can make sentences one after another with connecting given resources. Figure 5 shows the very simple relationships between document group and writer group. Two properties used in Figure 5, which are subPropertyOf and subClassOf, show the hierarchy of resources. They are defined by RDF Schema, and a key to express the relationships between resources. We will go into detail about RDF Schema later. The other three properties (hasAuthor, hasCreated, hasWritten) are user-defined properties.

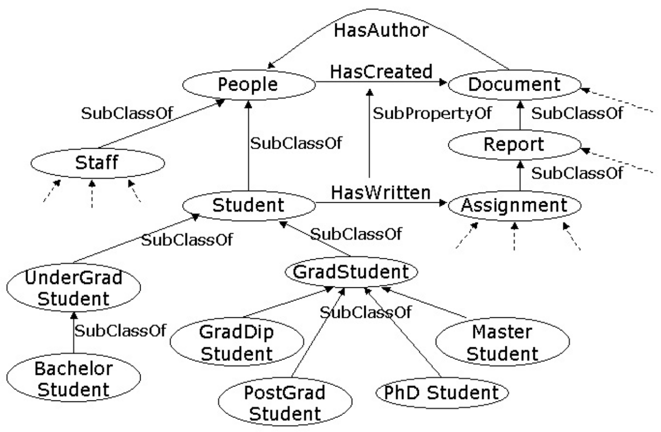


Figure 5 Graph with many resources

Classes are connected by the arrows (properties). There are only 13 classes and 14 properties in this case. But even such simple relations, the diagram is already complicated enough to confuse the reader. Therefore the new issue arises in here: how can we extract the sentences, which contain a particular meaning? We need a certain level of abstraction of resources to extract the meaningful information out of complicated resource connection. Now we talk about the topic of how to organize resources.

(C) HOW TO ORGANIZE RESOURCES

To organize the relations of resources, we apply two dimensions on resources. One dimension is vertical: the hierarchy of resources. More abstracted resource locates in higher position of hierarchy. The other dimension is horizontal: other than the hierarchical relations. We map resources with these two dimensions. Each mapped resources are connected to URI, which identify address of them. We organize resources in three steps; (1) identify resources, (2) abstract one resource to another, and (3) relate them each other.

Identify Resources: (Where is it?)

To identify the resources, the Semantic Web refers the location (URI) of them. In the Semantic Web, identification refers to the resources knowing "where it is". 'Identifying' means to describe resource's location with the combination of the namespace and their local name in the form of URI. (See Figure 6)

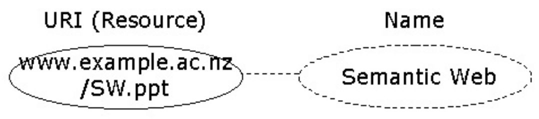


Figure 6 Identify Resources

Abstract Resources: (What is it?)

In abstraction, a resource should be classified into two different categories: class or property. (See Figure 7) The publishers of RDF documents make their own rules about how to sub-classify resources. For example, the document with title 'The Semantic Web' can be a sub-class of 'assignment report'. 'Assignment report' can be the sub-class of the 'report', which is again the sub-class of 'text document'. Then the 'text document' can be the sub-class of 'document', which is type 'Class'. In same manner, properties can be split into the sub-properties, and made into hierarchy. 'Edit' can be the sub-property of 'write'. And 'write' can be the sub-property of 'create', which is of type 'Property'. Through the abstraction, resources are semantically structured into the tree. Then the resources, whose addresses spread whole over the world, become available to search from the semantic tree.

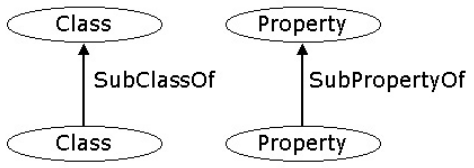


Figure 7 Abstract Resources

Relate Resources: (How do they relate?)

After resources are identified appropriately, then we can make sentences using those resources. (Figure 8) Connecting two resources (subject and object) with a property (predicate), we can make sentence in any level of hierarchy. That means, we can say either "Person – Created - Document" or "John - Wrote – Assignment report". 'Relating' can be done earlier than abstraction. The combination of relation and abstraction makes it possible to extract the meaningful information (i.e. knowledge representation).

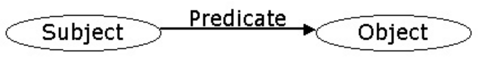


Figure 8 Relate Resources

Web information are sorted in these three steps (identify, abstract, relate), and described in the form of RDF documents. RDF documents are actually supported by three technologies: RDF, RDF Schema, and Ontologies. In Section 2.1.1.2, we discuss technical issues based on the conceptual framework.

2.1.1.2. Architectural Framework of the Semantic Web

The latest version of RDF and RDF Schema recommended by World Wide Web Consortium (W3C) is introduced in “RDF Vocabulary Description Language 1.0: RDF Schema” [6]. Vocabulary of RDF/RDFS contains 29 resources: 13 classes and 16 properties. The vocabulary shown in the following table is ones that we mainly deal with in this project.

	Class	Property
RDF	Property	type
RDFS	Resource, Literal, Class	subClassOf, subPropertyOf, domain, range

Syntax of RDF documents is extended from XML. Similarly to XML, user can define vocabulary of RDF documents extending W3C recommendation. A sample of RDF document in XML is introduced in Figure 9. The graph representation of Figure 9 is shown in Figure 13 with some extension. Both RDF and RDF Schema are used to write this RDF documents. Developing the vocabularies of users’ own, the RDF documents yield the meaning on the sentences.

(A) RESOURCE DESCRIPTION FRAMEWORK
Resource Description Framework (RDF) is an infrastructure that enables the encoding, exchange and reuse of structured metadata [1]. It is used to identify the resources showing the type of resource (whether class or property) and the URI of the resource. W3C provides the good introductory guide of how to write the RDF documents [7]. We see the simple example in here. The following sentence says, “Resource ‘People’ is a ‘class’, and is located in the address of `http://www.example.ac.nz/terms`”.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.example.ac.nz/terms#People">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
.....
```

In this example, ‘rdf:’ is a pre-defined namespace which is defined at "`http://www.w3.org/1999/02/22-rdf-syntax-ns`”. Terms ‘Description’, ‘about’, ‘type’, and resource are defined at the Web page represented by namespace ‘rdf’. Term ‘People’ is defined by user at "`http://www.example.ac.nz/terms`".

The following sentence says same thing as the above. The abbreviation is used in this case. Where many terms are defined in a page, this way is efficient as it shortens the sentence. In the rest of this report, we use the abbreviation.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xml:base="http://www.example.ac.nz/terms">
  <rdf:Description rdf:ID="People">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
.....
```

```
<?xml version="1.0"?>
<rdf:RDF xml:lang="en"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xml:base="http://www.waikato.ac.nz/tk27/terms">
  <rdf:Description rdf:ID="People">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Student">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#People"/>
  </rdf:Description>
  <rdf:Description rdf:ID="Document">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="WebPage">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#Document"/>
  </rdf:Description>
  <rdf:Description rdf:ID="hasCreated">
    <rdf:type rdf:resource="http://www.w3.org/...22-rdf-syntax-ns#Property"/>
    <rdfs:domain rdf:resource="#People"/>
    <rdfs:range rdf:resource="#Document"/>
  </rdf:Description>
  <rdf:Description rdf:ID="hasWritten">
    <rdf:type rdf:resource="http://www.w3.org/...22-rdf-syntax-ns#Property"/>
    <rdfs:subPropertyOf rdf:resource="#hasCreated"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 9 RDF Document

(B) RDF SCHEMA

RDF Schema (RDFS) is a semantic extension of RDF. It provides mechanisms for describing groups of related resources and the relationships between these resources. [6] It expresses the relations of class and property. Term ‘subClassOf’, which is defined by RDFS, is used to express the hierarchical relation of the classes. For example, where the ‘Student’ class is the sub class of the ‘People’ class, this relation is described as:

```
  <rdf:Description rdf:ID="Student">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:subClassOf rdf:resource="#People"/>
  </rdf:Description>
.....
```

Similarly, the hierarchy of properties is expressed by the term ‘subPropertyOf’. For example, the following sentence means ’hasWritten’ property is the sub-property of ‘hasCreated’”.

```
  <rdf:Description rdf:ID="hasWritten">
    <rdf:type rdf:resource="http://www.w3.org/...22-rdf-syntax-ns#Property"/>
    <rdfs:subPropertyOf rdf:resource="#hasCreated"/>
    </rdf:Description>
.....
```

RDFS is used to describe not only the hierarchical relation. It applies to all kind of sentences. Conceptually, sentences are structured with three components: subject, predicate, and object as introduced in Section 2.1.1.1. More precisely, this relation is described in RDF document using five components: two classes (subject and object), property, domain, and range. The terms ‘Domain’ and ‘Range’ are defined by RDF Schema. Domain represents the relationship between subject and predicate, and range represents the relationship between object and predicate. These relations between classes and properties are, again, supported with triple structure (i.e. Property – Domain – Class or Property – Range - Class), and property bridges two Classes in the middle. (Figure 10)

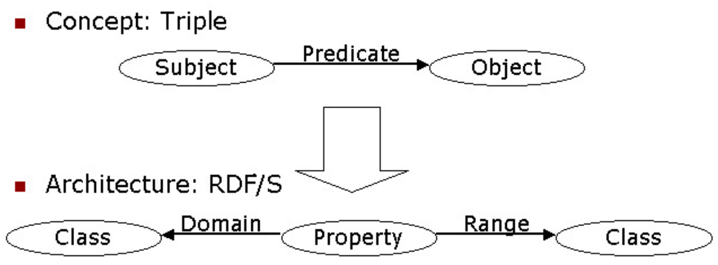


Figure 10

Splitting a sentence into two triples, the process of sentence-making becomes more flexible. When Web pages are created, they are not described as the part of the sentence initially. Until somebody publishes the document to connect resources, they are not a part of sentence. In the other case, resources, which are part of sentence, may be deleted. In such case, remainder of resources would be no longer a part of sentence. Building subject-predicate-object relation with five components, construction/destruction of sentences would become much more adaptable to the real-world situation.
Please note, 'Domain', 'Range', 'subClassOf', and 'subPropertyOf' are typed as 'Property'. They are used to represent the hierarchical relation and sentence relation of two resources. It is confusing to call them as property as it easily mixes up with properties other than these four properties. So, we call them as 'relational property' in this report.

2.1.2. Ontology

Ontologies deal with equivalency of resources. What person 'A' calls "create" can have the same meaning as when person 'B' says "write".

Looking back at Figure 4, there are two relations, which have same meaning, are drawn on the diagram. One is [Document – hasAuthor – People], and the other is [People – hasCreated – Document]. The order of subject and object is inversed, and classes are bridged by different properties, but the meaning of sentences is the same. Sometimes, even the sentence order is the same: such example is [People – isAuthorOf – Document]. (See Figure 11) When the user is searching for the document written by People, all these sentences provide the valid result 'Document'. In other case, the schema A and schema B may use the different terms to express the same meaning. For example, "People" in schema 'A' can be same thing as "User" in schema 'B'. Ontology describes the relationship among resources (e.g. if 'A' then 'B', etc.), and clarifies equality of two classes or interprets the multiple values of properties. Two technologies to support ontology are available: ***Web Ontology Language*** (OWL), which is W3C recommended [11] and is based on ***DARPA Agent Markup Language*** (DAML), and ***Ontology Inference Layer*** (OIL). [14]
In this report, we do not go further for this topic. Our main focus is to create the filter engine to notify about changes. Documents using Ontologies will be left for further study.

	subject	**predicate**	**object**
1	Document	hasAuthor	People
2	People	hasCreated	Document
3	People	isAuthorOf	Document

Figure 11 Equivalent sentences

2.1.3. Storage

RDF documents for the Semantic Web are usually stored in a storage system. After collecting the RDF documents, the system builds the database including the documents. RDF document can be stored in XML, triples, or graphs. The storage system can therefore use an XML Database, a *Relational Database* (RDB), or an *Object-Relational Database* (ORDB). Considering RDF in triple format, RDB or ORDB would be used.

For our adaptive Semantic Web, RDB and ORDB (for example, Oracle or SQL 2000) have strong advantage. They have trigger function, which enable to manipulate data upon insertion or deletion. This function is considered to be quite useful to build the filtering / notification system.

ORDB is similar to Relational Database. Object is closely equivalent to Entity of RDB. ORDB is known to have advantage in processing speed of triples comparing to the RDB. [5] Since the Semantic Web has hierarchy, the system needs to access to the database until reaching the end of hierarchy (either highest or lowest). In Object-Oriented programming, each object is directly connected to the relative resources. Its simple structure enables the system to process faster. (Figure 12)

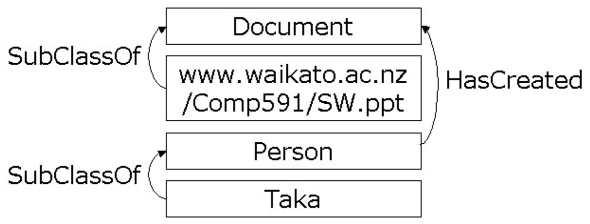


Figure 12 Object Relational Database

We employed Oracle for our storage system. Oracle is adaptable either to RDB or ORDB. Despite of the advantage of faster processing speed in ORDB, we actually used Oracle as RDB for our implementation. As we will discuss later, we used *Sesame* as platform. Sesame used to provide two different schemas: RDB-based and ORDB-based. So, we initially intended to compare the performance of RDB to ORDB. But ORDB-based schema is no longer available in the latest version. So, we justified the storage system to the currently available one (i.e. RDB).

2.1.4. Query

If we give the author name and topic of article to the search engine as keyword, it will return any documents, which contains the given keywords. Instead, the Semantic Web returns the article with the given topic and author. So, the returned documents are expected to be more relevant than the one from search engine. Moreover, the Semantic Web would return even the data which does not contain the given keywords, but relevant to the topic. Suppose the user would search for the 'report about the Semantic Web written by John who is a student', and we do not have exact match of this domain/range relation. However the higher level resources of them, say [People – HasCreated – Document], may have the domain / range relationship. Then the system deductively infers that the higher-level relationship makes the lower- level relation valid. (Figure 13)

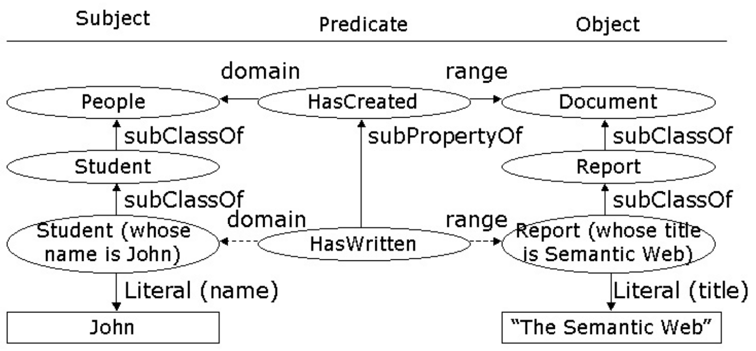


Figure 13 Find domain/range relations from super classes/properties

We identify four patterns for the system to infer the valid domain/range relations (Figure 14).

(1) Exact match (class and property have domain/range relation)
(2) Super property match (class and super property have domain/range relation)
(3) Super class match (super class and property have domain/range relation)
(4) Super class and super property match (super class and super property have domain/range relation)

When user passes the query to the system in the form of the triple, the system first checks the exact match of valid relation. If it is not found, then it traces any combination of the super-classes/super-properties of the triple. And if the valid relation would be found in the higher hierarchy, then returns all the sub classes of that combination, which are super classes of the given triple, as valid.

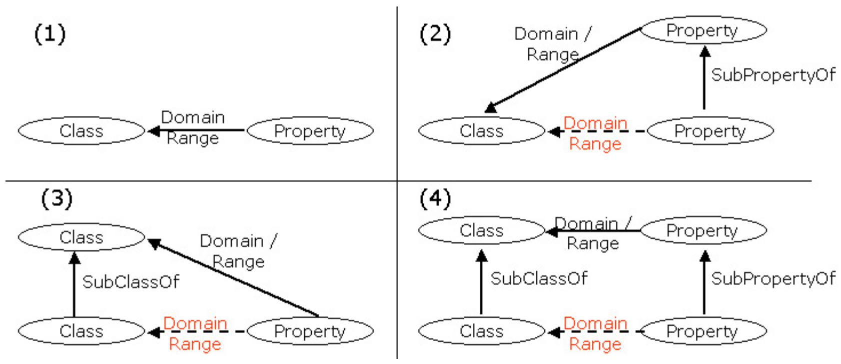


Figure 14 Four patters to find valid relation

We will revisit this approach later again when discussing the design of the Adaptive Semantic Web.

2.2. Event Notification

Traditional system provides the information upon the request of the client. Information delivery is outcome of request/reply-interaction. ***Event Notification System*** (ENS) [23] reacts to ***events***, not to requests, and delivers the information of client's interests/needs selectively. There are three key components in ENS: ***event generation/observation***, ***filter***, and ***notification***. (See Figure 15)

In ENS, any changes made on the known data are treated as event. Incoming events are observed by the system and transferred to the filter.

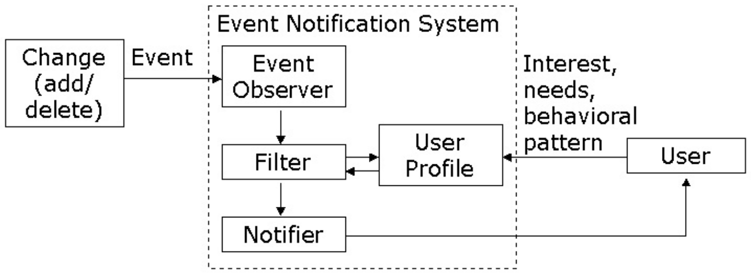


Figure 15 Event Notification System

There are many different types of events. Events can be the text data, or number. For example, in medical care system, the pulse of the patient and body temperature and the particular patterns of brainwave can be events. Pulse can be counted and reported to the system as the numeric data. Body temperature can be reported when it exceeds certain level. Brainwave would be reported when the wave pattern would become particular pattern. Or, the events can be generated by the patient themselves or his/her caregiver with pressing the button to tell the emergency. Events may be independently reported. Or they may be reported when they occur in a particular sequence, or when the combination of the change matches to the pre-defined event pattern. In the case that incoming events is not available, the system would fire the event generator, and collects data from outside system.

Not all events are relevant to the users' interests. So, when the occurrence of the event is observed, the system filters out the irrelevant information that is not the user's interests/needs. The condition of filtering is called ***user profile***. User profile in here has broader meaning than common English term. Rather than physical factors like age or sex, it focuses on users' interests or needs. User's interests or needs may be pre-registered by user, or the system infers from users' behavioural

patterns. If the matching data to the user profile are monitored, then they are transferred to the notifier, and the notifier delivers the event notice to the user.
In this process, definition of event and profile are the keys to receive the relevant event notifications. Event can be single event, or a combination of sequential events. Event can be fired by itself, or needs to be called to extract. We are interested in the changes of sentences in RDF documents. Then, more precisely which changes do we need to detect?
Event definition decides the input of the ENS. In contrast, user-profiling affect on the output of the ENS. What conditions to apply? How to describe events? Which language to use? Answering these questions, we can detect the valid results of event notices.

2.3. Related Work

The key processes in event notification are the change detection and the filtering process. Our intention is to extract the changes dynamically and selectively. So, we visited works related to each processes individually, and intend to integrate idea of them.

Concepts/tools for change-detection have been developed for some RDF related technologies: XML, database, Ontology, and for the Semantic Web. Cobena offered supporting tool for Xyleme (XML filtering system). [22] Active database is the most generally applicable tool for change detection. [19] Detection of changes guided by Ontology, which is one of three layers in the Semantic Web, is studied by Li Qin et al. [17]. Papamarkos et al offered event-condition action rule language, which is dedicated for the Semantic Web. [16]

Several query languages are offered for RDF, XML, and Attribute-value pairs. Karvounarakis et al proposed RQL, a query language for RDF documents [4]. He and his colleagues also discussed RQL for modelling a community portal [3]. Broekstra provided a guide for the RQL language dedicated to Sesame (SeRQL), which is a storage and querying system of RDF documents [9].

Only a few systems and languages have been proposed for XML filtering: Aguilera et al introduced Xyleme, which is the filter of XML documents [20]. Existing filters for XML documents do not detect changed or deleted data. No filters exist for RDF documents in triple or graph format.
Hayes et al introduced the automated collaborative filtering of attribute-value pairs [21]; this system is not suitable for RDF documents.

In this project, we intend to joint the two concepts of ‘change detection’ and ‘query language’; we offer the dynamic filtering of the changes in RDF documents using its triple structure. To the best of our knowledge, this is the first attempt for filtering RDF documents for changes and deletion.

3. System Design

In this section, we discuss about the design of the system we prepare in this dissertation. First, we introduce the overall system architecture in Section 3.1. Then we discuss the *Sesame* system, which is used as the storage and querying system for RDF documents. Our notification engine is developed adapting to the Sesame storage system. Sesame is introduced in Section 3.2. Then we discuss details of the conceptual design: event definition (Section 3.3), event observation (Section 3.4), profiling, filtering, and event notification (Section 3.5). Finally we introduce our implementation design in Section 3.6 (Storage) and in Section 3.7 (Trigger/Procedure).

3.1. System Architecture

In this project, we extended the existing system *Sesame*. Sesame is a RDF storage system, which provides the platform for the Semantic Web. [2] Details of Sesame will be introduced in Section 3.2. Event Notification Engine uses the data of RDF documents, which are translated and stored in the repository by Sesame, to develop the Event Notification Engine. (Figure 16, shaded area)
Event Notification Engine has three key modules: profiler, filter, and notifier. Profiler interprets user's queries into attribute-value pairs and store into user profile repository. Filter reacts to events (changes on the RDF records), and finds the matching records to user interest referring user profile records. Notifier receives the matching records from filter, and sends messages to the user. Our task in this project is to implement the filter engine in this project. Profiler and notifier, which are shown in the dot line in Figure 16, have not been implemented. We inserted user queries directly to the profile repository instead of parsing and storing through profiler.
Relevant events (matching records to user profiles) are stored into the query results database. These records are ready to be exported to user upon calling by notifier.

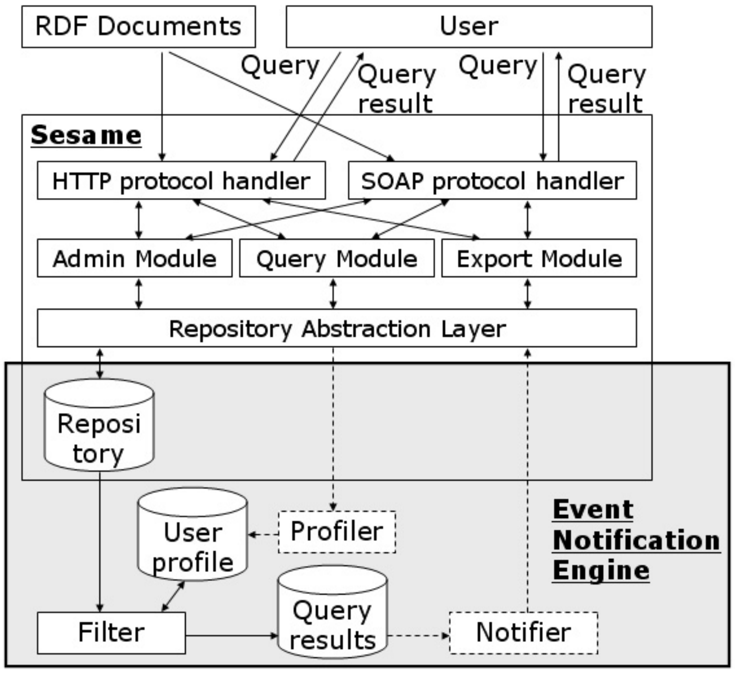


Figure 16 Architecture of the Semantic Web Filter Engine

3.2. Sesame

We used Sesame version 1.0.4 in our project. Hereinafter, we refer to this version as Sesame. Sesame supports insertion, storage, and querying of RDF documents. [2]. Two protocols are supported (HTTP and SOAP). It has three main modules: RDF administration module, RQL query module, and RDF export module. (See Figure 16) Administration module allows insertion / deletion of RDF data and schema information. Query module evaluates queries posed by users. Export module allows extraction of the schema and/or data. Information is translated in Repository Abstraction Layer (RAL) before storing into / extracting from database. This enables Sesame to handle different DBMSs.

RDF documents with any syntax errors are unacceptable by administration module. Three options are available in insertion interface: read from www, read from local file, and type directly to the textbox. Removal options are remove-triple option, which deletes one triple at a time, and clear-all-triples option, which deletes all triples from database.

Theoretically Sesame supports any DBMS because of abstraction by RAL, but actually current version 1.0.4 supports only three DBMS: MySQL, PostgreSQL, and Oracle 9i. [8] MySQL is categorised to Relational Database, and PostgreSQL to Object-Relational Database. Oracle can be used either as RDB or ORDB. Broekstra introduced two different schema; Relational database and Object-Relational Database [10]. However, its current version 1.0.4 handles only Relational Database schema.

In default, Sesame holds the information of RDF- and RDFS-defined resources as initial data. After user's first login and selection of the database, Sesame automatically creates 25 tables and inserts initial data into 16 tables. The remainder of nine tables are used for the temporary data manipulation. Hence we do not focus on these tables.

Entity Relationship Diagram (E-R Diagram) for valid 16 tables of Oracle is introduced later in Figure 22. An E-R Diagram is "*a model of entities in the business environment, the relationships or associations among those entities, and the attributes or properties of both the entities and their relationships. A rectangle is used to represent an entity and a diamond is used to represent the relationship between two or more entities*". [18] An oval represents an attribute. An underlined attribute is an identifier (primary key) of an entity.

There are some noteworthy issues about Sesame tables. The E-R Diagram indicates the referential integrity, but actually there is no foreign key set in the tables. No constraint is set in Sesame except primary keys. The Sesame parser engine is written in Java, and the java program controls referential integrity. Due to the restriction of the Sesame source code, no foreign key constraints can be added. Otherwise, Sesame reports the error upon deletion of data from the referred table. Even when the cascade option is selected, still Sesame returns the error message. So, all references are removed from tables, and maintained by Sesame Java programs.

There are some redundancies among tables. For, all records in SubClassOf table are derived from Direct_SubClassOf table. Direct_SubClassOf table contains only directly related sub-super class relationships. SubClassOf table, on the other hand, contains any combinations of sub-super classes. If there are two sub-super relations, then SubClassOf table holds all possible combinations making six records in total while Direct_SubClassOf table holds two records. (Figure 17) In other word, data in SubClassOf table can be inferred from Direct_SubClassOf table. RDF documents only state the direct sub-super relation. So, Sesame parser derives extra records and insert into SubClassOf table.

Sesame inserts 29 resources into tables in default, which represent classes and properties defined by RDF and RDF Schema. Examples of default classes are 'Resource', 'Class', 'Property', 'Literal', etc, and properties are 'type', 'subClassOf', 'subPropertyOf', 'domain', 'range', etc. Hierarchy to these resources are also generated by Sesame. For example, Sesame generates the 'Class' class in default. Then any classes newly added into Sesame system, say 'People' class, are defined as (direct) sub class of 'Class', and the record [People, Class] is added into the table Direct_SubClassOf table and SubClassOf table in relation of [sub, super] as well as adding record [People] into Class table. Same issues can be seen between Direct_SubPropertyOf table and SubPropertyOf table, and between ProperInstanceOf table and InstanceOf table.

Direct_SubClassOf

sub	**super**
Student	People
People	Class

SubClassOf

sub	**super**
Student	Student
Student	People
Student	Class
People	People
People	Class
Class	Class

Figure 17 SubClassOf vs Direct_SubClassOf

There are several database operation problems caused by integrity issue and the redundancy of table structure. This topic will be discussed later in Section 3.5.2 (Filtering).

Sesame supports five query languages: SeRQL-S, SeRQL-C, RDQL, Extract, and Explore. These languages extract the data of user's interests. Selection of query language is given in toolbar as read-action options. When the database is selected, Sesame shows the textbox under toolbar. Typing the query in this textbox according to the grammar of each language, and press evaluate button, then Sesame return the query results under the text box.

3.3. Conceptual System Design: Event Definition

In our system, we focus on Classes and Properties, but not "Literals" because:
(1) Our platform (Sesame) does not capture referential integrity between literal and resource.
(2) Class- Property relationships are central to the documents. They are very complex.
(3) Literal is merely the attribute of resources, so that it does not affect the meaning of sentences.

So, we only focus on the class, property, and their relationships (domain, range, sub-class, and sub-property). Possible changes of these components are insertion, deletion, and updating. Updating in Sesame is a combination of deletion and addition. It is therefore covered by insertion and deletion events.

We identify four insertion patterns for Domain relationship (See Figure 18(1) – (4)). Dotted lines represent the inserted resources. Range relation also has four patterns. (Figure 18(5) – (8)).

(1) Insert a relational property
(2) Insert a property and a relational property
(3) Insert a class and a relational property
(4) Insert all three components (i.e. a class, a property and a relational property)

For SubClassOf and SubPropertyOf, we identify four patterns each. Patterns of SubPropertyOf property are similar to Domain relation, but need to replace 'class' to 'property'. (See Figure 18 (9)-(12)).

Similarly, SubClassOf property has four insertion patterns. In this case, the above four patterns need to replace 'property' to 'class'. (Figure 18 (13) - (16)) In summary, there are total of 16 different events.

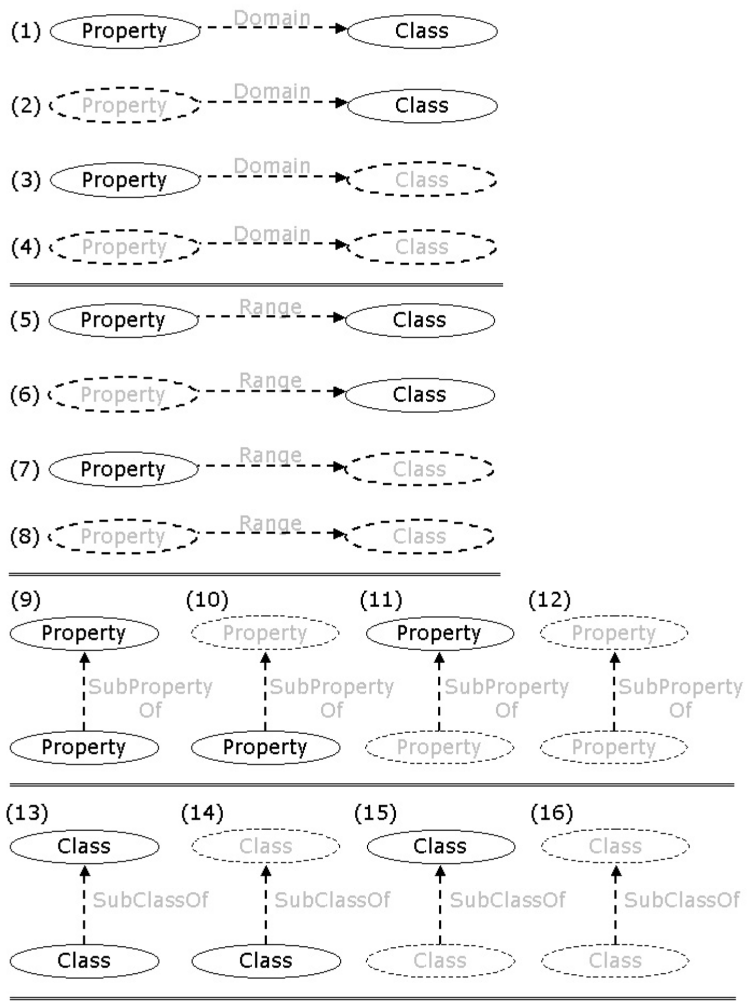

(1) Property Domain Class
(2) Property Domain Class
(3) Property Domain Class
(4) Property Domain Class
(5) Property Range Class
(6) Property Range Class
(7) Property Range Class
(8) Property Range Class
(9) Property SubProperty Of Property
(10) Property SubProperty Of Property
(11) Property SubProperty Of Property
(12) Property SubProperty Of Property
(13) Class SubClassOf Class
(14) Class SubClassOf Class
(15) Class SubClassOf Class
(16) Class SubClassOf Class

Figure 18 Event Patterns of Triple: Insertion

We identify six patterns for deletion from Domain relationship and Range relationship. The six patterns are described in Figure 19, with showing deleting resource with dot lines.

(1) Delete relational property only
(2) Delete property and relational property
(3) Delete class and relational property
(4) Delete class, property, and relational property
(5) Delete property only
(6) Delete class only

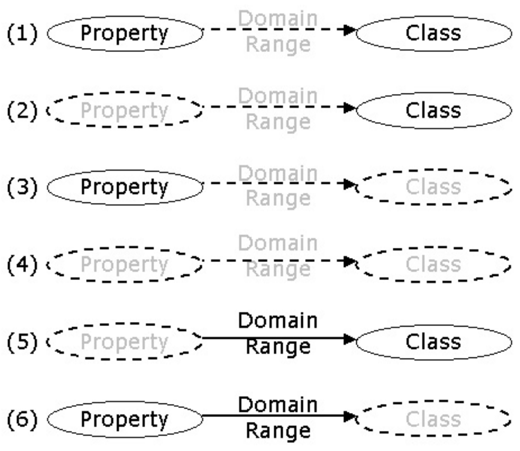


Figure 19 Event Patterns of Triple: Deletion

Pattern (5) represents the case that only the property is eliminated. In this case, the domain relation remains leaving garbage data. Pattern (6) eliminates classes only. Both (5) and (6) leave an invalid domain relation, but these patterns may happen in Sesame.

Similarly to Domain and Range, SubClassOf and SubPropertyOf relations also have six patterns each. So, there are total of 24 different event patterns.

As mentioned, updating is the combination of deletion and insertion. Thus, the two events (deletion and insertion) can be generated at the same time. Pattern (5) and (6) of deletion pattern do not apply for updating, because the corresponding patterns in insertion are not appeared.

3.4. Conceptual System Design: Event Observation

The proposing system uses Sesame as its platform. In Sesame, data are stored in a database. We will use the database - internal trigger to watch the events and manipulate the data.

In summary, the number of identified event patterns is 40 in total: 16 insertions and 24 deletions. This number is counted based on the events in RDF documents as described in the last section. Theoretically we need to observe 40 events. However, we need to reconsider the event definition from the system side. As we use triggers for monitor of events, events definition needs to be represented by changes in the database. There are eight relevant tables in Sesame's database: Class, Property, Domain, Range, Direct_SubClassOf, SubClassOf, Direct_SubPropertyOf, and SubPropertyOf. (Refer Appendix A1) As we discussed before, SubClassOf table is derived from Direct_SubClassOf table causing the redundancy. Thus actually we need to observe only one of them. We decided to use the Direct_SubClassOf table because the number of records is smaller. The same issue exists between Direct_SubPropertyOf table and SubPropertyOf table. Consequently, we need to observe the events on six tables. Each table may experience delete events and insert events. In conclusion, we need to observe 12 events on six tables in total.

3.5. Conceptual System Design: Profiling, Filtering, and Event Notification

The data processing flow of the event notification engine contains three modules: profiling, filtering, and notification.

3.5.1. Profiling

User Profile describes a user's interest. The profile may be expressed by profile definition languages. Profile definition may use one of the extensions of Sesame's five query languages. Currently, these language extensions do not exist. In the Adaptive Semantic Web, it is necessary to store the parsed profile in the database. Hence the extended language and their parser are not available, we manually insert the query into a Queries table. (The table design and the database schema will be discussed later in Section 3.6). Part of the future work is an implementation of a user interface for profile definition in Sesame, as well as the language extension and the parser. Then, this manual operation will be automated.
As shown in Appendix A3 (Table No 2), we designed table as to accept query (profile) in strings. Strings are interpreted into resource ID referring the Resources table. Filtering action can be fired only when Ids for all three components are identified. For example, where user wants the matching documents to the sentence [John – hasWritten - Report], components of tripe are stored as subject, predicate, and object respectively. Then triggers search resource ID of them. Given the resource ID, say 1, 2, and 3, then query becomes ready to search matching relations from Domain and Range tables.

3.5.2. Filtering

Filtering is the process to extract the matching records to profile (query) from valid domain and range relations. Where domain/range relationships exist in super classes/properties, they also need to be extracted as matching records.
Insertion/deletion of the data into tables fires the trigger on tables. Then the next issue is how to screen out the irrelevant data. There can be two approaches to filter the data. One approach is to use a built-in query language. Sesame prepared several built-in query languages. One of them is the dedicated query language for Sesame (called SeRQL – pronounce as circle) [8]. This query language would need to be extended to serve as profile language for the filtering. The other approach is to use the SQL query language of database with support of triggers and procedures. In our project, we employed the later approach. There are several reasons to select triggers for our approach as we will explain now in detail:

(1) Support of selective extraction
Query languages so far cannot select the part of matching documents selectively. Given the query, they return all matched results, but what we need is only the changed sentences. To extract the changes, the system has to hold all data and compare to the result after changes are made. It is impossible to hold data for query languages so far. On the other hand, database programs hold all data before changing, thus it is possible to extract changes selectively.

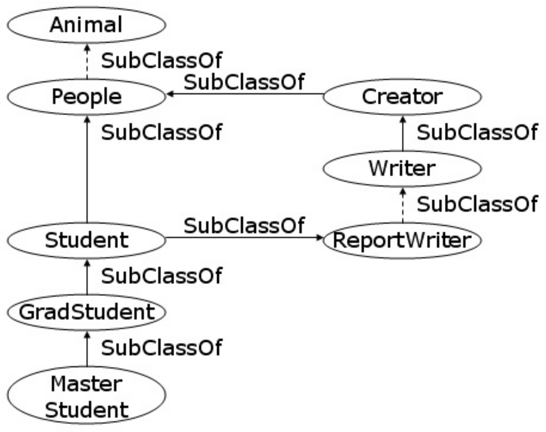


Figure 20 Deletion of Sub-Class

(2) Selective extraction of deletion
When a change was made, users need only be notified on that change. However, the current query languages return the whole result. So, the notification system would need to compare the results before the changes are made and after

the changes are made. Triggers allow for simpler solution of comparing within the database. Classes and properties sometimes can be connected in very complex way. Consider the situation that shown in the Figure 20. If sub-super relation between ‘Animal’ and ‘People’ is deleted, ‘Animal’ class loses whole sub-super relation between subclasses of ‘People’ class. However, where the sub-super relation between ‘Writer’ and ‘ReportWriter’ is deleted, new relations are more complicated. ‘People’ has two sub classes: ‘Creator’ and ‘Student’. Sub classes of ‘ReportWriter’ (i.e. ‘Student’ and its sub classes) are no longer the sub class of ‘Writer’ and ‘Creator’. Similarly, super classes of ‘Writer’ are no longer the super classes of ‘ReportWriter’. However, ‘Student’ and its sub classes are still sub classes of ‘People’ because there is another direct sub-super relationship between ‘People’ and ‘Student’. Because of potential existence of such direct relationships, we need to check the validity of each relationship one by one. Query languages, in general, are not designed to extract this sort of changes selectively.

(3) Inability of query languages to describe the multi level matching

Sesame returns the result of a query in the form of a triple. However it is uncertain whether it expresses the real relationship of the classes and properties. Triple is actually structured with the five components: class (subject), property (predicate), class (object), domain, and range. Where the class / property has super class / property, if the upper level of class / property would have valid domain (or range) relation, then the lower level class / property naturally have the valid relationship. Validity of domain and range should be evaluated individually without concerning levels. Then there is the case that a triple becomes valid even when the single property does not have both domain and range relations. Valid relation of super class/property in any upper level can make that triple valid. In Figure 21, the bottom line resources do not have the exact match of the relations. (i.e. John, HasEdited, and SemanticWeb) However, each resource has valid relation of domain or range in upper class. [HasWritten - Domain - Student] and [HasCreated - Range - Document]. Thus the bottom line relation is also valid. In this case, valid relationship is confirmed in different levels. If we selectively indicate only one property to show the validity of this statement, the triple does not express this relation accurately. It is considered to be more appropriate to indicate two properties to show the valid relation.

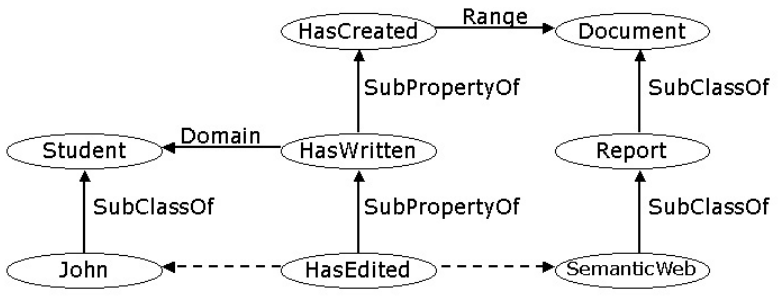


Figure 21 Matching in multi-level

(4) Less workload

Sesame supports five query languages and three database programs. Using the query languages for filtering means, it is necessary to analyse five query languages. It is more time consuming to analyze five query languages rather than convert the triggers to three database programs.

(5) Ease of future maintenance

The Semantic Web is not the established and stabilized technologies. It is still on the way of development. So, more query languages would be introduced in future. However the Database has already established the high quality level and got the stability. So, the less efforts will be expected for adoption than query languages.

Considering the above issues, this project challenged to develop the filter with trigger and procedures. Algorithm of trigger/procedure will be introduced in Section 3.7.

3.5.3. Event Notification

In Sesame, the result of queries is given directly on the screen. In the Adaptive Semantic Web, it is necessary to send the notification of changes to users. There can be several options for event notification. One is to send e-mail to users. Another option is to store the changes into the database, and show them at the time of user's next login. In our project, we stored relevant changes into the Notices table. (The table design and the database schema will be discussed later in Section 3.6) Upon implementation of a notifier interface in Sesame, this operation will be automated. Insertion of the changes into Notices table is handled by trigger with the assist of procedures. Algorithms of triggers/procedures will be introduced in Section 3.7.

3.6 Implementation Design: Storage

In addition to the Sesame generated tables, we created 14 more tables to run the event notification feature. The relations among the extended tables are introduced in the E-R Diagram (Appendix A2). The table design is also shown in the Appendix A3. Following the Sesame regulation, none of the additional tables for filtering engine has foreign keys. Instead, triggers maintain the referential integrity. Upon deleting the referred data, triggers delete the records, which is in referring tables. So the foreign keys indicated in E-R Diagram (Appendix A1 and A2) are not physically set, but they are conceptually referred.

The roles of tables are categorised into four groups: (1) store user queries (profiles), (2) intermediate the queries and query results, (3) store query results, and (4) error protection. The relation among tables is described in Figure 22. We also explain the relation among tables applying sample data in Figure 23. Figure 23 indicates both table of Sesame and extended tables for our Event Filtering Engine. Extended tables are underlined to distinguish from Sesame's ones.
In the following explanations, we sometimes use the term 'valid', for example, valid subject, valid domain relation, etc. We call valid when the resources or relations exist in Sesame tables. Domain relation is valid if combination of property and class exists in Domain table. Subject is valid if the resource name exists in Resources table and that resource's ID exists in Class table. Similarly, sub-super property relation is valid if the combination of sub property and super property exists in Direct_SubPropertyOf table.

Roles of tables for Event Notification engine are as follows:

(1) Tables for Queries

a. Requesters
Store the data about the requesters (users) including the e-mail address. Requesters' details are taken from log-in accounts

b. Queries
Store the user profiles as queries (subjects, predicates, and objects) as well as requester's id. Query details are stored being identified by query ID. Query ID is sequential number to identify the query. (Shown as 201 in Figure 23) Requester ID refers to Requesters table. (Shown as 101 in Figure 23) Predicate, subject, and object are stored as string. Shown as John, hasWritten, Report respectively in Figure 23)

When the same words as predicate is found in (inserted into) the Resources table and if it is the property (i.e. exist in Property table), then the resource ID of that property is stored into QueryPredicates table together with query ID. Similarly resource IDs of subject and object are stored into QuerySubjects and QueryObjects tables respectively when the resource IDs are found and they exist in Class table.

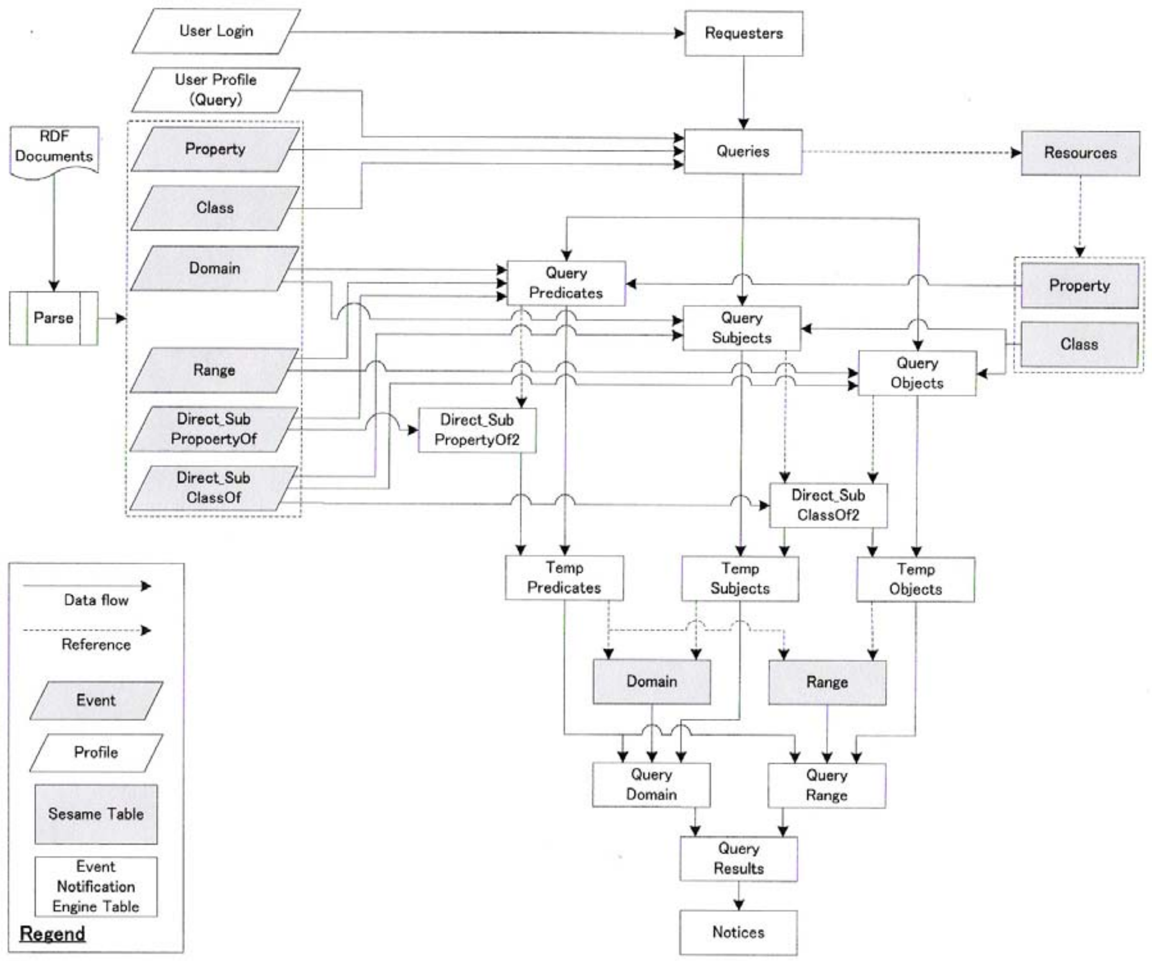
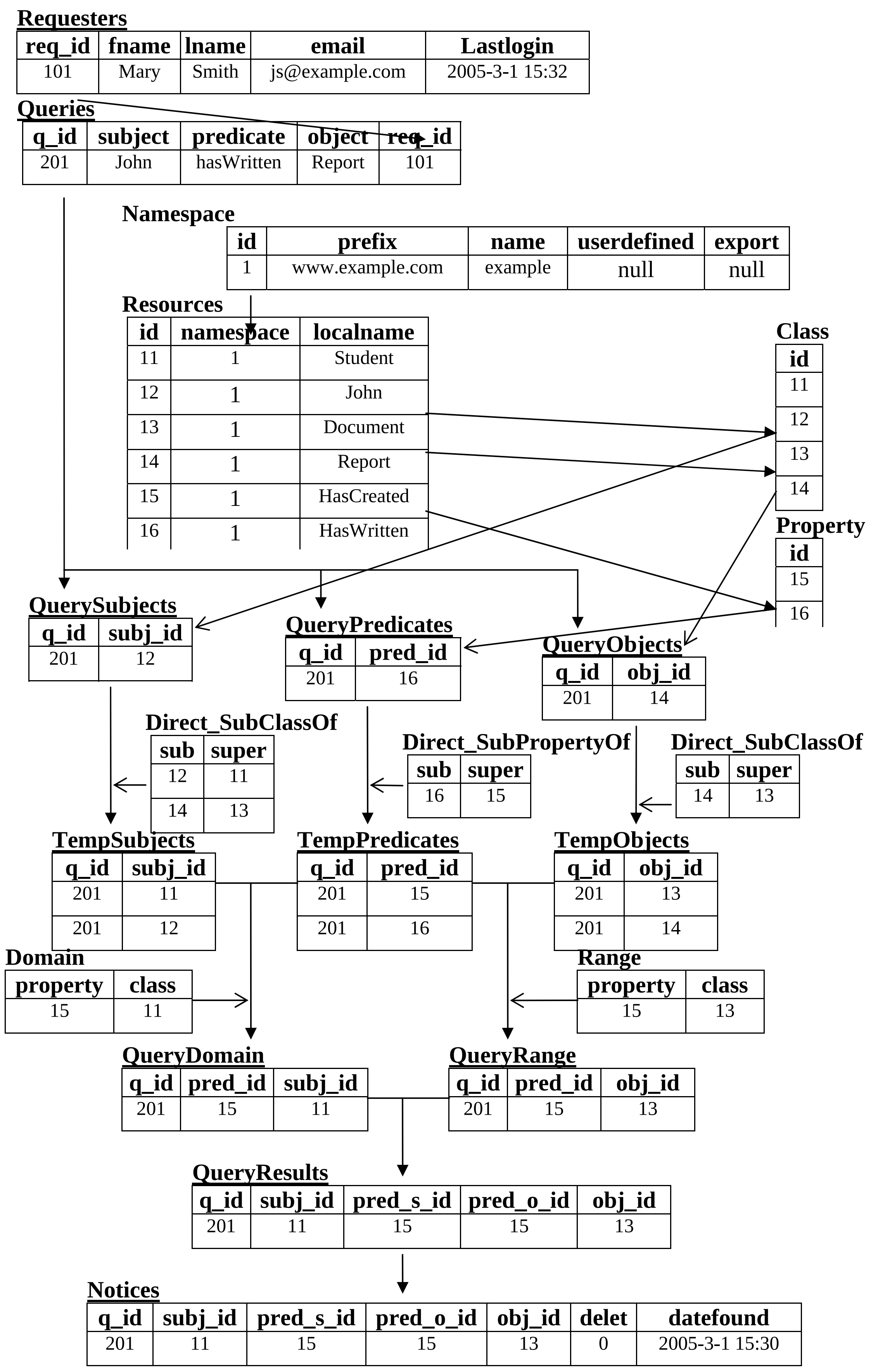


Figure 22 Relationship among tables created by Sesame/ tables for Event Filtering Engine



Requesters

req_id	fname	lname	email	Lastlogin
101	Mary	Smith	js@example.com	2005-3-1 15:32

Queries

q_id	subject	predicate	object	req_id
201	John	hasWritten	Report	101

Namespace

id	prefix	name	userdefined	export
1	www.example.com	example	null	null

Resources

id	namespace	localname
11	1	Student
12	1	John
13	1	Document
14	1	Report
15	1	HasCreated
16	1	HasWritten

Class

id
11
12
13
14

Property

id
15
16

QuerySubjects

q_id	subj_id
201	12

QueryPredicates

q_id	pred_id
201	16

QueryObjects

q_id	obj_id
201	14

Direct_SubClassOf

sub	super
12	11
14	13

Direct_SubPropertyOf

sub	super
16	15

Direct_SubClassOf

sub	super
14	13

TempSubjects

q_id	subj_id
201	11
201	12

TempPredicates

q_id	pred_id
201	15
201	16

TempObjects

q_id	obj_id
201	13
201	14

Domain

property	class
15	11

Range

property	class
15	13

QueryDomain

q_id	pred_id	subj_id
201	15	11

QueryRange

q_id	pred_id	obj_id
201	15	13

QueryResults

q_id	subj_id	pred_s_id	pred_o_id	obj_id
201	11	15	15	13

Notices

q_id	subj_id	pred_s_id	pred_o_id	obj_id	delet	datefound
201	11	15	15	13	0	2005-3-1 15:30

Figure 23 Relationship among tables with sample data

(2) Tables for intermediate process

c. QuerySubjects
Store query ID and valid class ID (i.e. resource ID) which has the same resource name in Resources table as subject name in Queries table. Query ID refers to Queries table. (Shown as 201 in Figure 23) Subject ID refers to Class table. (Shown as 12 in Figure 23)
Combination of query ID and subject ID must be unique.

d. QueryPredicates
Store query ID and valid property ID (i.e. resource ID) which has the same resource name in Resources table as predicate name in Queries table. Query ID refers to Queries table. (Shown as 201 in Figure 23) Predicate ID refers to Property table. (Shown as 16 in Figure 23)
Combination of query ID and predicate ID must be unique.

e. QueryObjects
Store query ID and valid class ID (i.e. resource ID) which has the same resource name in Resources table as object name in Queries table. Query ID refers to Queries table. (Shown as 201 in Figure 23) Subject ID refers to Class table. (Shown as 14 in Figure 23)
Combination of query ID and object ID must be unique.

f. TempSubjects
Temporally store subject ID and all valid super classes of subject ID that appears in QuerySubjects table. (Shown as 12 and 11 in Figure 23) Query ID refers query ID of corresponding subject. (Shown as 201 in Figure 23)

g. TempPredicates
Temporally store predicate ID and all valid super properties of predicate ID that appears in QueryPredicates table. (Shown as 16 and 15 in Figure 23) Query ID refers query ID of corresponding subject. (Shown as 201 in Figure 23)

h. TempObjects
Temporally store object ID and all super classes of valid object ID that appears in QueryObjects table. (Shown as 14 and 13 in Figure 23) Query ID refers query ID of corresponding subject. (Shown as 201 in Figure 23)

i. QueryDomain
Store all valid domain relations, which are combinations of resources held in TempPredicates table and TempSubjects table. (Shown as 15 and 11 respectively in Figure 23) Query ID refers query ID of corresponding predicate/subject. (Shown as 201 in Fig. 23)
Combination of query ID, predicate ID, and subject ID must be unique.

j. QueryRange
Store all valid range relations, which are combinations of resources held in TempPredicates table and TempSubjects table. (Shown as 15 and 13 respectively in Figure 23) Query ID refers query ID of corresponding predicate/object. (Shown as 201 in Figure 23)
Combination of query ID, predicate ID, and object ID must be unique.

(3) Tables for Query Results

k. QueryResults
Store query results, which are combination of records in QueryDomain table and QueryRange table. Any combinations of records in QueryDomain table and QueryRange table with mutual query ID are valid query results. In Figure 23, there is only one record each for query ID ‘201’ in QueryDomain and QueryRange tables. So, we get only one combination, which is [11, 15, 15, 13]. Combination of query ID, subject ID, domain-property ID, range property ID, and object ID must be unique.

l. Notices
Store logs of newly inserted /deleted query results into/from QueryResults table. Flag to indicate event type (1:delete or 0:insert) and date/time of filtering the event are also stored.
No referential integrities or unique constraints are set because a record may occur several times.

(4) Tables for Error Protection

m. Direct_SubClassOf2
Duplicate copy of Direct_SubClassOf table. We will discuss the reason to prepare the duplicated copy later in this section.

n. Direct_SubPropertyOf2
Duplicate copy of Direct_SubPropertyOf table

The Notices table and the Requesters table contain a column for sysdate (i.e. system date). In Notices table, sysdate is stored when the new insertion/deletion record is inserted. In Requesters table, the sysdate shows the time that user logged in last time. Comparing two sysdate, the system can select the only records, which have not been shown to the users. This feature offers different options to receive the event notice: Users can receive e-mails, or they can have the summary of the un-seen event notices upon logging-in.

Tables TempSubjects, TempPredicates, and TempObjects tables do not hold any records. They are used to store the relevant data temporarily when comparing the valid records ‘before insertion (or deletion)’ to ‘after insertion (or deletion)’. After insertion of relevant records after operation into these tables, records are compared to existing records (i.e. records before changes occur).

Last two tables are duplicate copies of existing tables. These introduce redundancies but some problems cannot be solved without these duplicated copies. One problem is caused by mutation, and the other problem is observed at the time of updating the table.

(1) Mutation problem
Mutation error occurs when the trigger/procedure tries to select the data from table, which is on the process of insertion/deletion and fired the trigger. For example, consider the case that the data is inserted into the Direct_SubClassOf table. Because an event (a newly added relation) may connect the query’s subject/object with the valid statement, the Direct_SubClassOf table fires the trigger to re-check the super classes of subject/object.
However, the Direct_SubClassOf table itself is in the process of insertion. In order to keep atomicity, Oracle will not treat the insertion transaction as complete until the trigger completes all duties. So, the Oracle returns the mutation error. Same problem happens in Direct_SubPropertyOf table, and in the deletion transaction, too.

To solve this mutation problem, we prepared the duplicate copy of Direct_SubClassOf table and Direct_SubPropertyOf table. When the data is inserted into Direct_SubClassOf table or Direct_SubPropertyOf table, the trigger inserts the same data into duplicate table. Then trigger can select the data from the duplicated copy any time.

(2) Update trigger
Another approach to solve the mutation problem could be proposed: Sesame prepares the two brotherly tables: SubClassOf table and Direct_SubClassOf table. All records in SubClassOf table are derived from Direct_SubClassOf table. So, we can select the records from the tables other than the one that is firing the trigger. However this approach is found to be invalid at the time of updating. The problem is the order in which the records are written into these tables. Sesame processes both deletion and insertion transactions first on the SubClassOf table and then on the Direct_SubClassOf table. We need to select the records before deletion and after insertion. If records are deleted before comparison, or if the records are inserted after comparison, the system loses the opportunity to compare the data before and after deletion (or insertion). So, we have to set deletion trigger on SubClassOf table, and insertion trigger on Direct_SubClassOf table. Otherwise the change is not reflected in the result.
However, in updating, this solution does not work. Updating is a combination of insertion and deletion. But in this solution, we have to set the trigger for updating in two different tables, which is not possible for triggers. Therefore, we prefer the solution to prepare the duplicated copy of Direct_SubClassOf table and Direct_SubPropertyOf table.

3.7 Implementation Design: Algorithm of Triggers

Since this project employs Oracle as database, naturally SQL is employed as query language. Initially triggers are fired upon insertion into/deletion from six tables: Property, Class, Domain, Range, Direct_SubClassOf, and Direct_SubPropertyOf. These tables fire the triggers, and manipulate other tables or call procedures. Inserted/deleted tables fire the trigger and manipulate other tables or call procedures. The Queries table is referred to at the time of table manipulation. Then the events that do not match the user profiles are filtered out. Filtering processes use different triggers depending on the incoming events. They will be introduced individually later in this section. Relevant changes are finally reflected to QueryResults table. QueryResults table is then updated removing irrelevant records and inserting the relevant records. QueryResults table holds the latest valid query results, and insert the insertion / deletion records into Notices table upon the occurrence of insertion/deletion.

The overall flow is described in the Appendix C1. Triggers and procedures are colour-coded. Green colour indicates the observation of the events. Yellow colour indicates event notification, and the rest of them are colour-coded to blue, which means intermediate processes. Please note, there are four sub procedures, which do not call other procedures. They assist the process of other procedures/triggers manipulating temporary tables or checking the records in tables. They are shown in the Appendix C2 with colour-code of grey. The flowcharts of individual triggers and procedures are introduced in the Appendix B1 to B28 with short explanations of process.
Now, we discuss the filtering algorithm. Event source and flow of data among tables can be found in Figure 22. We introduce the algorithms referring the example introduced in Figure 24. In this example, we interpreted RDF Document into the triple form to simplify the statement. We show the only insertion events in this example. Scenario of example is to filter the matching records to the query inquired by Mary, which is [John - hasWritten – Report]. This query and its requester's details are stored into the Queries table and Requesters table with the following details.

Profile (Query):

subject	**predicate**	**object**
John	hasWritten	Report

Event Notification Engine operation

Insert into Requesters table

req_id	fname	lname	e-mail	lastlogin
101	Mary	Smith	…	…

Insert into Queries table

q_id	subject	predicate	object	req_id
201	John	hasWritten	Report	101

As introduced before, there are 12 event observers: six tables with two operations each (i.e. insertion and deletion), and each event takes the unique process for filtering. Each event refers to the flowchart in the appendix B and Figure 24. The step numbers after Figure 24 represent numbers shown in the left end of Figure.

(1) Insertion into Domain table

When the new record is inserted into Domain table, trigger first checks if any valid record exists in the QuerySubjects table and if any valid record exists in QueryPredicates table. If both of them exist which have mutual query ID, then insert the new records into QueryDomain table. Insertion trigger of QueryDomain table checks the QueryRange table if it contains the query ID which is newly added into QueryDomain table. If it does, then insert the combination of records in QueryDomain and QueryRange tables into QueryResults table. QueryResults table fire the trigger to insert new notice record into Notices table. Repeat this process for sub classes of new class and sub properties of new property. If the transaction is updating, delete the old resources in domain relationship from QueryDomain table before insertion transaction. In Figure 24, no action is taken by our system because there is no record in QueryPredicates table at the time of insertion into Domain table.
For more information, refer to Appendix B9, B25, B26 / Figure 24 Step 8.

(2) Insertion into Range table

Algorithm is much like insertion into Domain table. Interpret as QuerySubjects to QueryObjects, Domain to Range, and QueryDomain to QueryRange, and then the rest of the explanation is same as the procedure to insert into Domain table. In Figure 24, no action is taken by our system because there is no record in QueryPredicates table at the time of insertion into Domain table.
For more information, refer to Appendix B11, B25, B26 / Figure 24 Step 9.

subject **predicate** **object** **Sesame Table Operation** **Event Notification Engine operation**

1 Student type class

Insert into Namespace table

id	prefix	name	…	…
1	www.example.com	example	…	…

Insert into Resources table

id	namespace	localname
11	1	Student

Insert into Class table

id
11

2 John type class

Insert into Resources table

id	namespace	localname
…	…	…
12	1	John

Insert into Class table → fire trigger

id
…
12

Insert into QuerySubjects table

q_id	subj_id
201	12

3 John subClassOf Student

Insert into Direct_SubClassOf table

sub	super
12	11

Insert into Direct_SubClassOf2 table

sub	super
12	11

4 Document type class

Insert into Resources table

id	namespace	localname
…	…	…
13	1	Document

Insert into Class table

id
…
13

Figure 24 (Part 1) Example of an Insertion of RDF Documents and Filtering Process

(3) Deletion from Domain table

Delete the relation from QueryDomain table where subject ID is class ID of deleted record and predicate ID is property ID of deleted record. Deletion trigger of QueryDomain table deletes the corresponding records from QueryResults table. Deletion trigger of QueryResults table then fires, and insert the full details of deletion records as well as the deletion time and the category to identify the deletion (i.e. set 1 in column 'delet') into Notices table.
For more information, refer to Appendix B10 and B26.

(4) Deletion from Range table
Algorithm is much like deletion from Domain table. Interpret the terms as same rules as insertion. For more information, refer to Appendix B10 and B26.

subject predicate object — **Sesame Table Operation** — **Event Notification Engine operation**

5 Report type class

Insert into Resources table

id	namespace	localname
…	…	…
14	1	Report

Insert into Class table

id
…
14

Insert into QueryObjects table

q_id	obj_id
201	14

6 Report subClassOf Document

Insert into Direct_SubClassOf table

sub	super
…	…
14	13

Insert into Direct_SubClassOf2table

sub	super
…	…
14	13

7 hasCreated type property

Insert into Resources table

id	namespace	localname
…	…	…
15	1	hasCreated

Insert into Property table

id
15

8 hasCreated domain Student

Insert into Domain table

property	class
15	11

9 hasCreated range Document

Insert into Range table

property	class
15	13

Figure 24 (Part 2) Example of an Insertion of RDF Documents and Filtering Process

(5) Insertion into Direct_SubPropertyOf table

In this procedure, we use the Direct_SubPropertyOf2 table, which is duplicated copy of Direct_SubPropertyOf table. Direct_SubPropertyOf2 table is automatically maintained by the trigger on Direct_SubPropertyOf table upon insertion, deletion, or updating. As discussed in Section 3.6, we use the duplicate copy to avoid the *mutation* problem.

When the new sub-super relation record is inserted, check the QueryPredicates table if new 'sub' exists. If it exists, then find all super properties of that predicate and insert into TempPredicates table. Also find super classes of corresponding subject / object from QuerySubjects / QueryObjects tables and insert into TempSubjects / TempObjects table respectively. Utilizing the records in temporary tables, insert the valid records into QueryDomain table and QueryRange table, then QueryResults table, which fire the trigger to insert event notice into Notices table.

Repeat the same procedure for the sub property of new 'sub' referring Direct_SubPorpoertyOf2 table until no sub property would be found.

If transaction is updating, then handle deleting transaction first.

In Figure 24, system finds the sub property ID '16' in QueryPredicates table, which is inserted in step 10. Then trigger of QueryPredicates table searches super properties of 16. '15' is found and '16' and '15' are inserted into TempPredicates table. Similarly, TempSubjects and TempObjects table get super classes; (12, 11) and (14, 13) respectively. Then all possible combinations of TempPredicates and TempSubjects are generated. Possible combinations of predicate-subject

<u>subject</u> <u>predicate</u> <u>object</u>
<u>Sesame Table Operation</u>

<u>Event Notification Engine operation</u>

10 hasWritten type property

Insert into Resources table

id	namespace	localname
…	…	…
16	1	hasWritten

Insert into Property table

id
16

Insert into QueryPredicates table

q_id	pred_id
201	16

Insert into TempSubjects table

q_id	subj_id
201	12
201	11

Insert into TempObjects table

q_id	obj_id
201	14
201	13

Insert into TempPredicates table

q_id	pred_id
201	16

Clear TempSubjects, TempObjects, and TempPredicates tables

Figure 24 (Part 3) Example of an Insertion of RDF Documents and Filtering Process

(domain) relations are (15, 11), (15, 12), (16, 11), and (16, 12). Among these combinations, one combination (15, 11) is found in Domain table. (Refer Figure 24 step 8) Therefore, this combination is stored into QueryDomain table together with query ID (i.e. 201). Similarly, range (predicate-object) relation (15, 13) is found to be valid in Range table. (Refer Figure 24 step 9) So, this relation is also stored into QueryRange table. QueryDomain table and QueryRange table have mutual query ID of 201 now. So, new record is created and inserted into QueryResults table. Then trigger of QueryResults table is fired, and it inserts the event notice into Notices table.
For more information, refer to Appendix B5, B6, B16, B19, B20, B28, B23, B24, B26, B27, B25 / Figure 24 Step 11.

(6) Insertion into Direct_SubClassOf table
Algorithm is much like insertion into Direct_SubPropertyOf table. Interpret property into class, class to property, Direct_SubPropertyOf2 to Direct_SubClassOf2, QuerySubjects/QueryObjects

subject predicate object / **Sesame Table Operation** — **Event Notification Engine operation**

11 hasWritten subPropertyOf hasCreated

Insert into Direct_SubPropertyOf table

sub	super
…	…
16	15

Insert into Direct_SubPropertyOf2 table

sub	super
…	…
16	15

Insert into TempPredicates table

q_id	pred_id
201	16
201	15

Insert into TempSubjects table

q_id	subj_id
201	12
201	11

Insert into TempObjects table

q_id	obj_id
201	14
201	13

Insert into QueryDomain table

q_id	pred_id	subj_id
201	15	11

Insert into QueryRange table

q_id	pred_id	obj_id
201	15	13

Insert into QueryResults table

q_id	subj_id	preds_id	predo_id	obj_id
201	11	15	15	13

Insert into Notices table

q_id	subj_id	pred_s_id	pred_o_id	obj_id	delet	date
201	11	15	15	13	0	…

Clear TempSubjects, TempObjects, and TempPredicates tables

Figure 24 (Part 4) Example of an Insertion of RDF Documents and Filtering Process

into QueryPredicates, and QueryPredicates into QuerySubjects/QueryObjects. In Figure 24, no action is taken by our system other than duplication of Direct_SubClassOf table, because there is no record in QueryPredicates table at the time of insertion into Direct_SubClassOf table.
For more information, refer to Appendix B7, B8, B17, B18, B21, B22, B28, B23, B24, B26, B27, B25 / Figure 24 Step3 and 6.

(7) Deletion from Direct_SubPropertyOf table
Deletion procedure is bit more complicate than insertion. Since the deleted sub-super relation may bridge other properties which are recognized as part of valid domain/range relations. (Refer Section 3.5.2) We need to screen all the predicates which are super class of deleted property. If any query uses the super property of deleted property, then re-build the valid combination of domain/range. We use the temporary tables (TempSubjects, TempPredicates, and TempObjects) for this purpose. We compare the existing query results with the temporary results, which now holds the latest valid results, and then delete any results which is not contained in the temporary results. It is much simpler to delete all results once, and then re-build the new results. But in that case, user shall receive the lots of notices of insertions and deletions. The above-mentioned procedure saves the number of insertion/deletion notices, and avoids the confusion of user.
For more information, refer to Appendix B6, B19, B20, B26, and B28.

(8) Deletion from Direct_SubClassOf table
Algorithm is much like deletion from Direct_SubPropertyOf table. Interpret the terms as same rules as insertion.
For more information, refer to Appendix B8, B21, B22, B26, and B28.

(9) Insertion into Property table
Upon insertion, trigger checks the corresponding ‘localname’, which is shown as rdf:ID in RDF document, from Resources table, then compare to predicate in the Queries table. If any predicates would match to property’s ‘localname’, then resource ID of property is inserted into QueryPredicates table as predicate ID. Then insertion event fires trigger of QueryPredicates table.
Insertion trigger of QueryPredicates table, then checks QuerySubjects table if any records with same query ID as predicate would exist. If records exist, then insert all super classes of subject into TempSubjects table, and all super properties of predicate into TempPredicates table. After that, system generates all possible combination of records in TempSubjects and TempPredicates tables, and compare to records in the Domain table whether any combinations would match to them. Where matching records exist, store those combinations into QueryDomain table.
Similarly, system checks QueryObjects table if any records with same query ID as predicate exist. If exist, then insert super classes of object into TempObjects table. Where the combinations of TempObjects and TempPredicates table would match to any records in the Range table, insert those combinations into QueryRange table. Where both QueryDomain and QueryRange table have any records with mutual query ID, then triggers of these tables create the combinations of the records from them, and insert them into QueryResults table. Insertion trigger of QueryResults table then fires, and insert the full details of insertion records as well as the insertion time and the category to identify the insertion (i.e. insert 0 into column ‘delet’) into Notices table.

In Figure 24 step 7, no action is taken by our system because no valid property can be found in Queries table at the time of insertion into Property table. In step 10, on the other hand, predicate ‘hasWritten’ is found in query ID ‘201’, and its resource ID is identified as 16. So, system inserts ‘16’ into QueryPredicates table. Trigger of QueryPredicates table then searches the records with query ID ‘201’ from QuerySubjects table and finds subject ID ‘12’. Then ‘12’ and super classes of ‘12’ (i.e. ‘11’) are inserted into TempSubjects table. Similarly, TempObjects table get super

classes (14 and 13), and TempPredicates table get super properties (16). Then all possible combinations of TempPredicates and TempSubjects are generated to find valid domain relations from Domain table. For query ID ‘201’, possible combinations of predicate-subject (domain) relations are (16, 11) and (16, 12). Neither of them have valid domain relation, so system does not insert any data into QueryDomain table. Similarly, TempPredicates and TempObjects tables are found not to have valid range relation, so the system stop further search, and clear TempSubjects, TempObjects, and TempPredicates tables.
For more information, refer to Appendix B1, B10, B6, B16, B26, B19, B20, B28, B23, B24, B27, B25 / Figure 24 Step 7 and 10.

(10) Insertion into Class table
Algorithm of insertion into Class table is much like insertion into Property table. Interpret terms as ‘property’ into ‘class’, ‘predicate’ into ‘subject’ (or ‘object’), ‘QueryPredicates’ into ‘QuerySubjects (or QueryObjects)’.
For more information, refer to Appendix B3, B17, B18, B10, B8, B28, B23, B24, B26, B21, B22, B27, B25 / Figure 24 Step 1, 2, 4, and 5.

(11) Deletion from Property table
Upon deletion of property, old property is deleted from Direct_SubPropertyOf table, Domain table, and Range table. Then the triggers of these tables process the rest of the data manipulation.
For more information, refer to Appendix B2, B10, B6, B19, B20, B26, and B28.

(12) Deletion from Class table
Algorithm is much like deletion from Property table. Interpret the terms as same rules as insertion.
For more information, refer to Appendix B4, B10, B8, B21, B22, B26, and B28.

Applying the knowledge of Event Notification System, RDF and RDF Schema technologies, we designed 12 triggers on Sesame tables. We also designed additional 14 tables to store user profiles and to filter matching events upon insertion/deletion events on the Sesame tables. In addition to triggers on Sesame tables, further 18 triggers are set on the tables for Event Notification System. The number of procedures prepared for Event Notification System is 25. In total, 57 triggers and procedures are prepared for our project.

4. Implementation and Evaluation

We implemented triggers and procedures based on the design introduced in previous section, and tested the functionalities of triggers/procedures. In this section, we summarise the tasks that we performed for the implementation, and study the performance of our implementation.

4.1. Implementation

We created 14 tables, and set the triggers on them as event observer/notifier. System is implemented only on database, and Sesame code is not modified at all. 25 triggers and 27 procedures (52 in total) are coded. Five of them (four triggers and one procedure) are not introduced in the system design. These four triggers and a procedure handle the queries input into Queries table. Triggers are fired upon (a) deleting query requester, (b) inserting new query, (c) updating query, and (d) deleting query. Trigger calls procedure after deleting query. These triggers/procedure prepare the initial records into the QueryResults table when the query is added

(or deleted / updated). They are necessary because the system cannot find any changes without the initial records. 12 triggers are for event monitoring, which are set on the tables provided by Sesame. (Property, Class, Domain, Range, Direct_SubPropertyOf, Direct_SubClassOf tables) Two triggers are for event notice, which are set on the QueryResults table. Other triggers are for filtering the irrelevant events.
Implementation had been carried out in the trial and error base, re-arranging to the design repeatedly. The mutation problem and the confusion caused by the lack of referential integrity caused these repetitions. Extremely slow processing, which we will discuss in Section 4.2.2 also caused the repetition for seeking the better solution.

4.2. Evaluation

The main focus of evaluation is to confirm the correct functioning of the Event Notification Engine. Functionalities of triggers/procedures are checked in two steps. First, we manipulated the database directly, and confirmed the insertion, deletion and updating tasks are appropriately performed on six Sesame-generated tables, which we set the event observers. We confirmed the performance with checking records inserted into Notices table. The updating was only tested in the database, since Sesame does not support updates via its interface.
Second, we generated valid RDF documents, and submitted them to Sesame for storage. Our filter engine matches the documents to the stored profiles, and the results are confirmed in the Notices table.

4.2.1. Test Planning

Test on Oracle Database
Three operations are tested on the eight tables using SQL language. Operations are insertion, deletion, and updating. Six tables observe events: Property, Class, Domain, Range, Direct_SubPropertyOf, and Direct_SubClassOf table. Additionally, we tested three operations on Queries table, and deletion on Requesters table. Operations on Queries table are expected to generate the base data, which are to be compared to the results after events occur. Requesters table contain the details of query requester. So, deletion of the requester should delete the queries requested by that requester from Queries table. So that function is also confirmed. The functionality is checked with selecting all records from Notices table. Where new notice is inserted, the function is confirmed to be working. All operations generate appropriate delete/insert notices, and all triggers/procedures' functions found to be working all right.

Inserting Test in Sesame interface
In the second stage, RDF documents are generated to test the insert function. Six tables are tested for insertion. This time we did not test Queries table because there is no interface to insert into Queries table in Sesame. Sesame has three options to insert data. We used the option to type directly to the textbox. The documents are copied and pasted into the textbox of Sesame interface. Since the notifier is not implemented in this project, the results are confirmed with querying the records from Notices table directly in Oracle database.

Abbreviations in the RDF documents are found to be unacceptable by Sesame. In Figure 25, both documents are valid RDF documents, but document (A) was not accepted by Sesame. There seems to be some parsing problem in Sesame. So, we tested without using abbreviation.

Deleting Test in Sesame interface
Deleting test is used the Sesame's option 'remove triple'.

Some constraints are found in deletion transaction of Sesame.

(1) When inserting the records, Sesame automatically generated the full combination of resources. For example, if we express the triple [XMLdocument – SubClassOf - WebPage] where the triple [WebPage- SubClassOf - Document] exists, then Sesame automatically generates the triples [XMLdocument – SubClassOf - Document] and [XMLdocument – SubClassOf - Class] even though this relation is not explicitly described. When deleting the record [XMLdocument – SubClassOf - WebPage], however, we have to explicitly delete two records [XMLdocument – SubClassOf - Document] and [XMLdocument – SubClassOf - Class] as well.

```
(A) RDF document with abbreviation
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xml:base="http://www.waikato.ac.nz/tk27/terms">
  <rdfs:Class rdf:ID="People"/>
  <rdf:Property rdf:ID="hasCreated"/>
</rdf:RDF>

(B) RDF document without abbreviation
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
         xml:base="http://www.waikato.ac.nz/tk27/terms">
  <rdf:Description rdf:ID="People">
    <rdf:type              rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
  </rdf:Description>
  <rdf:Description rdf:ID="hasCreated">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Property"/>
  </rdf:Description>
</rdf:RDF>
```

Figure 25 Abbreviation

(2) Invalid property would remain in Sesame. For example, if class [People] would be deleted where the domain relation [HasWritten – Domain – People] exists, domain relation must be removed, but Sesame requires the deleting statement explicitly. This problem is actually solved by the trigger implemented in this project.

(3) When storing data, definition of namespace simplifies the triple expressions, but removal transaction does not accept namespace and requires the full descriptions (means, with full URI address without namespace).

The functions are tested putting the above constraints into account. The URIs of records are typed into textboxes. The results are confirmed querying the Notices table.

4.2.2. Test Result and Evaluation

In the first test, tables in database are manipulated directly and all the triggers / procedures are confirmed to function in deletion, insertion, and updating. System date storage is confirmed to function appropriately. Queries handled by triggers/procedures of Database are confirmed to function appropriately without the dedicated query languages for RDF documents (e.g. RQL, SeRQL). For the multi-level domain/range relations, triple are confirmed to indicate two different properties accordingly. Triggers/procedures extracted the relevant relations appropriately upon insertion/deletion where the sub-super relations exist. In short, the triggers/procedures were found to execute the expected works appropriately.

Then the tests are carried out through Sesame interface. In both insertion and deletion, the triggers/procedures are confirmed to fire the event notices appropriately.

It was not the focus of this project to measure the performance, but it is worth to report it. Fairly saying, processing speed in the system is very slow. There are two potential reasons. Because all records in SubClassOf table can be derived from Direct_SubClassOf table, SubClassOf table is not really necessary. SubClassOf table holds all combination of sub-super relation derived from Direct_SubClassOf table, so the number of records exponentially increases suffering scalability. Same problem exists in SubPropertyOf table and InstanceOf table. Records in Triples table can be derived from other tables, too. There are total of 29 resources initially set by Sesame into Resources table. Only 29 records generate extra 432 records in other tables (that makes 461 records in total!). (Figure 26) This enormous number of records is considered to be the main reason of the slow processing.

Table	**Number of records**
Namespaces	2
Resources	29
Property	14
Direct_SubPropertyOf	1
Sub Property Of	15
Class	13
Direct_SubClassOf	12
SubClassOf	31
Domain	9
Range	8
Proper_InstanceOf	28
InstanceOf	57
Literal	0
Triples	120
Depend	121
RepInfo	1
Total	**461**

The second reason possibly is the algorithm to check the super class/property. As we discussed in Section 3.6, we used Direct_SubClassOf / Direct_SubPropertyOf table to find the super class / super property. These tables store only the direct sub-super relations. So, the system has to access to the table while super class/property of key resource exists. This repetitive operation possibly is another reason to spoil the processing speed. However, if we would use SubClassOf / SubPropertyOf tables, then the number of insertion/deletion shall increases. So, it is unsure which approach is more efficient.
Considering the number of Web pages existing in the World Wide Web, scalability is the big issue for the Semantic Web. Further development of more efficient platform and performance test of the system is expected.

Through the tests in database, we confirmed that the insertion, deletion, and updating functionalities work appropriately without any errors (including mutation). The triples in two different levels are recognised and extracted to the valid domain/range relations. The tests are successful in Sesame as well. Events inserted via Sesame's interface are appropriately filtered in our filter engine (according to the user profiles) and the matching events are inserted into Notices table.
One concern is the processing speed. It might negatively influence the scalability of the whole system. We propose to investigate alternative designs to database triggers in future work (see next Section).

5. Summaries and Future Work

Through this project, we completed four tasks. Our achievements are reported in Section 5.1. A summary of issues observed through the project implementation and recognised future works are discussed in Section 5.2. Finally in Section 5.3, we also summarise our recommendations for further improvements of the Sesame system.

5.1. Achievements

Our major achievements in this project are as follows:

(1) Design and Implementation of a Event Notice System for RDF.
We add the feature to reflect the changes made on RDF documents to the search results upon inserting/deleting/updating.

(2) Synchronized Deleting
When the user removes the class from Class table, for example, 'People', System automatically delete it from Domain, Range, Direct_SubClassOf, SubClassOf table without explicit removal query.

(3) Expression of multi level triple
Where the domain /range relations exist in the different level of super property, our system explicitly indicates the properties which has those relations.

(4) Query language-free solution
Since the triggers/procedures extract the relevant records to the user's query from database, the system does not need the extra query language. Upon designing user interface to accept the user's query in the form of [subject – predicate, object], the system will be free from developing further query languages.

5.2. Future Work on the Event Notification System

Here, we propose future steps to enhance our ENS:

(1) Extension for ontology
In this project, we did not deal with Ontology, as we did not wish to complicate the event notice engine. However it does not mean it is minor issue; rather it is very important part to interpret the RDF documents. Consider the three triples shown in Figure 11. These three sentences have same meaning. Then where the sentence (1) is valid, the document, which contains sub classes/property of sentence (2), is also valid. Where the sentence order is reversed and the sentence (1) is valid, the document, which contains sub classes/properties of sentence (3), is also valid. So, the involvement of Ontology changes the outcome of the Adaptive Semantic Web. We need to put this factor into account in the future study.

(2) Limited notification system
Our system is confirmed to observe the events and filter them. However, the interaction between Sesame is not completed. The query and requester data need to be inserted into the database using SQL language. Event notices, which are stored in the Notices table, should be delivered to users.

Involving the user interface design in Sesame, we need to upgrade our system as to work together with Sesame.

(3) Partial synchronization on triples manipulation.
We synchronised the deletion of property/class to the four tables, which are Domain, Range, Direct_SubClassOf, and Direct_SubPropertyOf tables. Because they would affect on the results of filtering we updated these tables. However, there are another tables, which contain the deleted resource. We need to analyze the relation of whole table, and should maintain other tables as well to maintain the consistency among tables.

(4) Scalability
The planned functionality of the system has been achieved. However, there exists the performance problem as we introduced in Section 4.2.2. Reasons for the slow processing probably are: Sesame's database table design, or trigger algorithm, or both. ORDB is considered to be faster in processing speed than RDB. So we initially intended to implement using ORDB, but changed the plan, as we found Sesame does not support ORDB in the latest version. As for the future work, ORDB needs to be studied adaptability to the Adaptive Semantic Web as well as the performance of ORDB with OQL language.

5.3. Future Work on Sesame System

Sesame is found to be good tool to experience the taste of the Semantic Web. However, some practical problems, which have to be solved to use in real world, are observed.

(1) Scalability
As we saw in Section 4.2.2, the database schema contains the large amount of redundant records, which shall cause the exponential increase of records. This problem is considered to be the critical fault to handle the large size of data spreading over the World Wide Web. The table design is seriously expected to improve.

(2) Usability in deleting/Updating operation
Sesame introduces the engine to implement deleting transaction. This engine requires user to type in the deleting triples in the form of URI. Where the insertion action automates the generation of inferred triples, it is hard for users to clear out such inferred triples. Simple and practical solution is expected for this issue. Considering the usability, the best solution is to analyse the removal of the sentence from RDF documents, and justify. Further development is expected for this theme.
Updating is not available in current version. Update is the combination of the deletion and insertion. It would be hard until the development of the automated deletion program. Further study of method to interpret the changes of the RDF documents is expected.

(3) Adaptability of abbreviation of the RDF documents
Abbreviation currently is not interpreted into database appropriately. It is recommended to upgrade the version to adapt to the valid abbreviation.

(4) Integrity to Literal
In current model, the literal does not have any referential integrity with resources. So, the key information like name or title are not extractable. In real situation, the user should need these information relating to resources. So, it would be recommended to keep referential integrity between literal and resources.

(5) Referential integrity
Foreign key cause error when data attempted to be removed. Even when cascading option was set on foreign key, still the foreign key causes error. It is easier for the researcher to develop the system further with analyse of the foreign keys, it would recommended letting Sesame work with the foreign key (if it is possible).

Acknowledgements

The project reported in this working paper has been performed by Takanori Kozuka as Dissertation for a Postgraduate Diploma in Computer Science. The project was supervised by Annika Hinze.

6. Bibliography

RDF and RDF Schema

[1] Eric Miller (1998) "An Introduction to the Resource Description Framework", D-Lib Magazine, May 1998, available at http://www/dlib.org/dlib/may98/miller/05miller.html

[2] Jeen Broekstra, Arjohn Kampman, Frank van Harmelen, (2001) “Sesame: An Architecture for Storing and Querying RDF Data and Schema Information”, MIT Press, Sep 2001, available at http://citeseer.nj.nec.com/broekstra01sesame.html

[3] Gregory Karvounarakis, Vassilis Christophides, Dimitris Plexousakis, Sofia Alexaki (2001) “Querying RDF Descriptions for Community Web Portals”, In Proceedings of the BDA'2001 (17iemes Journees Bases de Donnees Avances - French Conference on Databases), Agadir, Morocco, 2001, available at http://www.ics.forth.gr/isl/publications/paperlink/bda2001.pdf

[4] Gregory Karvounarakis, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl (2002) "RQL: a declarative query language for RDF", Proceedings of the 11th international conference on World Wide Web, May 2002, available at http://portal.acm.org.ezproxy.waikato.ac.nz:2048/ft_gateway.cfm?id=511524&type=pdf&coll=portal&dl=ACM&CFID=31607531&CFTOKEN=44744179

[5] Valerie Bonstrom, Annika Hinze, Heinz Schweppe (2003) “Storing RDF as a Graph”, First Latin American Web Congress (LA-WEB 2003) , Apr. 2003, available at http://www.la-web.org/2003/stamped/04_boenstroem_v.pdf

[6] Dan Brickley, R.V. Guha, Brian McBride (2004) “RDF Vocabulary Description Language 1.0: RDF Schema”, W3C recommendation, Feb.2004, available at www.w3.org/TR/rdf-schema/

[7] Frank Manola, Eric Miller, Brian McBride (2004) “RDF Primer”, W3C recommendation, Feb.2004, available at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/

[8] Aduna B.V. (2004) “User Guide for Sesame”, Sirma AI Ltd

[9] Jeen Broekstra, “Sesame RQL: A Tutorial”, available at http://openrdf.org/doc/rql-tutorial.html

[10] Jeen Broekstra, Arjohn Kampman, Frank van Harmelen (2002) "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema", In Proceedings International Semantic Web Conference 2002 available at http://www.openrdf.org/doc/papers/Sesame-ISWC2002.pdf

Ontology

[11] Michael K Smith, Chris Welty, Deborah L McGuinness (2004) "OWL Web Ontology Language Guide", W3C recommendation 2004, Available at http://www.w3.org/TR/owl-guide/

The Semantic Web

[12] Tim Berners-Lee, James Hendler, Ora Lassila, (2001), "The Semantic Web", Scientific American, May 2001, available at http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[13] Aaron Swartz (2002), "The Semantic Web In Breadth", HTML documents found on Sep 2004, available at http://logicerror.com/semanticWeb-long

[14] A.M. Kuchling (2004), "Introduction to the Semantic Web and RDF", ZPUG DC, Dec 2004, available at http://www.amk.ca/

Change Detection

[15] Peter Dolog, Nicola Henze, Wolfgang Ndjdl, Michael Sintek, (2003) "Towards the Adaptive Semantic Web", 1st Workshop on Principles and Practice of Semantic Web Reasoning. (2003) available at http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/2003/ppswr03.pdf

[16] George Papamarkos, Alexandra Poulovassillis, Peter T. Wood, "Event-Condition Action Rule Language for the Semantic Web", Proceedings of SWDB' 03 Very Large Data Bases The first International Workshop on
Semantic Web and Databases, Sep 2003

[17] Li Qin, Vijayalakshmi Atluri (2004) "Ontology-guided Change Detection to the Semantic Web Data", 23rd International Conference on Conceptual Modeling (ER2004), Shanghai, China, Nov 2004

Database

[18] Joseph S. Valacich, Joey F. George, Jeffrey A. Hoffer (2001) "Essentials of System Analysis & Design", Prentice Hall, Published July 2000

[19] Norman W. Paton, Oscar Diaz (1999) "Active database systems", ACM Computing Surveys (CSUR), Volume 31 Issue 1,March 1999, available at http://portal.acm.org.ezproxy.waikato.ac.nz:2048/citation.cfm?id=311623&coll=portal&dl=ACM&CFID=31607531&CFTOKEN=44744179

Filtering

[20] V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, F. Wattez (2000) “Querying XML Documents in Xyleme” ACMSIGIR Workshop, July 2000, available at http://www.haifa.il.ibm.com/sigir00-xml/final-papers/xyleme/XylemeQuery/XylemeQuery.html

[21] Conor Hayes, Barry Smyth (2001) "A Case-Based Reasoning View of Automated Collaborative Filtering", In Proceedings of 4th International Conference on Case Based Reasoning, ICCBR2001 (2001) 243-248, available at http://www.cs.tcd.ie/publications/tech-reports/reports.01/TCD-CS-2001-09.pdf

[22] Gregory Cobena, Amelie Marian (2002) "Detecting Changes in XML Document", In 18th Int.l Conf. on Data Engineering (ICDE 2002), 2002

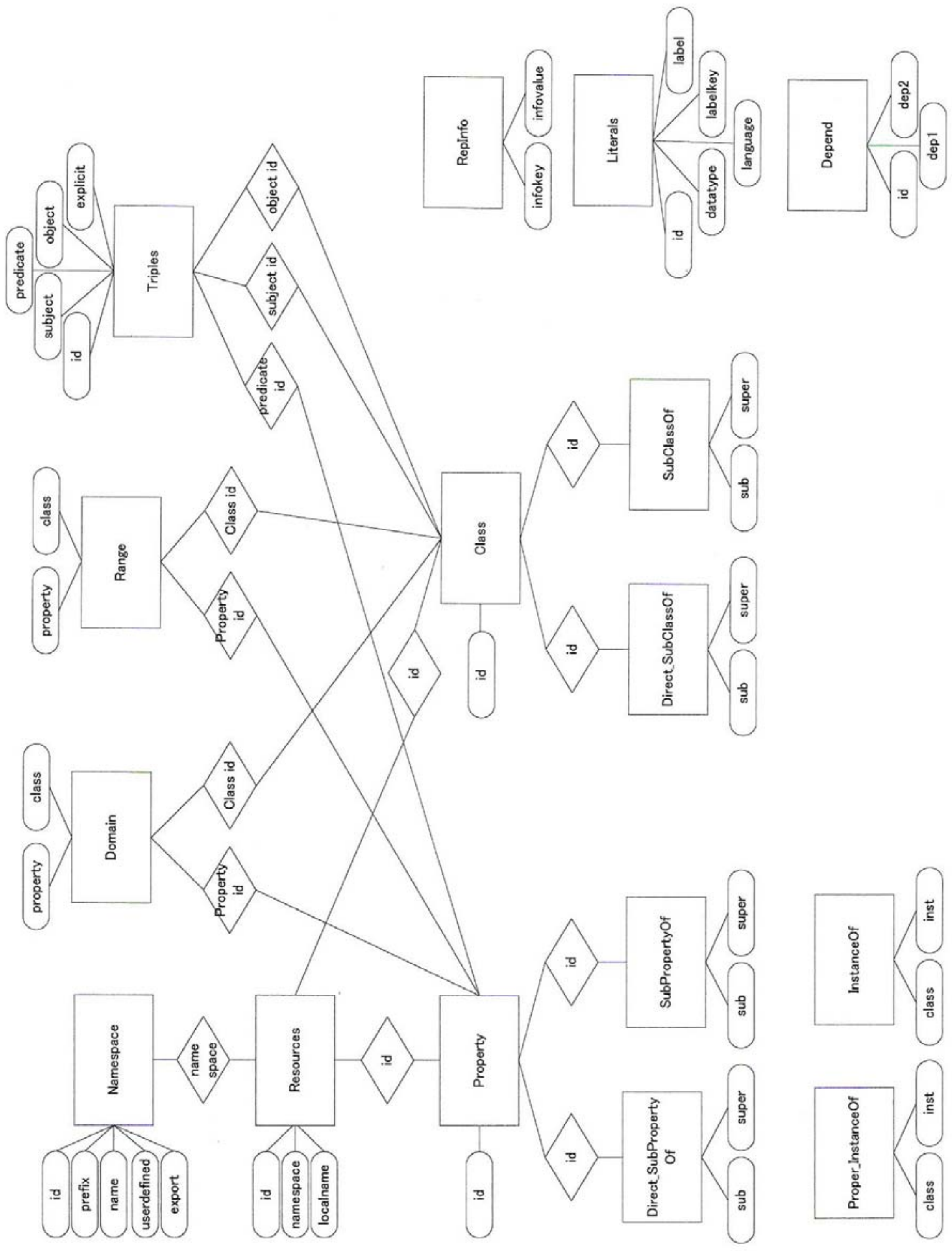
[23] Annika Hinze (2003) “AmediAS: An Adaptive Event Notification System”, Proceedings of the 2nd international workshop on Distributed event-based systems, July 2003

7. Appendices

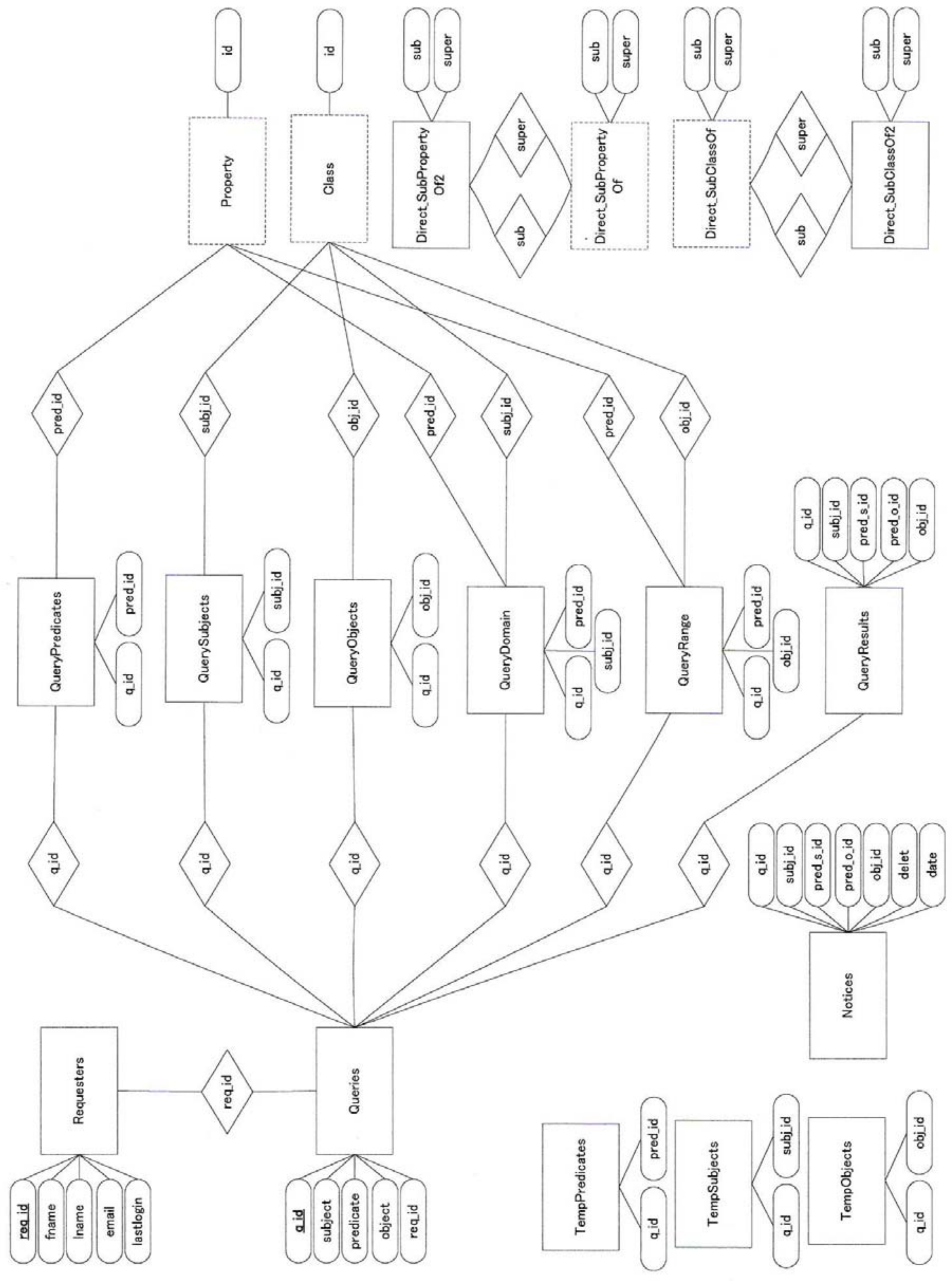
Appendix A. Database Design
Appendix B. Flowchart – Triggers and Procedures
Appendix C. Relation among Triggers and Procedures

Appendix A: Database Design

A1.Entity Relation Diagram for Sesame Database



A2. Entity-Relation Diagram for Event Notification Engine



A3. Table Design

1. Requesters

Attribute	Data Type	null	Constraint / Remarks
req_id	number (38)	not null	Primary key Requester ID
fname	varchar2 (100)	not null	First name of requester
lname	varchar2 (100)	not null	Last name of requester
email	varchar2 (255)	not null	E-mail address of requester
lastlogin	datatime	not null	sysdate, date and time to log-in last time

2. Queries

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key Query ID
subject	varchar2 (100)	not null	Subject in string
predicate	varchar2 (100)	not null	Predicate in string
object	varchar2 (100)	not null	Object in string
req_id	number (38)	not null	Requester ID Refer Requesters (req_id)

3. QuerySubjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
subj_id	number (38)	not null	Subject ID Refer Class (id)

Combination of q_id and subj_id must be unique

4. QueryPredicates

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key / Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate in string Refer Property (id)

Combination of q_id and pred_id must be unique

5. QueryObjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key / Query ID Refer Queries (q_id)
obj_id	number (38)	not null	Object ID Refer Class (id)

Combination of q_id and obj_id must be unique

6. TempSubjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
subj_id	number (38)	not null	Subject ID

7. TempPredicates

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
pred_id	number (38)	not null	Predicate in string

8. TempObjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
obj_id	number (38)	not null	Object ID

9. QueryDomain

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate ID Refer Domain (property)
subj_id	number (38)	not null	Subject ID Refer Domain (class)

Combination of q_id, pred_id and subj_id must be unique

10. QueryRange

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate ID Refer Range (property)
obj_id	number (38)	not null	Subject ID Refer Range (class)

Combination of q_id, pred_id and obj_id must be unique

11. Requesters

Attribute	Data Type	null	Constraint / Remarks
req_id	number (38)	not null	Primary key Requester ID
fname	varchar2 (100)	not null	First name of requester
lname	varchar2 (100)	not null	Last name of requester
email	varchar2 (255)	not null	E-mail address of requester
lastlogin	datatime	not null	sysdate, date and time to log-in last time

12. Queries

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key Query ID
subject	varchar2 (100)	not null	Subject in string
predicate	varchar2 (100)	not null	Predicate in string
object	varchar2 (100)	not null	Object in string
req_id	number (38)	not null	Requester ID Refer Requesters (req_id)

13. QuerySubjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
subj_id	number (38)	not null	Subject ID Refer Class (id)

Combination of q_id and subj_id must be unique

14. QueryPredicates

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key / Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate in string Refer Property (id)

Combination of q_id and pred_id must be unique

15. QueryObjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Primary key / Query ID Refer Queries (q_id)
obj_id	number (38)	not null	Object ID Refer Class (id)

Combination of q_id and obj_id must be unique

16. TempSubjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
subj_id	number (38)	not null	Subject ID

17. TempPredicates

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
pred_id	number (38)	not null	Predicate in string

18. TempObjects

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
obj_id	number (38)	not null	Object ID

19. QueryDomain

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate ID Refer Domain (property)
subj_id	number (38)	not null	Subject ID Refer Domain (class)

Combination of q_id, pred_id and subj_id must be unique

20. QueryRange

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
pred_id	number (38)	not null	Predicate ID Refer Range (property)
obj_id	number (38)	not null	Subject ID Refer Range (class)

Combination of q_id, pred_id and obj_id must be unique

21. QueryResults

Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID Refer Queries (q_id)
subj_id	number (38)	not null	Class ID for Subject Refer QueryDomain (subj_id)
pred_s_id	number (38)	not null	Property ID which makes domain relation Refer QueryDomain (pred_id)
pred_o_id	number (38)	not null	Property ID which makes range relation Refer QueryRange (pred_id)
obj_id	number (38)	not null	Class ID for Object Refer QueryRange (obj_id)

Combination of q_id, subj_id, pred_s_id, pred_o_id and obj_id must be unique.

22. Notices

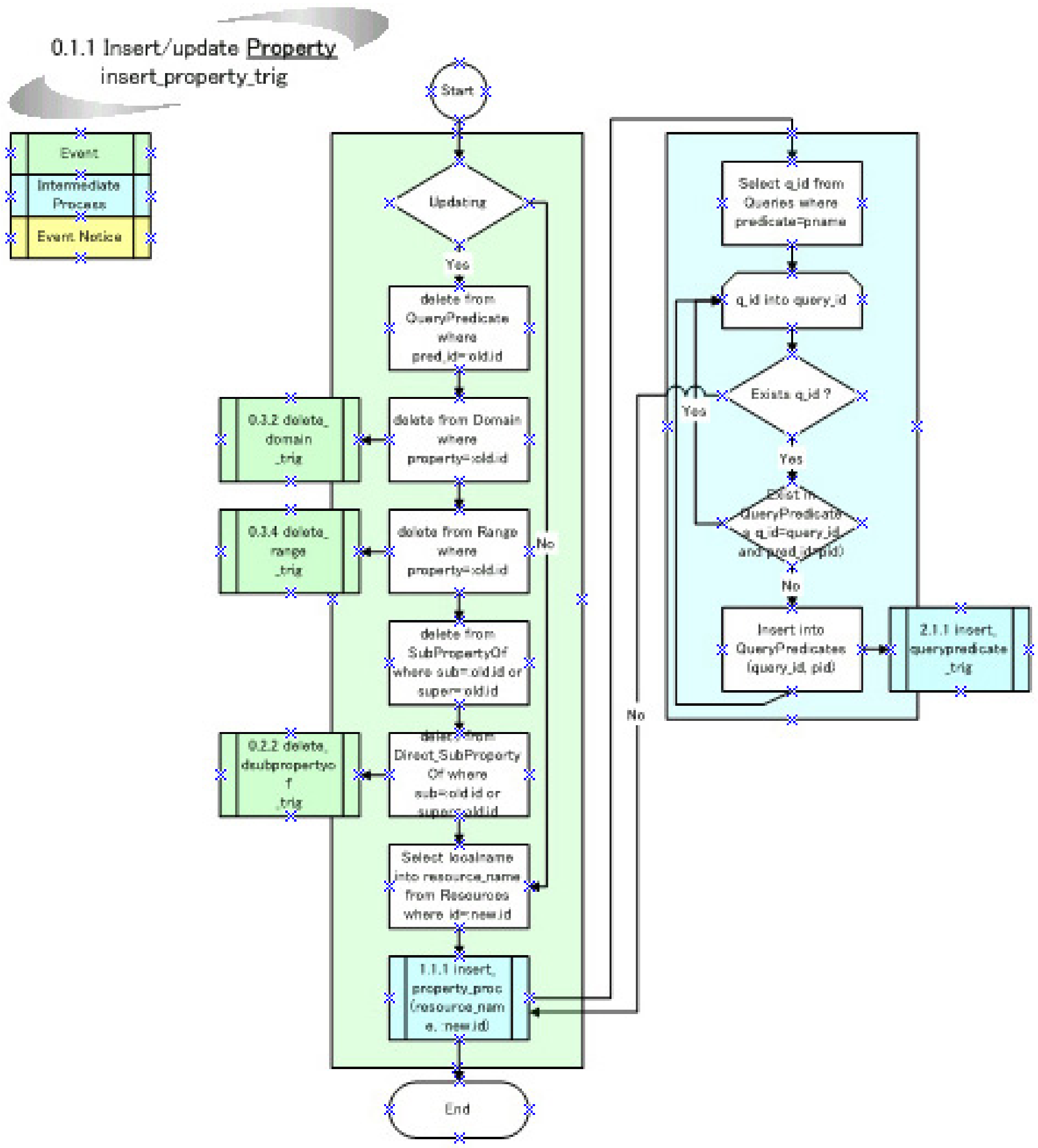
Attribute	Data Type	null	Constraint / Remarks
q_id	number (38)	not null	Query ID
delet	number (1)	not null	1=delete, 0=insert
datefound	date	not null	Sysdate, data that system inserts this record
subj_id	number (38)	not null	Class ID for Subject
pred_s_id	number (38)	not null	Property ID which makes domain relation
pred_o_id	number (38)	not null	Property ID which makes range relation
obj_id	number (38)	not null	Class ID for Object

23. Direct_SubPropertyOf2

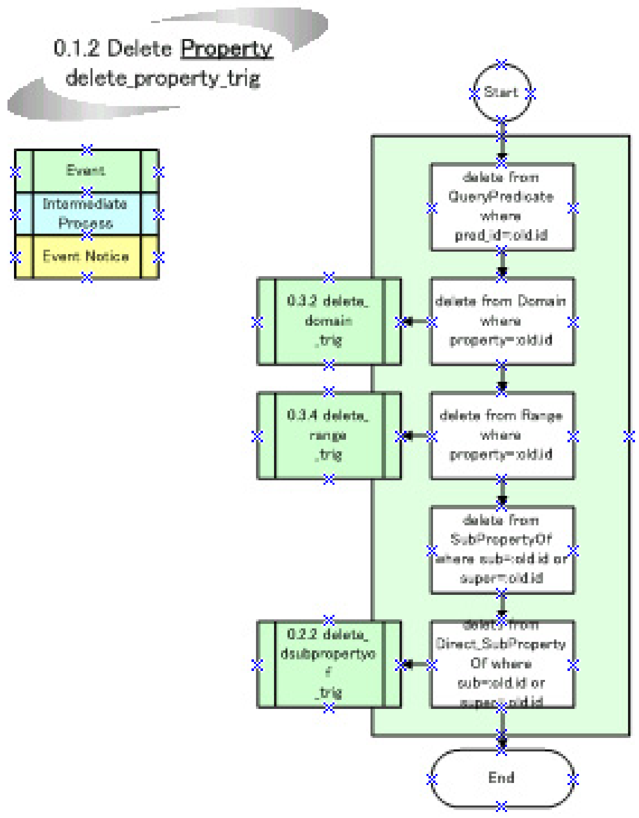
Attribute	Data Type	null	Constraint / Remarks
sub	number (38)	not null	Sub property ID
super	number (38)	not null	Super property ID

24. Direct_SubClassOf2

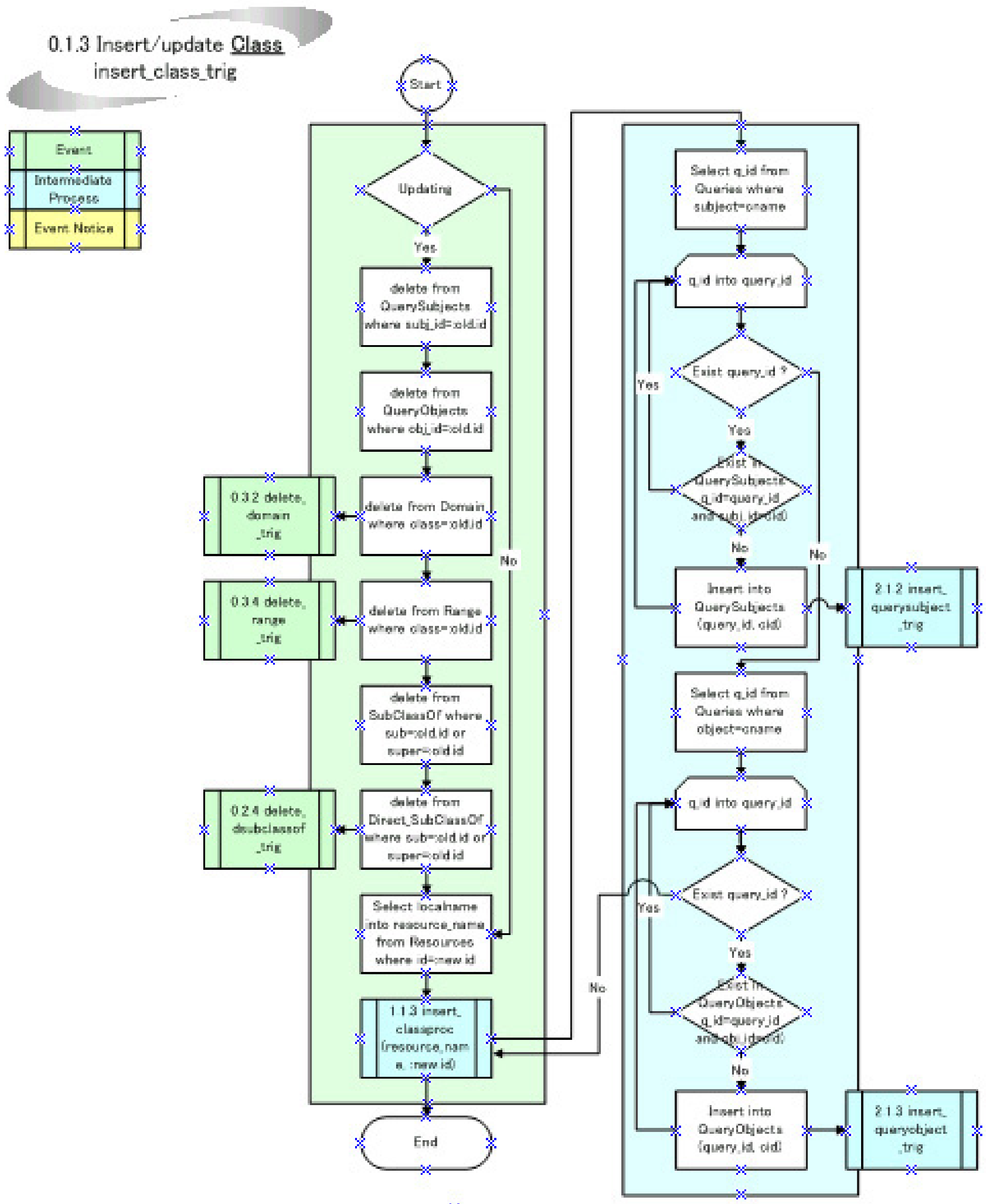
Attribute	Data Type	null	Constraint / Remarks
sub	number (38)	not null	Sub class ID
super	number (38)	not null	Super class ID


0.1.1 Insert/update Property
insert_property_trig
Event
Intermediate Process
Event Notice
Start
Updating
Yes
No
delete from QueryPredicate where pred_id=old.id
delete from Domain where property=old.id
0.3.2 delete_domain_trig
delete from Range where property=old.id
0.3.4 delete_range_trig
delete from SubPropertyOf where sub=old.id or super=old.id
delete from Direct_SubPropertyOf where sub=old.id or super=old.id
0.2.2 delete_dsubpropertyof_trig
Select localname into resource_name from Resources where id=new.id
1.1.1 insert_property_proc (resource_name, new.id)
End
Select q_id from Queries where predicate=pname
q_id into query_id
Exists q_id ?
Yes
No
Exist in QueryPredicate (q_id=query_id and pred_id=pid)
Yes
No
Insert into QueryPredicates (query_id, pid)
2.1.1 insert_querypredicate_trig

```
0.1.1 insert_property_trig
Event: insert or update into Table: Property
Role:
  For updating, Delete corresponding property from five tables, and fire
  the trigger of them
  Find the property name and pass to the procedure for comparing to the
  predicate name of existing queries.

--------------------------------------------------------------------

1.1.1 insert_property_proc
Role:
  Insert property_id into QueryPredicates Table when new property name
  matches to the predicate of the Queries Table.
```

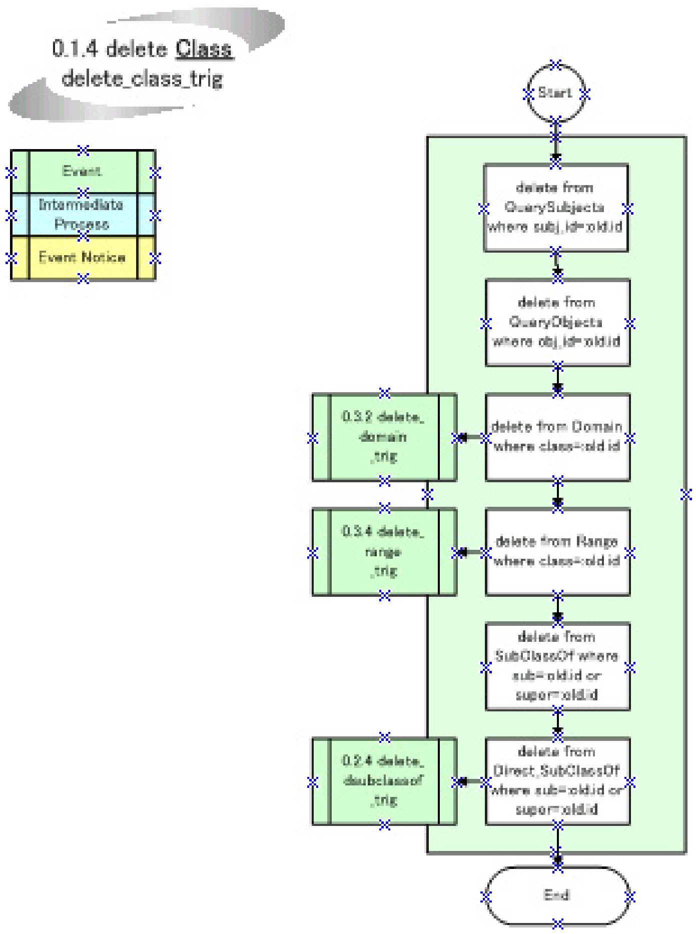

0.1.2 Delete Property
delete_property_trig
Event
Intermediate Process
Event Notice
Start
delete from QueryPredicate where pred_id=:old.id
delete from Domain where property=:old.id
0.3.2 delete_domain_trig
delete from Range where property=:old.id
0.3.4 delete_range_trig
delete from SubPropertyOf where sub=:old.id or super=:old.id
delete from Direct_SubPropertyOf where sub=:old.id or super=:old.id
0.2.2 delete_dsubpropertyof_trig
End

```
0.1.2 delete_property_trig
Event: delete from Table: Property
Role:
  Delete corresponding property from five tables, and fire the triggers
  of them.
```

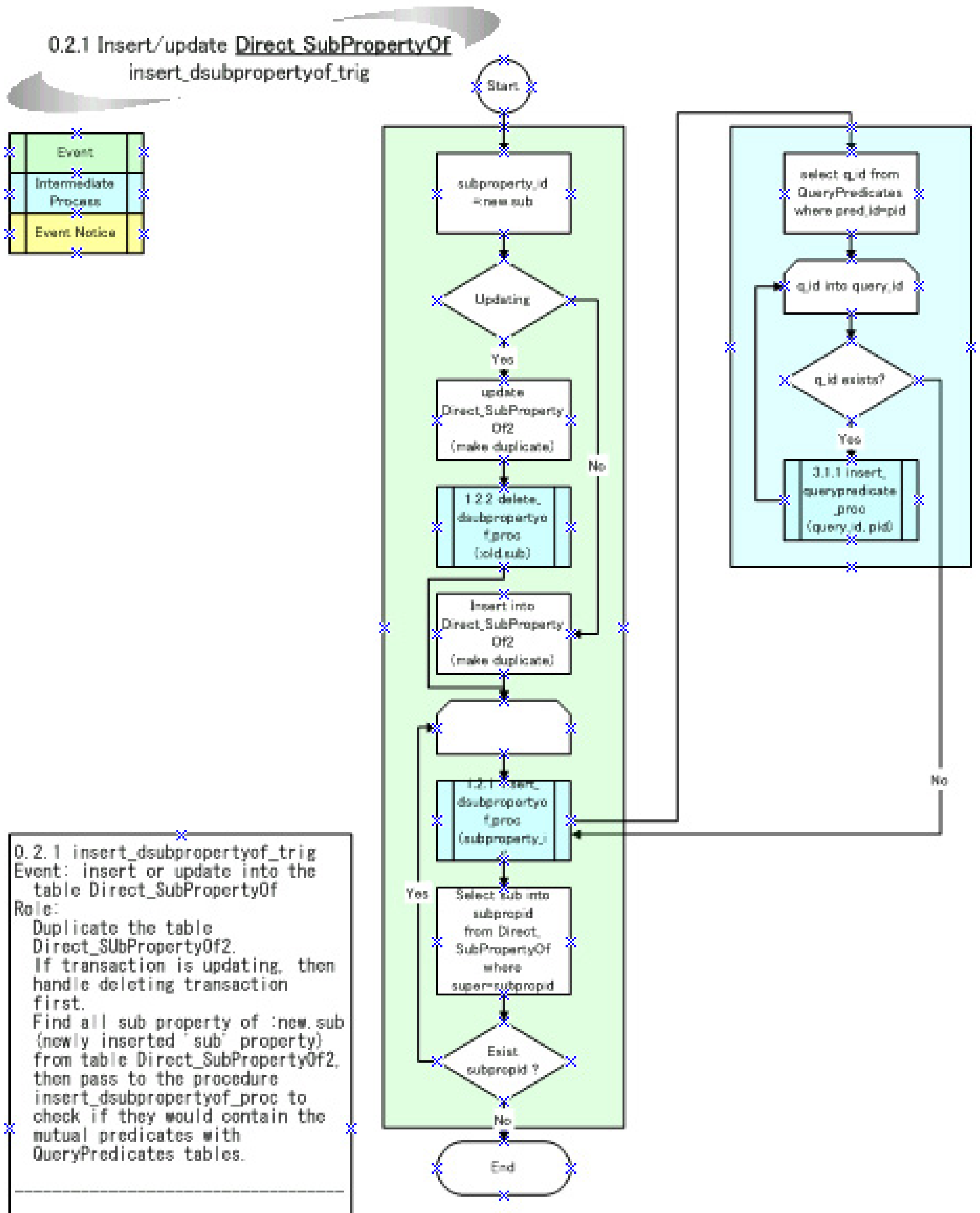

0.1.3 Insert/update Class
insert_class_trig
Event
Intermediate Process
Event Notice
Start
Updating
Yes
No
delete from QuerySubjects where subj_id=:old.id
delete from QueryObjects where obj_id=:old.id
delete from Domain where class=:old.id
0.3.2 delete_domain_trig
delete from Range where class=:old.id
0.3.4 delete_range_trig
delete from SubClassOf where sub=:old.id or super=:old.id
delete from Direct_SubClassOf where sub=:old.id or super=:old.id
0.2.4 delete_dsubclassof_trig
Select localname into resource_name from Resources where id=:new.id
1.1.3 insert_classproc (resource_name, :new.id)
End
Select q_id from Queries where subject=cname
q_id into query_id
Exist query_id ?
Yes
Exist in QuerySubjects (q_id=query_id and subj_id=cid)
No
Insert into QuerySubjects (query_id, cid)
2.1.2 insert_querysubject_trig
Select q_id from Queries where object=cname
q_id into query_id
Exist query_id ?
Yes
Exist in QueryObjects (q_id=query_id and obj_id=cid)
No
Insert into QueryObjects (query_id, cid)
2.1.3 insert_queryobject_trig

0.1.3 insert_class_trig
Role:
For updating, Delete corresponding class from six tables, and fire the trigger of them Find the Class name and pass to the procedure to compare the classname to existing queries.

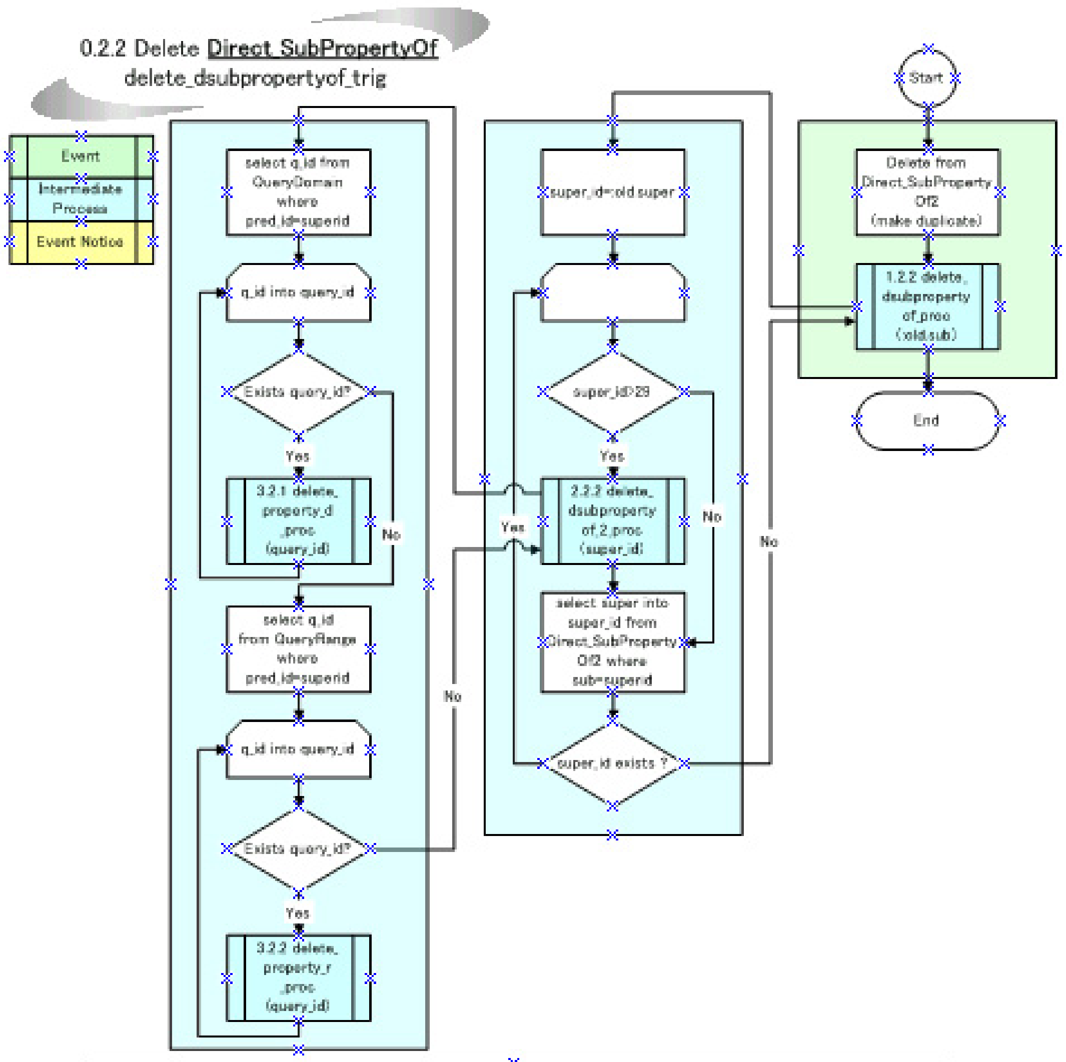
1.1.3 insert_class_proc
Role:
Insert class_id into QuerySubjects Table when new class name matches to the subject of the Queries Table. Insert class_id into Queryobjects Table when new class name matches to the object of the Queries Table.


0.1.4 delete Class
delete_class_trig
Event
Intermediate Process
Event Notice
Start
delete from QuerySubjects where subj_id=:old.id
delete from QueryObjects where obj_id=:old.id
delete from Domain where class=:old.id
0.3.2 delete_ domain _trig
delete from Range where class=:old.id
0.3.4 delete_ range _trig
delete from SubClassOf where sub=:old.id or super=:old.id
delete from Direct_SubClassOf where sub=:old.id or super=:old.id
0.2.4 delete_ dsubclassof _trig
End

```
0.1.4 delete_class_trig
Event: delete from Table: Class
Role:
  Delete corresponding class from six tables, and fire the trigger of them
```


0.2.1 Insert/update Direct SubPropertyOf
insert_dsubpropertyof_trig
Event
Intermediate Process
Event Notice
Start
subproperty_id =new.sub
Updating
Yes
No
update Direct_SubProperty Of2 (make duplicate)
1.2.2 delete_dsubpropertyof_proc (:old.sub)
Insert into Direct_SubProperty Of2 (make duplicate)
1.2.1 insert_dsubpropertyof_proc (subproperty_id)
Select sub into subpropid from Direct_SubPropertyOf where super=subpropid
Exist subpropid ?
Yes
No
End
select q_id from QueryPredicates where pred_id=pid
q_id into query_id
q_id exists?
Yes
No
3.1.1 insert_querypredicate_proc (query_id, pid)

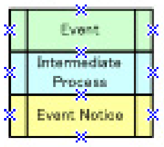
```
0.2.1 insert_dsubpropertyof_trig
Event: insert or update into the
  table Direct_SubPropertyOf
Role:
  Duplicate the table
  Direct_SUbPropertyOf2.
  If transaction is updating, then
  handle deleting transaction
  first.
  Find all sub property of :new.sub
  (newly inserted 'sub' property)
  from table Direct_SubPropertyOf2,
  then pass to the procedure
  insert_dsubpropertyof_proc to
  check if they would contain the
  mutual predicates with
  QueryPredicates tables.
------------------------------------

1.2.1 insert_dsubpropertyof_proc
Role:
  Given the property ID of sub
  property, check if they would be
  contained in the QueryPredicates
  table. If contained, pass to
  procedures
  insert_querypredicate_proc to
  insert into QueryDomain/
  QueryRange tables.
```

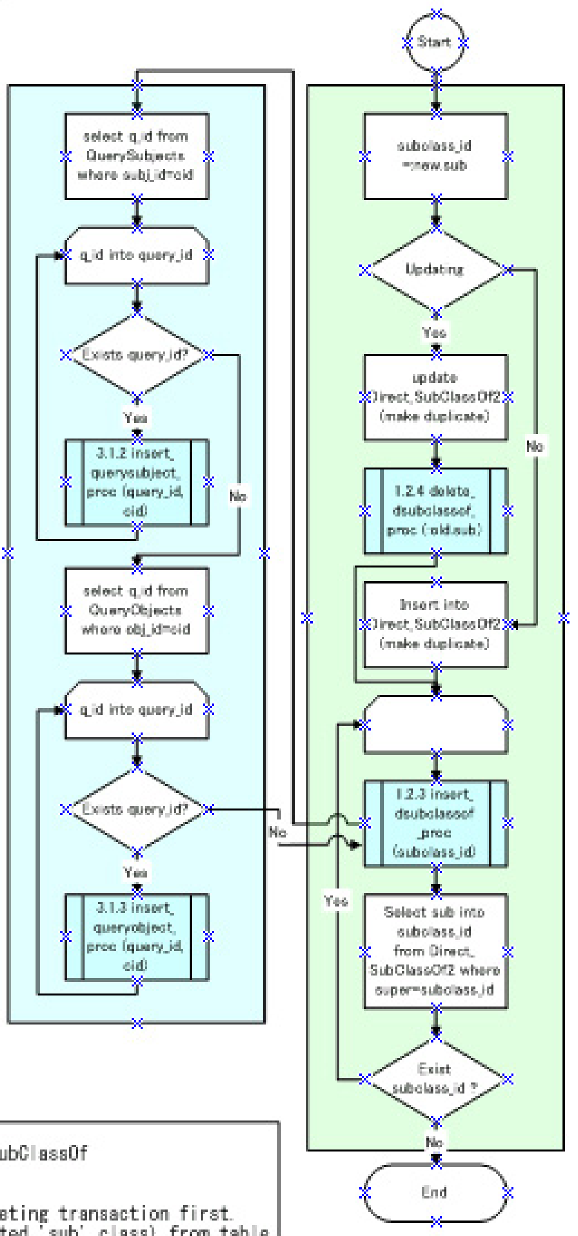

0.2.2 Delete Direct SubPropertyOf
delete_dsubpropertyof_trig
Event
Intermediate Process
Event Notice
Start
Delete from Direct_SubProperty Of2 (make duplicate)
1.2.2 delete_ dsubproperty of_proc (old.sub)
End
super_id=:old.super
super_id>29
Yes
No
2.2.2 delete_ dsubproperty of_2_proc (super_id)
select super into super_id from Direct_SubProperty Of2 where sub=superid
super_id exists ?
Yes
No
select q_id from QueryDomain where pred_id=superid
q_id into query_id
Exists query_id?
Yes
No
3.2.1 delete_ property_d _proc (query_id)
select q_id from QueryRange where pred_id=superid
q_id into query_id
Exists query_id?
Yes
No
3.2.2 delete_ property_r _proc (query_id)

```
0.2.2 delete_dsubpropertyof_trig
Event: delete from the table Direct_SubPropertyOf
Role:
  Duplicate the table Direct_SUbPropertyOf2.
  Call procedure, which is shared with update trigger

--------------------------------------------------------------------------

1.2.2 delete_dsubpropertyof_proc
Role:
  Records of super propertiess of deleted property will be deleted by
  procedure delete_dsubpropertyof_2_proc from Query tables (i.e.
  QuerySubjects, QueryObjects, QueryDomain, QueryRange, and QueryResults)

--------------------------------------------------------------------------

2.2.2 delete_dsubpropertyof_2_proc
Role:
  Given the super property :old.sub (deleted property), if any of them
  are contained in the QueryDomain/QueryRange, find the query ID of it.
  Then pass query ID to the procedure delete_property_d_proc/
  delete_property_r_proc.
```

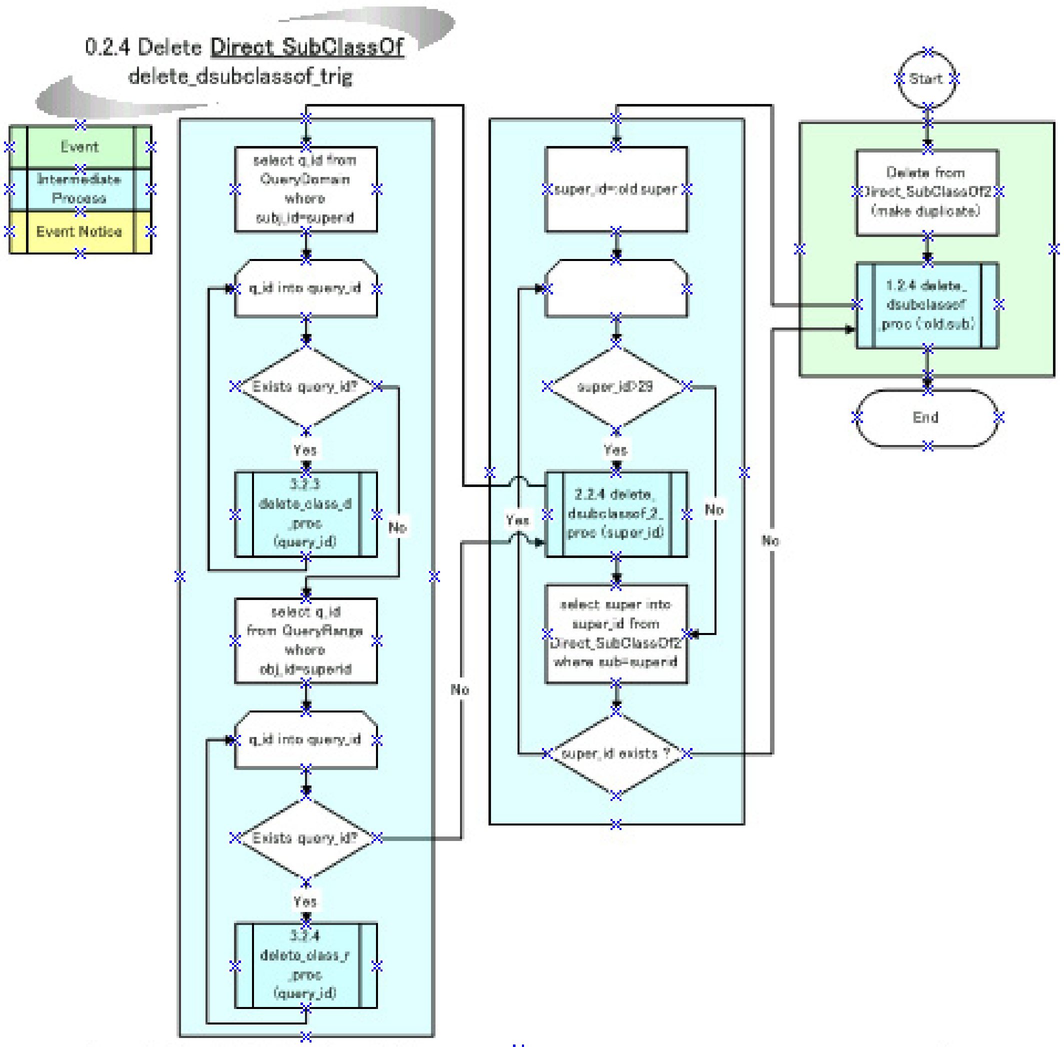
0.2.3 Insert/update Direct_SubClassOf

insert_dsubclassof_trig




Start
subclass_id =:new.sub
Updating
Yes
update Direct_SubClassOf2 (make duplicate)
1.2.4 delete_dsubclassof_proc (:old.sub)
No
Insert into Direct_SubClassOf2 (make duplicate)
1.2.3 insert_dsubclassof_proc (subclass_id)
Select sub into subclass_id from Direct_SubClassOf2 where super=subclass_id
Yes
Exist subclass_id ?
No
End
select q_id from QuerySubjects where subj_id=cid
q_id into query_id
Exists query_id?
Yes
3.1.2 insert_querysubject_proc (query_id, cid)
No
select q_id from QueryObjects where obj_id=cid
q_id into query_id
Exists query_id?
Yes
3.1.3 insert_queryobject_proc (query_id, cid)
No

```
0.2.3 insert_dsubclassof_trig
Event: insert or update on the table Direct_SubClassOf
Role:
  Duplicate the table Direct_SUbClassOf2.
  If transaction is updating, then handle deleting transaction first.
  Find all sub class of :new.sub (newly inserted 'sub' class) from table
  Direct_SubClassOf2, then pass to the procedure insert_dsubclassof_proc
  to check if they would contain the mutual subjects/objects with
  QuerySubjects/QueryObjexts tables.

--------------------------------------------------------------------------------

1.2.3 insert_dsubclassof_proc
Role:
  Given the class ID of sub class, check if they would be contained in
  the QuerySubjects/QueryObjects table. If contained, pass to rocedures
  insert_querysubject_proc/insert_queryobject_proc to insert into
  QueryDomain/QueryRange tables.
```


0.2.4 Delete Direct SubClassOf
delete_dsubclassof_trig
Event
Intermediate Process
Event Notice
Start
Delete from Direct_SubClassOf2 (make duplicate)
1.2.4 delete_dsubclassof_proc (:old.sub)
End
super_id=:old.super
super_id>29
Yes
No
2.2.4 delete_dsubclassof_2_proc (super_id)
select super into super_id from Direct_SubClassOf2 where sub=superid
super_id exists ?
select q_id from QueryDomain where subj_id=superid
q_id into query_id
Exists query_id?
3.2.3 delete_class_d_proc (query_id)
select q_id from QueryRange where obj_id=superid
3.2.4 delete_class_r_proc (query_id)

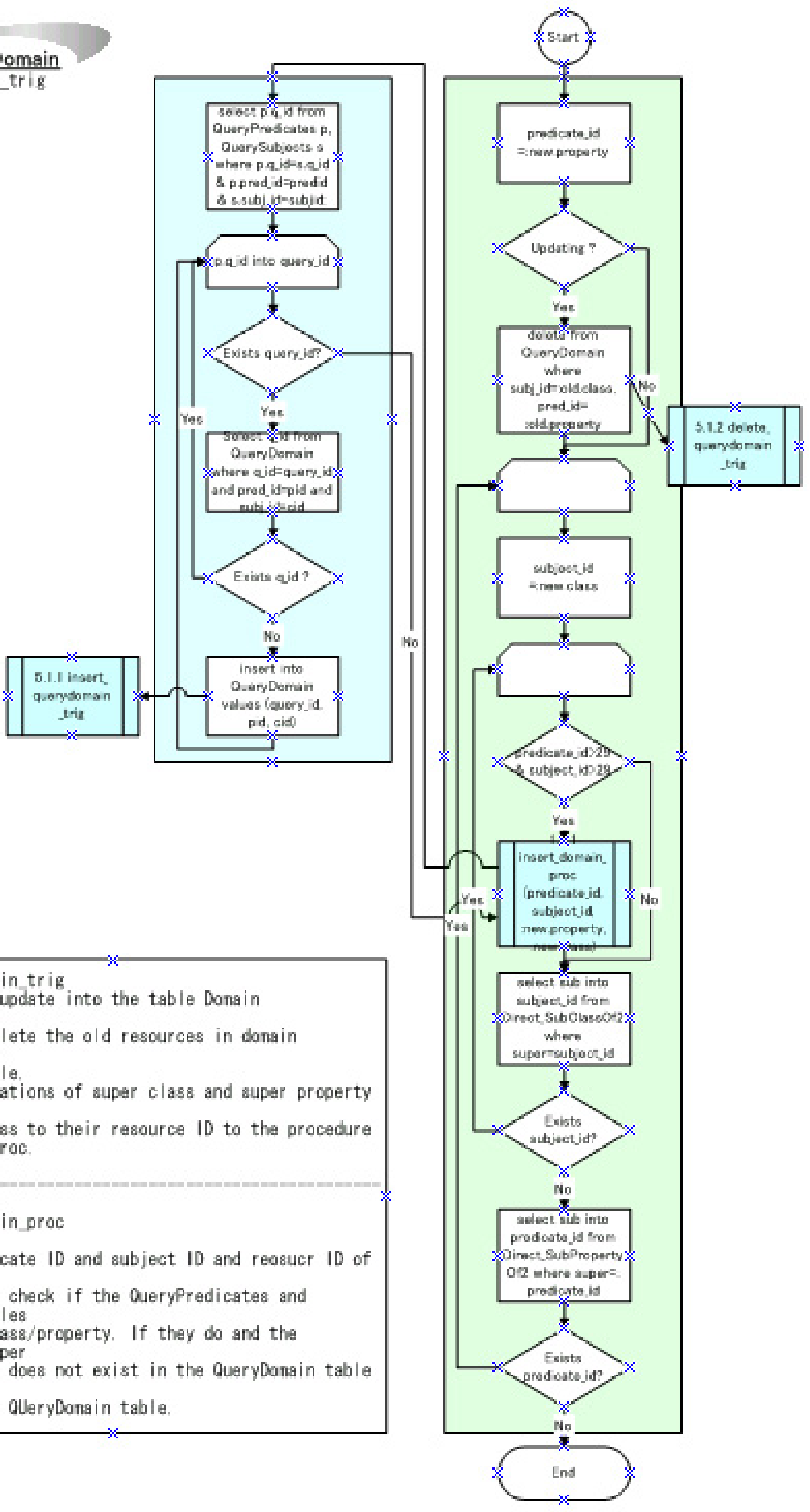
0.2.4 delete_dsubclassof_trig
Event: delete from the table Direct_SubClassOf
Role:
  Duplicate the table Direct_SUbClassOf2.
  Call procedure, which is shared with update trigger

1.2.4 delete_dsubclassof_proc
Role:
  Records of super classes of deleted class will be deleted by procedure
  delete_dsubclassof_2_proc from Query tables (i.e. QuerySubjects,
  QueryObjects, QueryDomain, QueryRange, and QueryResults)

2.2.4 delete_dsubclassof_2_proc
Role:
  Given the super class of :old.sub (deleted sub class), if any of them
  are contained in the QueryDomain/QueryRange, find the query ID of it.
  Then pass query ID to the procedure delete_class_d_proc/
  delete_class_r_proc.

0.3.1 Insert Domain
insert_domain_trig

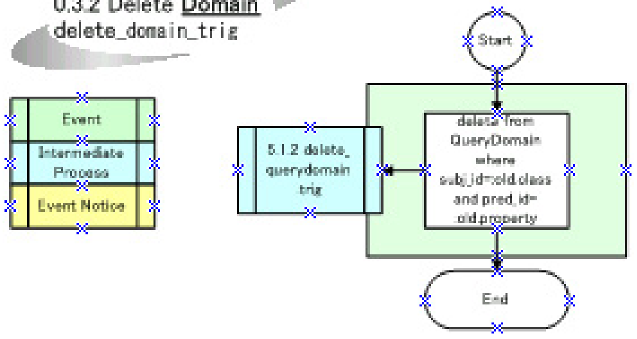
Event
Intermediate Process
Event Notice


Start
predicate_id =:new.property
Updating ?
Yes
No
delete from QueryDomain where subj_id=:old.class, pred_id= :old.property
5.1.2 delete_querydomain _trig
subject_id =:new.class
predicate_id>29 & subject_id>29
Yes
No
insert_domain_proc (predicate_id, subject_id, :new.property, :new.class)
select sub into subject_id from Direct_SubClassOf2 where super=subject_id
Exists subject_id?
No
select sub into predicate_id from Direct_SubPropertyOf2 where super=:predicate_id
Exists predicate_id?
No
Yes
Yes
End
select p.q_id from QueryPredicates p, QuerySubjects s where p.q_id=s.q_id & p.pred_id=predid & s.subj_id=subjid
p.q_id into query_id
Exists query_id?
Yes
No
Select q_id from QueryDomain where q_id=query_id and pred_id=pid and subj_id=cid
Exists q_id ?
Yes
No
insert into QueryDomain values (query_id, pid, cid)
5.1.1 insert_querydomain _trig

```
0.3.1 insert_domain_trig
Event: insert or update into the table Domain
Role:
  If updating, delete the old resources in domain
relationship from
  QueryDomain table.
  Find all combinations of super class and super property
of inserted
  domain, then pass to their resource ID to the procedure
  insert_domain_proc.
```

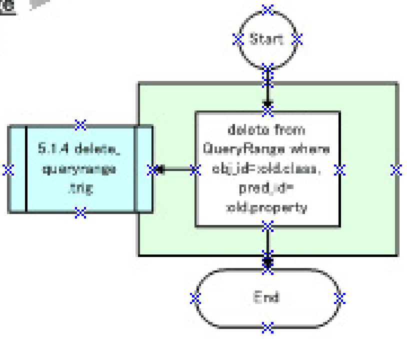
```
1.3.1 insert_domain_proc
Role:
  Given the predicate ID and subject ID and reosucr ID of
their super
  class/property, check if the QueryPredicates and
QUerySubjects Tables
  hold the sub class/property. If they do and the
combination of super
  class/property  does not exist in the QueryDomain table
yet, then
  insert the into QUeryDomain table.
```

0.3.2 Delete Domain
delete_domain_trig

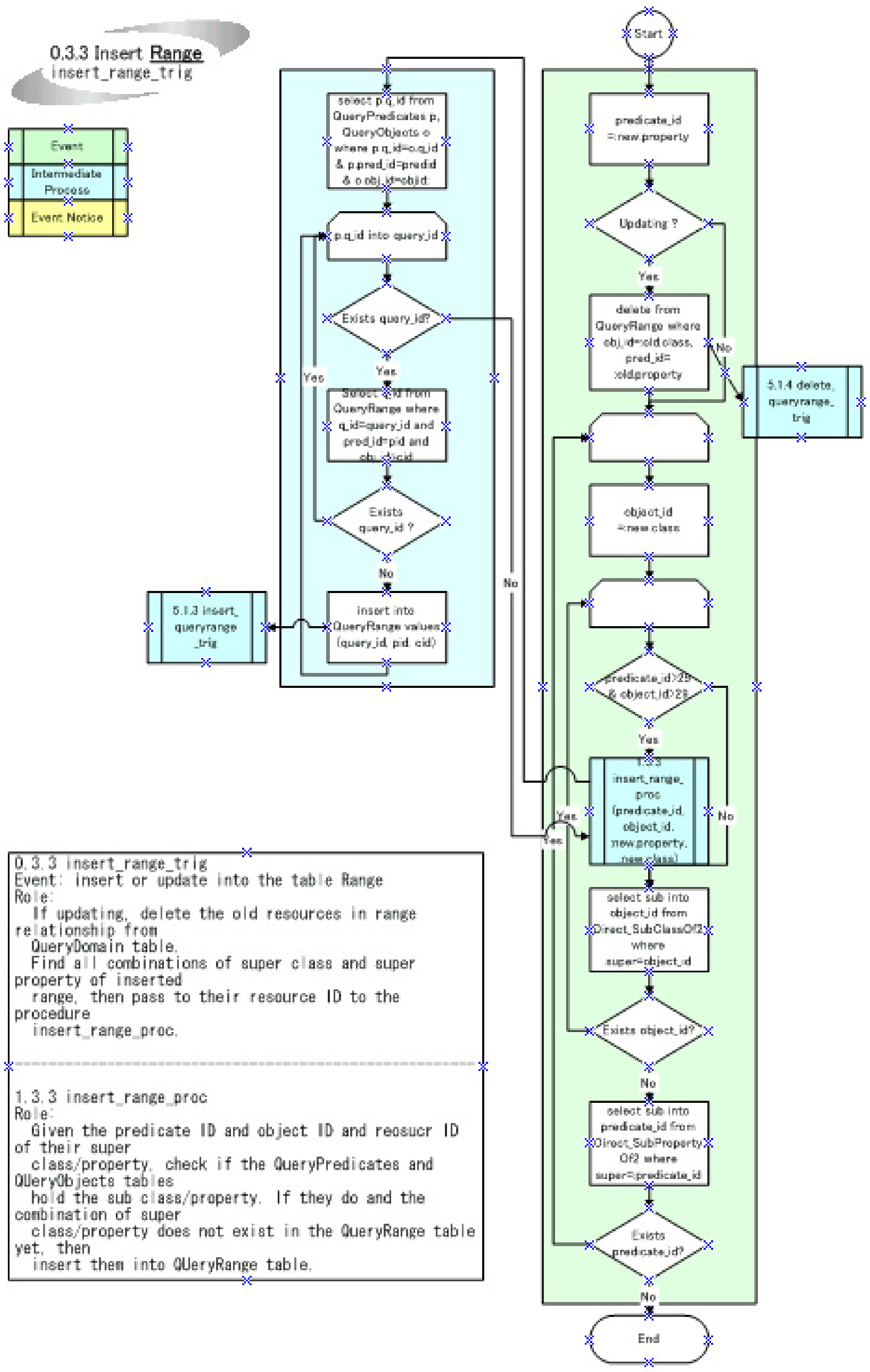

Event
Intermediate Process
Event Notice
Start
5.1.2 delete_querydomain_trig
delete from QueryDomain where subj_id=:old.class and pred_id=:old.property
End

```
0.3.2 delete_domain_trig
Event: delete from the table Domain
Role:
  Deleted class/property, delete same combination of resources from
  QueryDomain Table.
```

0.3.4 Delete Range
delete_range_trig

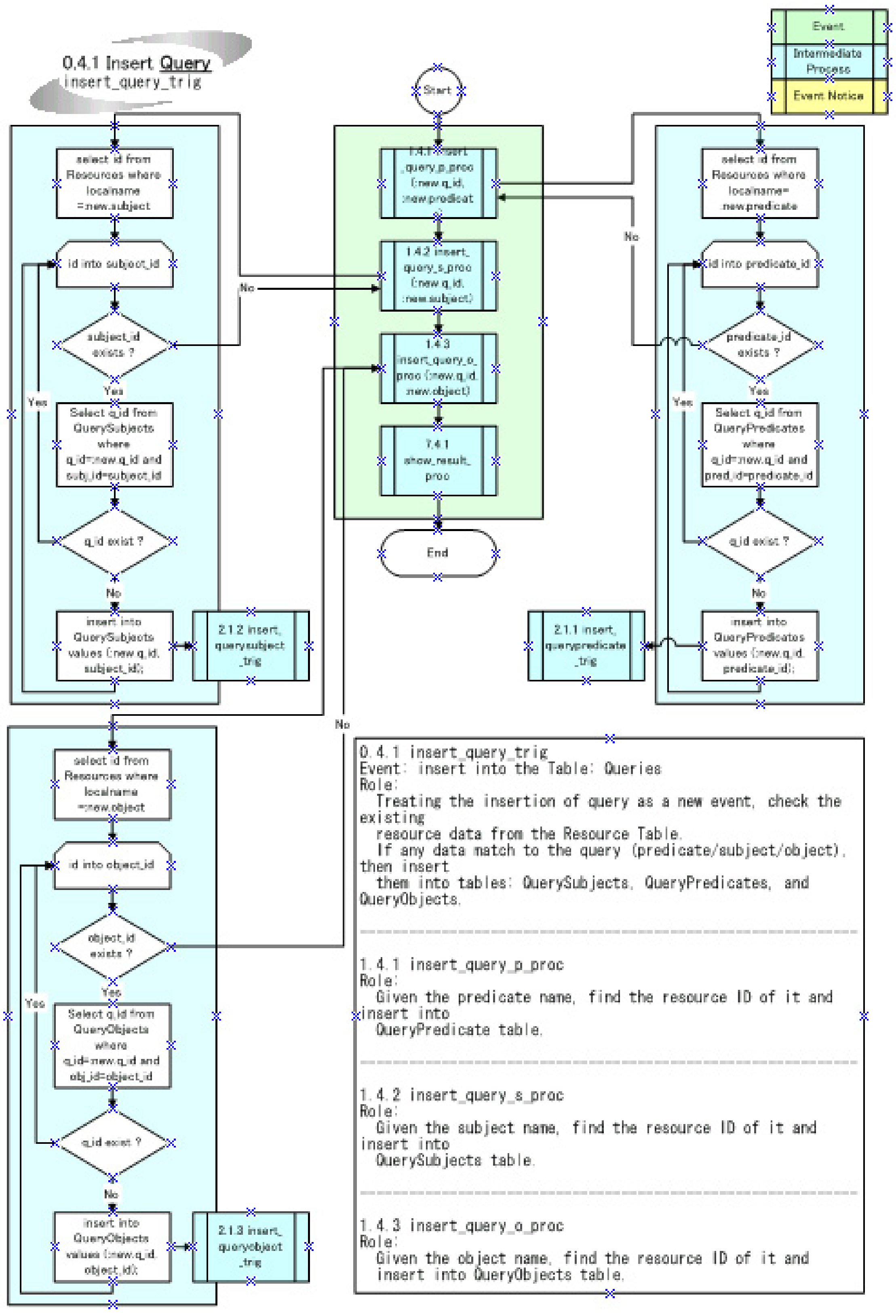

Start
5.1.4 delete_queryrange_trig
delete from QueryRange where obj_id=:old.class, pred_id=:old.property
End

```
0.3.4 delete_range_trig
Event: delete from the Table: Range
Role:
  Deleted class/property, delete same combination of resources from
  QueryRange Table.
```


0.3.3 Insert Range
insert_range_trig
Event
Intermediate Process
Event Notice
Start
predicate_id =:new.property
Updating ?
Yes
delete from QueryRange where obj_id=:old.class, pred_id= :old.property
No
5.1.4 delete_ queryrange_ trig
object_id =:new.class
predicate_id>25 & object_id>25
Yes
1.3.3 insert_range_ proc (predicate_id, object_id, :new.property, :new.class)
No
Yes
Yes
select sub into object_id from Direct_SubClassOf2 where super=object_id
Exists object_id?
No
select sub into predicate_id from Direct_SubPropertyOf2 where super=predicate_id
Exists predicate_id?
No
End
select p.q_id from QueryPredicates p, QueryObjects o where p.q_id=o.q_id & p.pred_id=predid & o.obj_id=objid;
p.q_id into query_id
Exists query_id?
Yes
Select q_id from QueryRange where q_id=query_id and pred_id=pid and obj_id=cid
Exists query_id ?
Yes
No
insert into QueryRange values (query_id, pid, cid)
5.1.3 insert_ queryrange _trig
No

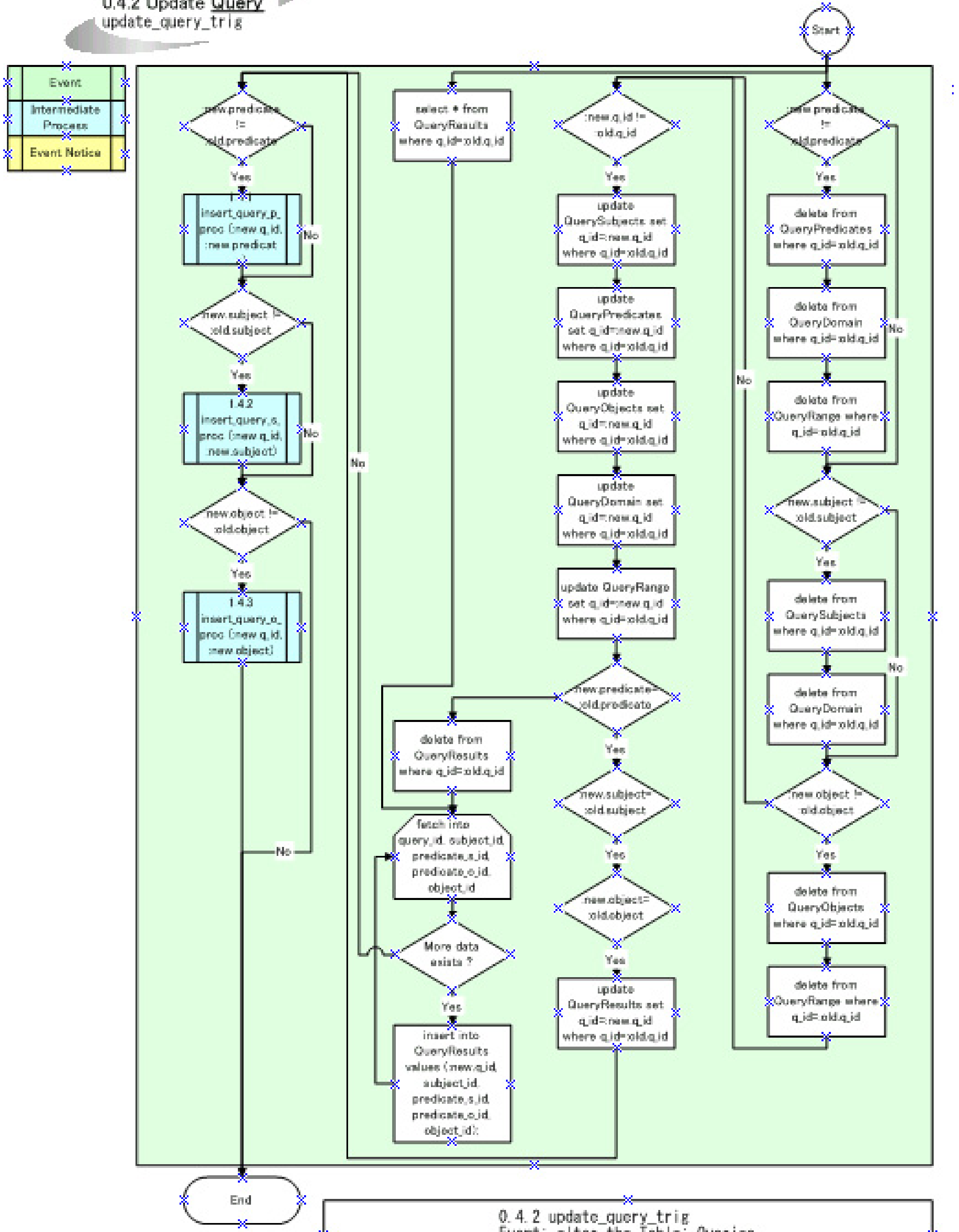
0.3.3 insert_range_trig
Event: insert or update into the table Range
Role:
  If updating, delete the old resources in range relationship from
  QueryDomain table.
  Find all combinations of super class and super property of inserted
  range, then pass to their resource ID to the procedure
  insert_range_proc.

1.3.3 insert_range_proc
Role:
  Given the predicate ID and object ID and reosucr ID of their super
  class/property, check if the QueryPredicates and QUeryObjects tables
  hold the sub class/property. If they do and the combination of super
  class/property does not exist in the QueryRange table yet, then
  insert them into QUeryRange table.


0.4.1 Insert Query
insert_query_trig
Event
Intermediate Process
Event Notice
Start
1.4.1 insert_query_p_proc (:new.q_id, :new.predicat
1.4.2 insert_query_s_proc (:new.q_id, :new.subject)
1.4.3 insert_query_o_proc (:new.q_id, :new.object)
7.4.1 show_result_proc
End
select id from Resources where localname =:new.subject
id into subject_id
subject_id exists ?
Yes
No
Select q_id from QuerySubjects where q_id=:new.q_id and subj_id=subject_id
q_id exist ?
Yes
No
insert into QuerySubjects values (:new.q_id, subject_id);
2.1.2 insert_querysubject_trig
select id from Resources where localname= :new.predicate
id into predicate_id
predicate_id exists ?
Yes
No
Select q_id from QueryPredicates where q_id=:new.q_id and pred_id=predicate_id
q_id exist ?
Yes
No
insert into QueryPredicates values (:new.q_id, predicate_id);
2.1.1 insert_querypredicate_trig
select id from Resources where localname =:new.object
id into object_id
object_id exists ?
Yes
No
Select q_id from QueryObjects where q_id=:new.q_id and obj_id=object_id
q_id exist ?
Yes
No
insert into QueryObjects values (:new.q_id, object_id);
2.1.3 insert_queryobject_trig

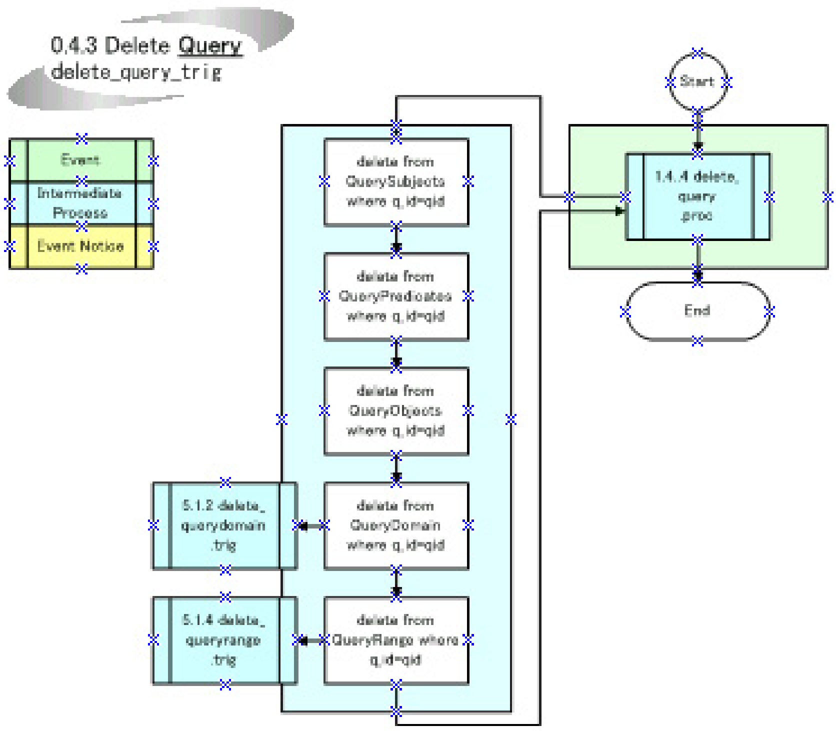
```
0.4.1 insert_query_trig
Event: insert into the Table: Queries
Role:
  Treating the insertion of query as a new event, check the
existing
  resource data from the Resource Table.
  If any data match to the query (predicate/subject/object),
then insert
  them into tables: QuerySubjects, QueryPredicates, and
QueryObjects.

------------------------------------------------------------

1.4.1 insert_query_p_proc
Role:
  Given the predicate name, find the resource ID of it and
insert into
  QueryPredicate table.

------------------------------------------------------------

1.4.2 insert_query_s_proc
Role:
  Given the subject name, find the resource ID of it and
insert into
  QuerySubjects table.

------------------------------------------------------------

1.4.3 insert_query_o_proc
Role:
  Given the object name, find the resource ID of it and
  insert into QueryObjects table.
```

0.4.2 Update Query
update_query_trig



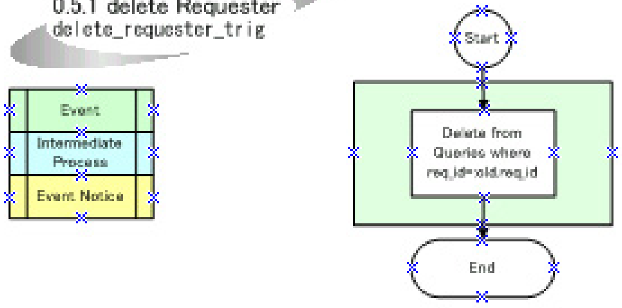
Start
Event
Intermediate Process
Event Notice
:new.predicate != :old.predicate
Yes
No
insert_query_p_proc (:new.q_id, :new.predicat
:new.subject != :old.subject
1.4.2 insert_query_s_proc (:new.q_id, :new.subject)
:new.object != :old.object
1.4.3 insert_query_o_proc (:new.q_id, :new.object)
select * from QueryResults where q_id=:old.q_id
:new.q_id != :old.q_id
update QuerySubjects set q_id=:new.q_id where q_id=:old.q_id
update QueryPredicates set q_id=:new.q_id where q_id=:old.q_id
update QueryObjects set q_id=:new.q_id where q_id=:old.q_id
update QueryDomain set q_id=:new.q_id where q_id=:old.q_id
update QueryRange set q_id=:new.q_id where q_id=:old.q_id
:new.predicate= :old.predicate
:new.subject= :old.subject
:new.object= :old.object
update QueryResults set q_id=:new.q_id where q_id=:old.q_id
delete from QueryResults where q_id=:old.q_id
fetch into query_id, subject_id, predicate_s_id, predicate_o_id, object_id
More data exists ?
insert into QueryResults values (:new.q_id, subject_id, predicate_s_id, predicate_o_id, object_id);
:new.predicate != :old.predicate
delete from QueryPredicates where q_id=:old.q_id
delete from QueryDomain where q_id=:old.q_id
delete from QueryRange where q_id=:old.q_id
:new.subject != :old.subject
delete from QuerySubjects where q_id=:old.q_id
delete from QueryDomain where q_id=:old.q_id
:new.object != :old.object
delete from QueryObjects where q_id=:old.q_id
delete from QueryRange where q_id=:old.q_id
End

0.4.2 update_query_trig
Event: alter the Table: Queries
Role:


0.4.3 Delete Query
delete_query_trig
Event
Intermediate Process
Event Notice
Start
1.4.4 delete_query.proc
End
delete from QuerySubjects where q_id=qid
delete from QueryPredicates where q_id=qid
delete from QueryObjects where q_id=qid
delete from QueryDomain where q_id=qid
delete from QueryRange where q_id=qid
5.1.2 delete_querydomain.trig
5.1.4 delete_queryrange.trig

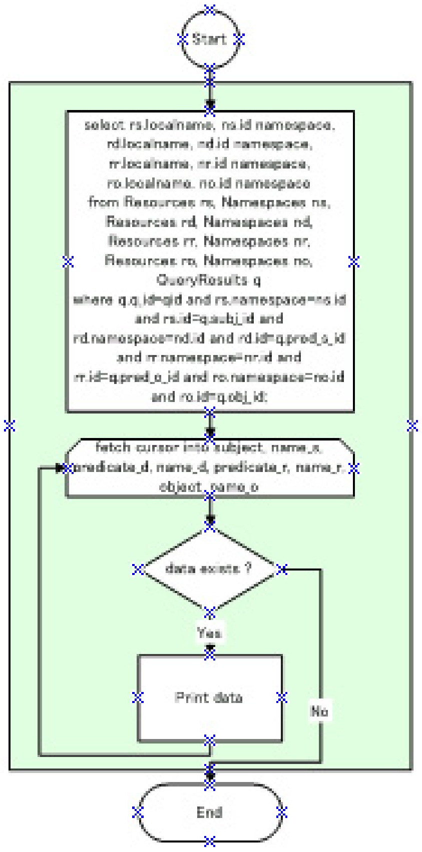
```
0.4.3 delete_query_trig
Event: delete from the Table: Queries
Role:
  call procedure delete_query_proc, which is shared with update trigger

------------------------------------------------------------------

1.4.4 delete_query_proc
Event: delete from the Table: Queries
Role:
  Cascade the requester ID from Query table.
```

0.5.1 delete Requester
delete_requester_trig



```
0.5.1 delete_requester_trig
Role:
  Cascade the requester ID from Query table.
```

7.4.1 Show Notices
show_result_proc

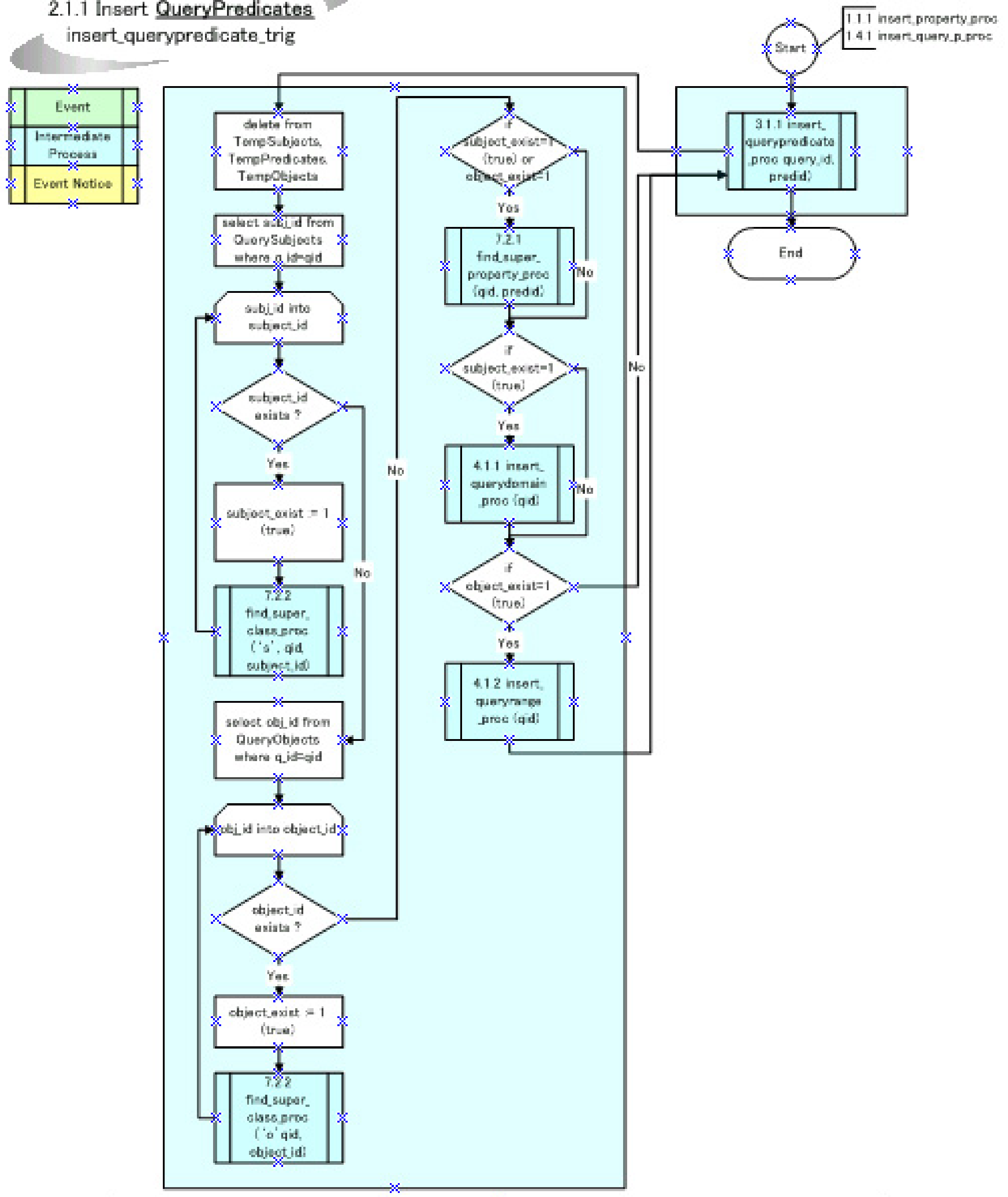


```
select rs.localname, ns.id namespace,
rd.localname, nd.id namespace,
rr.localname, nr.id namespace,
ro.localname, no.id namespace
from Resources rs, Namespaces ns,
Resources rd, Namespaces nd,
Resources rr, Namespaces nr,
Resources ro, Namespaces no,
QueryResults q
where q.q_id=qid and rs.namespace=ns.id
and rs.id=q.subj_id and
rd.namespace=nd.id and rd.id=q.pred_s_id
and rr.namespace=nr.id and
rr.id=q.pred_o_id and ro.namespace=no.id
and ro.id=q.obj_id;
```

```
fetch cursor into subject, name_s,
predicate_d, name_d, predicate_r, name_r,
object, name_o
```

```
7.4.1 show_result_proc
Role:
  Show the query results on console when the user input the query.
```

2.1.1 Insert QueryPredicates
insert_querypredicate_trig

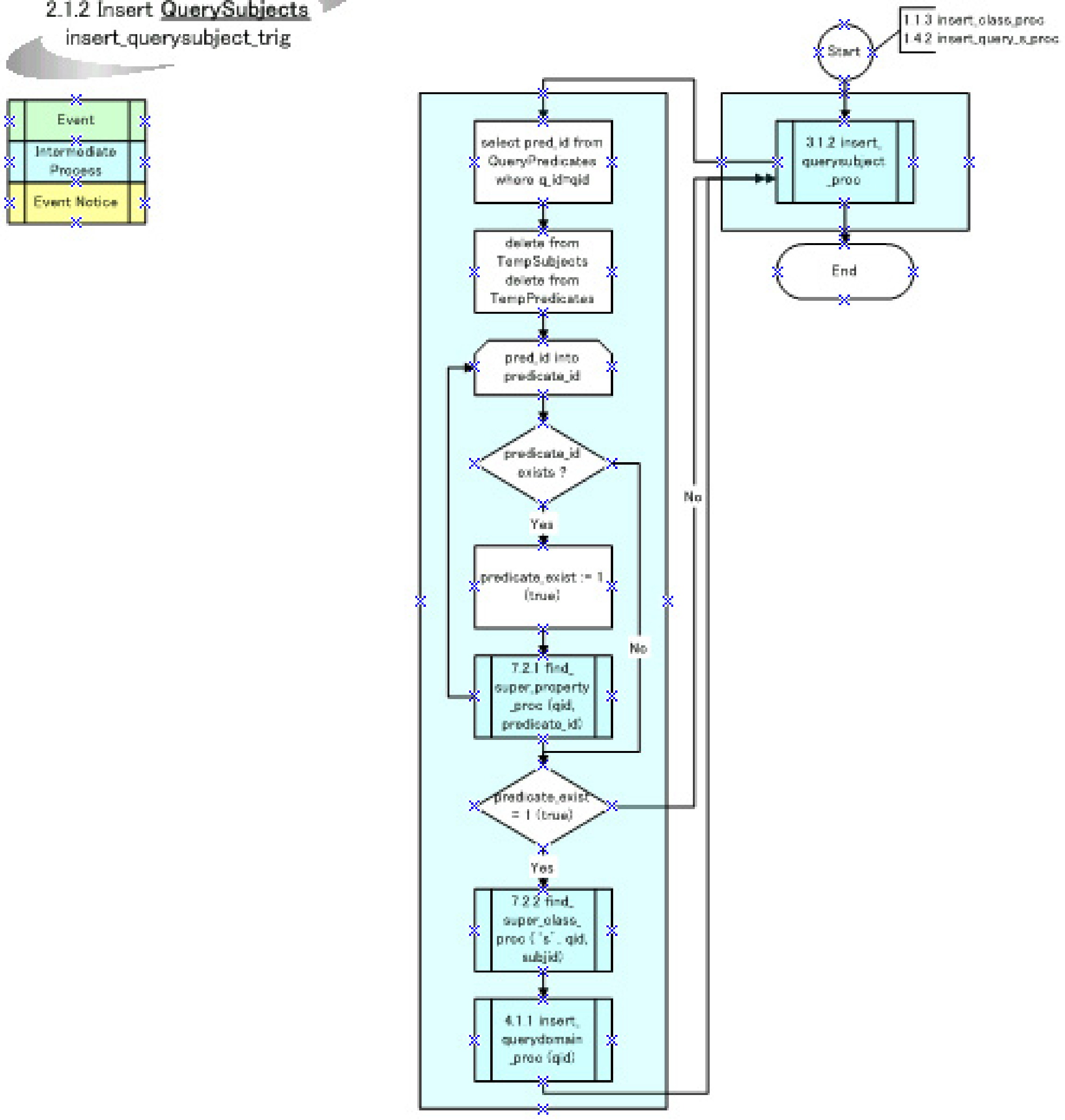


Event
Intermediate Process
Event Notice
Start
1.1.1 insert_property_proc
1.4.1 insert_query_p_proc
delete from TempSubjects, TempPredicates, TempObjects
select subj_id from QuerySubjects where q_id=qid
subj_id into subject_id
subject_id exists ?
Yes
No
subject_exist := 1 (true)
7.2.2 find_super_class_proc ('s', qid, subject_id)
select obj_id from QueryObjects where q_id=qid
obj_id into object_id
object_id exists ?
Yes
No
object_exist := 1 (true)
7.2.2 find_super_class_proc ('o' qid, object_id)
if subject_exist=1 (true) or object_exist=1
Yes
No
7.2.1 find_super_property_proc (qid, predid)
if subject_exist=1 (true)
Yes
No
4.1.1 insert_querydomain_proc (qid)
if object_exist=1 (true)
Yes
No
4.1.2 insert_queryrange_proc (qid)
3.1.1 insert_querypredicate_proc query_id, predid)
End

2.1.1 insert_querypredicate_trig
Event: insert into Table: QueryPredicates
Role:
  Insert is processed in procedure insert_querypredicate_proc, which
  is shared with other triggers/procedures
  Upding is an invalid transaction in this table.

3.1.1 insert_querypredicate_proc
Role:
  Find the super-classes of subject/object and super-property of predicate,
  and pass to procedures insert_querydomain_proc / insert_queryrange_proc
  to insert the valid pair into tables: QueryDomain QueryRange.

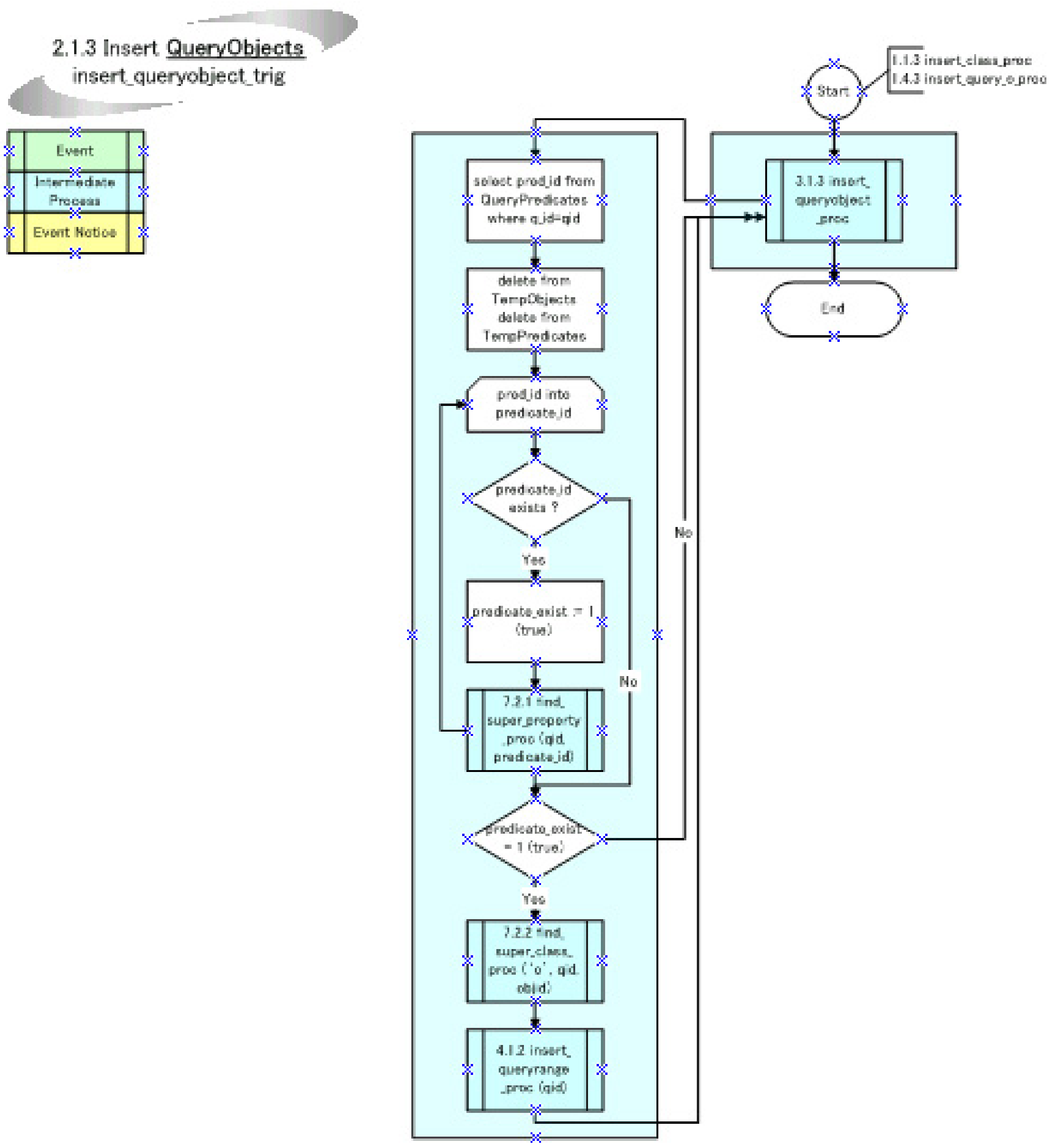
2.1.2 Insert QuerySubjects
insert_querysubject_trig

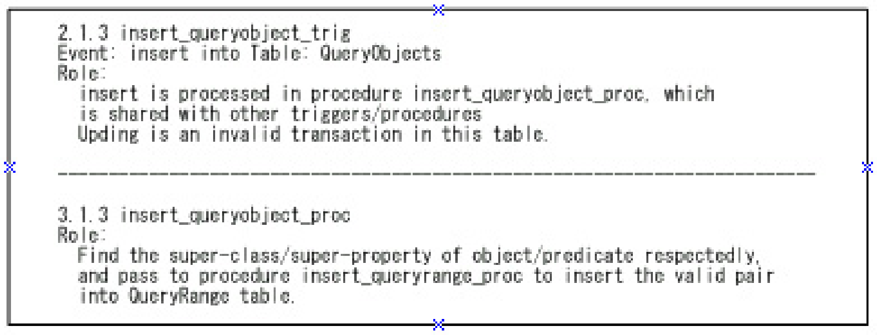


Event
Intermediate Process
Event Notice
Start
1.1.3 insert_class_proc
1.4.2 insert_query_s_proc
select pred_id from QueryPredicates where q_id=qid
delete from TempSubjects delete from TempPredicates
pred_id into predicate_id
predicate_id exists ?
Yes
No
predicate_exist := 1 (true)
7.2.1 find_ super_property _proc (qid, predicate_id)
predicate_exist = 1 (true)
Yes
No
7.2.2 find_ super_class_ proc ('s', qid, subjid)
4.1.1 insert_ querydomain _proc (qid)
3.1.2 insert_ querysubject _proc
End

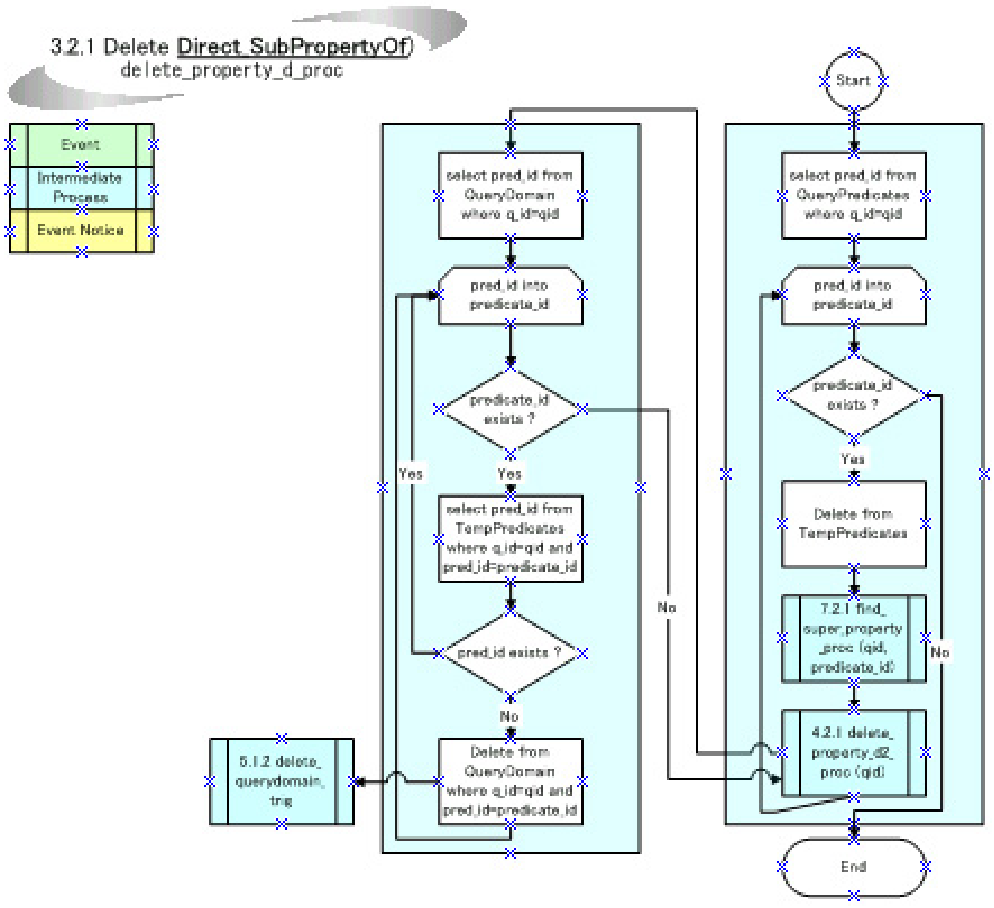
2.1.2 insert_querysubject_trig
Event: insert into Table: QuerySubjects
Role:
  Insert is processed in procedure insert_querysubject_proc, which
  is shared with other triggers/procedures
  Upding is an invalid transaction in this table.

3.1.2 insert_querysubject_proc
Role:
  Find the super-class/super-property of subject/predicate respectedly,
  and pass to procedure insert_querydomain_proc to insert the valid pair
  into QueryDomain table.


2.1.3 Insert QueryObjects
insert_queryobject_trig
Event
Intermediate Process
Event Notice
Start
1.1.3 insert_class_proc
1.4.3 insert_query_o_proc
3.1.3 insert_queryobject_proc
End
select prod_id from QueryPredicates where q_id=qid
delete from TempObjects delete from TempPredicates
prod_id into predicate_id
predicate_id exists ?
Yes
No
predicate_exist = 1 (true)
7.2.1 find_super_property_proc (qid, predicate_id)
predicate_exist = 1 (true)
Yes
No
7.2.2 find_super_class_proc ('o', qid, objid)
4.1.2 insert_queryrange_proc (qid)


2.1.3 insert_queryobject_trig
Event: insert into Table: QueryObjects
Role:
 insert is processed in procedure insert_queryobject_proc, which
 is shared with other triggers/procedures
 Upding is an invalid transaction in this table.

3.1.3 insert_queryobject_proc
Role:
 Find the super-class/super-property of object/predicate respectedly,
 and pass to procedure insert_queryrange_proc to insert the valid pair
 into QueryRange table.

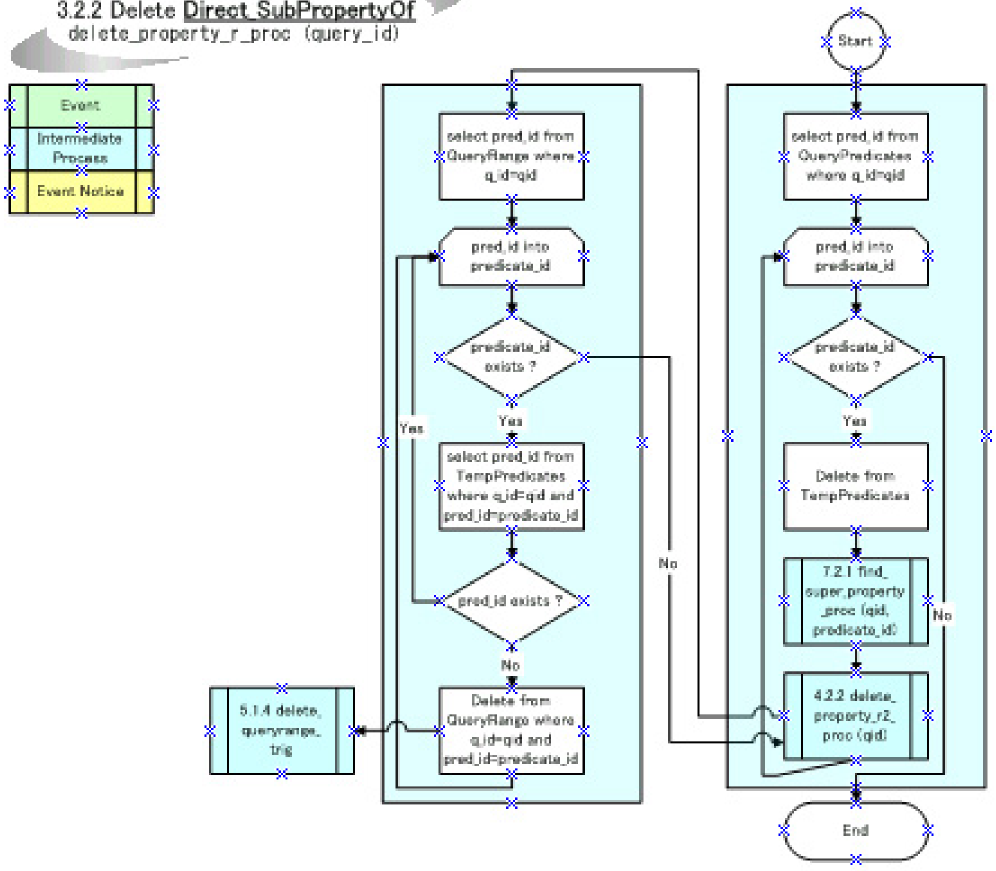


3.2.1 Delete Direct SubPropertyOf)
delete_property_d_proc
Event
Intermediate Process
Event Notice
Start
select pred_id from QueryPredicates where q_id=qid
pred_id into predicate_id
predicate_id exists ?
Yes
Delete from TempPredicates
7.2.1 find_super_property_proc (qid, predicate_id)
4.2.1 delete_property_d2_proc (qid)
No
End
select pred_id from QueryDomain where q_id=qid
pred_id into predicate_id
predicate_id exists ?
Yes
select pred_id from TempPredicates where q_id=qid and pred_id=predicate_id
pred_id exists ?
Yes
No
Delete from QueryDomain where q_id=qid and pred_id=predicate_id
5.1.2 delete_querydomain_trig
No

```
3.2.1 delete_property_d_proc
Role:
  Given the query ID, which is contained in the QyeryDomain table and
  whose pred_id is super property of deleted item, find out all valid
  super properties of predicate for that query. Then pass to
  delete_property_d2_proc to check if the super property is still valid.

---------------------------------------------------------------------

4.2.1 delete_property_d2_proc
Role:
  Given the query ID, check if the property ID of predicate in
  QueryDomain table is still valid, comparing with the latest valid super
  properties which are stored in TempPredicates table. If no longer valid,
  then delete it.
```

3.2.2 Delete Direct SubPropertyOf

delete_property_r_proc (query_id)

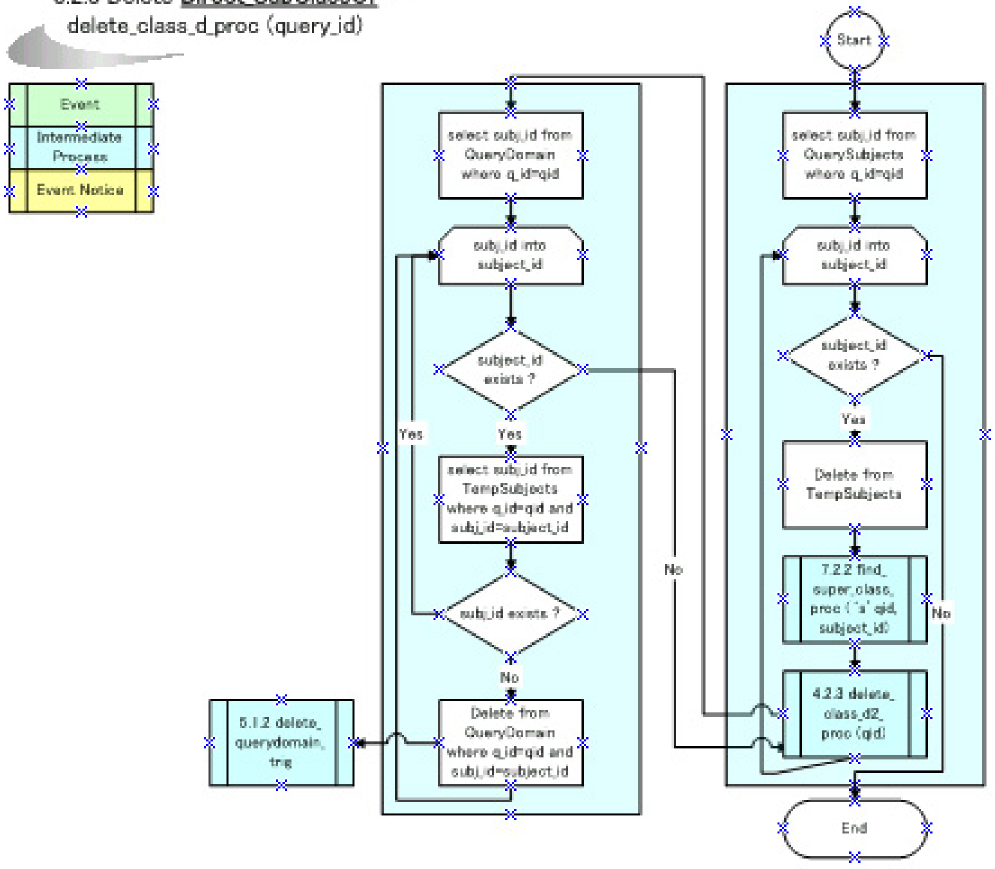

Event
Intermediate Process
Event Notice
Start
select pred_id from QueryPredicates where q_id=qid
pred_id into predicate_id
predicate_id exists ?
Yes
Delete from TempPredicates
7.2.1 find_super_property_proc (qid, predicate_id)
4.2.2 delete_property_r2_proc (qid)
No
End
select pred_id from QueryRange where q_id=qid
pred_id into predicate_id
predicate_id exists ?
Yes
select pred_id from TempPredicates where q_id=qid and pred_id=predicate_id
pred_id exists ?
Yes
No
Delete from QueryRange where q_id=qid and pred_id=predicate_id
No
5.1.4 delete_queryrange_trig

3.2.2 delete_property_r_proc
Role:
Given the query ID, which is contained in the QyeryRange table and whose pred_id is super property of deleted item, find out all valid super properties of predicate for that query. Then pass to delete_property_r2_proc to check if the super property is still valid.

4.2.2 delete_property_r2_proc
Role:
Given the query ID, check if the property ID of predicate in QueryRange table is still valid, comparing with the latest valid super properties which are stored in TempPredicates table. If no longer valid, then delete it.

3.2.3 Delete Direct_SubClassOf

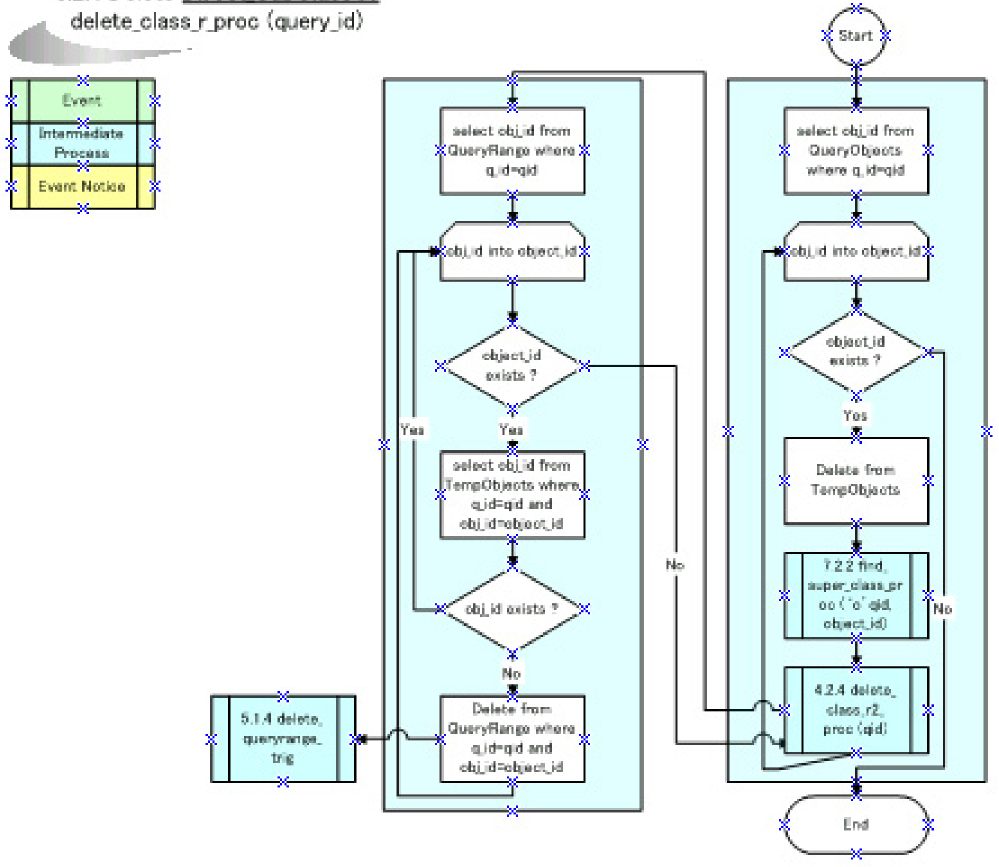
delete_class_d_proc (query_id)


Event
Intermediate Process
Event Notice
Start
select subj_id from QuerySubjects where q_id=qid
subj_id into subject_id
subject_id exists ?
Yes
Delete from TempSubjects
7.2.2 find_super_class_proc ('s' qid, subject_id)
No
4.2.3 delete_class_d2_proc (qid)
End
select subj_id from QueryDomain where q_id=qid
subj_id into subject_id
subject_id exists ?
Yes
No
select subj_id from TempSubjects where q_id=qid and subj_id=subject_id
subj_id exists ?
Yes
No
Delete from QueryDomain where q_id=qid and subj_id=subject_id
5.1.2 delete_querydomain_trig

```
3.2.3 delete_class_d_proc
Role:
  Given the query ID, which is contained in the QyeryDomain table and
  whose subj_id is super class of deleted item, find out all valid super
  class of subject for that query. Then pass to delete_class_d2_proc to
  check if the super class is still valid.

-----------------------------------------------------------------------

4.2.3 delete_class_d2_proc
Role:
  Given the query ID, check if the class ID of subject in QueryDomain
  table is still valid, comparing with the latest valid super classes
  which are stored in TempObjects table. If no longer valid, then delete
  it.
```

3.2.4 Delete Direct SubClassOf

delete_class_r_proc (query_id)


Event
Intermediate Process
Event Notice
Start
select obj_id from QueryObjects where q_id=qid
obj_id into object_id
object_id exists ?
Yes
Delete from TempObjects
7.2.2 find_super_class_proc ('o' qid, object_id)
No
4.2.4 delete_class_r2_proc (qid)
End
select obj_id from QueryRange where q_id=qid
obj_id into object_id
object_id exists ?
Yes
No
select obj_id from TempObjects where q_id=qid and obj_id=object_id
obj_id exists ?
Yes
No
Delete from QueryRange where q_id=qid and obj_id=object_id
5.1.4 delete_queryrange_trig

3.2.4 delete_class_r_proc

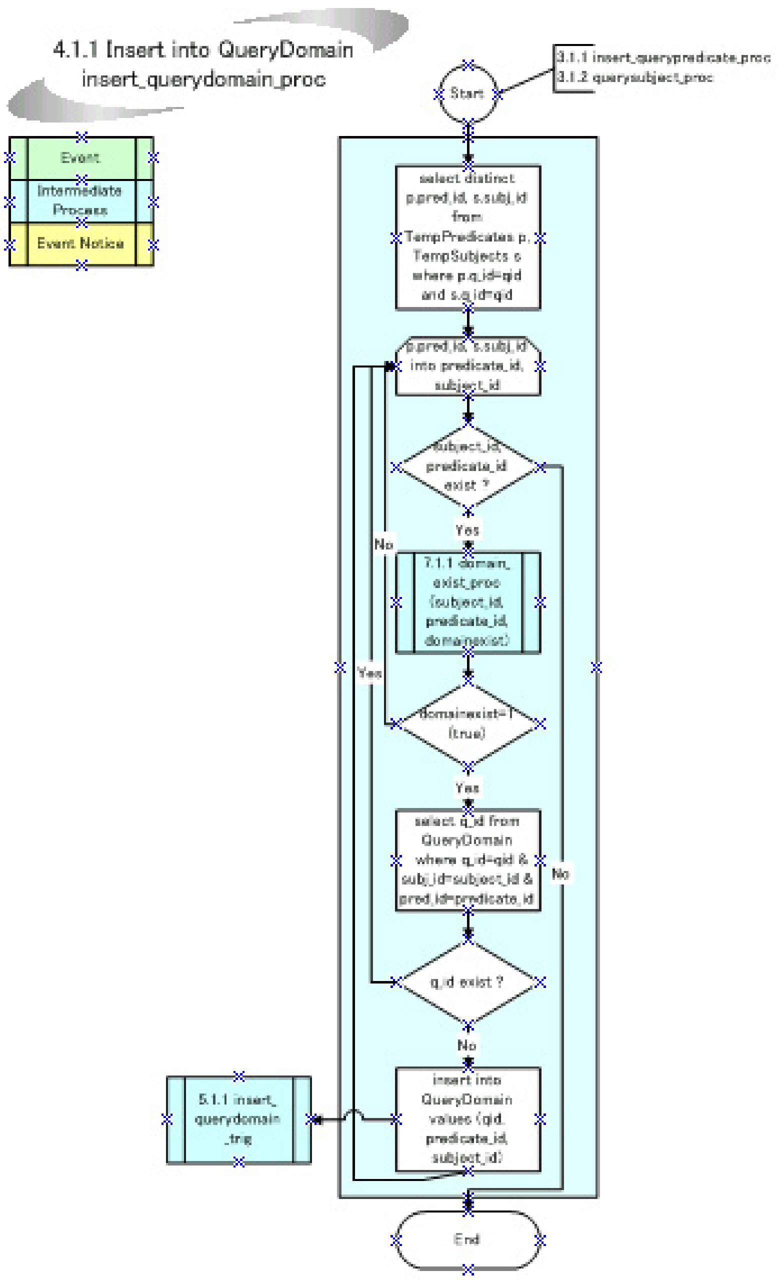
Role:

Given the query ID, which is contained in the QyeryRange table and whose obj_id is super class of deleted item, find out all valid super class of object for that query. Then pass to delete_class_r2_proc to check if the super class is still valid.

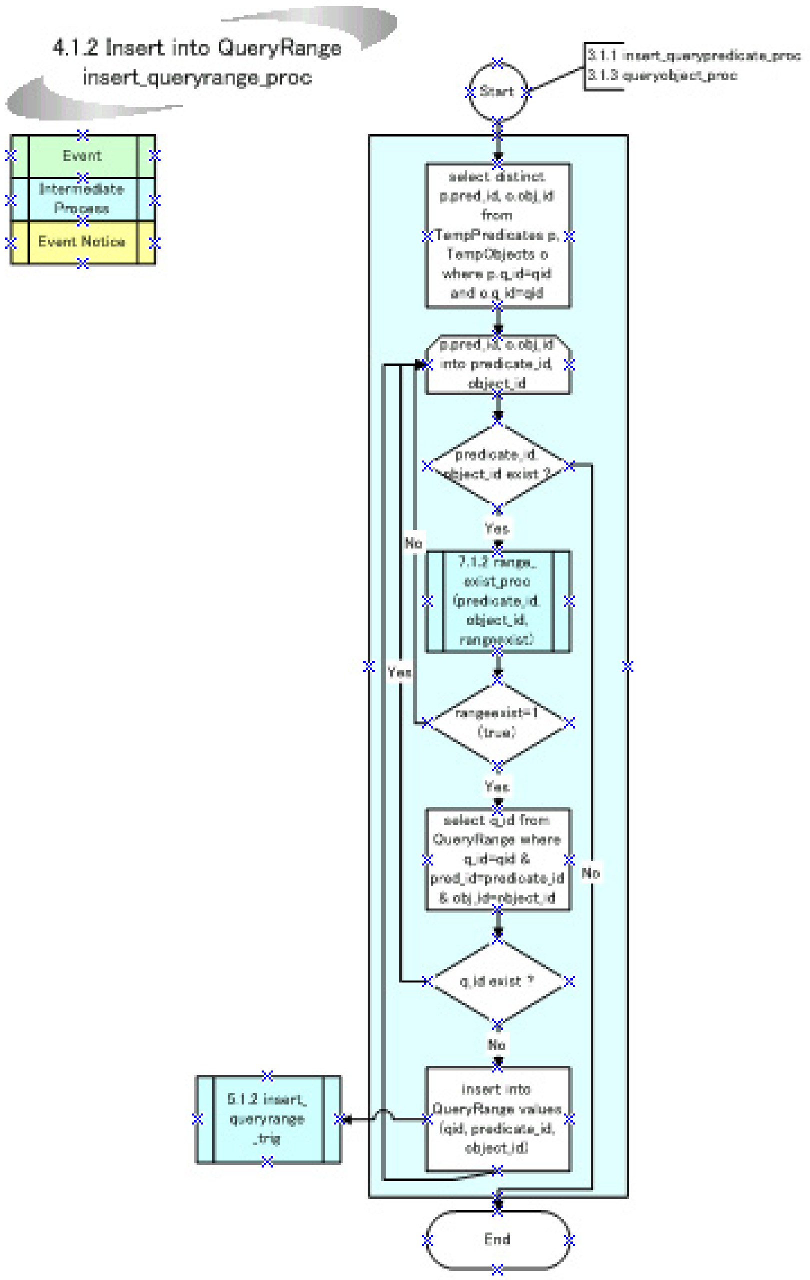
4.2.4 delete_class_r2_proc

Role:

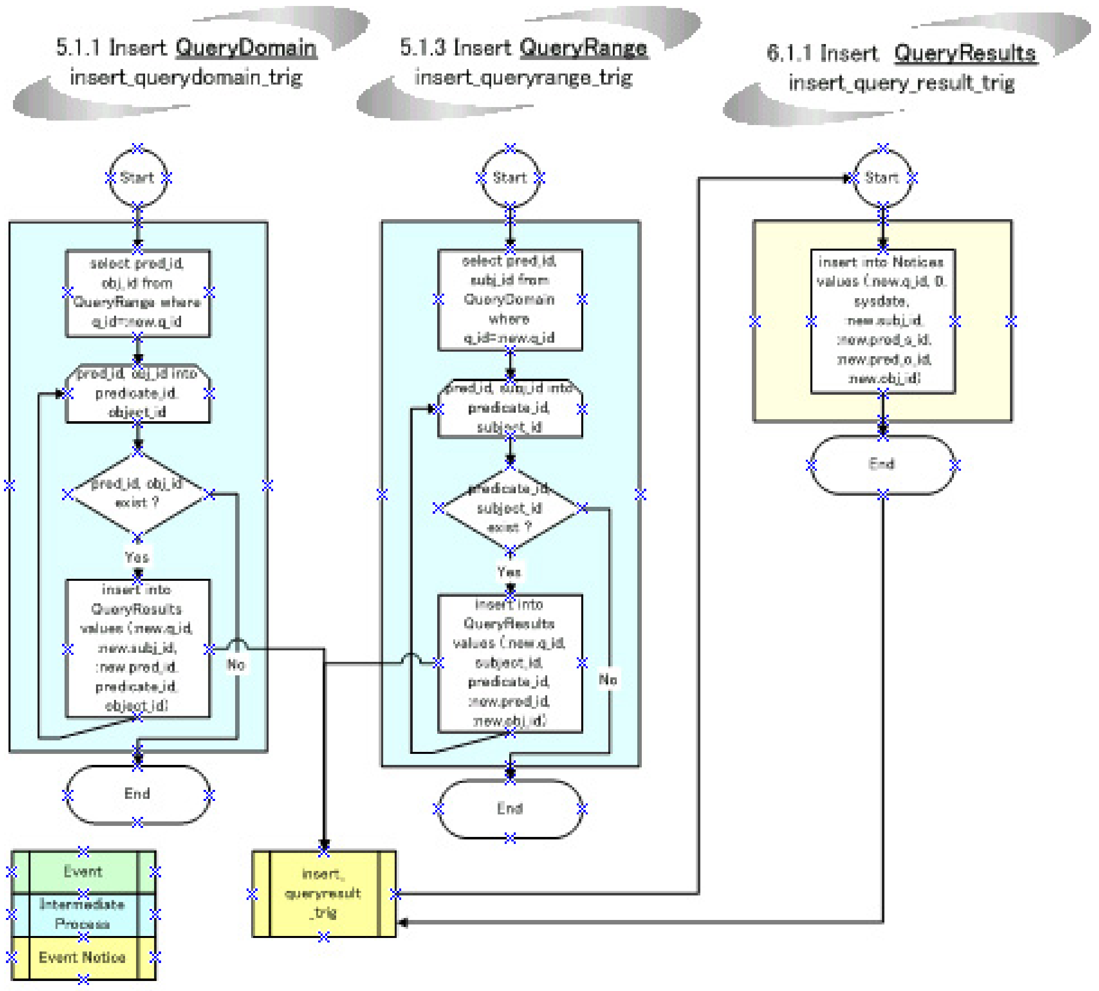
Given the query ID, check if the class ID of object in QueryRange table is still valid, comparing with the latest valid super classes which are stored in TempObjects table. If no longer valid, then delete it.


4.1.1 Insert into QueryDomain
insert_querydomain_proc
Event
Intermediate Process
Event Notice
3.1.1 insert_querypredicate_proc
3.1.2 querysubject_proc
Start
select distinct p.pred_id, s.subj_id from TempPredicates p, TempSubjects s where p.q_id=qid and s.q_id=qid
p.pred_id, s.subj_id into predicate_id, subject_id
subject_id, predicate_id exist ?
Yes
No
7.1.1 domain_exist_proc (subject_id, predicate_id, domainexist)
domainexist=1 (true)
Yes
Yes
select q_id from QueryDomain where q_id=qid & subj_id=subject_id & pred_id=predicate_id
No
q_id exist ?
No
insert into QueryDomain values (qid, predicate_id, subject_id)
5.1.1 insert_querydomain_trig
End

```
4.1.1 insert_querydomain_proc
Role:
  Given the query ID, retric the possible pairs of subject and predicate
  from TempSubjects/TempPredicates Tables (which contains the super
  classes/properties of subject/predicate). If any pairs exist in Domain
  table, and it is not contained in QueryDomain table yet, then insert
  it into QueryDomain table.
```

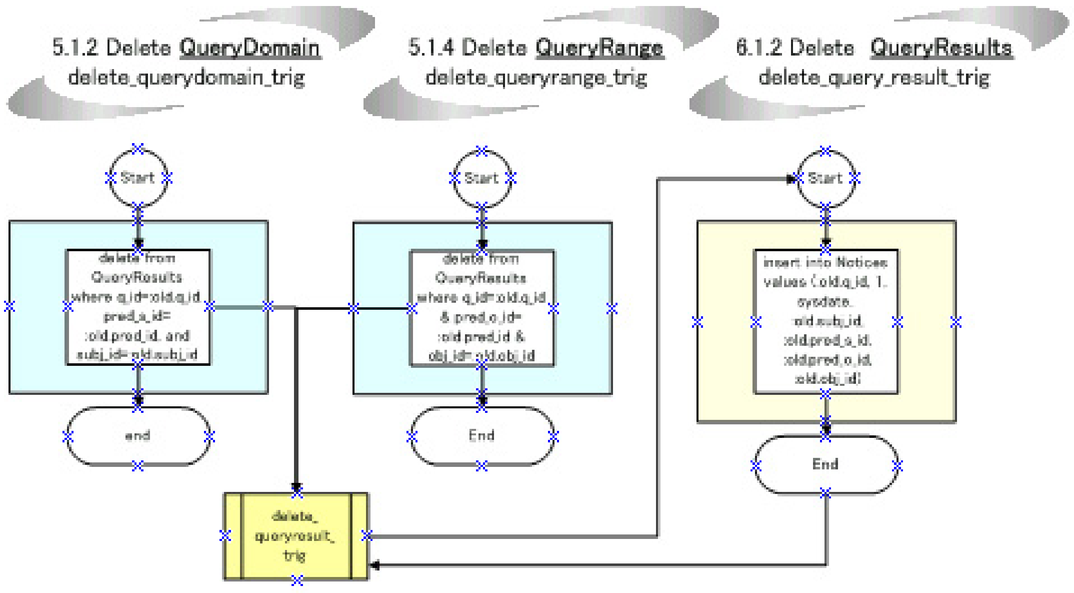

4.1.2 Insert into QueryRange
insert_queryrange_proc
Event
Intermediate Process
Event Notice
Start
3.1.1 insert_querypredicate_proc
3.1.3 queryobject_proc
select distinct p.pred_id, o.obj_id from TempPredicates p, TempObjects o where p.q_id=qid and o.q_id=qid
p.pred_id, o.obj_id into predicate_id, object_id
predicate_id, object_id exist ?
Yes
No
7.1.2 range_exist_proc (predicate_id, object_id, rangeexist)
rangeexist=1 (true)
Yes
Yes
select q_id from QueryRange where q_id=qid & pred_id=predicate_id & obj_id=object_id
No
q_id exist ?
No
insert into QueryRange values (qid, predicate_id, object_id)
5.1.2 insert_queryrange_trig
End

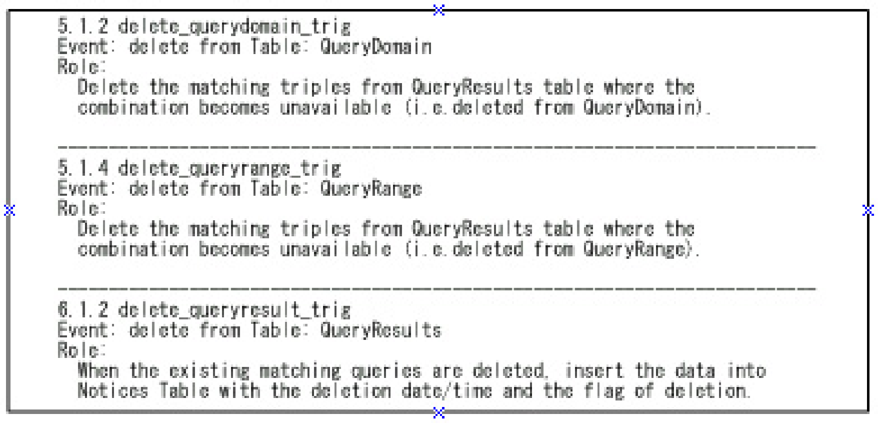
4.1.2 insert_queryrange_proc
Role:
  Given the query ID, retric the possible pairs of object and predicate from TempObjects/TempPredicates Tables (which contains the super classes/properties of object/predicate). If any pairs exist in Range table, and it is not contained in QueryRange table yet, then insert it into QueryRange table.



5.1.1 Insert QueryDomain
insert_querydomain_trig
5.1.3 Insert QueryRange
insert_queryrange_trig
6.1.1 Insert QueryResults
insert_query_result_trig
Start
select pred_id, obj_id from QueryRange where q_id=:new.q_id
pred_id, obj_id into predicate_id, object_id
pred_id, obj_id exist ?
Yes
insert into QueryResults values (:new.q_id, :new.subj_id, :new.pred_id, predicate_id, object_id)
No
End
Start
select pred_id, subj_id from QueryDomain where q_id=:new.q_id
pred_id, subj_id into predicate_id, subject_id
predicate_id, subject_id exist ?
Yes
insert into QueryResults values (:new.q_id, subject_id, predicate_id, :new.pred_id, :new.obj_id)
No
End
Start
insert into Notices values (:new.q_id, 0, sysdate, :new.subj_id, :new.pred_s_id, :new.pred_o_id, :new.obj_id)
End
insert_queryresult_trig
Event
Intermediate Process
Event Notice

```
5.1.1 insert_querydomain_trig
Event: insert into Table: QueryDomain
Role:
  Update transaction is not available
  Find the objects/predicate with same query ID as new resources from
  QueryRange table, then insert into QueryResults table that combination.

--------------------------------------------------------------------------

5.1.3 insert_queryrange_trig
Event: insert into Table: QueryRange
Role:
  Update transaction is not available
  Find the subjects/predicate with same query ID as new resources from
  QueryDomain table, then insert into QueryResults table that combination.

--------------------------------------------------------------------------

6.1.1 insert_queryresult_trig
Event: insert into Table: QueryResults
Role:
  When the new results is added, insert the data into Notices Table with
  the insertion date/time and the flag of insertion.
```


5.1.2 Delete QueryDomain
delete_querydomain_trig
5.1.4 Delete QueryRange
delete_queryrange_trig
6.1.2 Delete QueryResults
delete_query_result_trig
Start
delete from QueryResults where q_id=:old.q_id, pred_s_id= :old.pred_id, and subj_id=:old.subj_id
end
Start
delete from QueryResults where q_id=:old.q_id & pred_o_id= :old.pred_id & obj_id=:old.obj_id
End
Start
insert into Notices values (:old.q_id, 1, sysdate, :old.subj_id, :old.pred_s_id, :old.pred_o_id, :old.obj_id)
End
delete_queryresult_trig

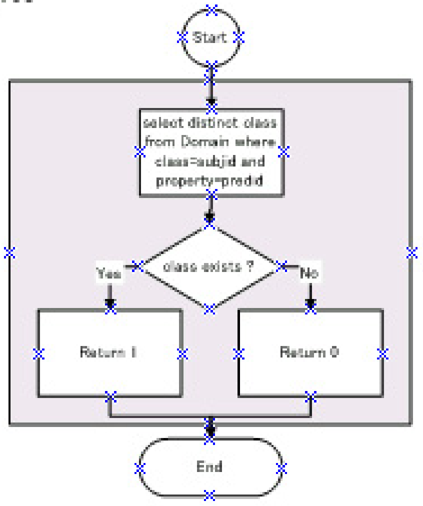

5.1.2 delete_querydomain_trig
Event: delete from Table: QueryDomain
Role:
Delete the matching triples from QueryResults table where the combination becomes unavailable (i.e. deleted from QueryDomain).

5.1.4 delete_queryrange_trig
Event: delete from Table: QueryRange
Role:
Delete the matching triples from QueryResults table where the combination becomes unavailable (i.e. deleted from QueryRange).

6.1.2 delete_queryresult_trig
Event: delete from Table: QueryResults
Role:
When the existing matching queries are deleted, insert the data into Notices Table with the deletion date/time and the flag of deletion.

7.1.1 Check if domain relation exists in Domain

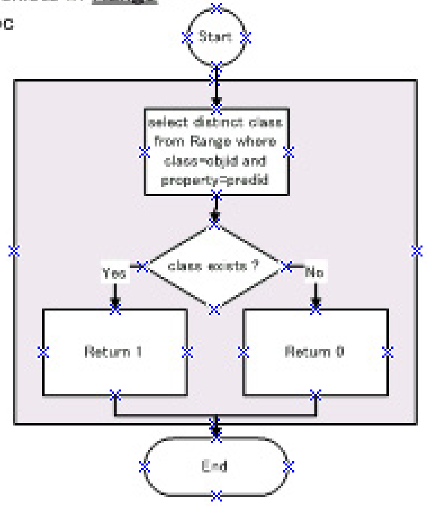
domain_exist_proc


Start
select distinct class from Domain where class=subjid and property=predid
class exists ?
Yes
No
Return 1
Return 0
End

```
7.1.1 domain_exist_proc
Role:
  Given the subject ID and Predicate ID, return true(1) if the combination
  exists in Domain table.
```

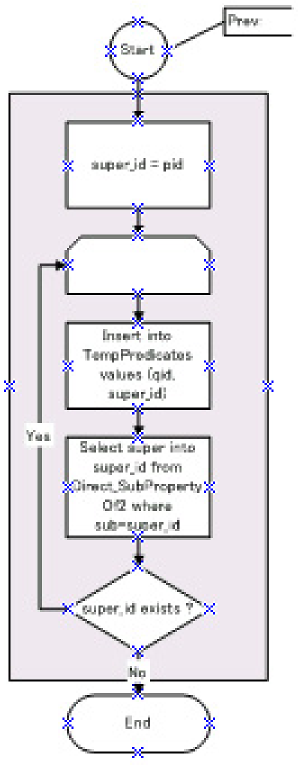
7.1.2 Check if range relation exists in Range

range_exist_proc

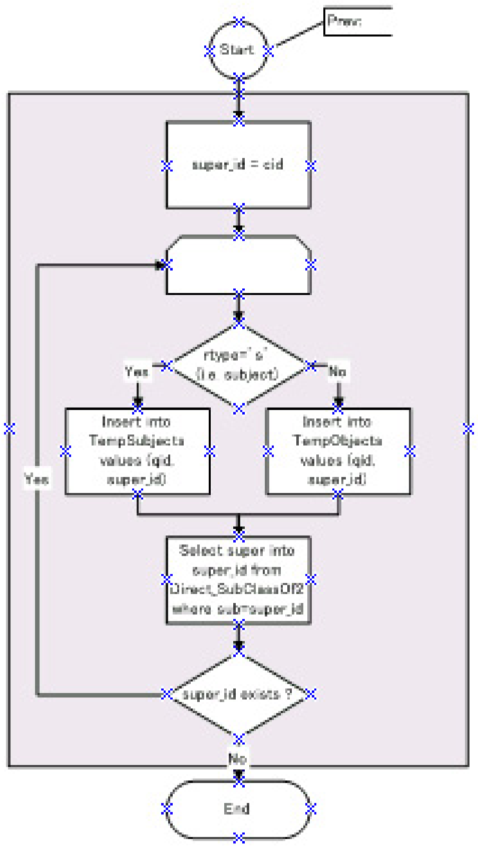

Start
select distinct class from Range where class=objid and property=predid
class exists ?
Yes
No
Return 1
Return 0
End

```
7.1.2 range_exist_proc
Role:
  Given the object ID and Predicate ID, return true(1) if the combination
  exists in Range table.
```

7.2.1 Insert into TempPredicates find_super_property_proc


Prev:
Start
super_id = pid
Insert into TempPredicates values (qid, super_id)
Select super into super_id from Direct_SubPropertyOf2 where sub=super_id
super_id exists ?
Yes
No
End

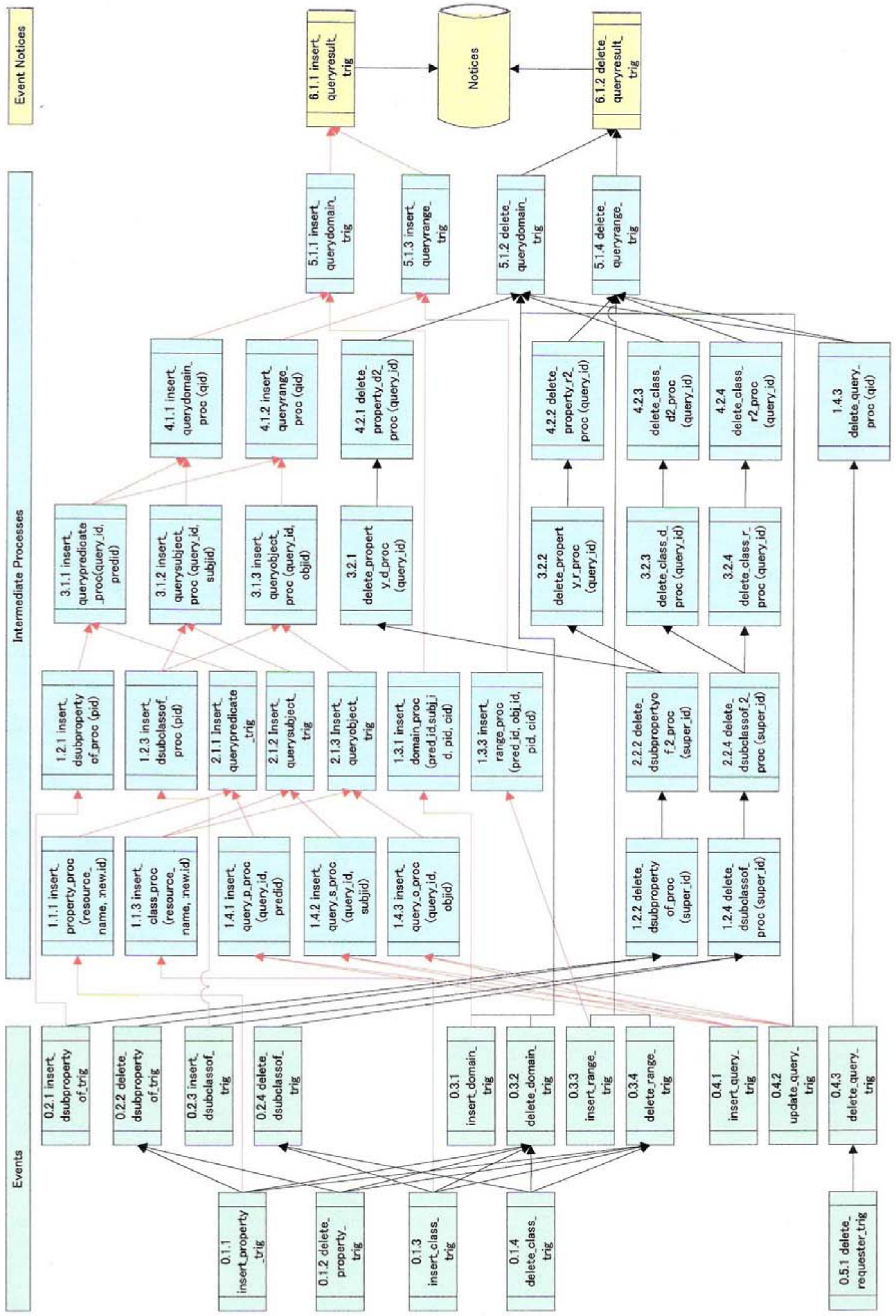
7.2.2 Insert into TempSubjects and TempObjects find_super_class_proc


Prev:
Start
super_id = cid
rtype='s' (i.e. subject)
Yes
No
Insert into TempSubjects values (qid, super_id)
Insert into TempObjects values (qid, super_id)
Select super into super_id from Direct_SubClassOf2 where sub=super_id
super_id exists ?
Yes
No
End

```
7.2.1 find_super_property_proc
Role:
  Find the super property of given predicate, and temporaly save into the
  TempPredicates Table

--------------------------------------------------------------------------

7.2.2 find_super_class_proc
Role:
  Find the super class of given subjects/objects, and temporaly save into
  the TempSubjects/TempObjects Table
```

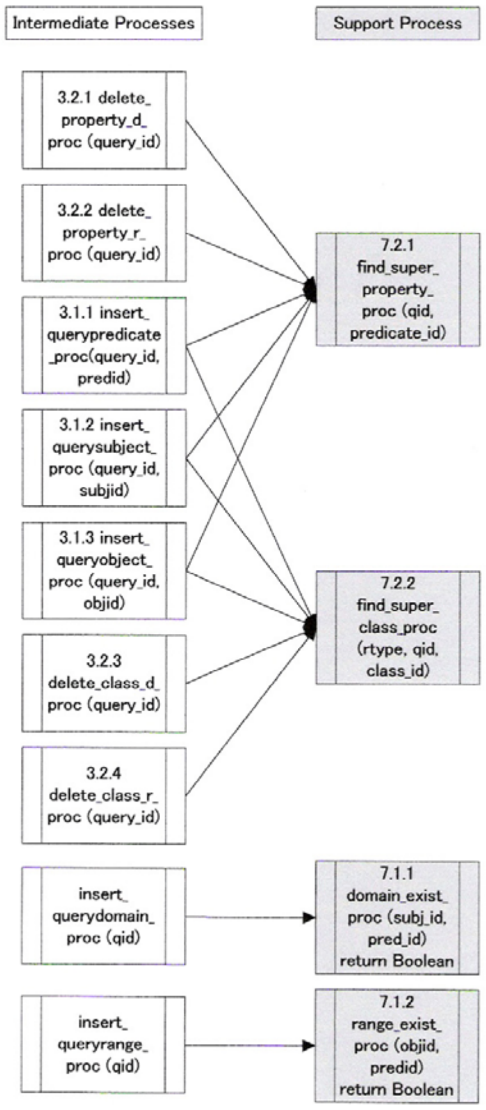
Appendix C. Relation among Triggers and Procedures

Appendix C1 Flow of Triggers Procedures



Events
Intermediate Processes
Event Notices
0.1.1 insert_property_trig
0.1.2 delete_property_trig
0.1.3 insert_class_trig
0.1.4 delete_class_trig
0.5.1 delete_requester_trig
0.2.1 insert_dsubpropertyof_trig
0.2.2 delete_dsubpropertyof_trig
0.2.3 insert_dsubclassof_trig
0.2.4 delete_dsubclassof_trig
0.3.1 insert_domain_trig
0.3.2 delete_domain_trig
0.3.3 insert_range_trig
0.3.4 delete_range_trig
0.4.1 insert_query_trig
0.4.2 update_query_trig
0.4.3 delete_query_trig
1.1.1 insert_property_proc (resource_name, :new.id)
1.1.3 insert_class_proc (resource_name, :new.id)
1.4.1 insert_query_p_proc (query_id, predid)
1.4.2 insert_query_s_proc (query_id, subjid)
1.4.3 insert_query_o_proc (query_id, objid)
1.2.2 delete_dsubpropertyof_proc (super_id)
1.2.4 delete_dsubclassof_proc (super_id)
1.2.1 insert_dsubpropertyof_proc (pid)
1.2.3 insert_dsubclassof_proc (pid)
2.1.1 insert_querypredicate_trig
2.1.2 insert_querysubject_trig
2.1.3 insert_queryobject_trig
1.3.1 insert_domain_proc (pred_id, subj_id, pid, cid)
1.3.3 insert_range_proc (pred_id, obj_id, pid, cid)
2.2.2 delete_dsubpropertyof_2_proc (super_id)
2.2.4 delete_dsubclassof_2_proc (super_id)
3.1.1 insert_querypredicate_proc(query_id, predid)
3.1.2 insert_querysubject_proc (query_id, subjid)
3.1.3 insert_queryobject_proc (query_id, objid)
3.2.1 delete_property_d_proc (query_id)
3.2.2 delete_property_r_proc (query_id)
3.2.3 delete_class_d_proc (query_id)
3.2.4 delete_class_r_proc (query_id)
4.1.1 insert_querydomain_proc (qid)
4.1.2 insert_queryrange_proc (qid)
4.2.1 delete_property_d2_proc (query_id)
4.2.2 delete_property_r2_proc (query_id)
4.2.3 delete_class_d2_proc (query_id)
4.2.4 delete_class_r2_proc (query_id)
1.4.3 delete_query_proc (qid)
5.1.1 insert_querydomain_trig
5.1.3 insert_queryrange_trig
5.1.2 delete_querydomain_trig
5.1.4 delete_queryrange_trig
6.1.1 insert_queryresult_trig
Notices
6.1.2 delete_queryresult_trig

Appendix C2 Sub Procedure


Intermediate Processes
Support Process
3.2.1 delete_property_d_proc (query_id)
3.2.2 delete_property_r_proc (query_id)
3.1.1 insert_querypredicate_proc(query_id, predid)
3.1.2 insert_querysubject_proc (query_id, subjid)
3.1.3 insert_queryobject_proc (query_id, objid)
3.2.3 delete_class_d_proc (query_id)
3.2.4 delete_class_r_proc (query_id)
insert_querydomain_proc (qid)
insert_queryrange_proc (qid)
7.2.1 find_super_property_proc (qid, predicate_id)
7.2.2 find_super_class_proc (rtype, qid, class_id)
7.1.1 domain_exist_proc (subj_id, pred_id) return Boolean
7.1.2 range_exist_proc (objid, predid) return Boolean