Lightweight Synchronization of Ontologies

Arun Sharma

Lightweight Synchronization of Ontologies

Master's Thesis

Arun Sharma
Mat. No. 262119

July, 2006

Under the supervision of:

Prof. Dr. Matthias Jarke
Lehrstuhl für Informatik V
RWTH Aachen University
&
Dr. Jérôme Euzenat
Project EXMO
INRIA Rhone-Alpes

Advisor:
Dr. Christoph Quix
Lehrstuhl für Informatik V,
RWTH Aachen University

Acknowledgements

I would like to express my special gratitude towards Dr. Jérôme Euzenat for providing me the opportunity to undertake my thesis at INRIA, Rhone-Alpes and supervising me on this thesis. I would also like to thank Dr. Jason J. Jung for his proactive suggestions during the course of my thesis work.

I would also like to express sincere gratitude to Prof. Dr. M. Jarke and Dr. Christoph Quix for their guidance, advice, encouragement and invaluable feedback during this thesis work. Lastly, I would like to thank all the members of the EXMO research team at INRIA for taking part in the PicSter experiment used to evaluate the system and for their constructive feedback during the prototype development.

Declaration

I hereby declare that this master thesis is my own written and all references or assistances received during this work are completely indicated in the bibliography.

Aachen, 4th July 2006
Arun Sharma

Abstract

The semantic web is based on the idea of having formalized knowledge expressed on the web (in languages like RDF). However, we know that people do not like to strictly comply with some ontology and they would tend to add their own tags within existing ontology descriptions. This thesis addresses the issue of heterogeneity within the domain of photo annotation. It presents a peer-to-peer infrastructure and client software that enables users to provide ontology based photo annotations in a free manner (by using the most convenient vocabulary) and share them with other users in a peer-to-peer environment. Moreover, the thesis presents an ontology alignment based mediator service to translate queries among the peers.

Contents

Chapter 1

Introduction

The aim of this thesis is to develop a peer-to-peer architecture for making semantic queries in a heterogeneous environment. Introduction of the concept is provided in this chapter, which is divided into four sections. Section 1.1 describes the background and a real life scenario of the problem. Section 1.2 states the contributions made by this thesis. In the last section of this chapter, the outline of the thesis is explained.

1.1 Background and Problem

The thesis addresses the issues involved in the areas of photo annotations, semantic peer-to-peer networks and ontology reconciliation within the semantic web.

Recent advances in the technology and storage media have enabled us to store and distribute a large number of photographs. Searching a particular set of photographs from such a large collection is increasingly problematic. Although large memory spaces (databases) containing images exist, the tools for searching an image are limited. Typically, the databases provide an indexing scheme that allows keyword search but do not offer much help to the user in finding the desired photograph.

The semantic web proposes use of explicit background knowledge as a way to address the search problems encountered in multimedia storages. It allows creating a syntactic format (in languages like RDF [MM04]) that specifies the background information of resources. However, most semantic web based applications are distributed and heterogeneous. For instance, within the photo annotation domain, people typically maintain their personal photo collection locally and want to share them with others in a distributed environment. The integration of heterogeneous resources found in such applications is one of the main problems faced by the semantic web community. For contributing a solution to this problem, data is expressed in the framework of ontologies (theories describing the vocabulary used for expressing data). However, in applications like photo annotation tools, people want to use personally modified ontologies without sticking to any reference

ontology. This makes the ontologies themselves heterogeneous and some work has to be done to achieve interoperability: hence, the problem is only lifted at the ontology level.

Ontologies reconciliation is achieved by finding the correspondences between ontologies (called alignments). Finding these alignments is only the first step of the process. Very often these alignments have to be evaluated, improved and finally transformed into some executable procedure before being used by applications. Such ontology alignment techniques can be for mediating queries and answers in peer-to-peer systems and federated databases by transforming an ontology in order to integrate it with another one; generating a set of bridge axioms that will help identify corresponding concepts; and translating messages sent from one peer to another.

Having discussed the background basics, we explain the problem dealt within this by using an example scenario described below which is applicable within the domain of photo annotations.

Example Scenario

1. In near future, almost every computer user will have a large collection of pictures. In such a large collection, finding the suitable photographs can be tedious. Semantic web standards propose a way to provide background knowledge for these pictures in the form of *semantic annotations* and hence make the search process for meaningful and quick.

2. In a typical scenario, users would normally like to share their pictures and annotations with their friends and relatives, and would always prefer to do so quickly and conveniently. Although there are a number of web portals[1,2] which allow this functionality, they are not so useful when the pictures have to be distributed only with a small number of people. Moreover, uploading all the pictures on a web portal is time consuming and tedious. *Peer to peer architectures* propose a way to distribute information and resources in more time efficient and convenient way.

3. The semantic photo annotations are expressed using a formalized knowledge in languages like RDF. However, people do not like to strictly comply with some standard vocabulary and ontology and they would tend to add their own tags within existing ontology description.

4. Hence, the scenario turns up as the following: The users want to share the pictures and their annotations with their friends but they do not know the vocabularies in which other users' pictures have been annotated.

---

[1] Flickr photo web portal: http://www.flickr.com
[2] Riya photo annotation portal: http://www.riya.com

Problem: *It is because of this tendency to have one's own vocabulary to express the metadata about the resources, digital pictures in our scenario, the problem of heterogeneity arises. Ontology alignment techniques propose a way to deal with this problem of semantic interoperability.*

This problem of heterogeneity occurs both in personal ontology scenario as well as in other corporate ontology scenarios. We address the issue in personal ontology based applications. Although similar concepts can applied to more complex ontology model based applications.

1.2 Contributions of this thesis

Although there are a couple of tools available for first two sub-scenarios mentioned in the previous section, to the best of our knowledge currently there is no tool that combines the functionalities of all of them and makes an interesting application. Tools based on first scenario [hWGS[+]05] [RDFPic] either put a restriction by providing a reference ontology for annotating pictures or ask the users to upload the annotations on to a web portal; tools based on second scenario [BEH[+]04] [NWQ[+]02] have not been applied to the domain of photography.

This thesis work combines the ontology based annotation techniques, p2p systems and ontology alignment techniques by developing a basic prototype. In particular it provides the following:

1. An architecture for solving the problem cited in the previous section. The architecture prototype is implemented in a peer-to-peer photo annotation scenario.

2. An application prototype for *personal ontology* based photo annotation and search: Although, a few ontology based photo annotation systems [SDWW01] [LB02] have been developed, all of them provide their own ontology or require the user to stick to a reference ontology. The prototype developed during this master thesis work allows the users to have their own ontologies by providing basic ontology editing functionalities in addition to the annotation functionality.

3. The architecture implemented can act as a test bed for evaluating ontology alignment algorithms by real users. A number of ontology algorithms have already been implemented in the alignment API developed by the researchers at INIRIA Rhone-Alpes. The architecture and real users' metadata developed and attained through this masters' work has been used to evaluate the algorithms implemented in the API.

4. The developed application tool has been used in a pilot experiment within the EXMO team of INRIA Rhone-Alpes.

1.3 Outline of thesis

This thesis is divided into six chapters. In this chapter, the contributions and introduction of the work was described. Also, the problem statement and the scenario were discussed.

Chapter 2 – The general concepts used in the thesis work are introduced. State of the art of the concepts used is then discussed. Two photo annotation tools are also discussed and a basis for the need of this thesis is developed.

Chapter 3 – The design tool and the prototype design are introduced. Moreover, the architecture of the application is constructed and explained in the following section.

Chapter 4 – Three implemented technologies are described in details. More specific functions of each approach and the realizations in the implementation are stated in this chapter.

Chapter 5 – An evaluation model of this thesis is explained. An experiment performed during the thesis is explained and the results are evaluated. The application developed is then compared with other similar systems.

Chapter 6 – The system developed is then compared with some of the existing systems.

Chapter 7 - The conclusion of this thesis and a summary of all experiences in this work is described. Finally, the future work stated as last section in this chapter. At the end of this thesis, the appendix and references of this thesis are presented.

Chapter 2

State of the Art and Fundamentals

This chapter presents an overview of state-of-the-art technologies which are related to the thesis work.

The World Wide Web has made a huge amount of information electronically available and the trend is growing continuously. So far, various communities have taken advantage of the current web functionalities to strengthen communication and information exchange not only within the community but also with external communities or individual users. Various web portals have appeared with the purpose of providing an open and effective communication forum for their members. In a typical case, a portal collects and presents relevant information for the community, and users can share the resources or information to the community.

However, current Web technology presents many limitations to make information accessible for users in an efficient manner. The general problems to find information on the Web are: searches are imprecise, often yielding matches to many thousands of hits. Moreover, it is not possible to directly retrieve a particular piece of information, instead users have to read through all retrieved documents and identify the information manually. These limitations naturally appear in existing web portals based on this technology, making information searching, accessing, extracting, interpreting and processing a difficult and time-consuming task. In this context, the Semantic Web aims at automated information access and use based on machine-processable semantics of data.

This chapter is divided into six sections. Section 2.1 discusses ontologies as the backbone of the Semantic Web. Section 2.2 gives an overview of the ontology query languages and briefly presents one of them which has been applied in this thesis work. Section 2.3 presents the currently applied techniques for dealing with the semantic interoperability problem as discussed in the last chapter. An overview of the peer-to-peer paradigm has been mentioned on section 2.4. Section 2.5 introduces the related work which also provides a basis and a scope for this thesis. Finally, section 2.6 discusses the conclusion of this chapter.

2.1 Ontology Languages

Semantic Web enables automated information access and use based on machine-processable semantics of data. Ontologies [Fen01] are backbone technology for the Semantic Web and - more generally - for the management of formalized knowledge in the context of distributed systems. They provide "machine-processable semantics" of data and information sources that can be communicated between different agents (software and people). In other words, information is made understandable for the computer, thus assisting people to search, extract, interpret and process information.

The proposed mechanism for meaningful communication between people and / or machines within the World Wide Web is to add semantic markup to Web resources in order to explicitly describe their content. This semantic markup makes use of terms for which ontologies provide a concrete specification of their meaning.

The significant term structure of ontology languages currently under development for the Web consists of at least two elements: *classes* and *relationships* (properties) that can exist among classes. Different types of ontologies are distinguished according to the purpose of their usage. The most common ontology types used in Semantic Web are domain ontologies and application ontologies. Several ontology languages have been developed during the last few years which are defined keeping the Web in mind. For applications on the web it is important to have a language with a standardized syntax. Because XML emerges to be the standard language for data interchange on the web, most of the ontologies use XML syntax. Two such languages RDFS [BG04] and OWL [McvH04] are briefly discussed in next subsections. For the purpose of this thesis OWL ontologies have been adapted because it is increasingly becoming a candidate for the standard vocabulary description language.

2.1.1 RDF Schema

The two basic structuring elements as mentioned above, namely classes and relationships are provided by the Resource Description Framework Schema (RDFS), the lowermost ontology language of the Semantic Web language layer architecture. As with the following ontology languages, RDFS usually is serialized as XML document in order to meet the syntactical requirements of today's Web communication protocols. RDFS can be considered as a very simple ontology language allowing the definition of class hierarchies via `subClassOf` statements. For example, `Toyota` can be defined as some kind of `Car` as follows:

```
<rdfs:Class rdf:ID="Toyota">
     <rdfs:subClassOf rdf:resource="#Car"/>
</rdfs:Class>
```

Semantically this can be expressed in first order logic (FOL) as an implication between two unary predicates: $\forall x: Toyota\,(x) \Rightarrow Car\,(x)$.

The possible combinations of classes and properties can be restricted by qualifying the domain and range of properties. An owner relationship for cars, called `Car-Owner`, is narrowed in its domain to the class `Human` and in its range to `Car` in the following:

```
<rdf:Property rdf:ID="Car-Owner">
    <rdfs:domain rdf:resource="#Human"/>
    <rdfs:range rdf:resource="#Car"/>
</rdf:Property>
```

The FOL correspondence to properties are binary predicates. According to that, the semantics of the property definition is as follows: $\forall x, y: Car - Owner\,(x, y) \Rightarrow Car\,(y)$ and $\forall x, y: Car - Owner\,(x, y) \Rightarrow Human\,(x)$. Property hierarchies can also be defined analogous to class hierarchies using `subPropertyOf` statements.

2.1.2 Web Ontology Language (OWL)

OWL is developed as a vocabulary extension of RDF Schema and is derived from the DAML+OIL Web ontology language. This extension covers class language constructs like conjunction, disjunction, negation, existential and universal qualified quantification and cardinality constraints of properties (plus some others). OWL itself provides three increasingly expressive sublanguages. The least expressive sublanguage is OWL Lite. We briefly discuss some of the constructs which are typical of OWL language. Syntax and semantics of all constructs can be found in [McvH04].

In OWL, a class can be defined as conjunction of other classes or class descriptions using the `intersectionOf` statement. For example, it might be rational to define `Toyota` as the conjunction of the classes `Car` and `Made-in-Japan`:

```
<owl:Class rdf:ID="Toyota">
    <rdfs:subClassOf>
    <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Car"/>
            <owl:Class rdf:about="#Made-in-Japan"/>
        </owl:intersectionOf>
    </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

Semantically, this corresponds to logical conjunction in FOL:
$\forall x: Toyota\,(x) \Rightarrow Car\,(x) \wedge Made - in - Japan\,(x)$.

OWL provides a way for specifying universal qualified quantification for locally restricting the range of a given property within a class definition. For example, a `GermanCar` is a car for which all fillers of the property `MadeIn` are of type `Germany`:

```
<owl:Class rdf:ID="GermanCar">
 <rdfs:subClassOf>
  <owl:Class>
   <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Car"/>
    <owl:Restriction>
     <owl:onProperty rdf:resource="#MadeIn"/>
     <owl:hasValue rdf:resource="#Germany"/>
    </owl:Restriction>
   </owl:intersectionOf>
  </owl:Class>
 </rdfs:subClassOf>
</owl:Class>
```

The fragment from above has the following semantics in FOL:
$$\forall x: GermanCar(x) \Rightarrow Car(x) \wedge \left(\forall y: MadeIn(x, y) \Rightarrow y = Germany \right).$$

Additionally, OWL Lite extends RDFS for transitive, inverse, and symmetric properties. A perfect example of a transitive property within the car world is the car-model relationship `olderModel`:

```
<owl:TransitiveProperty rdf:ID="olderModel"/>
```

Semantically, a transitive property enforce that:
$\forall x, y, z: olderModel(x, y) \wedge olderModel(y, z) \Rightarrow olderModel(x, z)$. An inverse property of a given property can be defined as follows:

```
<owl:ObjectProperty rdf:ID="Is-Car-Of">
     <owl:inverseOf rdf:resource="#Car-Owner"/>
</owl:ObjectProperty>
```

In FOL this means: $\forall x, y: Is - Car - Of(x, y) \Rightarrow Car - Owner(y, x)$ and vice versa. A symmetric property can be introduced as follows:

```
<owl:SymmetricProperty rdf:ID="Friend-Of"/>
```

This definition imposes the following FOL semantics:
$\forall x, y: Friend - Of(x, y) \Rightarrow Friend - Of(y, x)$

The underlying semantics of these definitions allows inferring logically entailed knowledge implicitly encoded in the cars domain ontology discussed here. Usually, logical reasoning systems are used for making such knowledge explicitly available for users.

2.1.3 Ontology Classification and Use

There are several classifications for ontologies extensively covered by available literature in different fields ([Fen01] [Bru03]). Some are based on quantity and conceptualization type, others are established based on the scope of their applicability, and still others based on their usability and/or reusability.

Two classification schemes which are along the goals of this thesis are mentioned below. The first is based on complex needs to represent the domains of interest and also on the type of use to be made of them in the system. The second focuses on the levels of decentralization of the system and the possible dissemination and spreading of concepts across the entire network.

2.1.3.1 Lightweight and heavyweight ontologies

Lightweight and *heavyweight* ontologies; both contain concepts, generalizations and relationships, but heavyweight ontologies, in general, also allow specification of cardinality constraints, refined statements and axioms, while lightweight do not:

- Heavyweight: the ontologies contain rules in formal computer language and are used to perform various forms of runtime automated reasoning. In defining heavyweight ontologies the language must be selected carefully because high expressiveness is needed to declare the large number of axioms. Since they can also be used for reasoning, the language must support them. Some examples of languages include OIL/DAML[1]. They are normally used as references, since their use is cumbersome.
- Lightweight: these only describe concepts, relations and functions, not rules. The concepts described by their attributes are organised using only the subclass of the relationship. These ontologies are used primarily as a base for organising and standardizing content. Taxonomies, concept hierarchies, and object models and diagrams are common ways to express lightweight ontologies. Their objective is to be shared by a series of users who converge on a particular common world view.

2.1.3.2 Local and shared ontologies

The classification based on local vs. shared ontologies addresses the decentralization issue of any ontology-based application. In open environments, ontologies are developed and maintained independently of each other in a distributed setting. Therefore, two systems may use different ontologies to represent their view of the domain.

---

[1] DAML + OIL reference: http://www.w3.org/TR/daml+oil-reference

If each source has its own *local ontology*, to achieve interoperability the ontologies must be brought together by mapping links. Here the link between the ontologies must be small.

If all the local ontologies are developed from a common vocabulary, we use the term *shared ontologies* to describe the ontologies that share the same vocabulary but are allowed to describe more complex semantics. This approach allows a degree of local autonomy and facilitates the mapping between two ontologies predetermined by their relationships with the shared mediated ontology.

2.1.4 Expressivity and Complexity

Expressiveness of an ontology language is a key decision factor in determining its applicability and usefulness in a particular domain description. Normally, there is always a trade-off between the expressive power and efficient reasoning support provided by a language. Generally, the richer the expressiveness of an ontology language, the higher is the complexity associated with it which ultimately results in lack of support for efficient reasoning.

Any ontology language has got two key requirements, namely, a well-defined syntax and a well-defined formal semantics. A *well-defined syntax* is a key requirement for making the ontology machine-readable. RDF/S certainly has a very well defined syntax. However, because of its lack of reasoning support, it lacks on the semantics issue. The issues typical of RDF/S are discussed later in this section.

Formal semantics is used to describe a precise meaning of knowledge. It is an important factor for ontology languages because the semantics does not refer to subjective perceptions and is interpreted in one single way by all machines and/or humans. A well-defined semantics allow machines to reason about the knowledge. For ontological knowledge, reasoning has to be done on concepts like class relations such as equivalence of classes, consistency and class memberships.

To support reasoning, proper semantics is required which should be able to check the consistency of the ontology and knowledge; and should be able to classify instances in classes. It should also be able to check unintended relationships between the classes. These requirements are important for the purpose of designing large ontologies which are developed by multiple authors. They are also important for integrating and sharing ontologies from multiple sources.

RDF/S has got its own limitations. RDFS expresses ontological knowledge by using an organization of vocabulary in typed hierarchy. The hierarchy is defined using primitives such as subclass and sub-property relations; domain and range restrictions; and instances of classes. However the language provided by RDFS is weaker because of a number of reasons like – Cardinality restrictions are impossible to express in RDF; RDFS does not allow Boolean combination of classes; it is not possible to express disjoint classes, and so on [Sha05].

Depending on the ontology formalism different mechanisms can be used to enhance systems usability. For example a reasoner can be employed to check cardinality constraints and class membership or an inference engine could interpret symmetric or transitive relationships. Reasoning is an important factor in determining the quality of an ontology. It is important to identify inter and intra ontology relationships and to check consistency of the annotations attached to the ontology.

Description logics provide the reasoning support and hence act as a very strong base on which latest ontological languages have been developed. [BHS03] defines Description logics as following: Description logics (DL) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way.

Since description logics provide a well-defined semantics and powerful reasoning tools which are needed for Semantic Web and its integration; researchers of the semantic web community decided to use its expressive power for developing ontology languages. OWL is one such language which is based on RDFS and its semantics is defined in expressive description logic. It is considered to be accepted as a potential standard for the Semantic Web.

Since OWL is developed on top of RDF, ideally the best approach to increase the semantic capability of language would be to use complete RDF syntax and a full logic together. However due the tradeoff discussed above, such a language will be computationally inefficient for many purposes. Because of these reasons, the W3C Web Ontology Group decided to define OWL as three different sublanguages *viz.* OWL Lite, OWL-DL and OWL Full; each of which has got varied level of expressiveness and complexity.

2.1.5 Editing

A number of ontology editors like Protégé, OntoEdit have been developed for ontology creation and their edition. These editors are quite exhaustive by nature and provide a number of ontology management functions. For the purpose of this master work, some stand-alone simple ontology editing functionalities required for ontology based photo annotation have been developed.

Some other aspects of ontologies especially when speaking in terms of interoperability problem are discussed in section 2.4

2.2 Ontology Query Languages

The support of querying facilities has always been a primary requirement for repositories of any kind. The proliferation of knowledge caused by the widespread use of the Web as a knowledge communication platform has posed the same and

even more imperative requirements for performing queries and thus locating desirable resources into the vast information space. However, the data models used to represent and encode knowledge on the Web differ from the traditional data structures. As discussed in previous section, RDF, RDFS and OWL are the emerging standards used to encode web-based data. Thus, the functionality a querying language should support must take into account the structure and the peculiarities of the new paradigms. In particular, RDF and OWL are based on a directed labeled graph data model, which allows labels both on nodes and edges, while XML uses labels only on edges. Both models can be serialized in a number of different representations, one of which is the use of triples in the form of <predicate, subject, object>.

This section briefly gives an overview of the existing ontology query languages and their requirements and explains the SPARQL [PS06] query language which is used in this thesis work for querying the ontology based metadata. It also briefly compares SPARQL with two other query languages. However before describing these languages, the features that are typical of query languages are mentioned as below:

- *Expressiveness.* Expressiveness indicates how complex queries can be formulated in a given language. Usually, expressiveness is restricted to maintain other properties such as safety and to allow an efficient execution of queries.
- *Closure.* The closure property requires that the results of an operation are again elements of the data model. This means that if a query language operates on the graph data model, the query results would again have to be graphs.
- *Adequacy.* A query language is called adequate if it uses all concepts of the underlying data model. This property therefore complements the closure property: For the closure, a query result must not be outside the data model, for adequacy the entire data model needs to be exploited.
- *Orthogonality.* The orthogonality of a query language requires that all operations may be used independently of the usage context.
- *Safety.* A query language is considered safe, if every query that is syntactically correct returns a finite set of results (on a finite data set). Typical concepts that cause query languages to be unsafe are recursion, negation and built-in functions.

2.2.1 Overview of ontology query languages

There are a number of ontology query languages which are based on the criteria for stating whether the query language can exploit the presence of ontology (schema) knowledge, i.e., if the ancestor/descendant traversal of class/property hierarchies can be performed and what filtering conditions can be posed on class/property hierarchies. The basic criterion of the *Data Querying* is the ability of the query language to provide constructs for calculating the extent of a property/class, i.e., the

12

ability to find all the resources defined as instances of a particular property/class. Furthermore, in this category fall criteria such as the support of complete Boolean filters (negation, conjunction, disjunction), set-based operations (union, intersection, difference), arithmetic operations on data values and container values constructors, i.e., language constructs that can be used to build sequences or bags.

Data/Ontology Querying refers to the competence of the query language in combining data and ontology querying in a query expression. The basic way to perform such an advanced query is to use generalized path expressions. A generalized path expression queries data and ontology at the same time, thus enabling the formulation of queries on description bases without exact knowledge of the ontologies employed. Generalized path expressions, as well as the support of existential/universal quantifiers and nested queries are indicative criteria of the expressive power of a query language. Furthermore, there are a number of additional features which are used to evaluate the effectiveness of the query language and the added value of its use when it comes to real, large scale applications. The support of aggregate (min, max, average, sum, count), grouping (e.g., an SQL-equivalent to group_by clause), sorting (for ordering querying results) and built-in data functions (e.g., math and string/date converting functions), in combination with the facilities to support the definition of arbitrary functions and user-defined inference rules are features that add to the expressive power of the query language.

Work on RDF query languages has been progressing for a number of years. Several different approaches have been tried, ranging from familiar looking SQL-style syntaxes, such as RDQL [Sea04] and Squish [Mil02], through to path-based languages like Versa . SPARQL follows this trend and is addressed in the next sub-section.

2.2.2 SPARQL Query Language

SPARQL is a query language and protocol for RDF being designed by the W3C and is undergoing standardization by the RDF Data Access Working Group (DAWG). As mentioned in the SPARQL working draft [PS06], SPARQL is a query language for getting information from such RDF graphs. It provides facilities to:
- extract information in the form of URIs, blank nodes, plain and typed literals.
- extract RDF subgraphs.
- construct new RDF graphs based on information in the queried graphs.

The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, which is like an RDF triple, but with the possibility of a variable instead of an RDF term in the subject, predicate or object positions. Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern.

SPARQL actually consists of two separate specifications. The query language specification makes up the core. Additionally, the query results can be described in XML format for serializing the results of a SPARQL SELECT (and ASK) query. This simple format is easily processable with common XML tools such as XSLT.

The second specification is the data access protocol which uses WSDL 2.0 to define simple HTTP and SOAP protocols for remotely querying RDF databases. The XML results format is used to generate responses from services that implement this protocol.

The focus on SPARQL in this thesis has been on the query language itself. The W3C defines four distinct types of SPARQL Queries depending on different keywords used in the query: select, ask, construct and describe. The type of the query determines the result format. Each of the query type has been shown below with a basic example. A detailed syntax and description can be found in [PS06].

- SELECT: returns a table of variable/value bindings. These bindings can also be returned in an XML format. The example below shows a SPARQL query to find the name and email address of a person from the information in the given RDF graph (graphG1). The query consists of four parts, the PREFIX clause determines the namespaces used in the query; the SELECT clause identifies the variables to appear in the query results; the FROM clause determines the RDF graph to be queried; and the WHERE clause has one triple pattern.

  Query:
  PREFIX ex: <example.org/ex1>
  SELECT ?name ?mbox
  FROM <http://example.org/graphG1.txt>
  WHERE {
  ?X foaf:name ?name .
  ?X foaf:mbox ?mbox .
  }

  Result:

?name	?mbox
Jerome	Jerome.Pierson@inrialpes.fr
Faisal	Faisal.Alkhateeb@inrialpes.fr
Arun	Arun.Sharma@inrialpes.fr

- ASK: returns a boolean value as the result, depending on the predicates used in the WHERE clause

  Query: The following SPARQL query checks if there exists an RDF subject in the given graph with name and mbox as the objects.

14

```
PREFIX ex: <example.org/ex1>
ASK
FROM <http://example.org/graphG1.txt>
WHERE {
    ?X foaf:name ?name .
    ?X foaf:mbox ?mbox .
}
```

Result: True

- CONSTRUCT: The CONSTRUCT result form returns a single RDF graph specified by a graph template. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables into the graph template and combining the triples into a single RDF graph by set union. The following SPARQL query return the complete RDF triples that satisfy the conditions mentioned in the WHERE clause.

  Query:
  ```
  PREFIX ex: <example.org/ex1>
  CONSTRUCT { ?X foaf:name ?name}
  FROM <http://example.org/graphG1.txt>
  WHERE {
      ?X foaf:name ?name .
      ?X foaf:mbox ?mbox .
  }
  ```

  Result:
  ```
  @prefix foaf: <http://xmlns.com/foaf/0.1/>
  _:b0 foaf:name "Jerome"
  _:b0 foaf:mbox <mailto: Jerome.Pierson@inrialpes.fr>

  _:b1 foaf:name "Faisal"
  _:b1 foaf:mbox <mailto: Faisal.Alkhateeb@inrialpes.fr>

  _:b2 foaf:name "Arun"
  _:b2 foaf:mbox <mailto: Arun.Sharma@inrialpes.fr>
  ```

- DESCRIBE: According to current conventions, DESCRIBE form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor. The query pattern is used to create a result set. Since there is not a common consensus on the use of DESCRIBE queries, they were not used by this project.

2.3 Semantic Interoperability

With current ontology standards, a manual or automatic ontology co-ordination is a challenging task. In evolving domains, it is expected that ontologies will not remain

static and various versions of ontologies will have to be tracked. Interdisciplinary ontologies may need to be created from existing domain-specific ontologies, domain specific ontologies may need to be merged with more general ontologies, different versions of single-domain ontology may need to be merged, and new information may need to be merged with existing ontologies. Furthermore, new ontologies may be built by merging information from heterogeneous databases or other information sources. Hence, these ontologies will have to be reconciliated. Such a semantic reconciliation has been termed as *ontology alignment* in [EP04] [EP04], which involves finding relationships between objects belonging to different ontologies. These ontology alignment results can be used for various purposes such as displaying the correspondences, transforming one source into another, creating a set of bridge axioms or rules between the ontologies, or generating query wrapper. [EP04] defines the problem of ontology alignment as following: given two ontologies each describing a set of discrete entities (which can be classes, properties, rules, predicates etc.), find the relationships (e.g. equivalence or subsumption) holding between these entities.

2.3.1 Ontology Matching vs. Schema Matching

Before introducing the specifics of ontology alignment techniques, it is important to differentiate two form of interoperability, *viz.* syntactic and semantic interoperability. While syntactic interoperability is mostly dealt with schema matching techniques [RB01] [RaB01] [EBB$^+$04] [Shv04], semantic interoperability is addressed by ontology matching/alignments. Ontologies provide a shared and common understanding of domain that can be communicated between people with distributed or heterogeneous application systems. On the other hand a schema defines a set of relations and certain integrity constraints. A detailed comparison of schema matching and ontology is out of the scope of this paper [see 7]. However, it is worthwhile to note that although (unlike ontologies) schemas do not provide explicit semantics for the data they represent; both schemas and ontologies provide a vocabulary of terms that describes the domain of interest.

2.3.2 Ontology Alignment

Ontology Alignment ([EP04] [KS03] [ES05] [Euz05]) is the task of establishing a collection of binary relations between the vocabularies of two ontologies. Since a binary relation can itself be decomposed into a pair of total functions from a common intermediate source, the alignment of two ontologies O_1 and O_2 can be described by means of a pair of ontology mappings which generates an alignment O_0. Such an aligned ontology is generated in a way that further merging of ontologies can be carried out. The intuitive way of such an alignment consists of defining a pair-wise distance between entities of ontologies and computing the best match between them by minimizing the distance of similarity measure between them. Roughly speaking, these entities correspond (but not limited) to the following: classes, relations, properties and instances of classes and properties.

2.3.3 Classification of ontology alignment techniques

The pair-wise similarity distance measure and matching described in previous section can be done by different methods. As discussed in [SE05] and [EBB[+]04], the use of appropriate methods depends on the objects to be compared, their context and their external semantics. Figure 2.1 taken from [SE05] presents such a context classification. The figure decomposes these methods in two perspectives: kind of techniques used (top-to-bottom) and the kind of manipulated entities (bottom-to-top). The lower layer is concerned with the type of input considered by the technique use which can be broadly classified into extensional, terminological, structural and semantics. The next layer further classifies it on language methods and distinguishes structural methods into external or internal structures. This lower classification is briefly described below.
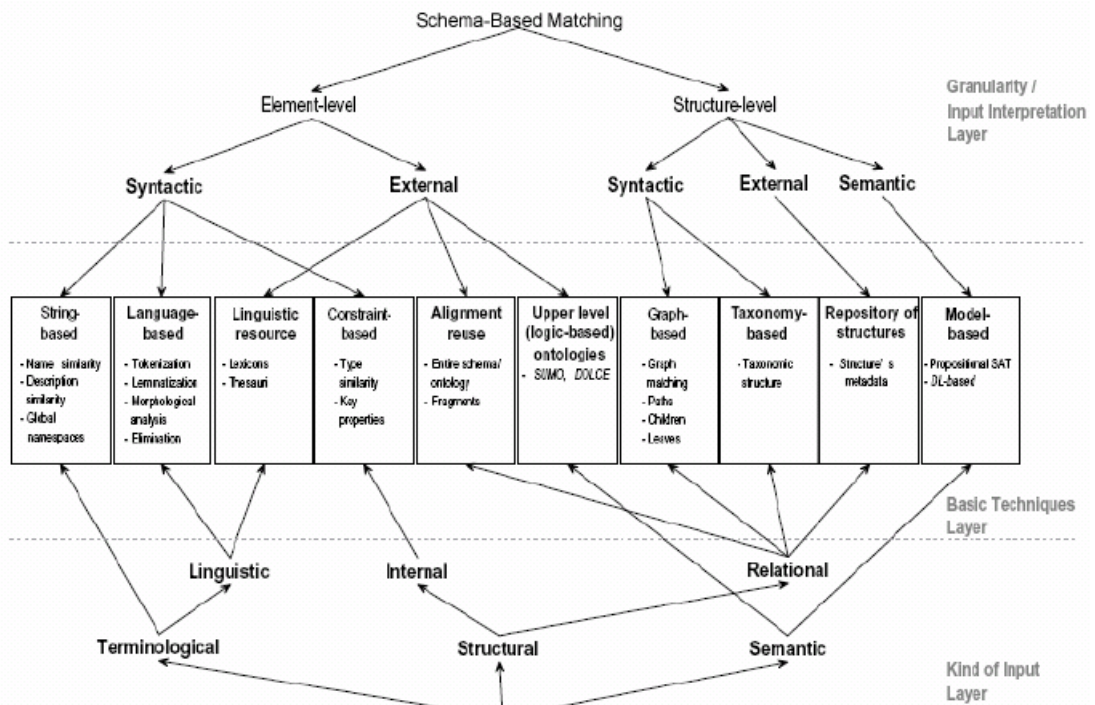
Figure 2.1: Classification of ontology alignment approaches [SE05]

Terminological methods

Terminological methods focus on string comparison. They can be applied to the name, the label or the comments concerning the entities to find those which are similar. This can be used for comparing class names and/or URI. In terminology, the relation between terms and concepts can have multi-meanings, i.e. terms can refer to more than one concept, and a single term can have many variants, all

related to a single concept. The concepts can be represented by different names in different ontologies.

In [EBB⁺04] term-comparison has been specified in two categories which depends on the consideration of character strings only or if they use some linguistic knowledge.

String based methods: These methods take advantage of the structure of the string as a letter sequence. String based methods normally find similar classes but not the alignment between them. Strings can be compared in many ways depending on ones perception, for e.g. an exact sequence of letters, an erroneous sequence of letters, a set of letters, etc.

Language based methods: These methods rely on using Natural Language Processing techniques to find association between instances of concepts. These methods essentially depend on the expressiveness and productivity properties of the natural language, which implies that even technical terms can be expressed in different ways without altering their meaning.

Structural Methods

These methods are based on comparing the structure of entities of ontology instead of comparing their names. This comparison can be subdivided in a comparison of the internal structure of an entity or the comparison of the entity with other entities to which it is related.

Internal Structure: These methods are constraint based aligning methods which use the criteria such as the range of properties, cardinality, transitivity and symmetry of properties to calculate similarities between the entities. Since the entities with similar internal structure or properties can be numerous, these methods are known to create alignment clusters rather than to discover accurate correspondence between the entities.

External Structure: This matching technique is based on the hierarchical positions of entities in their respective ontologies. It is based on the criteria that if two entities from two ontologies are similar, their neighbours might also be similar. The criteria which can be used to find such similarity may include finding similarities at super and sub entities, sibling entities, descendant and leaf entities.

Semantic Approaches

The key characteristics of semantic methods are that they have model-theoretic semantics which is used to justify their results. Hence they are deductive methods. Propositional satisfiability (SAT) and modal SAT techniques represent a few examples of such techniques. In SAT techniques the alignment is described as a translation of the matching problem and mapping queries into a propositional

18

formula and then to check its validity. Modal SAT can be used for extending the methods related to propositional SAT to binary predicates. Its basis is to delimit propositional SAT from the case of trees which allows handling only unary predicates (like classes) by admitting binary predicates (like slots). The idea hence is to enhance propositional logics with modal logic or description logic operators. Therefore the matching problem is translated into a modal logic formula which is further checked for its validity using sound and complete satisfiability search procedures. Description logic techniques can be used to establish the relations between classes in a purely semantic manner.

2.3.4 Ontology Alignment API (OLA)

OLA [EP04] developed within the EXMO team at INRIA is a class of algorithms for ontology alignments which targets the following characteristics:
- covering all the possible characteristics of ontologies (i.e., terminological, structural and extensional);
- taking care of collection structures (lists, sets) and accounting for them during matching;
- expliciting all recursive relationships and finding the best matching through iteration.

OLA is implemented for ontologies described in OWL-Lite [EP04]. It uses the Alignment API and implementation that was recently developed by the Exmo team at INRIA-Rhone Alpes [Euz04] and the University of Montréal. The algorithm first compiles the OWL ontologies into graph structures unveiling all relationships between entities. These graph structures produce the constraints for expressing a similarity between the elements of the ontologies. The similarity between nodes of the graphs follows two principles: (i) it depends on the category of node considered (e.g., class, property), and (ii) it takes into account all the features of this category (e.g., superclasses, properties). This similarity is a weighted linear aggregation of the similarity measures between all the entities a couple of entities is in relation. This accounts for all the relationships between entities. However, these features (like subclasses) are sets of entities, the similarity between these sets of entities, thus depends on a local matching between these entities. A matching of both sets is considered which is: (i) of maximal total similarity, (ii) exclusive, and (iii) of maximal size.

Similarity between labels can be produced by any kind of particular terminological method (e.g., string distance, linguistic evaluation). Similarity between data values and data types can be provided by specialized external similarity measures (e.g., Euclidean distance, symmetric difference distance).

The definition of this similarity provides a set of equations whose variables are the similarity values between entities of the ontologies. This set of equations cannot be solved directly due to local matching. Depending on the currently computed similarity, the matching as defined above can be different. An iterative algorithm

has been proposed in [Euz04] which compute a first approximation of the similarity (without the local matching), then compute the local matching and reiterate.

From this solution, it is possible to extract an alignment between the two ontologies. This ontology alignment API will be used in the master's thesis work to compare the existing alignment algorithms.

2.4 Peer to Peer Systems for Knowledge Exchange

Peer-to-Peer computing (P2P) allows devices to communicate and collaborate through a connecting network without reference to a central server. It also provides the functionality as a decentralized discovery system and point-to-point messaging system. The peers are software components that have to agree upon a common set of rules to publish, share and access "resources" like services, data or applications, and communicate among each others. P2P infrastructures are especially suitable for all areas where there can't be a centralized control. These infrastructures allows for the much needed degree of autonomy. A detailed description on advantages of P2P systems can be found elsewhere in the literature.

Napster [NAP01] was among the first real-world applications that used P2P technology to share music files. But in organizational/personal context, business applications for P2P computing fall into a handful of scenarios like:

- Peer-to-peer collaboration among the users.
- Helping businesses deliver services and capabilities more efficiently across diverse geographic boundaries.
- Allowing computing networks to dynamically work together using intelligent agents. Agents reside on peer computers and communicate various kinds of information back and forth. Agents may also initiate tasks on behalf of other peer systems.

Lately, P2P infrastructures have been considered to be especially suitable for distributed knowledge management applications, as used in this thesis.

But today's P2P solutions have some issues:
- In many projects, P2P is just a solution for distributing disk space. But this is the minor importance for knowledge management. It is the sharing of information and knowledge which needs to be supported.
- At the moment, search and retrieval are limited just to keyword-based search. No support for metadata is provided.
- Query answering uses resources in a single location, while many queries require combination of information from different sources.
- Knowledge sharing is limited to the exchange of entire files.

Theses issues make traditional P2P solutions invalid for knowledge sharing purposes. It is necessary to design and implement the system over a P2P framework, (like JXTA [JXTA]) to fit the distributed knowledge resources.

2.5 Closely Related Work

This section describes some of the work that has been done in the areas which are dealt within the master's thesis *viz.* annotation with ontologies and semantic peer-to-peer systems techniques.

2.5.1 Image annotation

Recent advances in storage media along with the web have enabled users to store and distribute photographic images worldwide. However, with this ever increasing content of digital images either locally or on the web, finding suitable photographs for a particular purpose is increasingly problematic. A number of annotation tools have been devised in this direction. Some of them are discussed below:

Standalone Tools: Lately, a number of tools have been developed that work as photo organisers. These tools, such as *iPhoto* and *Picasa*, provide a number of basic functionalities to annotate personal photo collection. Simple keyword based search is also provided. Such tools are good when used only for personal uses. However, they do not provide any functionality for sharing the photos with other users.

Web Portals: Since last few years, a number of websites have come-up which provide means to publish a collection of digital photos online in a centralized and organized fashion but with no formal knowledge specifications. These web portals (e.g. flickr.com) are based on semi-structured indexing schemes that allow a keyword search. Flickr's popularity has been fueled by its innovative community tools that allow photos to be *tagged* and browsed by *folksonomic* means. Although currently a hit among the masses, such portals lack the personalization aspect of searching and sharing. In a normal scenario tags are not sufficient to annotate the pictures properly. Hence although such portals are good for sharing the pictures with a larger group like on world-wide-web, they are not very useful when one wants to share the picture only with a limited number of people. Finally, keyword based searches return hundreds of pictures that may not be of interest to the user.

Ontology based image annotation: With the emergence of semantic web standards such as resource description framework and related languages like OWL, creating a syntactic format with semantic background knowledge is possible. This makes the process of searching and annotating more precise. This technique can be used to index and search collections of photographs by storing background knowledge about the pictures in ontologies. This section briefly describes some of the existing tools which provide the capability to annotate images based on ontological descriptions.

Photo Annotation Tool (University of Amsterdam) [SDWW01]

This tool was one of the first photo annotation applications to make use of ontologies. The goals of the tool development were intended for assisting magazine editors and others to search the photographs quickly from their collection. Following are the key highlights of this tool:

1. The tool used the notion of structured annotation by using subject matter ontologies written in RDF-Schema. Two kinds of ontologies were used: (a) A *photo annotation ontology*, which specifies an annotation's structure independent of the particular subject matter domain. This ontology provides the basic template for the annotation construction, and (b) *subject matter ontology* which specifies the vocabulary and background knowledge of the photo's domain. For this purpose, the tool developers developed a domain ontology based on phylum hierarchy of animal species through a set of subclass relations between the species classes.
2. The tool reads an RDFS file containing ontology specification which conforms to W3C standard. From the RDFS specifications, the tool generates a user interface for annotating photos. The user can then annotate the pictures based on the ontologies loaded by the tool. The user interface generator is defined independently of the ontology. The tool just reads the RDFS representing annotation schema. For each property of this schema that represents another compound schema, it generates a tab or sentence item.
3. The tool also provides a simple query interface to test the annotations. It searchers the database for annotations that have all properties specified in the target description filled with values that are equal to or specializations of the value in the target.

Although this tool did not provide extensive query mechanisms and was based on static ontologies, it was first of its kind which triggered research in ontology based annotation approaches in other field like music as well.

RDFPic (INRIA-Sophia Antipolis, MIT, Keio) [RDFPic]

RDFPic is also one of the fist tools providing ontology based annotation facility. It is a tool to ember RDF descriptions of a picture into the image itself. It is built around the concept of an RDF "schema". Each schema has a number of attributes whose values can be set. For instance, a schema "rendering" could include a "Renderer used" field. Each of these schemas has a tab in the left bottom section of the RDFpic interface. The tool provides two default schemas *viz*. Dublin Core and a Technical Schema which specifies the technical. Each of the attributes of a schema has a row in the tab pane. The tool can be plugged in with web-servers like Jigsaw [LM99] to serve either the either the JPEG image data or the RDF description that is stored in it. The choice can be made by using HTTP content negotiation to determine which of the two a client wants.

However, the tool doesn't take care of the existing annotation in the image files, for examples those generated by the camera in the EXIF tags. Moreover it is based on very basic schemas which don't take advanced semantic relations between the entities.

PhotoStuff (Mind Lab, University of Maryland) [hWGS[+]05]

PhotoStuff is an image annotation tool (written in Java) which uses an ontology to provide the expressiveness required to assert the contents of an image, as well information about the image (date created etc.). In order to provide this capability, an underlying OWL ontology has been defined that describes various concepts such as *region, image* etc. and properties including *hasRegions, depicts* etc. This allows assertions to be made stating that an image contains a region that depicts some concept.

PhotoStuff allows users to annotate regions of an image with respect to concepts in any ontology specified in RDFS or OWL. It provides the functionality to import images, ontologies, instance-bases, perform markup, and export the resulting annotations to disk or a Semantic Web Portal. PhotoStuff can extract the existing metadata from the images and encode them in RDF/XML formats. It also maintains a loose coupling with a Semantic Web portal and can retrieve all the instances that have been submitted to the portal and submit generated RDF image markup.

2.5.2 Semantic peer-to-peer systems

Numerous advantages have been cited in literature about the advantages a peer-to-peer system over centralized applications: scalability in data volumes, robustness against failure of single component; to name a few. However, besides being the solution to many problems, the large degree of distribution of Peer-to-Peer systems is also the cause of a number of new problems: the lack of a single coherent schema for organizing information sources across the Peer-to-Peer network hampers the formulation of search queries, duplication of information across the network results in many duplicate answers to a single query, and answers to a single query often require the integration of information residing at different, independent and uncoordinated peers.

The research community has recently turned to the use of semantics in Peer-to-Peer networks to alleviate these problems [PJR[+]04] [SWAP[2]]. The use of semantic descriptions of data sources stored by peers and indeed of semantic descriptions of peers themselves helps in formulating queries such that they can be understood by other peers, in merging the answers received from other peers, and in routing queries across the network. In particular, the use of ontologies and of Semantic

---

Web technologies has been identified as promising for Peer-to-Peer systems. This section briefly reviews two such existing systems.

Edutella

The Edutella network implements an RDF-based metadata infrastructure for P2P networks based on JXTA framework. In order to access content stored on the network it uses the query language RDF-QEL.

Edutella network connects highly heterogeneous peers (in their up time, performance, storage size, number of users, …) implementing a set of services: *Query service* (this is the most basic service and allow the peers register the queries it may be asked and standardized query and retrieval of RDF metadata), *Replication service* (providing support to storage by replicating data in different peers and workload balancing), *Mapping service* (translating between different metadata vocabularies to enable interoperability between different peers), *Mediation service* (to join data from different sources and reconcile conflicting and overlapping information), *Clustering service* (to set up semantic routing and semantic cluster using semantic information) and *Annotation service* (to annotate materials stored anywhere in the network).

An Edutella peer has some kind of local storage for RDF triples (e.g., a relational database), as well as some kind of local query language. (e.g., SQL). One of the Query service purposes is to abstract the metadata that describes the content of a peer, from various possible RDF storage layer language and user lever query language. For that, it uses the Edutella Query Exchange Language and Edutella Common Data Model (ECDM) that provide syntax and semantics needed to achieve this goal.

Edutella wrappers enable the peer to translate queries and results from Edutella query and result exchange format to the local format of the peer and vice versa, and to connect the peer to the Edutella network. For communication with the Edutella network the wrapper translates the local data model into the ECDM and connects to the network using JXTA P2P primitives, transmitting the queries based on ECDM in RDF/XML form.

In considering the functionality provided by Edutella, we can say that its main asset is that it favours broad interoperability among heterogeneous peers. Edutella Mapping Service will handle the mapping among peers so that the system properly manages the use of multiple ontologies with overlapping information. This is really its primary objective.

The ability to use metadata to specify the type of queries that a peer can receive favours enormously an adequate selection of peers to obtain the most precise response. However, there are no reliability mechanisms in one peer or another when faced with possible contradictory or imprecise responses.

24

Using a common language for exchanging queries and results requires local translation in each peer. This could result in reduced results performance, as well as limited capacity in expressiveness in the language of each peer. Additionally, each peer can only describe resources from a single source (which is known in Edutella as RDF repositories).

Another of the problems with Edutella is its possible inefficiency in managing traffic from queries over the network or Peer selection service routing: the selection is done not at the selecting peer but at the receiving peer. This means that the query is broadcasting through the network and then, this only scales well in small groups. It uses the standard services of JXTA to find peers and to route messages.

Bibster

Bibster [BEH[+]04] is a Peer-to-Peer system based on the SWAP architecture [SWAP], which allows to easily integrate, share and search bibliographic metadata using semantic technologies for the representation of the bibliographic instances and the peers' expertise to allow effectively routing of the queries. It addresses a typical problem in the daily life of a computer scientist, where one regularly has to search for publications or their correct bibliographic metadata. Semantic similarity measures identifying duplicates allow to visualize and to integrate the heterogeneous search results from the peers. Bibliographic entries are extracted from BibTex into an ontology. The query results themselves represent small ontologies, containing duplicates.

Finding duplicates is related to finding corresponding mappings. In both cases it is necessary to recognize identical objects despite their different identifiers. In the given scenario duplicates are bibliographic entries which refer to the same publication or person in the real world, but are modeled as different resources. The similarity function is based on different features of the respective instances. For persons one can refer to the name. For publications to title, authors, editors, journal, address, type of publication, etc. The function returns a value between 0 and 1 by applying specific heuristics to every feature: Strings are compared using the Levenshtein distance; the authors of publications are compared by comparing the two sets. Some domain specific features require special heuristics: if the type of one publication is "Misc", this only means that no further information about the type was available. If another publication is e.g. type "Article" the similarity is set to 0.5 rather than 0. Besides individual functions, the approach focuses on applying an aggregation function to achieve an overall similarity. Through transitive closure a set of identical entities is gained and duplicate query results are visualized as one merged resource. A relatively detailed explanation of similarity measure techniques is describes in the next section.

2.6 Conclusion

This chapter introduced some concepts according to the thesis requirement. Also, state-of-the-art of used technologies has been described.

A requirement case was formed after studying that current ontology based annotation systems do not allow a scope for personalization of the vocabulary used. Moreover, to the best of our knowledge, we have not come across any peer-to-peer and ontology based photo annotation tool. A comparison between the existing semantic P2P systems and the one developed during this thesis work is drawn in chapter 7.

Chapter 3

Design and Architecture

This chapter presents the high level system architecture of the application prototype called *PicSter* developed during the thesis work. Section 3.1 introduces the problem and their possible solutions along with the requirement specifications followed by section 3.2 gives the system design of the architecture. Implementation related information is described in the next chapter.

3.1 Introduction & Requirement Specifications

In this section, we revisit the example scenario described in section 1.2. The scenario is split into sub-scenarios as user requirements with each of them followed by possible solutions to the problem faced and a recommendation to the PicSter. Each sub-scenario is presented as a user requirement, following which a system requirement will be specified after analysing the possible solutions.

User requirement: *Users want to annotate local as well as web resources, images in the case of PicSter.*

Possible solutions: Schema based metadata; ontology based annotation.

The importance of data is quite well known but data without meaning are not very useful on their own, since they are ambiguous. The data used to describe the data itself is termed "metadata". One of the options used most to define metadata is the XML language. The metadata plays an important role in interoperability among different communities and applications by structuring the contents of the data.

However, something more powerful than metadata is required for a formal description of contents. Metadata allow us to structure the contents, but they do not help us define their semantics. One of the solutions is to use ontologies. The advantage of these is that they represent a standard and they are widely accepted by the research community.

Chapter 2 describes various aspects of ontologies in depth. Considering that the ontologies provide a form of understanding a particular domain, their use in information systems provides a series of benefits including the following:

- They provide mechanisms that, using a common vocabulary, allow access knowledge and information.
- They allow using a knowledge exchange format between different communities, providing communications protocols between groups with different needs and points of view stemming from their different contexts. Improved consistency and a lack of ambiguity improve communications and, thus, facilitate the exchange of knowledge.
- They facilitate interoperability and information integration between systems.
- They allow knowledge to be reused.
- Additionally, the knowledge, through the different ontology representations, becomes usable by automatic systems, which makes it possible to make inferences with stored knowledge, as well as reasoning, checking inconsistencies, and so on.

Given the above mentioned advantages, we decided to develop an ontology based system where users could annotate their images.

System requirement: An ontology based system for photo annotation.

Ontology Based Architecture: Figure 3.1 shows a generic architecture for ontology based applications. The figure presents a decomposed design of ontology based application. The ontology layer is concerned with the creation and maintenance of the model of the application. The middleware layer supplies common ontology-related services, while the application layer builds on the ontology and related services to provide some kind of ontology functionality to the end user.

The ontology layers' main goal is the acquisition of ontologies. We assume that most of the users' of PicSter will not be knowledge engineers and hence will acquire their ontologies from elsewhere (e.g. web).

User Requirement: *Although users will not be ontology experts, they would typically want to adapt existing ontologies to their needs; which will allow them to annotate their photos in a free way.*

These requirements lead to the following system level requirements.

System Requirement at Ontology Layer: The system should provide basic ontology editing functionalities.

System Requirements at Middleware Layer:
- A local repository to organise knowledge and information on a common vocabulary
- Access to optimized retrieval of knowledge
- Reusing existing knowledge and facilitating reasoning and inferences on existing knowledge.

System Requirements at Application Layer:
- Provide means to browse and visualize the ontologies.
- Embed other functionalities of PicSter like searching the metadata and to share the same with other users.
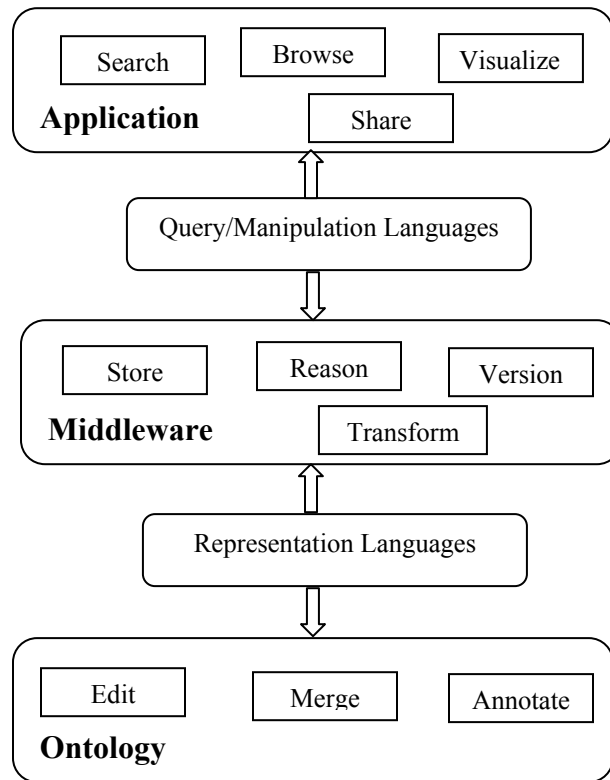
Figure 3.1: Architecture for Ontology-based Applications

User Requirement: *Users want to share the image metadata and possibly the resources with other users having similar interests.*

Possible Solutions: Submit the image metadata on to a web portal; a decentralized system using web services; a direct P2P system.

There exist a number of metadata storage portals like the Kowari[1] Metadata Store which act as a centralized database for metadata submission. However, given the needs for personalisation of photo annotation, metadata submission to a central server does not propose a sufficient solution. Additionally such portals do not provide the privacy required for domains such as photo annotations.

Although web services and semantic peer to peer systems are driven by similar needs of decentralisation, there exist a number of issues that distinguish them.

Most of the current solutions under the title of Web services have a number of elements that are common to their construction. For communication, they use XML to format their messages and deliver them over HTTP. If an interface language is required, another XML syntax called WSDL is used. To deal with the bootstrap issues associated with service and endpoint discovery, clients may access a public UDDI database. This basic framework allows clients to discover and communicate with a service on the public network

However, the characterisation of the web service operation is the classic client/ server model. The software fulfilling the role of the server will register with a centralized but replicated datastore. Software that fulfills the role of the client will contact the datastore to discover the server location and can then contact the server. The mechanism to ensure that the client and server can speak intelligibly to each other, or interoperate, is enforced by well-known standards.

Unlike Web services, where the client and server roles mean the system deployment requires two separate pieces of software, P2P networks have just one deployable software unit - the peer code. If the peer algorithm can provide for it, P2P networks are easy to extend. Peers may do exactly the same thing or there may be transient role assignments. The key element of a P2P system is the peer algorithm, the programmed behavior that makes the system fulfill its intended purpose. The algorithm enforces a logical network topology by defining the concept of neighbouring peers. The neighbour relationship enables the flow of the essential data in the network, a vital requirement in a system with no central data resource. The fact that the network may be a fluctuating and dynamic one, with peer neighbour relationships breaking and reforming as the load or infrastructure stability changes, is the core attraction of P2P systems.

The above mentioned discussion leads to the following PicSter specific requirement.

System Requirement: A semantic peer-to-peer architecture for metadata sharing and search processes.

For the purpose of searching the metadata on such a peer-to-peer network, users will have to send queries which are understandable by the other peers. As mentioned previously that the users would typically do not want to stick to any

---

[1] http://www.kowari.org/

30

reference ontology for annotation, each peers will have its own set of local ontologies.

<u>User Requirement</u>: *Each user will maintain its own sets of personal ontologies in the local repository.*

Since such a set of personal ontologies will be heterogeneous, there exists a need for a mediator service to provide interoperability support. Such a mediator service should provide correspondences between the ontologies.

<u>System Requirement</u>: A query mediator to provide translations of queries written for one ontological model to another.

<u>Possible Solutions</u>: Manual ontology mediation; Automatic ontology mediation.

Manual mediation needs an ontology engineer for the purpose of finding correspondences between the ontologies. This is because the ontology design and maintenance require an in depth knowledge of ontology languages and formalization aspects. Since the users are not expected to have such knowledge, manual mediation does not propose a solution for the PicSter.

A number of automatic ontology alignment algorithms have been proposed as is discussed in chapter 2. Such algorithms can be implemented as an ontology alignment service which could then be used by the query mediator to perform the query translations as mentioned above. Among the various existing tool, PicSter will be using the Ontology Alignment API – OLA.

<u>System Requirement</u>: An ontology alignment service.

3.2 System Architecture

Having discussed the user and the system requirements, this section describes a high level design architecture which addressed these requirements. The implementation specific details are described in next chapter. The modular architecture of PicSter can be seen in Figure 3.2. At a high level, there are three core components:

1. Client Interface: This module deals with the image annotation and search functionalities of the prototype. It has following sub-components:
   - Ontology and Metadata Management: PicSter uses background knowledge from ontologies which provide the vocabulary to annotate images. It provides mechanism for metadata generation, storage and retrieval functionalities.
   - Image/Media Management: This component provided functionalities that assist in the annotation process of images.

- User Involvement and Interaction: The user sets up all the initial arrangements.

2. Communication Infrastructure: This module deals with the P2P communication aspects of PicSter. In particular it provides the following functionalities:
    - P2P Communication: This component provides mechanism to communicate with other peers that are running PicSter. It deals with basic P2P functionalities like: peer discovery, message transmission and reception.
    - User Involvement and Interaction: The user sets up all the initial arrangements required to publish its presence to other peers. Additionally, the user may also manually select the peers to which she wants to communicate.

3. Mediator Service: This module provides alignments between personal ontologies and is used in rewriting the queries and the results before sending them to the remote peers.
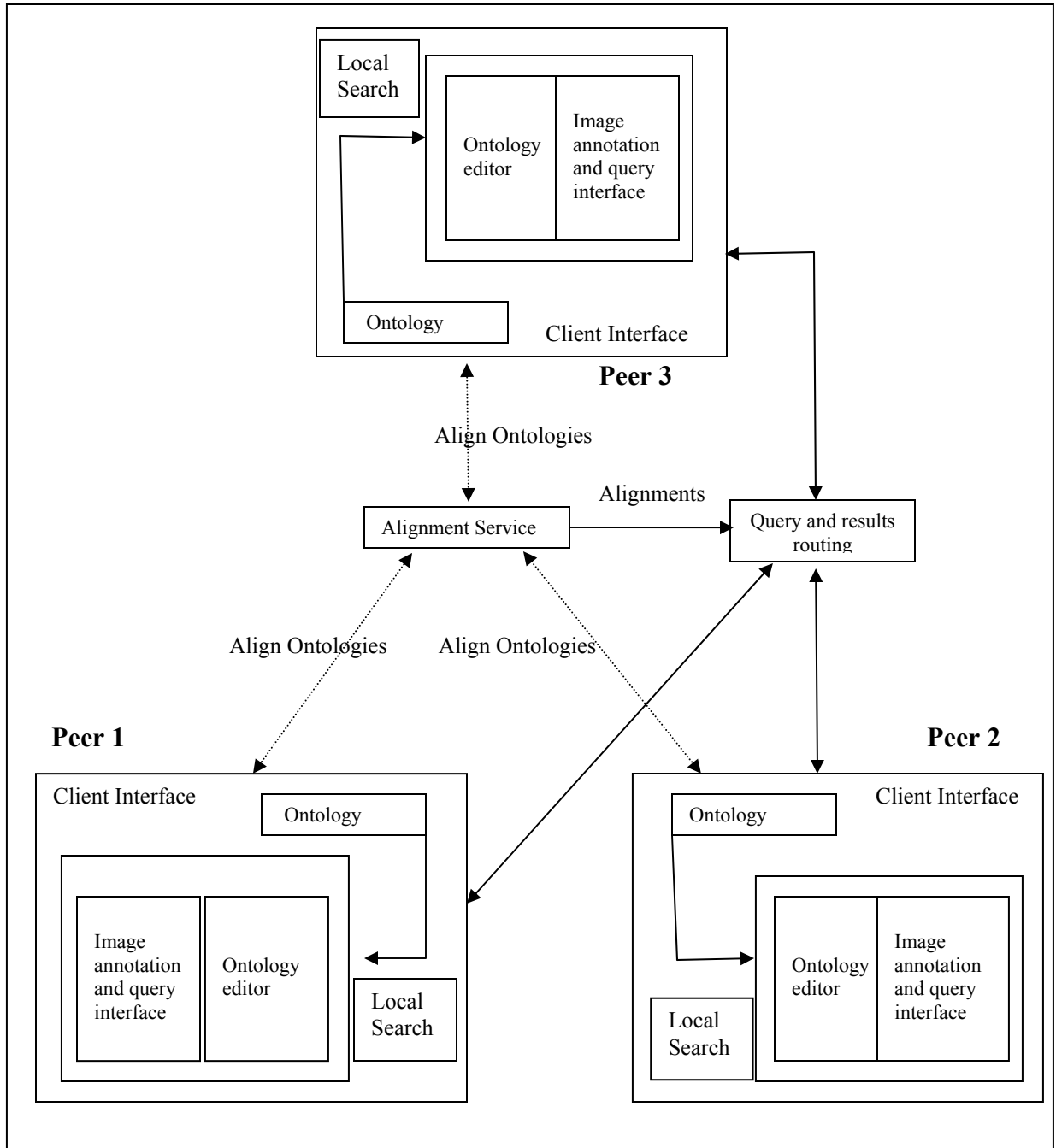
Figure 3.2: System Architecture

3.2.1 Client Interface

PicSter client interface is based on PhotoStuff [hWGS[+]05] tool and is extended to address the personalization issue required for photo annotation. The overall approach of the client interface module is shown in Figure 3.3. It consists of three

sub-components, namely (ontology-based) image annotation, ontology edition and metadata management. All of these components can be used in an interactive manner during the annotation process.
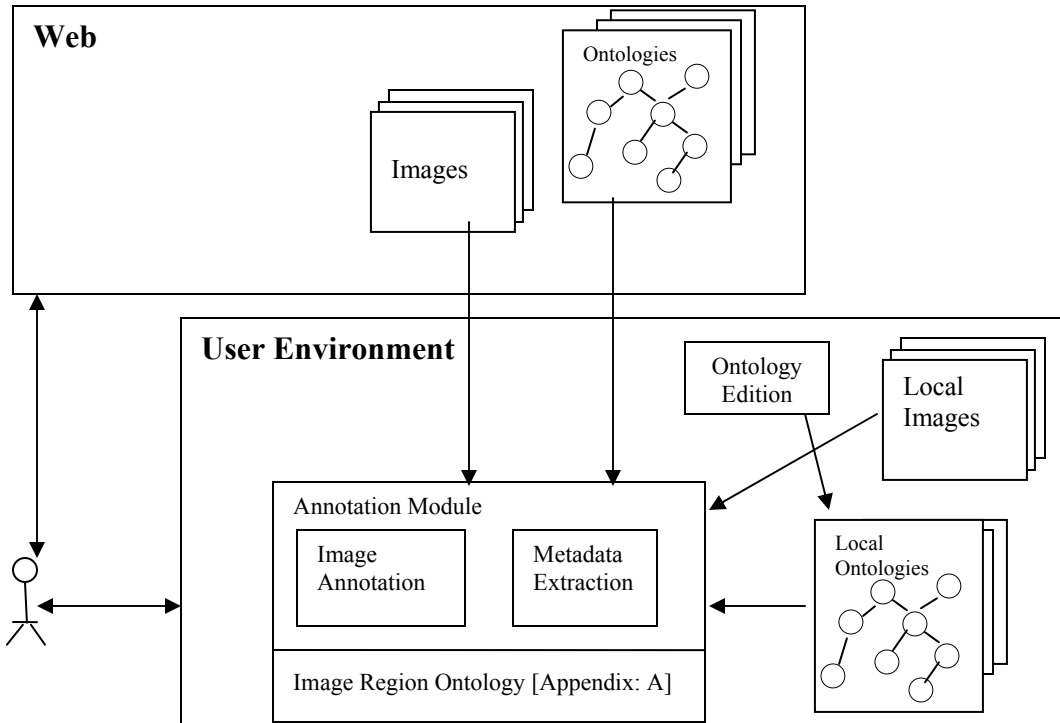
Figure 3.3 Image Annotation Approach

Image Annotation: PicSter uses an image ontology [Appendix: A] specified in OWL which defines a set of concepts for images, videos, regions, depictions etc. This ontology provides the expressiveness required to assert the contents of images (in forms of regions) as well as the whole image.

Users can annotate regions of images with respect to concepts in any ontology written in OWL. The client extracts the embedded metadata within the images and serialise it in RDF syntax. The resulting annotations are stored on the local disk and can be exported to other peers that are looking for these annotations. Annotation search and export process is discussed later in this chapter.

Ontology Management: PicSter allows users to load multiple ontologies at once, which enables the users to mark-up the images with concepts distributed across any of the loaded ontologies. These ontologies may exist either locally or on the World Wide Web.

PicSter provides some basic ontology edition functionalities which are useful for quick personalisation of the photo annotation process. In particular it allows users to create their own hierarchy of classes and properties by:

- Creating new classes: The new class can be specified as a subclass of any class among the loaded ontologies and can associate labels and comments with it.
- Creating new properties: PicSter allows users to create new properties for any of the existing or newly created ontology classes. Users can specify the range and domain of the property and hence can create both OWL Data-type and OWL Object properties. The property can also be specified as a sub-property of any property among the loaded ontologies.

This basic ontology editing functionality ensures that the users of the system do not have to stick to a base ontology for annotating their pictures. It also removes the need for the users to be dependent on external complex ontology editors like Protégé for creating a simple class hierarchy required for photo annotations.

Metadata management: PicSter extracts the existing metadata embedded in image files and serialise this information in RDF/XML. This permits the embedded metadata to be directly incorporated into the tool without any user involvement. This is done by mapping the embedded metadata to the Dublin Core[2] and EXIF[3] schemas. Implementation specific details for the metadata management are specified in the next chapter.

3.2.2 Communication Infrastructure

This component performs the tasks which are related to P2P communication. The design principles for this component are the ones which are typical for P2P systems, namely:

- A peer announces its presence to other peers when joining the network
- Peer advertisements are sent and received in an asynchronous manner.
- All peers maintain a local cache of known peers, so that they can send queries to known peer even if they have not received an advertisement for that peer.
- Certain peers can act as super peers within a group

In addition to the above mentioned functions, PicSter infrastructure relies on a specific protocol for message exchange between the peers. This message exchange protocol is explained in the next chapter.

---

[2] Dublin Core Metadata Initiative: http://dublincore.org/
[3] EXIF- Exchangeable Image File Format: http://www.exif.org/

This component also provides a GUI for involving users in this peer communication process. A user can typically select the peers with which it is interested in and can also specify the type of search she is interested in.

3.2.3 Mediator Service

The mediator service is the core component of the architecture which addresses the issue of heterogeneity. In a peer-to-peer PicSter scenario each peer maintains a different data model based on it personal ontologies. For peers to communicate among themselves, messages/queries transferred among them should be stated in the respective peers' metadata model and vocabulary. This requires that the queries must be translated into a query that is understandable by the respective peer. The mediator service component of the PicSter architecture provides this translation.

Figure 3.4 illustrates how PicSter deals with the interoperability problem. In order to achieve interoperability between peers, data needs to be exchanged. This data needs to be interpreted by the receiver in the way it was intended by the sender. Having ontologies is not enough to achieve full interoperability between applications, because of the differences in the ontologies used by various peers. The mediator provides the reconciliation of differences between ontologies.
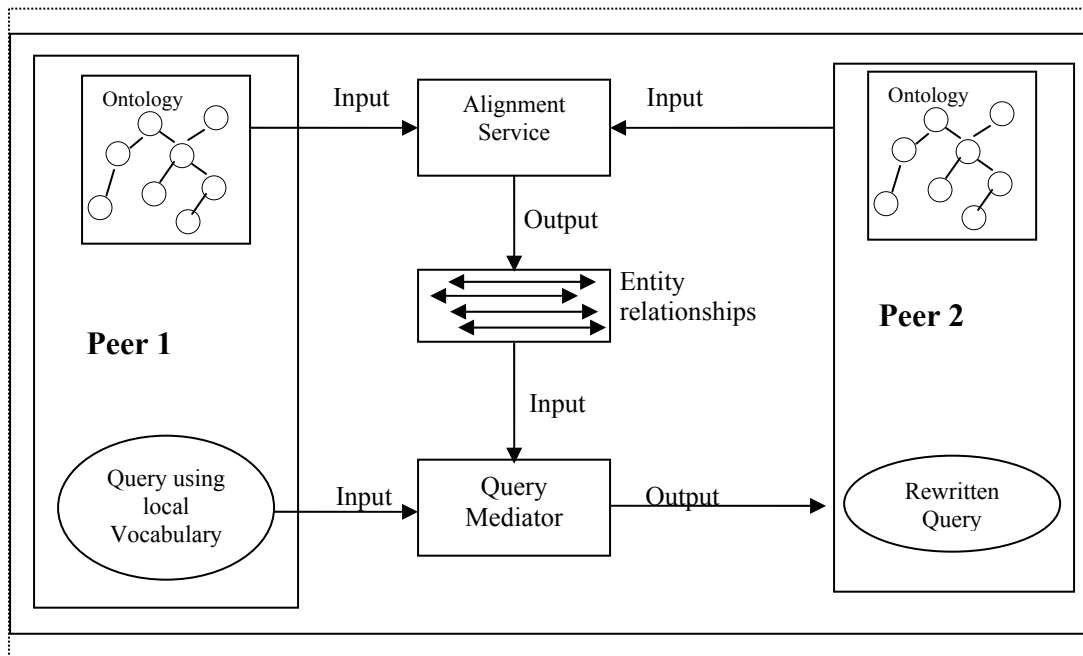
Figure 3.4: PicSter Query Mediator

The mediator consists of two parts:

- *An alignment service*: The alignment service takes two ontologies as input and generates alignments between them as the output. The current version

36

of PicSter has not implemented such a service but uses the pre-computed alignments given to system. That is, the alignments are generated out of the system either by using the ontology Alignment API [Euz04] or by the users themselves as reference alignments. A further step in this direction is to implement a service which computes the alignments on the fly as a user performs a query operation.

- *A Query Mediator*: The main task of the mediator within the PicSter application is to translate the queries. For this purpose, it takes an ontology alignment and a query as the inputs and generates the rewritten query as the output. Current version of the mediator performs translations on the basis of equivalent correspondences given by the input alignment. The URI references within a query are replaced by their corresponding equivalent URI as mentioned in the input alignment. Further step in this direction would be to allow translations based on more enhanced correspondences (for instance, subset). The mediator is implemented for queries written in the SPARQL [PS06] query language. The details of query constructs allowed in PicSter can be seen in section 4.3. Implementation specific details of the Query Mediator are mentioned in the next chapter.

3.3 Conclusion

This chapter explains the design and architecture of the PicSter prototype. A requirement analysis is done by analyzing the problems and their possible solutions. High level system architecture is initially described followed by the description of each module; namely; the client interface, the peer-to-peer architecture and the mediator service. Next chapter explains the implemented technologies in this thesis. The implementation detail of each module is described. After finishing the implementation part, system will be evaluated using the criteria which are shown in evaluation chapter.

Chapter 4

Implemented Technology

This chapter discusses the implementation strategy used to develop the prototype application. Three main implementation specific technologies have been used by PicSter which are explained in each section. The first section describes the Jena API that is used by the client interface component of the prototype. The second section describes the JXTA Framework that is used for P2P communication and the protocol used by PicSter. The third section describes the SPARQL query language and its use in PicSter.

4.1. Client Interface Implementation and used Technology

4.1.1 Introduction

PicSter client interface is a platform independent prototype and it uses the Jena Ontology API[1] for internal representation of its ontology model. Jena is a toolkit for developing applications within the semantic web. It essentially provides an application programming interface (API) for manipulating RDF models. It is written in Java programming language and implements the interpretation of the RDF specifications. Jena was developed to satisfy two goals:

- to provide an API that was easier for the programmer to use than alternative implementations
- to be conformant to the RDF specifications

A brief description of the API is mentioned here to provide a basis which is used by Jena. The section also illustrated how Jena is used in PicSter.

---

[1] http://jena.sourceforge.net/

4.1.2 Jena Ontology API Architecture and basics

The structure of Jena implementation is shown in figure 4.1. The implementation has been designed to permit the easy integration of alternative processing modules such as parsers, serializers, stores and query processors.

The API itself consists of a collection of Java interfaces representing resources, properties, literals, containers, statements and models. A common set of classes implement these interfaces, though these may be sub-classed or replaced to optimize particular implementations. The model class is a generic implementation of an RDF graph. A standard interface connects model to classes which implement storage and basic querying of RDF statements. A standard interface also enables integration of specialized query processors.
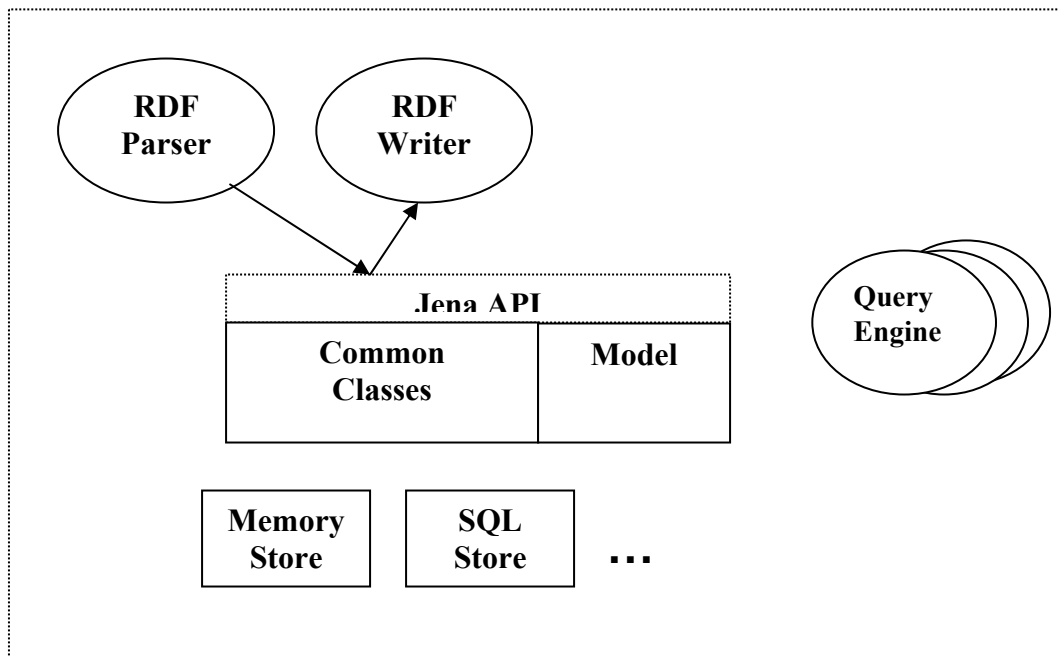
Figure 4.1: Structure of Jena API Implementation

4.1.3 Ontology Languages and the Jena API

The Jena Ontology API is language-neutral: the Java class names do not mention the underlying language, e.g: `OntClass` and `ObjectProperty`). To represent the differences between the various representations, each of the ontology languages has a profile, which lists the permitted constructs and the URI's of the classes and properties. Thus in the DAML profile, the URI for object property is `daml:ObjectProperty`, in the OWL profile is it `owl:ObjectProperty` and in the RDFS profile it is null since RDFS does not define object properties.

The profile is bound to an ontology model, which is an extended version of Jena's Model class. The general Model allows access to the statements in a collection of RDF data. `OntModel` extends this by adding support for the kinds of objects expected to be in an ontology: classes (in a class hierarchy), properties (in a property hierarchy) and individuals. The properties defined in the ontology language map to accessor methods. For example, an `OntClass` has a method to list its super-classes, which corresponds to the values of the subClassOf property. This point is worth emphasizing: no information is stored in the `OntClass` object itself. When the `OntClass listSuperClasses()` method is called, the information is retrieved from the underlying RDF statements. Similarly adding a subclass to an `OntClass` asserts an additional RDF statement into the model.

The statements that the ontology Java objects see depend on both the asserted statements in the underlying RDF graph, and the statements that can be inferred by the reasoner being used (if any). Figure 4.2 shows how the statements are seen by the `OntModel`.
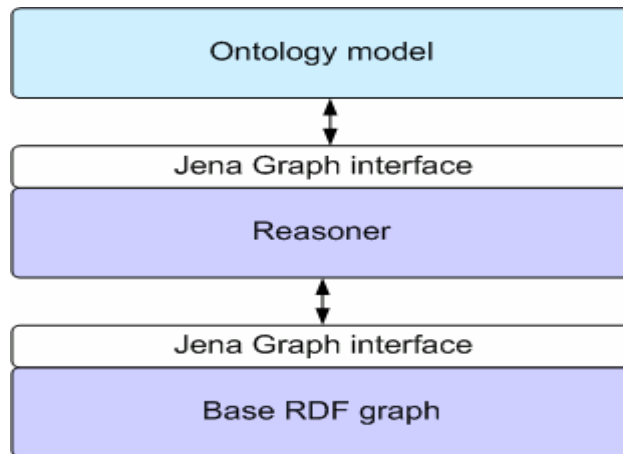
Figure 4.2: Statements in Jena Ontology Model [Jena]

The asserted statements are held in the base graph. This presents a simple internal interface, Graph. The reasoner, or inference engine, can use the contents of the base graph and the semantic rules of the language, to show a more complete set of statements - i.e. including those that are *entailed* by the base assertions. This is also presented via the Graph interface, so the model works only with that interface. This allows us to build models with no reasoner, or with one of a variety of different reasoners, without changing the ontology model. It also means that the base graph can be an in-memory store, a database-backed persistent store, or some other storage structure altogether again without affecting the ontology model.

4.1.4 PicSter Markup Model and Jena

PicSter uses Jena to manipulate ontology based image annotations and the associated instances in RDF syntax. It loads the ontologies and allows the users to

annotate the loaded images with respect to the concepts in any of the loaded ontologies. PicSter stores these annotations and instance within the memory as a Jena model while the prototype is running which then writes them to the local repository when it quits. Any changes to the loaded ontologies are also stored in the repository. Internally, PicSter maintains a Java model for images and forms. These models are used for interacting with user and assisting her in the process of annotation and edition.

Figure 4.3 shows the overall architecture as implemented by the client interface of the PicSter prototype.
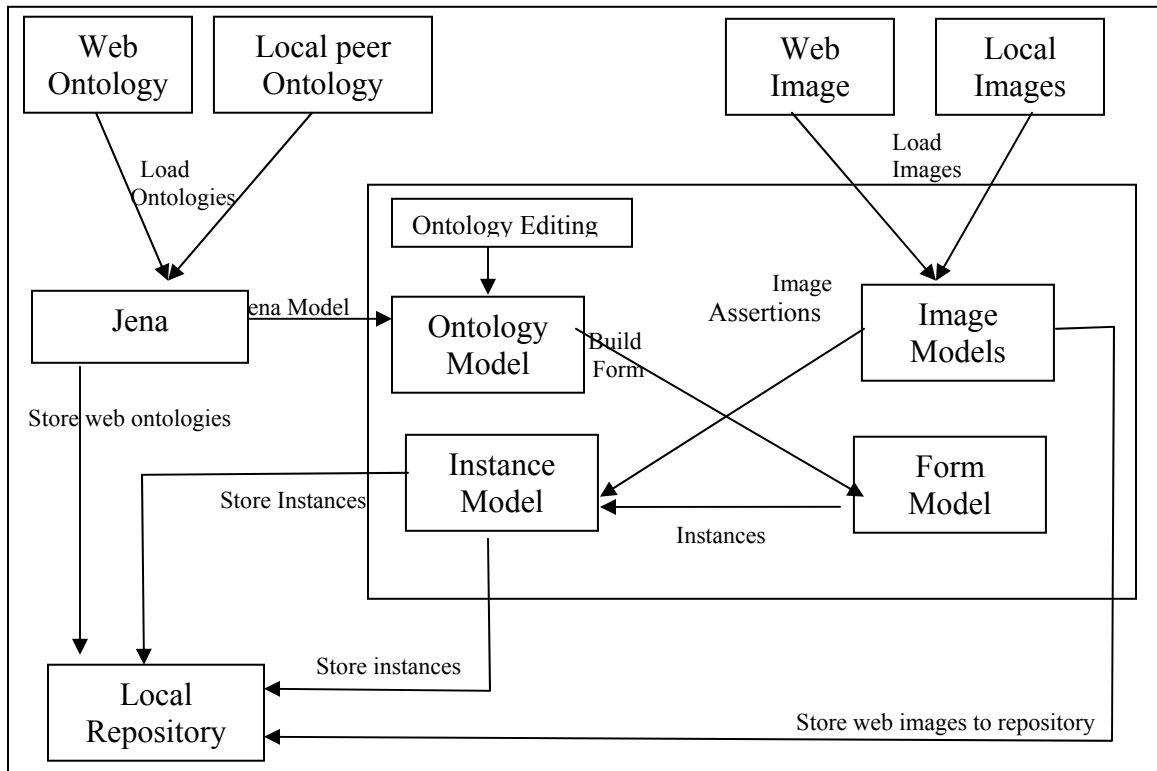
Figure 4.3: Implementation Architecture of PicSter Client Interface

4.2 PicSter Peer to Peer Communication using JXTA

This section describes the technology used by PicSter for implementing the peer-to-peer communication component of the system. PicSter uses JXTA technology for implementing the P2P component. After an overview of JXTA, a protocol used by the PicSter is discussed later.

4.2.1 JXTA Introduction

JXTA (Juxtapose)[2] is a project launched by the Sun Corporation to create an open network programming platform. Following are some of the salient features of the JXTA:

- JXTA is just a protocol; therefore the implementation is independent.
- Exchange language of JXTA is XML.
- Monitoring of the status of peers or network possible.
- It allows self-organization of several peers into peergroups.
- It provides mechanism for advertisement and discovery of services.
- JXTA is not based on particular network protocol (e.g. TCP/IP or HTTP), which makes it easier to deal with firewalls.
- It does not require a particular programming language, which allows independence of the hardware environment.
- Security aspects can be implemented into the protocols, on the other hand also meaning no specific security concept is enforced.
- Its current implementation is in Java, hence can be used on many operating systems.

A brief overview of the JXTA terms has been mentioned here which is followed by the PicSter specific protocol description.

Figure 4.4 shows the layered structure on which the JXTA protocols are based. Protocols used by the PicSter are mentioned briefly.
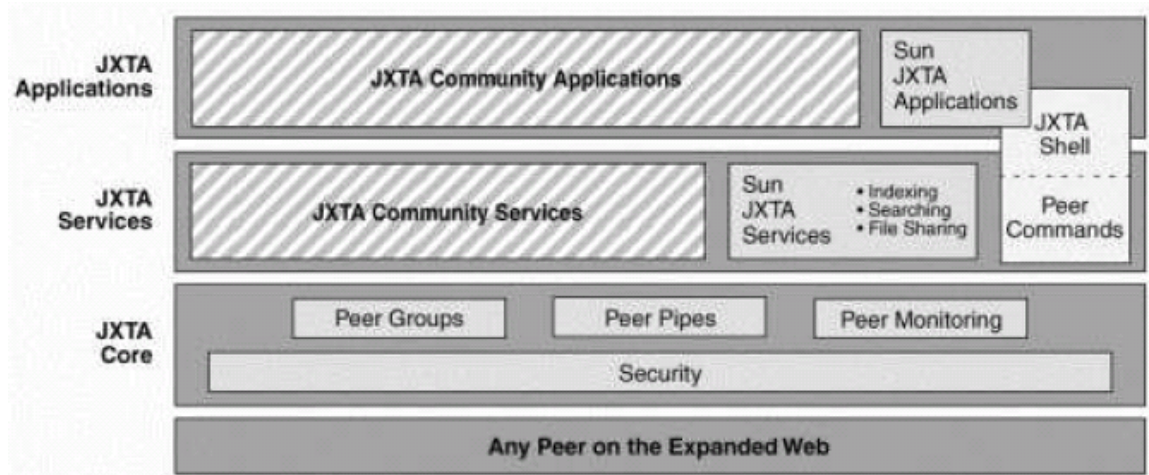

Figure 4.4: Layers in JXTA [JXTA]

---

[2] www.jxta.org

42

Platform Layer

The platform layer provides the basic functionality for the peer to peer system. This so called JXTA core is necessary for any use. It normally comprises peer establishment, communication management such as routing and other low-level plumbing. Some definitions from this layer are:

Peer: Peers are the basic entity of the network. They act independently and provide and consume from the peer-to-peer network.
Peergroup: Peers can gather creating so called peergroups. These peergroups normally share a common interest and use own protocols on top of the JXTA protocols.
Pipe: Pipes are communication channels on top of the basic network. Peers advertise their possibility to connect to other peers. As soon as the pipe is established two peers can easily communicate with one another while the lower network is fully transparent. Pipes are basically unidirectional.
Advertisement: Advertisements are sent by peers connecting to the network to publish the services they provide. These can be basic services like the pipes or high-level services understood only by a specific peergroup.
Message: Messages are constructed in XML and are interchanged between peers. They are transferred using the described pipes.

Another part of the JXTA core is the use of identifiers. For each peer, peergroup, or object an ID is created (UUID). With this ID every location in the peer-to-peer network can be uniquely identified and reached.

Service-layer

On top of the basic layer the so called service layer is located. A list of common protocols from the service-layer is provided at this point. Even though they are on this level, many of them are important to have a smooth running peer-to-peer network.

Rendezvous Protocol: This protocol allows peers to join the network or a peergroup. It is used to check the local environment and finally log on to the network. This protocol can be processed in several ways one being recognized by special rendezvous peers, that take over all shake-hands tasks. Another possibility is to use cascading starting from one direct neighbor and passing the information on through the network.

Peer Discovery Protocol: After being accepted to the network the peer advertises the resources it has as well as discovering the resources from other peers. This is the idea of this protocol.

Endpoint Routing Protocol: If a peer wants to know which way to send its message to another specific peer, it can contact routers which will then provide a list of peers

forming the path the message has to follow. Any peer can become a routing peer with its own desire.

There are other protocols defined in this layer such as Pipe Binding Protocol, Peer Information Protocol, etc. A detailed description is left and can be found in literature [JXTA].

Application-layer

Finally on top of the service layer is the application layer. Examples can be messaging services, email applications, or other high-level programs. The applications can be plugged into the JXTA framework. It is this layer where the P2P component of PicSter resides.

4.2.2 JXTA in PicSter

PicSter uses the default JXTA shell for peer configuration and an application level message exchange protocol developed during this project. Figure 4.5 shows the startup PicSter peer configuration window. Peers can set a secure username and a password for configuring their peers. They can also specify if a peer will act as a rendezvous peer.

The PicSter specific JXTA pipes and messaging protocol is described below.

Pipes

PicSter uses JXTA 'propagation' pipes to broadcast queries to a peergroup. A message sent through a propagation output pipe will be received by all peers in the group that have created a matching propagation input pipe.
PicSter creates its propagation input and output pipes using advertisements containing known pipe IDs. By using these hard-coded IDs, PicSter can immediately communicate with other PicSter peers in the peergroup, without having to use discovery to find their pipe advertisements. An application-specific string is included as part of the ID, allowing different PicSter instances to hold separate conversations within the same peer group

JXTA's propagation pipes are not guaranteed to be reliable. Messages may arrive out of order. Some may be dropped, and some may arrive in duplicate. In addition, there is an upper limit to the message size. PicSter contains mechanisms to work around these obstacles.
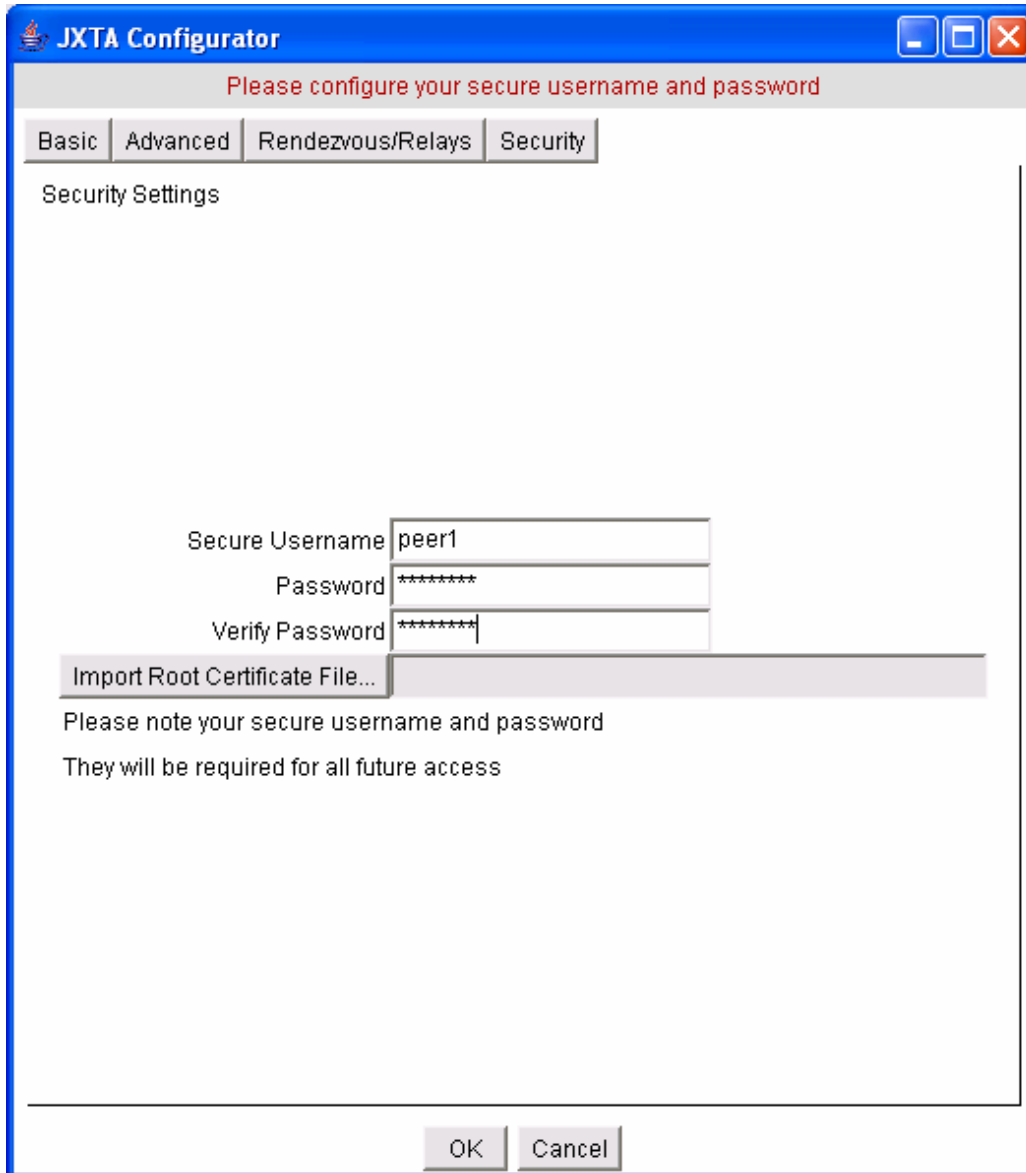
Figure 4.5: PicSter Configuration Window

Messages

PicSter uses a messaging protocol to exchange queries and their answers among the peers. Although PicSter uses JXTA 'propagation pipes' for message communication, it is the message fields which enable a peer to identify if the message was addressed to it.

PicSter uses the messages described below for peer to peer communications. Each message contains several fields, or 'elements'. The first element contains the message type, so the receiving peer will know which other elements to expect. All the elements are stored as strings.

Specifications of messages

1. **Query Message:** This message is used to send the query message to other peers.

Message Element	Contents
MessageType	"QUERY"
PicSterSenderName	Name of the sending peer
SenderID	ID of sending peer
PicSterVersion	PicSter version
Dest	Destination peer
Query	Query to be sent

Table 4.1: PicSter JXTA Query Message

2. **Annotation Message:** This message is used to send the image annotations to the peer whose query has been processed.

Message Element	Contents
MessageType	"ANNOTATION"
PicSterSenderName	Name of the sending peer
SenderID	ID of sending peer
PicSterVersion	PicSter version
Dest	Destination peer
FileName	Name of the annotation file
Annotation	<annotation>*RDF annotations*</annotation>

Table 4.2: PicSter JXTA Annotation Message

3. **Instance Message:** This message is used to send all the instances returned by a query to the peer whose query has been processed.

Message Element	Contents
MessageType	"INSTANCES"
PicSterSenderName	Name of the sending peer
SenderID	ID of sending peer
PicSterVersion	PicSter version
Dest	Destination peer
FileName	Name of the instance file
Instance	<instances><instance> *RDF instance* <instance> <instance> *RDF instance*</instance>…..</instances>

Table 4.1: PicSter JXTA Instance Message

4. **File Message:** In addition these messages which return the annotations and the associated instances of a particular query results, the PicSter JXTA protocol also

46

defines a mechanism to transfer images as well across the peer-to-peer network. This message has not been implemented robustly, as it causes significant delay in transferring images when images to be returned are big in size.

4.3 PicSter Query Mechanism

PicSter uses SPARQL query language for sending queries to other peers or for local search. A generic SPARQL query processor API was developed during the thesis work. PicSter provides an ARQ[3] implementation for this API. ARQ is a query engine for Jena that supports the SPARQL RDF query language.

Searching

PicSter permits three types of functionalities for querying the loaded data model:

1. Keyword search
2. SPARQL text search
3. Ontology based search

Currently, only first two functionalities have been implemented. Ontology based search queries will be a subset of the set of queries allowed by the SPARQL text search functionality.

4.3.1 Keyword search

This functionality allows the user to search the markup which has been created by him or others (querying remotely) to find the instances and images that match the search criteria. The entered keyword is matched against the labels and id's of all instances and images loaded in the local or selected peers' PicSter model. Any instances that match the entered keyword will be returned in the set of search results. If the query is done on a remote peer, the returned instances and the image annotations are stored locally in the `PICSTER_HOME/.jxta/instances` and `PICSTER_HOME/.jxta/annotations` directories respectively.

The keyword search is internally translated into the following SPARQL query:

```
SELECT ?uri
WHERE {
 ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type .
 OPTIONAL {
   ?uri <http://www.w3.org/2000/01/rdf-schema#label> ?label .
  }
 FILTER(regex(str(?uri), "keyword", "i") || regex(?label, "keyword",
"i") ) .
FILTER (!isBlank(?uri)) .
}
```

---

[3] ARQ – A SPARQL Processor for Jena: http://jena.sourceforge.net/ARQ/

No FROM clause is used by the SPARQL query because, the default model to be searched has already been loaded by the PicSter (local or remote).

4.3.2 SPARQL text search

PicSter allows an advanced user to enter any valid SPARQL query with the following restrictions:

1. Only SELECT queries are accepted by the system.
2. All queries must have a variable `?uri` in the SELECT clause. Although other variables may be used to restrict the number of results.
3. No FROM clause is permitted in entered query because the system uses the default model loaded by PicSter.

Example queries:

1.
```
SELECT ?uri
WHERE
{
 ?uri http://www.semantech.org/ontologies/foaf.owl/name "Paul".
 ?uri http://www.semantech.org/ontologies/foaf.owl/gender "male".
 ?uri http://www.semantech.org/ontologies/foaf.owl/worksAt "INRIA".
}
```

This query returns the set of instances that are male and work for INRIA and are Paul.

2.
```
PREFIX j.1: < http://exmo.infrialpes.fr/people/euzenat#>
SELECT ?uri ?symbol ?number
WHERE
{
 ?uri j.1:postcode "38300".
 ?uri j.1:has-pretty-name "Meylan".
}
```

This query shows that namespace prefixes are permitted. It returns instances that have their postcode property value as "38300" and has-pretty-name property value as Meylan.

3.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?uri ?depiction
WHERE {
    ?person foaf:name ?uri .
    OPTIONAL {
      ?person foaf:depiction ?depiction .
    } .
  }
```

This query returns a set of uris of people who may or may not have a picture associated with them. Please note that although the query returns the values of `?depiction` variable also in the result set, only the `?uri` variable is used by the PicSter to show the results.

4.3.3 Ontology Based Search

This functionality has yet not been implemented. The idea is to allow the user to do a form based search, where a user can search for a class name in the ontologies loaded by the PicSter. An instance search form is generated on the basis of the selected class, where the user can enter values in the property fields. Each such query will be generated dynamically depending on the values entered in the property fields. However, internally these queries will be a subset of the queries already accepted by the PicSter; hence the only missing functionality is the user interface.

Each such query will be internally formulated as the following SPARQL query:

```
SELECT ?uri
WHERE
{
 ?uri property1Uri "entry1".
?uri property2Uri "entry2".
?uri property3Uri "entry3".
--
--
--
?uri propertyNUri "entryN".
}
```

4.4 Query Mediator

As described in Chapter 3, PicSter Mediator Service acts as a query translator. The query mediator is based on the top of the Alignment API [Euz04]. The mediator takes an alignment as an input that has been specified by the API. The alignment provides the set of correspondence between entities of the two ontologies. The API implements a number of alignment algorithms to provide alignments. A new alignment algorithm can also be easily added and implemented within the API. Some of the available alignment algorithms are:

NameEqAlignment: It compares the equality of class and property names and aligns the objects with the same name.
EditDistNameAlignment: It uses an editing distance between entity names. It builds a distance matrix and chooses the alignment from the distance.
SubsDistNameAlignment: It computes a substring distance on the entity name.
StrucSubsDistNameAlignment: It computes a substring distance on the entity names and uses and aggregates this distance with the symmetric difference of properties in classes.

The `QueryMediator` is written for queries written in the SPARQL query language. The implemented mediator relies on an embedded generic SPARQL query processor interface that was developed during the thesis period. This makes the `QueryMediator` independent of the query processor implementation and hence can be used in applications other than PicSter that need to perform alignment based query translations. The `QueryMediator` class diagram is mentioned below in figure 4.6.
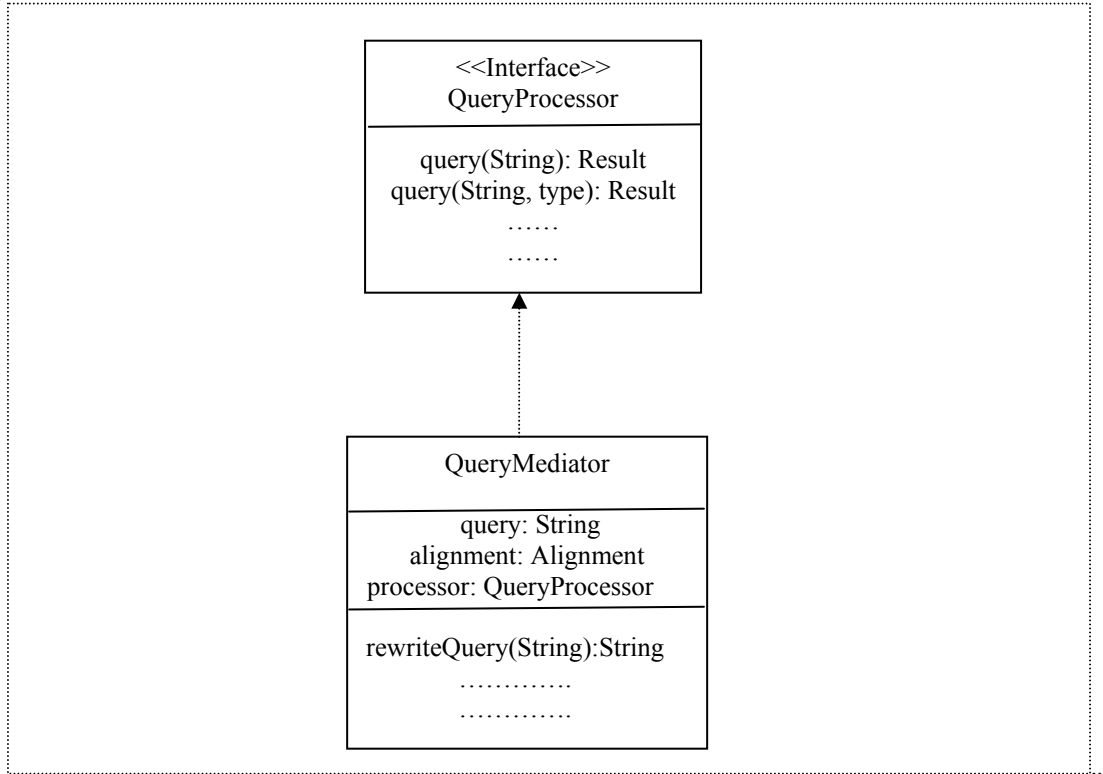
Figure 4.6: Query Mediator Class Diagram

PicSter provides the ARQ [Arq] based implementation for the `QueryProcessor`. It essentially provides a wrapper to the ARQ SPARQL processor for Jena API. This implementation is passed as a parameter to the mediator, as shown in the code.

The input query is rewritten on the basis of the alignment given. Currently, PicSter uses only the equivalent entity correspondence to perform the translations. It replaces the entity URI by the corresponding equivalent URI from the metadata model of the queried peer. It works only for the alignments with '=' and does not care for other relationships such as subclass etc.; a functionality which could be added as a future work.

An example of the input alignment correspondences is shown in figure 4.6. The figure shows entity correspondences with confidence measure between two ontologies. The XML serialization of the same as produced by the alignment API can be seen in Appendix B.

50

An example SPARQL query and its rewritten query based on the alignment cited in figure 4.7 is shown below:
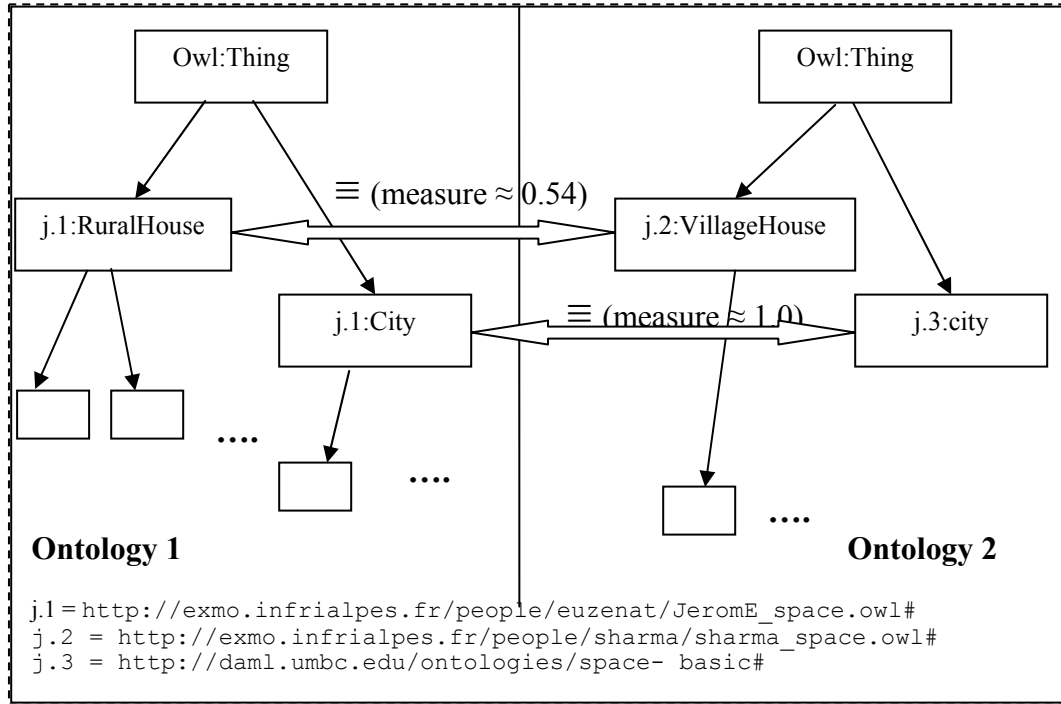
Figure 4.7: Example Alignments

Input query: The query returns the URI of all the resources from the ontology model (of the originating peer) which have "`xyz`" as the value for the property `j.1:RuralHouse` and "Grenoble" as the value for the property `j.1:City`.

```
PREFIX j.1: < http://exmo.infrialpes.fr/people/euzenat/JeromE_space.owl#>
SELECT ?uri
WHERE
{
 ?uri j.1:RuralHouse "xyz".
 ?uri j.1:City "Grenoble".
}
```

Rewritten query: The query returns the URI of all the resources from the ontology model (of the destination peer) which have "`xyz`" as the value for the property `j.2:VillageHouse` and "Grenoble" as the value for the property `j.3:City`.

```
SELECT ?uri
WHERE
{
?uri http://exmo.infrialpes.fr/people/sharma/sharma_space.owl#VillageHouse
"xyz"
?uri http://daml.umbc.edu/ontologies/space- basic#city "Grenoble".
}
```

4.5 PicSter by Example

The section shows the process of annotation, ontology editing and search process through an example.

4.5.1 Peer Configuration

When a PicSter peer is started for the first time, it launches a JXTA Configuration window which allows the user to configure and bootstrap the peer. This will enable the system to know the network on which it will run. In addition to the mandatory peer name and password fields, a user can specify which port the peer listens on, where any rendezvous peers are and whether this peer will be operating as a rendezvous. The entered information is then stored in a local directory.

For every successive runs, PicSter just asks the username and password information and other peer information is loaded from the local directory. Two different instances of PicSter on one machine can only be launched from different directories.

4.5.2 Annotation Process

This section describes the process of photo annotation using PicSter.

4.5.2.1 Loading Images

A user can load both local and remote images. If the URL of the image is known, it can be entered into the address bar in the main window which will then be loaded by PicSter and displayed in the interface. Likewise, multiple images can be loaded at once from the local machine and displayed in the main window. PicSter also allows the drag-drop mechanism for loading the file from a directory into the main window. To load a local file, a user can go to "File->Load Image" and get a URL dialog. Images loaded from the web are stored locally in a cache.

4.5.2.2 Loading Ontologies

The process of loading ontologies is similar to the one of loading images. Ontologies loaded from the web are also stored locally in a cache. Multiple ontologies can be loaded and PicSter displays a tree view of all the ontology classes.

Figure 4.8 shows the PicSter main window snapshot with three ontologies and four images loaded by the system. The figure also shows a selected region in the image which will be asserted during the annotation process.
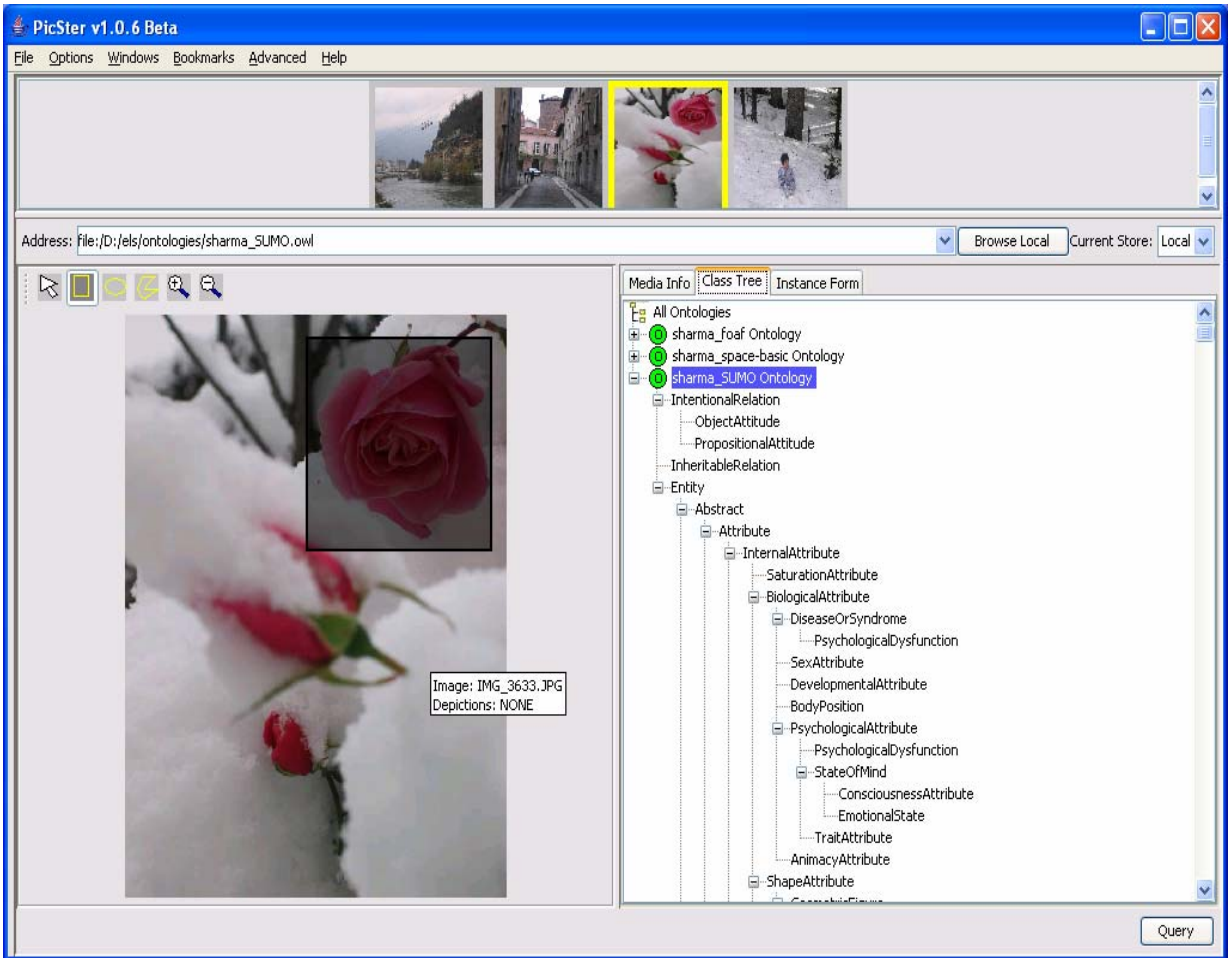
Figure 4.8: PicSter main window

4.5.2.3 Creating Instance

Users can either annotate a whole image or its parts by selecting regions within the image. A user can create a new depiction by dragging a class to one of the selected regions or to the whole image. This will enable her to create a new instance of the dragged class type. When the instance is saved, the depiction assertion is also saved.

Right-clicking on the image/region will bring up a pop-up menu. One of the items in this menu is "Add Depicts" which has a submenu of all the classes currently loaded into the tool. This is a quick way to add annotations to an image/region. If the user right clicks inside a region, the depicts assertion will be made for that region, otherwise the assertion will be made for the entire image itself. This is analogous to dragging a class from the class tree and dropping it somewhere within the media itself. The user can also drag classes from the class tree to media thumbnails in the image list. This will create a depicts assertion for that image using the new instance created by the user.

A particular instance can also be associated with multiple images by 'Select All' command from the image thumbnail right click menu or by using Ctrl-Clicks. In this case the same depicts assertion will be made for all the selected images. However, this process can not be applied for associating the same instance to multiple regions.
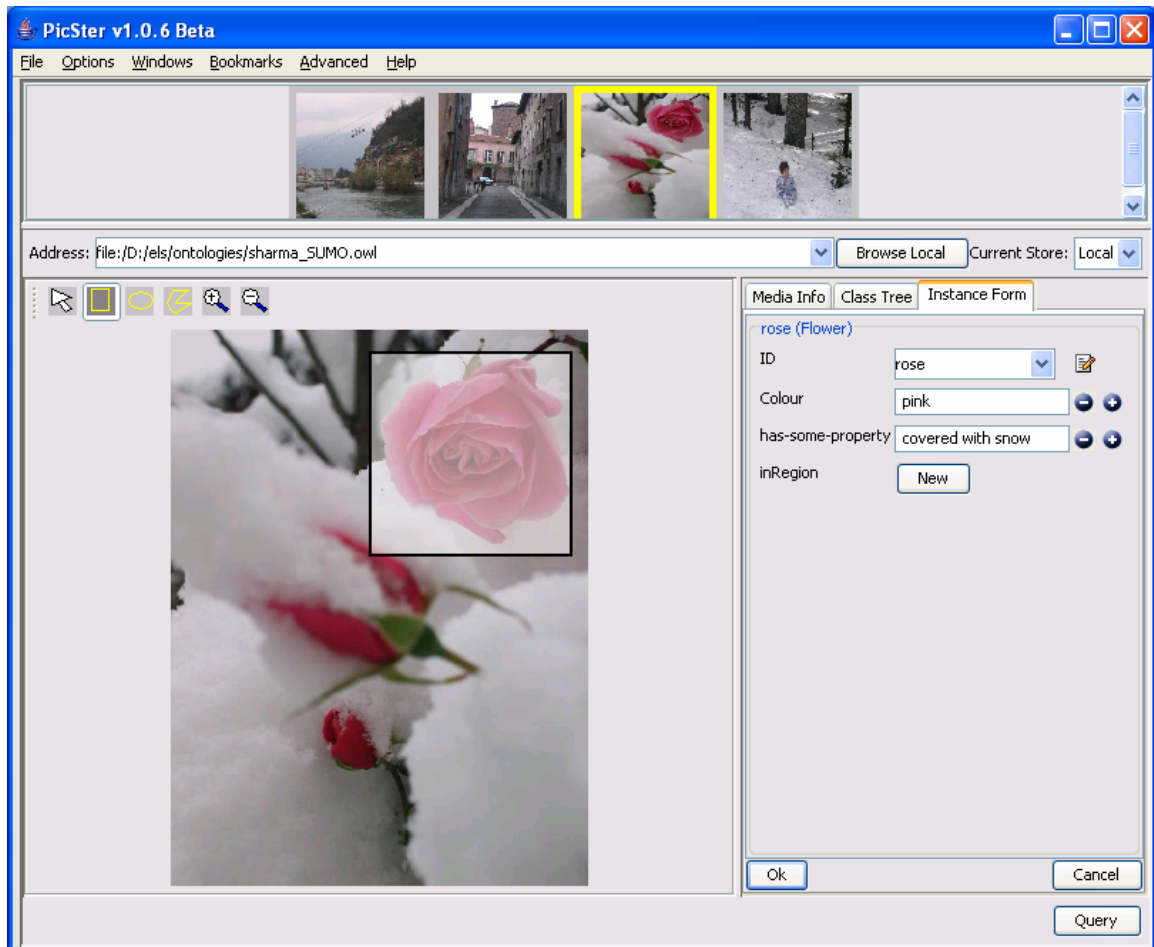
4.5.2.4 Instance Form

The instance form is where a user can enter and edit data for any new or existing instance. The form is based on properties that are in the domain of any of the types of the instance, or are associated with any of the types of the instance via some restriction. Widgets are shown for each property, and a specific editor widget is shown based on the type of the property, whether it is an object or data property.

Figure 4.9 shows the snapshot of the instance form used to create an instance for the Flower class of the loaded ontology. Multiple values for the same property can also be entered using the instance form.

The created instance is stored within memory till the system is running and is written to local repository in the RDF syntax when it quits. The RDF serialization of the created instance is mentioned below:

```
<rdf:RDF
  xmlns:j.0="http://www.mindswap.org/~glapizco/technical.owl#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:j.1="http://exmo.inrialpes.fr/people/sharma/ontologies/p
    icster/space-basic/" >
 <rdf:Description
    rdf:about="http://exmo.infrialpes.fr/people/sharma/elster/inst
    ances/Flower.rdf#rose">
    <j.0:depiction
        rdf:resource="http://exmo.infrialpes.fr/people/sharma/elste
        r/images/IMG_3633.JPG/region2087"/>
  <rdf:type
    rdf:resource="http://exmo.inrialpes.fr/people/sharma/ontologie
    s/picster/space-basic/Flower"/>
  <j.1:Colour>pink</j.1:Colour>
  <j.1:has-some-property>covered with snow</j.1:has-some-property>
  <rdfs:label>rose</rdfs:label>
 </rdf:Description>
</rdf:RDF>
```

54

Figure 4.9: PicSter Instance Form

4.5.3 Ontology Editing
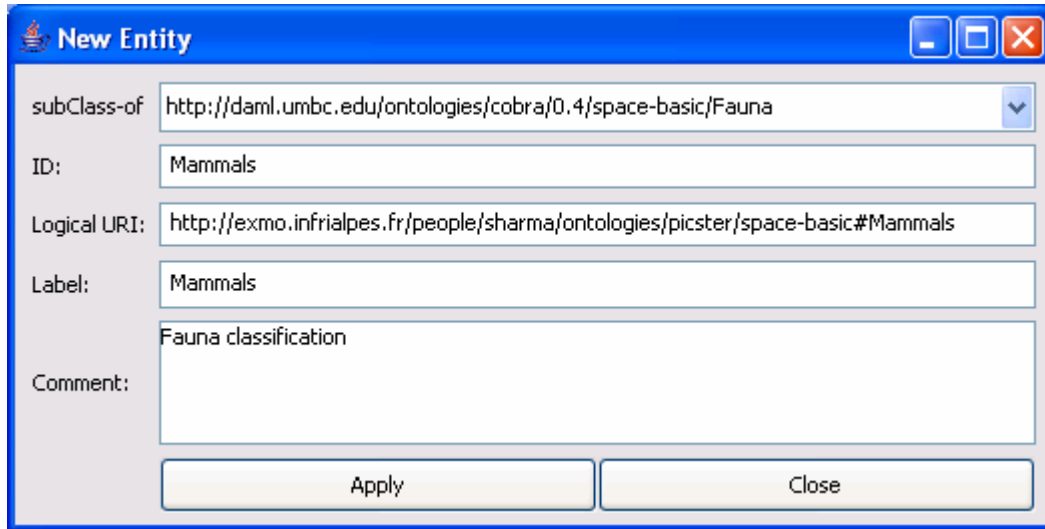
PicSter provides a limited number of ontology editing functionalities. These are basic editing functions required for creating a class hierarchy during the process of annotation. It also provides autonomy to the users by not asking it to use some reference ontology for the

4.5.3.1 Creating subclasses

PicSter allows a user to create a class hierarchy by creating subclasses of the existing classes. Subclass menu can be seen by right clicking on any ontology class. Figure 4.10 shows a snapshot of the PicSter user interface for creating new classes. Among all the fields, ID and the logical URI are mandatory for class creation. Instance form uses the value in label field while interacting with the user.

The snapshot creates a subclass `Mammals` for the class `Fauna`. Any ontology once changed is referred as a personal ontology and is stored in the local repository. Clearly, it inherits all the properties of it superclasses.

Figure 4.10: Form for creating a new subclass

4.5.3.2 Creating Properties

PicSter allows users to create both OWL DataType and OWL ObjectType properties. Subclass menu can be seen by right clicking on any ontology class. Figure 4.10 shows the snapshot of the property creation window. The new property can be assigned as an OWL sub-property of any of the existing ontology properties. As with sub-class creation, the ID and the logical URI fields are mandatory.

The default selected value for the domain is the class for which the property is being created. For `DataType` properties, the pull-down menu for the range field is the set of all XML Schema data types as defined in [XMLS]. For `ObjectType` properties, the user can select the range as any of the existing classes in loaded ontologies.

Figure 4.11 created an OWL `DataType` property with ID as `livesIn` with the newly created class `Mammal` as its domain. Any ontology edition is written in the local repository when the system quits.

Figure 4.11: Form for creating a new ontology property

4.5.4 Searching

PicSter allows a user to search for photo annotations both locally and in a peer to peer network. The PicSter Query Interface is shown in figure 4.12.

Scope: A user can select the scope of the query using the panel options shown in the figure:

- *Local:* Local images URIs are returned when the scope is selected as local.
- *All:* The entered query will be sent to all other peers currently running a PicSter instance and which have been discovered by this peer.
- *Selected:* A user can manually select the peers to which the query has to be sent by selecting peers in the peer-list.

Type: PicSter currently supports two types of queries:

- *Keyword search:* Allows a basic keyword search, which is internally translated to a fixed SPARQL query.
- *SPARQL Text Query:* A user can enter a valid SPARQL SELECT query in the text area. The set of queries excepted by PicSter can be found in section 4.3

Ontology Based Query: The current version of PicSter does not support this type of query. However, the idea is to allow the user to perform a form-based query. An instance form is created based on the ontology class the user is interested in. The user can then enter the property values to restrict the number of results returned by the query.

Results:

- *Local query:* For local queries the result is a set of resource (image or regions) URIs which are shown in the result panel.
- *Remote query:* Remote queries return the set of instances and image annotations in RDF serialized syntax. The returned instances and annotations are stored in a local cache. The results panel shows the list of image or region URIs as a list.

If the images identified by the result URIs are stored locally, then they can be viewed in the PicSter main window by clicking the 'Show selected' or 'Show All' button.
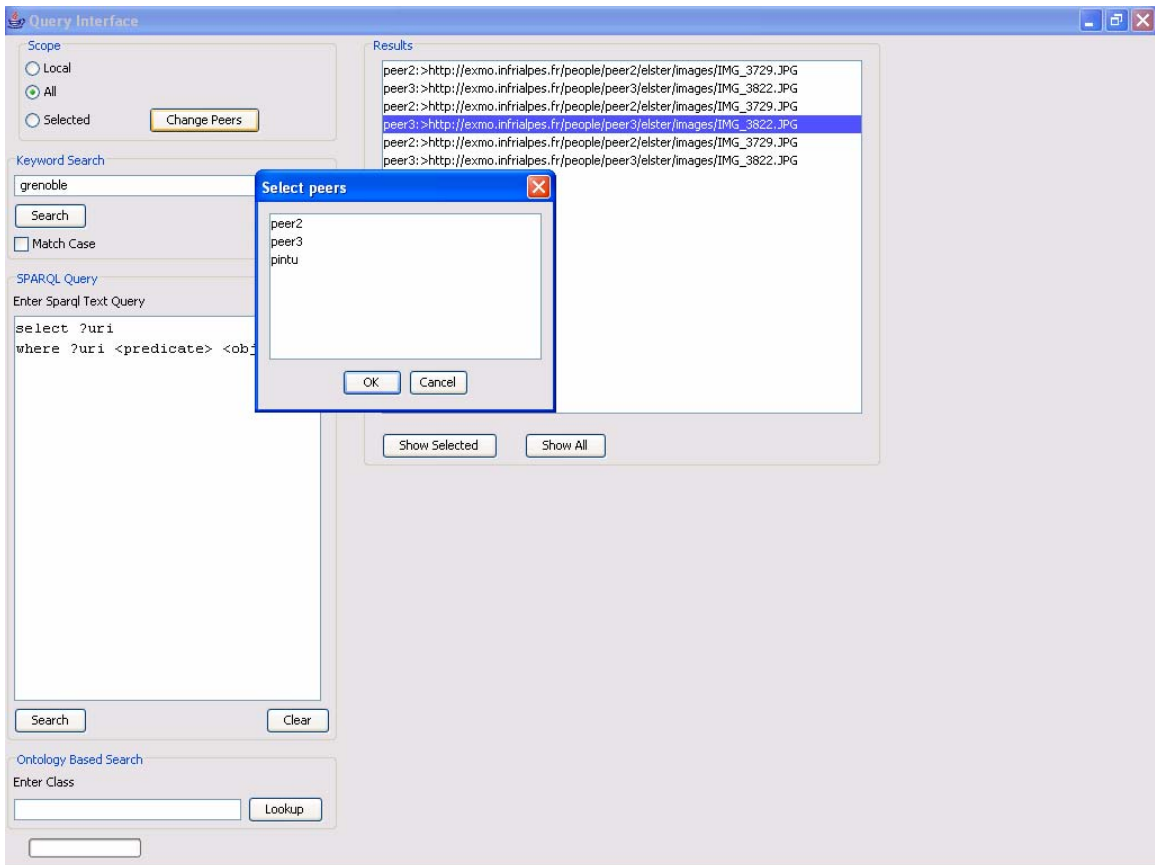
Figure 4.12: PicSter Query Interface

4.6 Conclusion

This chapter provides the implementation details of the PicSter prototype in details. Technologies involved in client interface implementation; P2P infrastructure implementation; and query mediator service are described. The PicSter query mechanism is described. Finally the functionalities provided by PicSter are shown through an example. The quantitative information about the implemented prototype is shown below:

Number of Code lines: 7737
Number of implemented classes: 68
Number of Interfaces: 15
Approximate man days: 100
Ratio (code lines/man days) = 77.37

Chapter 5

Evaluation

In this chapter, the evaluation of the master thesis is presented. PicSter system evaluation was done through an experiment performed during its development phase which is described in first section. The main objectives of such an experiment were the following:

- To get the user feedback during the application development.
- To collect the data required for testing the alignment algorithms implemented by the Ontology Alignment API [Euz04]. Although the thesis did not develop an alignment algorithm and such algorithms are not used within the current version of the application prototype, the data collected through the experiment described below was used to analyse the efficiency of such algorithms. The preliminary results of this analysis are shown in section 5.1.
- The system acts as a test bed for further research activities within the EXMO team at INRIA. The PicSter experiment will be continued on termination of this thesis and the data collected will be used to analyse semantic social networks and alignment compositions.

The chapter ends with a summary of the PicSter user feedback and its comparison with similar existing systems.

5.1 PicSter Experiment and Data Collection

To evaluate the application prototype developed during the thesis work, a data collection and evaluation experiment was done. The members of the EXMO team at INRIA were involved in the experiment. In addition to PicSter evaluation, the experiment was also done in order to collect data for a number of research activities in which the EXMO team is involved in, like semantic social network analysis, alignment composition etc.

Annotating the pictures:

A bunch of 113 photographs were taken in Grenoble and INRIA. Seven participants {AZ, AS, FAK, JJ, JE, JP and SL} from the EXMO team had annotated them through PicSter. They had a subset of 47 randomly selected pictures to annotate. The way to annotate was left free and the participants were free to use existing ontologies, design their own ontologies or extend existing ones.

In addition, the users were also asked the time taken to annotate these pictures and also to name other participants who they feel to be the closest with, ontology-wise. This was done in order to analyse the social networking possibilities.

Data Preparation:

All the ontologies used with by each participant were merged along with the annotations and instances, so that each participant is associated with an ontology file that represents its personal ontology and data model.

Aligning Ontologies:

In this step, each participant was asked to evaluate and provide two alignments with the ontologies of other participants. The participants are organized in a circle and each one has to provide the alignment between its ontology and that of its neighbors.

Each team member had to align his own ontology with an assigned participant. Alignments are a set of correspondences, which was provided in the following format:

uri1 \t rel \t uri2 \n

URI of a class or property of the first ontology (ones own), then tabulation, then a relation (like =, >, < or anything that one deem necessary), then tabulation, then URI of a class or property of the second ontology (the one assigned).

These alignments acted as reference alignments for evaluation with the ones created by the Alignment API.

The assignment for the alignments was the following:

- AZ aligns his ontology with JP's.
- AS aligns his ontology with FAK's.
- FAK aligns his ontology with JE's.
- JJ aligns his ontology with AS's.
- JE aligns his ontology with SL's.
- JP aligns his ontology with JJs.
- SL aligns his ontology with AZ's.

So we have the cycle JP → JJ → AS → FAK → JE → SL → AZ → JP. Since, the experiment with PicSter at INRIA will be continued even after the termination for the thesis, the results from this alignment evaluation cycle could later be used to evaluate compositions between the alignments as a future work (Chapter 6).

Alignment Evaluation:

The most prominent criteria to evaluate the performance of the alignment algorithms are to calculate their precision and recall values; a technique adapted from information retrieval area. Precision and recall are based on the comparison of the alignment obtained through the algorithm with a reference alignment; hence finding which correspondences are found correctly/missing.

There exist a number of metrics which are used to calculate the precisions and recall values, namely:

- *True positives:* The number of correct alignments an alignment file contains.
- *False positives:* The number of incorrect a alignments an alignment file contains.
- *False negatives:* The number of correct alignments missed in an alignment file.

DEFINITION (PRECISION, RECALL) [EE05]. *Given a reference alignment R, the precision of some alignment A is given by*

$$Pr(A, R) = \frac{|R \cap A|}{|A|}$$ i.e. (truePositives/(truePositives + falsePositives))

and recall is given by

$$Re(A,R) = \frac{|R \cap A|}{|R|}$$ i.e. (truePositives/(truePositives + falseNegatives))

Essentially, precision gives the proportion of correct alignments among those found and recall give the proportion of correct alignments found.

A summary of the alignment evaluation results is given below in table 5.2. The results do not seem very interesting. There were a number of reasons responsible for the bad precision and recall values. The most important reason for these results was the bad quality of reference alignments provided by the experiment participants. This was mainly because most of the participants used and modified already available ontologies that were having a huge number of classes (e.g SUMO). Since there was no interface available for participants to align these big ontologies, browsing through such ontologies was difficult; hence the quality of reference alignments was bad. Due to this reason, the number of alignments provided in the reference alignments was very less to obtain good results. This can be seen from the table 5.1; the number of alignments provided by most of the participants (except

AZ-JP and AS-FAK) is very less when compared with the number of entities in the respective ontologies to be aligned. The number of alignments between AZ-JP and AS-FAK is comparable to the number of entities in the ontologies; hence the precision and recall values for these are relatively better than others. Therefore, most of the values in table 5.2 do not hold any relevance. Also, in some cases, the images were annotated in different languages; English and French; which resulted in a number of missing alignments by the algorithms. This increased the number of `falseNegatives`; thus lowering the recall value.

Additionally, the alignment algorithms gave a number incorrect alignments due the terminology used in the annotations. For example the Levenshtein algorithm gave a confidence measure of 0.56 for equality relation to the following semantically unrelated instances (`house3` and `houseDoor`):

```
http://exmo.infrialpes.fr/people/sharma/elster/instances/House.rdf#
house3
http://exmo.infrialpes.fr/people/faisal/elster/instances/Doors.rdf#
houseDoor
```

Similarly, the JaroMeasure algorithms gave a confidence measure of equality of 0.82 for equality relation to the following unrelated classes (`Country` and `Counting`):

```
http://daml.umbc.edu/ontologies/cobra/0.4/space-basic#Country
http://exmo.inrialpes.fr/people/JP/Picster/SUMO.owl#Counting
```

Such incorrect alignments increased the number of `falsePositives`; thus lowering the precision value. Finally, it is clear that the precision and recall values given by simple algorithms such as equalDistance are relatively better because they are based on string equality conditions.

Align	Number of Cells	#Class1/#Class2
AZ-JP	138	141/1449
JP-JJ	14	1449/1468
JJ-AS	17	1468/158
AS-FAK	50	158/738
FAK-JE	75	738/390
JE-SL	--	390/1395
SL-AZ	19	1395/158

Table 5.1: Number of correspondences in reference alignments

Algo/Test		AZ-JP	AS-FAK	FAK-JE	JJ-AS	JE-SL	JP-JJ	SL-AZ	H-mean
refalign	**Prec.**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Rec.	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
equalDistance	**Prec.**	0.83	0.40	0.04	0.09	--	0.00	0.00	0.01
	Rec.	0.13	0.04	0.10	1.00	--	0.07	0.00	0.10
subStringDistance	**Prec.**	0.09	0.03	0.00	0.00	--	0.00	0.00	0.00
	Rec.	0.25	0.06	0.10	1.00	--	0.29	0.25	0.18
hammingDistance	**Prec.**	0.10	0.03	0.00	0.00	--	0.00	0.00	0.00
	Rec.	0.28	0.06	0.10	1.00	--	0.29	0.25	0.18
jaroMeasure	**Prec.**	0.10	0.03	0.00	0.00	--	0.00	0.00	0.00
	Rec.	0.28	0.06	0.10	1.00	--	0.29	0.50	0.19
ngramDistance	**Prec.**	0.10	0.05	0.00	0.00	--	0.00	0.00	0.00
	Rec.	0.25	0.08	0.10	1.00	--	0.29	0.00	0.18
levenshteinDistance	**Prec.**	0.08	0.03	0.00	0.00	--	0.00	0.00	0.00
	Rec.	0.23	0.06	0.10	1.00	--	0.21	0.50	0.18
smoaDistance	**Prec.**	0.11	0.05	0.00	0.00	--	0.01	0.00	0.00
	Rec.	0.28	0.08	0.10	1.00	--	0.36	0.25	0.20

Table 5.2: Summary of alignment algorithm evaluation results

5.2 PicSter evaluation by users

The feedback and information given by the users were collected to serve two purposes, namely, to analyse the usability of the application developed; and, to gather data and lay a basis for further research at INRIA.

Participants of the experiment described in section 5.1 were asked to give a general feedback on the application prototype. Since the experimentation was done at the development phase, not all functionalities of the developed prototype could be evaluated.

As discussed in section 5.1, participants were asked to complete a set of tasks and their experience with the tool was collected. A set of questions were asked to the users. There were two main subjects in this evaluation, which are usability in terms of user satisfaction and data collection for social network analysis.

The overall evaluation evaluates the usability aspects as listed in table 5.3. Users were asked to rate their satisfaction levels on a scale of 1 (very poor) to 5 (very good). The set of question asked can be found in Appendix C. The table shows that users were satisfied with the annotation process in the range of 3 and 3.5. Since the searching capabilities were in the rudimentary stage during the experimentation phase, most of the users could not evaluate this functionality; the average satisfaction level for searching was 3. The satisfaction level with ontology edition functionality was between 2.5 and 3.

Three users wanted to have additional ontology editing functionalities. One user wanted to use the ontology editing functionality of PicSter for other uses. Two users

had used other ontology based systems and only one user currently annotates his photos with other photo annotation tools.

In addition, the users mentioned that the annotation process involves many number of clicks which makes the job more time consuming. Four users mentioned that familiarising with the system takes more time. Additionally, the user feedback involved bug-reporting, most of which have been removed while others have been documented in the source code. Overall, the users were satisfied with idea behind the experiment and all of them agreed that experiments such as the one done with PicSter are useful as semantic annotation software will grow in future.

Aspect	Question	Average User Rating
Annotation	How would you rate your overall satisfaction with the annotation process?	3.25
Search	How would you rate your overall satisfaction with the search process?	3
Ontology Editing	How would you rate your overall satisfaction with the ontology editing process?	2.67
Usability	How satisfied are you with the overall usability of the prototype?	2.67

Table 5.3: Users satisfaction rating

Since the peer-to-peer and query mediator modules of PicSter were developed in the second half of the thesis, user feedback about these functionalities was not possible.

The participants of the PicSter experiment were asked to provide additional information which will be used in further research. A summary of this information is mentioned here for the purpose of documentation only. These parameters might assist in semantically analysing the social networks (e.g. two people using the SUMO ontology are alike) and further experimentation.

Participant	No. of Ontologies used	No. of instances created	No of classes in the ontologies	No. of properties in the ontologies	Time taken for annotation	Closest affinity with
AS	3(SUMO,FOAF,SPACE)	66	158	242	4	JJ+AZ
AZ	5(AKT, office, SPACE, others)	87	141	45	8	JE
FAK	2(earthrealm, Travel)	81	738	38	11	JE+AS
JE	6(SPACE,FOAF+4)	78	390	30	35	AZ+SL
JJ	1(SUMO)	49	1468	208	6	AS+SL
JP	1(SUMO)	27	1449	208	5	SL+AZ
SL	2(SUMO)	24	1395	416	--	--

Table 5.4: Parameters for photo annotation and social network analysis

5.3 Comparison with existing systems

This section examines the PicSter prototype by comparing it with other systems along the lines of the technologies involved in this thesis. Although PicSter has not yet been evolved as a fully developed and tested system, it still provides a start to address the issue of heterogeneity in peer-to-peer environment using ontology alignment and mediator services.

A number of systems have been developed to address different issues associated with (semantic) peer-to-peer systems in their own way. Some of them are still in their infancies while others have evolved among the users of a particular community. The comparison cited in this section is based on the following main considerations:

- *Search Model:* Keyword based search or metadata based search?
- *Heterogeneity:* Does the system specify a fixed shared schema or allows multiple metadata schemas support?
- *Query handling:* Does the system allow semantic querying? Does the system allow expertise aware query forwarding?
- *Degree of autonomy:* Does the system accept arbitrary incoming queries?
- *Level of personalization:* Does the system allow full autonomy to users to edit the vocabulary used for annotation and search process?
- *Query Translation:* Does the system provide a query meditation service? If so, what type of translation does it apply?

Some of the well known Peer-to-Peer systems are briefly described here on the basis of above mentioned criteria and then later discussed with respect to PicSter. Table 5.5 summarizes this comparison.

System	Markup-Scheme	Semantic Routing	Query Forwarding	Semantic Query	Ontology Query Language	Reference Ontology	Ontology Alignment	Ontology Edition
Gnutella	Keyword	No	Yes	No	-	-	-	-
Napster	Keyword	No	No	No	-	-	-	-
Edutella	RDF	Yes	Yes	Yes	RQL	Yes	No	No
Bibster	RDF/ DAML+OIL	Yes	Yes	Yes	SeRQL	Yes	No	No
Piazza	Database	No	Yes	Yes	XQuery	-	No	No
C-OWL (not a system)	C-OWL	Application specific	Application specific	Application specific	Application specific	-	Explicit mappings	-
PicSter	RDF/OWL	No	No	Yes	SPARQL	**No**	**Yes**	**Yes**

Table 5.5: Comparison of PicSter with existing peer-to-peer systems

66

Gnutella, Napster

Gnutella [Lim00] and Napster [NAP01] are ancestors in P2P computing. They only support keyword-based search. Gnutella is representative instance for the query flooding which can not scale well. Napster adopts central servers to maintain a centralized directory from which connected peers can register their expertise and also retrieve a list of peers of users' interest. Since keyword bases search systems do not allow any semantic level querying, they are not discussed further in this thesis.

Edutella and Bibster

Edutella [NWQ+02] as described in chapter 2 is built on the JXTA framework and aims to combine metadata with P2P networks. In Edutella, each peer stores locally data (educational resources) that are described in RDF relatively to some reference ontologies (e.g., http://dmoz.org). For instance, a peer can declare that it has data related to the concept of the dmoz taxonomy corresponding to the path Computers/Programming/ Languages/Java, and that for such data it can export the author and the date properties. The overlay network underlying Edutella is a hypercube of superpeers to which peers are directly connected. Each super-peer is a mediator over the data of the peers connected to it. When it is queried, its first task is to check if the query matches with its schema: if that is the case, it transmits the query to the peers connected to it, which are likely to store the data answering the query; otherwise, it routes the query to some of its neighbour super-peers according to a strategy exploiting the hypercube topology for guaranteeing a worst-case logarithmic time for reaching the relevant super-peer.

Bibster[RaB01], like Edutella is also built on the JXTA framework and focuses on bibliographic records. It is embedded in the general SWAP architecture. It exploits ontologies in data representation, query formulation, query-routing and answer presentation. However like Edutella, it uses reference ontologies (like SWRC, ACM Topic Hierarchy) for each such module. Hence, it also does not provide users complete autonomy for metadata vocabulary.

Piazza

Piazza [HITM03] does not consider that the data distributed over the different peers must be described relatively to some existing reference schemas. Each peer has its own data and schema and can mediate with some other peers by declaring mappings between its schema and the schemas of those peers. The topology of the network is not fixed but accounts for the existence of mappings between peers: two peers are logically connected if there exists a mapping between their two schemas. The underlying data model of the first version of Piazza is relational and the mappings between relational peer schemas are inclusion or equivalence statements between conjunctive queries. Such a mapping formalism encompasses the Local-as-View and the Global-as-View formalisms used in information integration systems

based on single mediators. The price to pay is that query answering is undecidable except if some restrictions are imposed on the mappings or on the topology of the network. The currently implemented version of Piazza relies on a tree-based data model: the data is in XML and the mappings are equivalence and inclusion statements between XML queries. Query answering is implemented based on practical (but not complete) algorithms for XML query containment and rewriting. It maintains a centralized storage of all the schemas and mappings in a global server.

C-OWL

In [BGvH$^+$03], a context extension is given to OWL, called C-OWL, which is based on local model semantics and distributes description logics. C-OWL allows us to contextualize ontologies. Hence it provides some formal semantics for distributed ontologies. However, the vocabularies of local ontologies are supposed to be pair-wise disjoint, and the globalization can only be obtained by using explicit mappings. It doesn't fit very well in with one of the basic architectural principles of the Web, which allows anyone be able to freely add information about an existing resource using any vocabulary they please.

Discussion

Keyword based P2P systems are sufficient for applications which do not need complex query languages and complex metadata, such as sharing MP3 files. However, photographic annotations in PicSter, educational resources in Edutella, or bibliographic records in Bibster require processing of complex semantic queries by making use of ontologies

While each of the metadata based systems described above were driven by different requirements, most of them address the issue of semantic heterogeneity in their own way. In Bibster, for instance, providing reference ontologies for bibliographic search might not affect the level of usage because, as the metadata domain is limited. However in most semantic web applications this assumption fails and a need to provide more autonomy to users arise.

Although PicSter is not tested and used as extensively as other systems like Edutella and Bibster, it does provide an initiative to provide full autonomy to the users with respect to the vocabulary used. Additionally it allows users to develop their ontologies on the fly without depending on any of the external ontology editors. The query mediator service provided by PicSter, albeit simple, marks a beginning of ontology alignment algorithms usage in real world applications.

Nevertheless, PicSter has its own limitations which need to be handled and tested before it can be considered as a more robust application. Some of them include semantic query routing and forwarding mechanism. Although query forwarding may not seem to be a necessity in the case of photo annotation tool like PicSter where the user manually selects the peers to be queried, it becomes an efficiency

requirement if the user wants the system to select the peers. Bibster and Edutella both provide expertise based query forwarding schemes which can be adopted in PicSter. Additionally since PicSter has not yet been tested extensively, its scalability remains unexplored.

5.4 Conclusion

The PicSter experiment described in section 1 was the basis for further evaluation of the developed prototype and the thesis in general. The problem faced by most of the users in the experiment was to find a suitable ontology to annotate pictures. Most of them were unsatisfied with the ontologies found on Swoogle[4]. There are very few quality ontologies and these are not "lightweight ontologies". So the experimenter cannot avoid falling into two traps: using heavy-weight ontologies, with a huge number of (and relatively useless) concepts and features; or using several toy-ontologies which lead to a very strange compilation of distinct small hierarchies made from different viewpoints etc. Using such huge ontologies in turn affected the experiment during the evolution of alignment algorithms phase. Given such large ontologies and lack of a user interface to manually align them, the quality of reference alignments provided by the participants was very poor to produce interesting results. The lessons learnt through this experiment could be used in further experiments with tools like PicSter to get more convincing results for alignment algorithms.

The feedbacks from the users conclude that the prototype application is satisfactory but the real application can be improved as per the comments received. Almost all users agree that such prototypes are useful in the continuously evolving area of semantic web based applications and photo annotation tools in particular. However, the current version of prototype needs further changes for it to be adopted and be useful for people outside the computer science domain.

Finally, the developed prototype was compared with other similar systems in section 5.3 and salient features of PicSter were discussed in detail.

---

[4] http://www.swoogle.umbc.edu

69

Chapter 6

Conclusion and Future Work

This chapter summarizes the previous chapters that described the approach to annotate and share personal photographic metadata and addressed the issue of heterogeneity in the semantic web. Since the field of semantic web and peer-to-peer infrastructures is still developing rapidly, some ideas for future research are discussed.

6.1 Summary

The thesis addressed three key issues, namely, ontology based photo annotation; peer-to-peer metadata sharing; and heterogeneity problem in the semantic web. The thesis work has resulted in development of an application prototype called PicSter, which combined these three research areas.

Image annotation is an important issue because of the rapid increase in number of personal pictures. A number of existing image annotation tools were studied and the important missing functionalities of each of them was stated. A case was developed to address the personalization aspect of semantic photo annotation. An existing photo annotation tool was extended into a new application that could allow users to annotate pictures by using and extending their own hierarchy of ontological concepts.

A number of semantic peer-to-peer systems were studied and a P2P infrastructure was developed for sharing photo annotations. Existing technologies like JXTA were used to develop a photo annotation sharing infrastructure.

An application was developed which could address the problem of heterogeneity in semantic web by providing and analyzing real users' data. For this purpose an ontology alignment based query mediator was written. To the best of our knowledge, this is the first such application which uses an ontology mediator to share ontology metadata.

The PicSter experiment performed during the thesis work was explained. The data obtained through this experiment was analyzed and used to evaluate the alignment algorithms implemented in the alignment API developed at INRIA. However, this evaluation of algorithms did not produce interesting results due to the lack of sound reference alignments with which the algorithmic alignments could be compared. Feedback from the experiment participants was used to evaluate the application prototype. The users were satisfied with idea behind the experiment and all of them agreed that experiments such as the one done with PicSter are useful as semantic annotation software will grow in future. Finally a comparison of PicSter with other similar systems was done where salient features and limitations of PicSter were discussed.

6.3 Future Work

The prototype developed during the masters' work provides a basis for further similar tools that could explore various ontology matching techniques to deal with problem of interoperability in personal as well as corporate environments.

However, there still needs a significant amount of work to done for PicSter to evolve as a completely robust photo annotation tool. Some of the ideas which could be added into PicSter are:

- Semantic Query Routing: Currently PicSter relies on the inbuilt JXTA query routing mechanism. An ideal routing scheme would be to route the queries on the basis of the expertise level each peer. In a large network such routing schemes could increase the efficiency manifolds.
- Query Forwarding: Currently PicSter relies on the assumption that users manually select the peers to which they want to communicate. While in a small network of peers this assumption appears valid, it might not be as simplistic in when the numbers of peers increase. In such cases, automatic peer selection techniques could be included in PicSter.
- Ontology editing: The ontology editing functionality could be extended to include a few additional capabilities.
- Alignment Service: The alignments used by the query mediator could be generated on the fly by implementing the alignment service.
- Extending Query Mediator: Currently the PicSter query mediator translates only equivalent correspondences. A more advanced version including the subclasses and other correspondences could be generated as the research area evolves.
- Different media: The infrastructure and techniques used in PicSter development could be applied to other types of media like the videos.

PicSter Experimentation: The experimentation done in during the thesis period with PicSter tool will be further extended along the lines of research activities of the EXMO team at INRIA. In particular, the results and data obtained will be used for the following tasks:

Semantic Social Network Analysis: The results and data obtained during this masters' work will be used to evaluate the distance between the ontologies of the users in order to compute a weighted social network. Such a semantic network would then be compared to the user provided affinity measure.

Additionally, PicSter could also be extended to act as a recommender system based on such ontological distances. For instance, *Peer B could recommend Peer A that Peer C uses similar ontology than him and hence Peer A could contact (befriend with) Peer C for metadata exchange.*

Alignment Composition: The cyclic alignments obtained through the PicSter experiment could be used for generating alignments by composition, inverse and intersection that can be compared with the generated alignments and can be used for assessing the properties of composition (is it weaker than the direct alignments).

References

[ACG+05]   Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset and Laurent Simon. SomeWhere in the Semantic Web. *Third International Workshop on Principles and Practice of Semantic Web Reasoning.*

[BEH+04]   Jeen Broekstra, Marc Ehrig, Peter Haase, Frank van Harmelen, Maarten Menken, Peter Mika, Bjoern Schnizler, and Ronny Siebes. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the SemPGrid 04 Workshop*, New York, May 2004

[BFM02]   Chris Bussler, Dieter Fensel, and Alexander Mädche. A conceptual architecture for semantic web enabled web services. *SIGMOD Records, 31(4):24–29, 2002.*

[BG04]   Dan Brickley, R.V. Guha. RDFS recommendation by W3C: http://www.w3.org/TR/rdf-schema/

[BGvH+03]   Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., and Stuckenschmidt, H. (2003). C-OWL: Contextualizing ontologies. In Sekara, K. and Mylopoulis, J., editors, *Proceedings of the Second International Semantic Web Conference*, number 2870 in Lecture Notes in Computer Science, pages 164–179. Springer Verlag.

[BHS03]   F. Badder, I. Horrocks, U. Sattler. Description Logics as Ontology Languages for the Semantic Web.
www.cs.man.ac.uk/~horrocks/ Publications/download/2003/BaHS03.pdf

[Bru03]   Jos De Bruyn. Using Ontologies. DERI Technical Report DERI-2003-10-29

[BSZ03]   Paolo Bouquet, Luciano Serafini, and Stefano Zanobini. Semantic coordination: A new approach and an application. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc 2nd ISWC*, volume 2870 of *Lecture Notes in Computer Science*, pages 130–145, Sanibel Island (FL, USA), October 2003. Springer Verlag.

[EBB+04]   J. Euzenat, J. Barrasa, P. Bouquet, R. Dieng, M. Ehrig, M. Hauswirth, M. Jarrar, R. Lara, D. Maynard, A. Napoli, G. Stamou, H. Stuckenschmidt, P. Shivako, S. Tessaris, S. Van Acker, I. Zaihrayeu, and T. Bach. D2.2.3: State of the art on ontology alignment. Technical report, NoE Knowledge Web project, 2004.

[EE05]   Marc Ehrig, Jérôme Euzenat. Relaxed Precision and Recall for Ontology Matching. *Proceedings of the Workshop on Integrating Ontologies*, volume 156, pp. 8

[EP04]   Jérôme Euzenat and Petko Valtchev. Similarity based ontology alignment in OWL-lite. In *Proc. 15th ECAI*, Valencia (ES), 2004.

[ES05]   M. Ehrig, Y. Sure. Framework for Ontology Alignment and Mapping. http://www.aifb.uni-karlsruhe.de/WBS/meh/foam/

[Euz04]   Jérôme Euzenat. An API for ontology alignment. In *Proc. 3rd international semantic web conference, Hiroshima (JP)*, 2004.

[Euz05]   J. Euzenat. Alignment Infrastructure for Ontology Mediation and other Applications. *First International Workshop on Mediation in Semantic Web Services, 2005*.

[Fen01]   D. Fensel, Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, Springer-Verlag, Berlin, 2001.

[GSY04]   Fausto Giunchiglia, Pavel Shvaiko, and Michael Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *Proceedings of ESWS 2004, Heraklion (GR)*, pages 61–75, 2004.

[hWGS+05]   Christian Halaschek-Wiener, Jennifer Golbeck, Andrew Schain, Michael Grove, Bijan Parsia, and Jim Hendler. Photostuff - an image annotation tool for the semantic web. In *4th International Semantic Web Conference - Poster Paper*, 2005.

[HITM03]   Halevy, A., Ives, Z., Tatarinov, I., Mork, P. Piazza: data management infrastructure for semantic web applications. In: *WWW'03.* (2003)

[KFHH01]   M. Klein, D. Fensel, F. Harmelen, I. Horrocks. Relation between Ontologies and Schema-Languages: Translating OIL-Specifications to XML-Schema. http://www.cs.vu.nl/~frankh/postscript/ECAI00-WS2.pdf

[KS03]   Y. Kalfoglou, M. Schorlemmer. Ontology Mapping: The state of art. *The Knowledge Engineering Review,* Vol 18, 2003, Pages 1-31

[LB02]   Y. Lafon, B. Bos. Describing and retrieving photos using RDF and HTTP. http://www.w3.org/TR/photo-rdf/

[Lim00]   LIME WIRE LLC. Gnutella - limewire 4.0. http://www.limewire.com/, 2000.

[LM99]   Yves Lafon & Benoit Mahe. Jigsaw 2.0 internal design. July 1999: http://www.w3.org/Jigsaw/Doc/Programmer/design.html

[McvH04]   Deborah L. McGuinness, Frank van Harmelen. Web Ontology Language: http://www.w3.org/2004/OWL/

[Mil02]   Libby Miller: RDF Squish Query Language: http://ilrt.org/discovery/2001/02/squish/

[MM04]   Frank Manola, Eric Miller. RDF Primer: http://www.w3.org/TR/rdf-primer/

[NAP01]   NAPSTER, LLC. Napster. http://www.napster.com, 2001.

[NWQ+02]   Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, Tore Risch. EDUTELLA: A P2P Networking Infrastructure Based on RDF, WWW 2002.

[PJR+04]   Matthew Perry, Maciej Janik, Cartic Ramakrishnan, Conrad Ibañez, Budak Arpinar, Amit Sheth. Peer-to-peer discovery of semantic associations.P2PKM'04. www.p2pkm.org/downloads/Perry2005.pdf

[PS06]   Eric Prud'hommeaux, Andy Seaborne. SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/

[RaB01]   Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4):334–350, 2001

[RB01]   Erhard Rahm and Philip Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[RDFPic]   RDFPic: http://jigsaw.w3.org/rdfpic/

[SDWW01]   Guus Schreiber, Barbara Dubbeldam, Jan Wielemaker, and Bob Wielinga. Ontology-based photo annotation. *IEEE Intelligent Systems*, May/June 2001.

[SE05]   P. Shvaiko, J. Euzenat. A Survey of Schema-Based Matching Approaches, *Journal of Data Semantics*, Springer Verlag, 2005. Pages 146-171.

[Sea04]   Andy Seaborne. RDQL – A Query Language for RDF: http://www.w3.org/Submission/RDQL/

[Sha05]   Arun Sharma. Ontology Based Annotation. Seminar paper, Informatik V. SS-2005. RWTH Aachen University

[Shv04]   Pavel Shvaiko. Iterative schema-based semantic matching. Technical Report DIT-04-020, University of Trento (IT), 2004.

[VP04]   P. Valduriez and E. Pacitti, Data Management in Large-scale P2P Systems, In Proc. 6th Int. Conf. on High Performance Computing in Computational Sciences, 2004

[XMLS]   XML Schema: http://www.w3.org/XML/Schema

Appendix A: Image Region Ontology

```xml
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF xmlns="http://www.mindswap.org/2005/owl/digital-media#"
        xml:base = "http://www.mindswap.org/2005/owl/digital-media"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
>

<owl:Class rdf:ID="DigitalMedia">
   <rdfs:label>Digital Media</rdfs:label>
   <rdfs:comment>The class of digital media data</rdfs:comment>
</owl:Class>

<owl:Class rdf:ID="Image">
   <rdfs:label>Image</rdfs:label>
   <rdfs:comment>The class of images</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#DigitalMedia"/>
   <rdfs:subClassOf
rdf:resource="http://xmlns.com/foaf/0.1/Image"/>
 </owl:Class>

 <owl:Class rdf:ID="Video">
   <rdfs:label>Video</rdfs:label>
   <rdfs:comment>The class of videos</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#DigitalMedia"/>
 </owl:Class>

<owl:Class rdf:ID="Segment">
   <rdfs:label>Segment</rdfs:label>
   <rdfs:comment>The    class    of    fragments    of    digital    media
content</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#DigitalMedia"/>
 </owl:Class>

 <owl:Class rdf:ID="StillRegion">
   <rdfs:label>Still Region</rdfs:label>
   <rdfs:comment>2D    spatial    regions    of    an    image    or    video
frame</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#Segment"/>
   <!--<rdfs:subClassOf rdf:resource="#Image"/>-->
 </owl:Class>
```

```
 <owl:Class rdf:ID="ImageText">
    <rdfs:label>Image Text</rdfs:label>
    <rdfs:comment>Spatial regions of an image or video frame that
correspond to text or
captions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
 </owl:Class>

<owl:Class rdf:ID="Mosaic">
    <rdfs:label>Mosaic</rdfs:label>
    <rdfs:comment>Mosaic    or    panaoramic    view    of    a    video
segment</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#StillRegion"/>
 </owl:Class>

 <owl:Class rdf:ID="VideoSegment">
    <rdfs:label>Video Segment</rdfs:label>
    <rdfs:comment>Temporal    intervals    or    segments    of    video
data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
    <rdfs:subClassOf rdf:resource="#Video"/>
 </owl:Class>

 <owl:Class rdf:ID="MovingRegion">
    <rdfs:label>Moving Region</rdfs:label>
    <rdfs:comment>2D    spatio-temporal    regions    of    video
data</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#Segment"/>
 </owl:Class>

 <owl:Class rdf:ID="VideoText">
    <rdfs:label>Video Text</rdfs:label>
    <rdfs:comment>Spatio-temporal    regions    of    video    data    that
correspond to text or captions</rdfs:comment>
    <rdfs:subClassOf rdf:resource="#MovingRegion"/>
 </owl:Class>


<owl:Class rdf:ID="VideoSegmentsOrStillRegions">
  <rdfs:label>VideoSegmentsOrStillRegions</rdfs:label>
  <owl:unionOf rdf:parseType="daml:collection">
    <owl:Class rdf:about="#VideoSegment"/>
    <owl:Class rdf:about="#StillRegion"/>
  </owl:unionOf>
</owl:Class>

<owl:Class rdf:ID="VideoFrame">
    <rdfs:label>VideoFrame</rdfs:label>
    <rdfs:comment> Frame of a video </rdfs:comment>
    <rdfs:subClassOf rdf:resource="#DigitalMedia"/>
    <rdfs:subClassOf rdf:resource="#Image"/>
 </owl:Class>

<!-- Properties -->

<owl:ObjectProperty rdf:ID="descriptor">
    <rdfs:domain rdf:resource="#DigitalMedia"/>
```

```
</owl:ObjectProperty>

 <owl:ObjectProperty rdf:ID="visualDescriptor">
    <rdfs:comment>Descriptor - applicable to images, videos, video
segments, still regions and moving
regions.</rdfs:comment>
    <owl:subPropertyOf rdf:resource="#descriptor"/>
 </owl:ObjectProperty>

<!-- Others -->

<owl:Class rdf:ID="ImagePart">
   <rdfs:label>Image Part</rdfs:label>
   <rdfs:comment>2D  spatial  regions  of  an  image  or  video
frame</rdfs:comment>
   <rdfs:subClassOf rdf:resource="#Segment"/>
   <!--<rdfs:subClassOf rdf:resource="#Image"/>-->
 </owl:Class>

<owl:ObjectProperty rdf:ID="depicts">
  <rdfs:label>depicts</rdfs:label>
  <owl:inverseOf rdf:resource="#depiction"/>
  <rdfs:subPropertyOf
rdf:resource="http://xmlns.com/foaf/0.1/depicts"/>
  <rdfs:subPropertyOf
rdf:resource="http://xmlns.com/foaf/0.1/depiction"/>
</owl:ObjectProperty>


<owl:ObjectProperty rdf:ID="depiction">
  <rdfs:label>depiction</rdfs:label>
  <rdfs:subPropertyOf
rdf:resource="http://xmlns.com/foaf/0.1/depiction"/>
</owl:ObjectProperty>


<owl:ObjectProperty rdf:ID="regionOf">
  <rdfs:label>regionOf</rdfs:label>
        <rdfs:range rdf:resource="#DigitalMedia"/>
        <rdfs:domain rdf:resource="#ImagePart"/>
  <owl:inverseOf rdf:resource="#hasRegion"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="frameOf">
  <rdfs:label>frameOf</rdfs:label>
        <rdfs:range rdf:resource="#Video"/>
        <rdfs:domain rdf:resource="#VideoFrame"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasRegion">
  <rdfs:label>hasRegion</rdfs:label>
        <rdfs:range rdf:resource="#ImagePart"/>
        <rdfs:domain rdf:resource="#DigitalMedia"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="segmentOf">
  <rdfs:label>segmentOf</rdfs:label>
```

```
        <rdfs:domain rdf:resource="#VideoSegment"/>
        <rdfs:range rdf:resource="#Video"/>
  <owl:inverseOf rdf:resource="#hasSegment"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasSegment">
  <rdfs:label>hasSegment</rdfs:label>
        <rdfs:domain rdf:resource="#Video"/>
        <rdfs:range rdf:resource="#VideoSegment"/>
</owl:ObjectProperty>


<owl:DatatypeProperty rdf:ID="startFrame">
  <rdfs:label>startFrame</rdfs:label>
        <rdfs:domain rdf:resource="#VideoSegment"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="endFrame">
  <rdfs:label>endFrame</rdfs:label>
        <rdfs:domain rdf:resource="#VideoSegment"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasDurationSeconds">
  <rdfs:label>hasDurationSeconds</rdfs:label>
        <rdfs:domain rdf:resource="#Video"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="hasTotalFrames">
  <rdfs:label>hasTotalFrames</rdfs:label>
        <rdfs:domain rdf:resource="#Video"/>
</owl:DatatypeProperty>


<owl:DatatypeProperty rdf:ID="frameNumber">
  <rdfs:label>frameNumber</rdfs:label>
        <rdfs:domain rdf:resource="#VideoFrame"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="svgOutline">
  <rdfs:label>svgOutline</rdfs:label>
        <rdfs:domain rdf:resource="#ImagePart"/>
</owl:DatatypeProperty>

</rdf:RDF>
```

Appendix B: Example alignment generated by the Alignment API

```
<?xml version='1.0' encoding='utf-8' standalone='no'?>
<rdf:RDF xmlns='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
     xml:base='http://knowledgeweb.semanticweb.org/heterogeneity/alignment'
     xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
     xmlns:xsd='http://www.w3.org/2001/XMLSchema#'>
<Alignment>
 <xml>yes</xml>
 <level>0</level>
 <type>11</type>
 <method>fr.inrialpes.exmo.align.impl.method.EditDistNameAlignment</method>
 <onto1> http://exmo.infrialpes.fr/people/euzenat/JeromE_space.owl</onto1>
 <onto2> http://exmo.infrialpes.fr/people/sharma/sharma_space-basic.owl</onto2>
 <uri1>http://space.frot.org/rdf/space.owl</uri1>
 <uri2>http://daml.umbc.edu/ontologies/cobra/0.4/space-basic</uri2>
 <map>
  <Cell>
   <entity1
rdf:resource='http://exmo.infrialpes.fr/people/euzenat/JeromE_space.owl#RuralHo
use'/>
   <entity2
rdf:resource='http://exmo.infrialpes.fr/people/sharma/sharma_space.owl
#VillageHouse'/>
   <measure
rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>0.5384615384615384
</measure>
   <relation>=</relation>
  </Cell>
 </map>
 <map>
  <Cell>
   <entity1 rdf:resource='http://exmo.infrialpes.fr/people/euzenat#City'/>
   <entity2 rdf:resource='http://daml.umbc.edu/ontologies/space- basic#city'/>
   <measure rdf:datatype='http://www.w3.org/2001/XMLSchema#float'>1.0
</measure>
   <relation>=</relation>
  </Cell>
  </map>
</Alignment>
</rdf:RDF>
```

Appendix C: Acronyms

API – Application programming interface
DAML - DARPA Agent markup language
DC – Dublin core
DL – Description logic
EXMO - Computer mediated exchange of structured knowledge (INRIA Research Team)
ECDM – Edutella Common Data Model
EXIF – Exchangeable image file format
FOAF – Friend of a Friend
FOL – First Order Logic
HTTP – Hypertext Transfer Protocol
JXTA - Juxtapose
OLA – Ontology Alignment API
OWL – Ontology Web Language
RDF/S – Resource Description Framework/Schema
SOAP – Simple Object Access Protocol
SPARQL – Simple Protocol and RDF Query Language
SQL – Structured Query Language
UDDI – Universal Description, Discovery
URI – Uniform Resource Identifier
W3C – World Wide Web Consortium
WSDL – Web Service Description Language
XML – Extensible Markup Language
XSLT – Extensible Style-sheet Language Transformation

Appendix D: Questionnaire

Where possible, please indicate a number between 1 (very poor) – 5 (very good)

- Considering all aspects of the experience, how would you rate your overall satisfaction with:

        A. Annotation Process
        B. Searching (if used)
        C. Ontology Editing

- How satisfied are you with the usability?

- Do you normally annotate pictures?

- Have you used any other ontology based systems?

- How much time did you take for annotating the given set of photographs

- Who among the other participants of the PicSter experiment do you feel closest (ontology-wise)?

- Any other comments?

List of Figures

List of Tables