DESIGN AND IMPLEMENTATION OF FALCON-AO++: AN IMPROVED ONTOLOGY ALIGNMENT SYSTEM

BY

FATSUMA JAURO

DEPARTMENT OF MATHEMATICS

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA.

JULY, 2014

DESIGN AND IMPLEMENTATION OF FALCON-AO++: AN IMPROVED ONTOLOGY ALIGNMENT SYSTEM

BY

Fatsuma JAURO, B.TECH COMPUTER SCIENCE (ATBU)

MSc/Scien/04690/2009-2010

A THESIS REPORT SUBMITTED TO THE SCHOOL OF POSTGRADUATE STUDIES, AHMADU BELLO UNIVERSITY, ZARIA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF M.Sc DEGREE IN COMPUTER SCIENCE.

DEPARTMENT OF MATHEMATICS,

FACULTY OF SCIENCE

AHMADU BELLO UNIVERSITY, ZARIA

NIGERIA

JULY, 2014

DECLARATION

I declare that the work in this thesis entitled, Design and Implementation of Falcon-AO++: An Improved Ontology Alignment System has been carried out by me in the Department of Mathematics. The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at this or any other Institution.

Name of student	Signature	Date

CERTIFICATION

This thesis entitled DESIGN AND IMPLEMENTATION OF FALCON-AO++: AN IMPROVED ONTOLOGY ALIGNMENT SYSTEM by Fatsuma JAURO meets the regulation governing the award of the degree of M.Sc Computer Science of Ahmadu Bello University, and is approved for its contribution to knowledge and literary presentation.

_____	_____
Chairman, Supervisory Committee	Date
_____	_____
Member, Supervisory Committee	Date
_____	_____
Head of Department	Date
_____	_____
Dean, Postgraduate school	Date

DEDICATION

This thesis is dedicated to my parents Hajiya Aishatu Adamu Boddoji and Late Mal. Jauro Gombe, brothers and sisters. It is also dedicated to my husband Dr. Shehu Muhammad Toro and my daughters Kawthar, Zahra and Siddiqa.

ACKNOWLEDGEMENT

I am infinitely grateful to Almighty Allah for giving me the strength and wisdom to carry out this research. First, I express my sincere gratitude to my major supervisor Professor Sahalu B. Junaidu, for his excellent guidance, tolerance, and encouragement throughout this research. Without his support, this work would not have existed. Special appreciation goes to my second supervisor Dr. S.E. Abdullahi, for his guidance and support.

I also acknowledge the effort of Absalom and Laide, for their technical contribution and encouragement. Special thanks to my colleague Bassama Bello Alhassan with whom we struggled together from the beginning to the end of MSc programme. Thanks also to all the lecturers of mathematics department especially, Mal. Muhammad Abdullahi, Mal. Mustafa Bagiwa, Aliyu Salisu, khalil Muh'd Zakari and Dr. H. G Dikko for their support from knowledge and experience.

I would like to express my warm appreciation to my husband Shehu Muhammad Toro, and my daughters; Kawthar, Zahra and Siddiqa for their patience, encouragement and prayers throughout the period of my study.

Special appreciation to my Mother Hajiya Aishatu Adamu Boddoji for her love, care, encouragement and special prayers. I also appreciate the support of my brothers; Abubakar Jauro, Adamu Jauro, Abdulrazaq Jauro,. Sani Jauro, Aliyu Jauro, Ahmad Jauro, and Imamu Jauro. Thanks also to my sisters; Aishatu Jauro, Kubura Jauro, Aishatu Jauro and also to all my nieces especially Maimuna Awwal Shehu and my Nephews especially Mustafa Abubakar Jauro and Zailani Adamu Jauro for their concern. I also thank Abubakar Sani Abdullahi for his brotherly assistance.

I would also like to thank Salamatu Isah, Mariya Ibrahim, Zainab Moh'd, Maryam Pulatan, Aisha Abu Gidado, Abubakar Sule and Hauwa Yakubu for their well wishes and encouragement. Thanks to all those whom I may have forgotten to mention their names.

ABSTRACT

In this thesis, Falcon-AO++: an extension of Falcon-AO alignment system that supports the interactive contribution of a domain expert in the matching process was developed and implemented. The two systems were evaluated using Ontology Alignment Evaluation Initiative (OAEI) Conference track data set and evaluation measures. The evaluation results have shown clearly that the contribution of an expert and the matching ability of matchers can improve alignment results.

TABLE OF CONTENTS

DECLARATION ..... **iii**
CERTIFICATION ..... **iv**
DEDICATION ..... **v**
ACKNOWLEDGEMENT ..... **vi**
ABSTRACT ..... **vii**
TABLE OF CONTENTS ..... **viii**
LIST OF FIGURES ..... **x**
LIST OF TABLES ..... **xi**

CHAPTER ONE **1**
INTRODUCTION **1**
1.1 Background of the Study **1**
1.2 Motivation and Problem Statement **4**
1.3 Thesis Goal **7**
1.4 Research Questions **7**
1.5 Research Aim and Objectives **8**
1.6 Methodology **8**
1.7 Research Contribution **8**

CHAPTER TWO **9**
LITERATURE REVIEW **9**
2.1 Introduction to Ontology **9**
2.1.1 Structure of Ontology 10
2.1.2 Benefits of developing an ontology 11
2.1.3 Ontology Languages 11
2.1.4 Ontology interoperability techniques 13
2.2 Ontology Alignment **13**
2.2.1 Applications of Ontology Alignment 15
2.3 Ontology matching Techniques **16**
2.3.1 Element level techniques 17
2.3.2 Structure level Techniques 19
2.3.3 Other Matching Techniques 20
2.4 Limitations of Some Matching Techniques **21**
2.5 Review of Ontology Alignment Systems **22**
2.5 Summary **29**

CHAPTER THREE **30**
DEVELOPMENT OF FALCON-AO++ **30**
3.1 System Requirement **30**

3.2 Architecture of Falcon-AO++ 31
3.2.1 User Input 31
3.2.2 User Restrictions 33
3.2.4 Parsing and Model Construction 43
3.2.5 Similarity Combination 44
3.2.6 Alignment Generation 45
3.3 Jena 46
3.3.1 Jena Reasoners and Rule Engines 47
3.4 Summary 48

CHAPTER FOUR 50
IMPLEMENTATION OF THE FALCON-AO++ 50
4.1 Graphical User Interface of Falcon-AO 50
4.2 Falcon-AO++ 51
4.2.1 Parsing and Entity Retrieval Process 52
4.2.2 Model Construction and Constraint Specification 53
4.2.3 Rule Implementation 54
4.2.4 User Input Data Structure and Integration with Falcon-AO 56
4.3 Evaluation of Alignment Systems 58
4.3.1 OAEI (Ontology Alignment Evaluation Initiative) 58
4.3.2 Data Set 58
4.3.3 Evaluation Measures 59
4.3.4 Results and discussion 60
4.4 Summary 63

CHAPTER FIVE 65
SUMMARY, CONCLUSION AND RECOMMENDATION 65
5.1 Summary 65
5.2 Conclusion 65
5.3 Recommendation 66
5.4 Limitations and Future work 66
APPENDIX ONE: Sample of Codes 70

List of Figures

Figure 1.1: Sample of Heterogeneous Ontologies Describing University Domain 5
Figure 2.1: An illustration of ontology alignment 14
Figure 2.2: Architecture of Falcon-AO 27
Figure 3.1 Architecture of Falcon-AO++ 32
Figure 3.2: Process Flow Diagram 35
Figure 3.3: PBM Process 36
Figure 3.4: Sample Ontologies 37
Figure 3.5: Structural Proximities of O_1 37
Figure 3.6: Sample of Clusters in O_1 38
Figure 3.7: Sample of Blocks 38
Figure 3.8: Block Matching 40
Figure 3.9: Directed Bipartite Graph of O_2 43
Figure 4.1: GUI of Falcon-AO 51
Figure 4.2: GUI of Falcon-AO++ 52
Figure 4.3: Creating an Empty Model 53
Figure 4.4: Adding User Selected Entities as Classes into Temporary Ontology File 53
Figure 4.5: Rule Implementation 54
Figure 4.6: Storing Equivalent Entities in Array List Data Structures 55
Figure 4.7: Storing Disjoint Entities in Array List Data Structures 55
Figure 4.8: Implementing Disjoint Constraint 56
Figure 4.9: Creating Data Structures for Equivalent Entities 57
Figure 4.10: Integrating User Input Data Structures with the Data Structure of Falcon-AO. 57
Figure 4.11: Comparative Results of Falcon-AO and Falcon-AO++ 63

List of Tables

Table 2.1: Summary of reviewed systems 28
Table 3.1: Some Jena packages and classes 47
Table 4.1: Alignment results 61
Table 4.2: Evaluation results 62

CHAPTER ONE

INTRODUCTION

This chapter gives an introduction of the research work. The background, motivation, problem statement and goal, research questions, objectives, and the research contributions are all addressed in this chapter.

1.1 Background of the Study

The present WWW (World-Wide Web) has over a billion pages containing information on almost every subject and is used by millions of people around the world. What hinders the World Wide Web from achieving its goal of being useful to both humans and machines is the fact that most of the information is designed only for human consumption (not understandable by machines). In many situations, the same information relating to a particular domain is presented heterogeneously (i.e it is contained in several documents with heterogeneous structures). Users can cope with cases of heterogeneity, but machines find it very difficult to understand the semantics of the information. Although the technologies used by keyword-based search engines (e.g. Yahoo, Google) have been improved, still the results provided by the search engines are sometimes insufficient or irrelevant hence not serving the expectation of the user. Antoniou and Harmelen, (2008) described the problems associated with keyword-based search engines as follows:

1. High recall, low precision. Even if the main relevant pages are retrieved, they are of little use if another 28,758 mildly relevant or irrelevant documents are also retrieved. Too much can easily become as bad as too little.
2. Low or no recall. Often it happens that we don't get any relevant answer for our request, or that important and relevant pages are not retrieved. Although low recall is a less frequent problem with current search engines, it does occur.

3. Results are highly sensitive to vocabulary. Often our initial keywords do not get the results we want; in these cases the relevant documents use different terminology from the original query. This is unsatisfactory because semantically similar queries should return similar results.
4. Results are single Web pages. If we need information that is spread over various documents, we must initiate several queries to collect the relevant documents, and then we must manually extract the partial information and put it together.

These problems remain the same despite improvements to search engines. To find the expected result, a user has to browse selected documents and extract the required information, this makes search to be very tedious and time consuming. The major cause of all these problems is because information contained on the web is not machine - understandable.

To overcome the existing problems, researchers suggested that the content should be used as it is and that techniques based on artificial intelligence and computational linguistics be developed. This approach has been used for sometime but still requires great effort.

The idea of semantic web evolved as a means of enhancing the web data structure. More concretely, research aimed at realizing the Semantic Web is an attempt to transform the current Web into an information space with a *semantic organisational foundation*; an information space that makes information semantically accessible to machines by considering its meaning (Maedche, 2002). The semantic web will contain both data and meta-data that describe the data. Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, enabling better cooperation between users and computers (Berners-Lee, 2001).

What is required by the semantic web is a well structured collection of information and set of inference rules so that machines can easily perform reasoning; this is provided by ontology (Berners-Lee, 2001).

Ontology is an essential tool for the semantic web. It describes a domain by defining terms and relations in that domain. It serves as a means of interaction between people and organisation by providing a common vocabulary over a domain. The benefits of using ontologies include reuse, sharing and portability of knowledge across platforms, and improved maintainability, documentation, maintenance, and reliability (Lambrix and Tan, 2007). Generally, ontologies give an improved understanding of a domain and information management in that domain. In computer science, the goal of ontology is that it adds meaning to the data such that an application can classify the information.

In the semantic web, ontology can be used to enhance the accuracy of web searches such that search engines can search for only those pages that describe a particular concept instead of the ones having ambiguous terms.

Various ontologies describing the same domain have been developed by different people and organizations. In most cases, the ontologies are not absolutely equal due to differences in used vocabularies, varying coverage of domain and many other reasons. This gives rise to heterogeneity problem. Information processing across the ontologies of a given domain cannot be possible without knowledge of the semantic mapping between them. Creating this mapping is the alignment problem, and solving it is the first step to semantic interoperability. Ontology alignment tries to solve the semantic heterogeneity problem by discovering the matching between entities of different ontologies that describe the same domain (Lambrix and Tan, 2007).

1.2 Motivation and Problem Statement

Large number of ontologies have been developed and made publicly available. Many of these ontologies may describe the same domain, but they are very different because they have been designed and developed independently by different ontology developers following diverse modeling principles and patterns.

Ontology heterogeneity could be due to use of different information representation formats (syntactic heterogeneity), depth of coverage or area of coverage (structural heterogeneity). The use of standard data representation languages such as Resource Description Framework (RDF), Resource Description Framework Schema (RDFS), Web Ontology Language (OWL) and some other components are being used to solve these kinds of heterogeneities. Another kind of heterogeneity issue which prevents the semantic web from achieving semantic interoperability is known as *semantic heterogeneity*. It occurs due to variation in used vocabulary; when different terms are used to describe the same thing or equal terms are used to describe different things, use of different languages, and use of abbreviation and so on. Figure 1.1 shows two simple heterogeneous ontologies describing the University domain. The two ontologies contain entities with different names (Faculty and School, NonAcademicStaff and AdminStaff, Professor and Prof.) having the same meaning. The concept 'Students' is included in both ontologies, but in the second ontology it is further grouped into PG_Students and UG_Students i.e the depth of coverage varies. There are also abbreviated terms; Professor abbreviated as Prof., AcademicStaff abbreviated as AcadStaff.

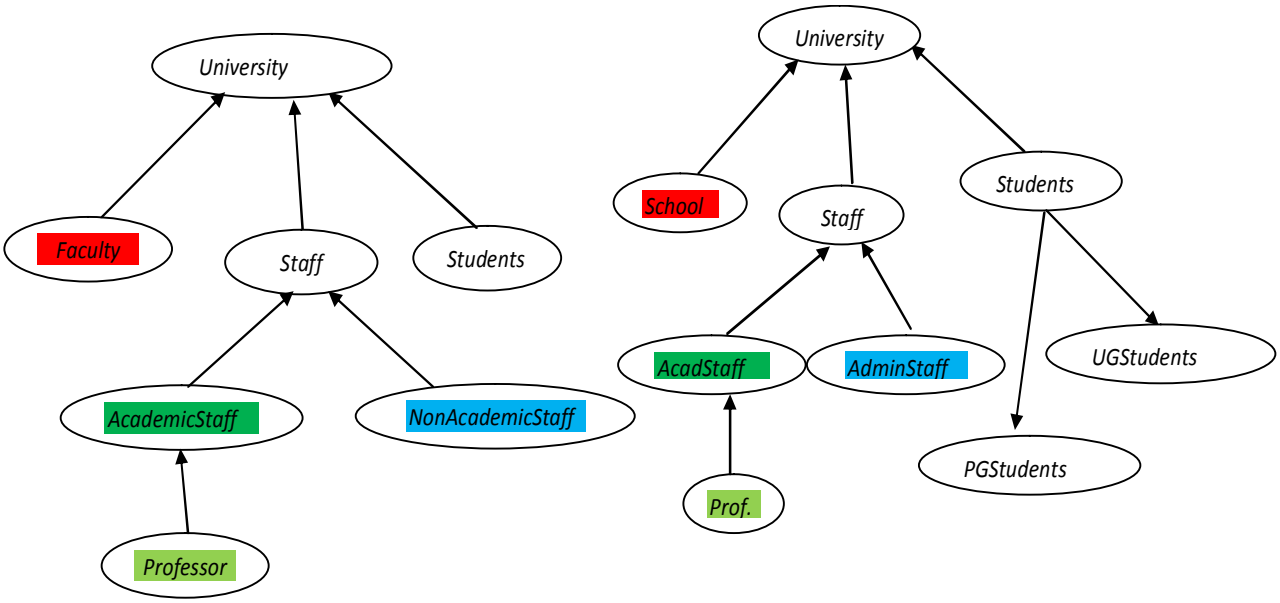


Figure 1.1: Sample of Heterogeneous Ontologies Describing University Domain
Similar terms are highlighted in same colours and arrows point to superclass.

The heterogeneity of ontologies mainly causes the problem of variation in meaning or ambiguity in entity interpretation and, consequently, it prevents information systems from sharing their own domain knowledge to the community. Therefore, without knowing the semantic mappings between entities of ontologies, information systems find it very difficult to understand and exchange information with one another.

Interoperability among different ontologies becomes essential to take full advantage of the semantic web. Semantic interoperability is necessary before multiple applications can truly understand data and treat it as information. This can only be achieved by resolving heterogeneity between ontologies.

One suggested solution is to develop a single ontology that covers an entire domain, but this is very hard or impossible. A widely accepted solution to the problem is *Ontology Alignment*.

Ontology alignment or matching tries to solve semantic heterogeneity problems by means of discovering a map between similar terms (entities) of two different ontologies of the same

domain. This enables people and software agents to work in a more smooth and collaborative way.

Ontology alignment process involves several steps which include feature engineering, searching and selection of matching candidates, similarity computation, etc. Manually handling of the overall process is highly tedious. Many fully automatic ontology alignment systems have been developed but the main problem is that the results provided by fully automatic techniques are rarely of sufficient quality. For many real-world datasets, fully automatic state-of-the-art tools still yield results of unsatisfactory quality for many applications (Paulheim *et. al*, 2013).

The challenges faced by fully automatic methods are manifold, including vocabulary differences (e.g., due to synonymy and homonymy), modeling differences (e.g., due to different model granularity or different attribute formats) and different points of view on the modeled reality (Granitzer, *et al.,* 2010). A matching system might align two equal terms with different interpretations.

Example1: Given two conference track ontologies to align. One of the ontologies focuses on documents published during conference, while the other ontology focuses on papers to be reviewed by a reviewer. Thus, a problem in aligning the two ontologies can be that an alignment system identifies equal concepts in both ontologies (e.g paper), but when manually comparing both ontologies the concept paper in the first ontology may convey different information content from that of the second ontology. The use of entities has significant impact on their interpretation; therefore, matching entities which are not meant to be used in the same context is often error-prone.

At the recent ontology alignment evaluation initiative (OAEI), the best fully automatic ontology matching tools have yielded a result quality of about 70% for single-language and

40% for multi-lingual matching tasks from the conference domain (Euzenat *et al.*, 2012, cited by; Paulheim *et al.,* 2013).

There is an upper bound to the quality of the alignment which is hard to exceed by fully automatic ontology matching tools, Ontology matching is "a very challenging problem for both man and machine" which calls for semi-automatic approaches combining the strengths of automatic matching algorithms and the expertise of domain experts in the matching process (Paulheim *et al.,* 2013).

Ontology alignment is still one of those problems that we cannot automate completely, and having a human in the loop might increase the quality of the results of machine-driven approaches (Sarasua *et al.,* 2012).

1.3 Thesis Goal

The goal of this research is to develop a semi-automatic ontology alignment system that combines the knowledge of domain expert and the matching ability of matchers to align ontologies using Falcon-AO (Hu and Qu, 2008) as base system. Falcon-AO is a fully automatic ontology alignment system that uses different strategies to align ontologies. The main aim here is to enhance the results produced by Falcon-AO.

1.4 Research Questions

This research aims at providing answers to the following questions:

1. In what ways can an expert assist an alignment system?
2. How can experts get easy access to entities in ontologies being aligned?
3. What is the performance evaluation of the user interactive system?

1.5 Research Aim and Objectives

The aim of this research was to design and implement Falcon-AO++ using Falcon-AO as base system. The objectives of this research were to:

1. Develop a framework for aligning ontologies with user assisted functionality
2. Extend Falcon-AO alignment system to adopt the user assisted functionality
3. Implement the extended Falcon-AO (Falcon-AO++)
4. Evaluate the two implementations (Falcon-AO and Falcon-AO++) using standard dataset and evaluation criteria.

1.6 Methodology

The following steps were adopted in order to provide answers to the aforementioned research questions.

1. Review of literature to understand the concept of ontology and ontology alignment
2. Review of different ontology alignment techniques and systems
3. Design a user assisted framework for aligning ontologies
4. Extend Falcon-AO with the user assisted framework
5. Evaluate the extended Falcon-AO using Ontology Alignment Evaluation Initiative (OAEI) Conference track dataset and reference alignment.

1.7 Research Contribution

The major contribution of this research is the development of a user assisted framework for ontology alignment. It explains the fact that human (domain expert) assistance can contribute to improving the effectiveness of ontology matching systems. Another contribution is the development and integration of an automatic entity retrieval method which automatically retrieves all entities of two given ontologies to provide easy access of the entities to domain experts.

CHAPTER TWO

LITERATURE REVIEW

This chapter discusses the basic concept of ontology including its structure and representation languages. It also provides explanation of ontology alignment and its application areas, different techniques used for alignment and the limitations of some of the techniques. Finally different ontology alignment systems are reviewed.

2.1 Introduction to Ontology

The term "ontology" originates from philosophy where it is defined as nature of existence. It is being used in computer science with a specific meaning different from that of its origin. Gruber (1993) defined the term as follows: "ontology is a formal, explicit specification of a shared conceptualization". In this definition, formal means the ontology should be machine-readable, explicit means the type of concepts used and constraints applied to them are clearly defined, shared means that an ontology captures consensual knowledge, i.e., it is not private to an individual, and conceptualization refers to an abstract model of some domain knowledge in the world that identifies that domain's relevant concepts (conceptualization models how people think about things of a particular subject area.). From this definition it can be said in simpler terms that ontology describes a domain by defining entities and relations between entities. Ontology specifies the meaning of the entities and relations relevant to an application domain by means of logical statements (expressed in Description Logics) (Morant, 2009). Automated reasoning tools can then exploit their semantics to make explicit information that was implicit in the ontology.

2.1.1 Structure of Ontology

As mentioned, ontology describes a domain by defining entities and relations that exist between the entities in that domain. In general, ontologies consist of the following components:

1. **Concepts:** A concept is also called a class. It is the major component of ontology. Classes are usually represented in form of hierarchy i.e having subclasses and belonging to a superclass. They are described by attributes.
2. **Instances:** An instance represents actual entity that exists in real world. Each instance is described by a class.
3. **Relations:** relations between concepts describe the concepts and their characteristics or the interaction between concepts or its properties
4. **Axioms:** Ontology includes a set of axioms to restrict the possible interpretations of its concepts. These axioms express the constraints that the concepts involved in those axioms must comply to. These axioms relate one concept with the other concepts of the ontology. A formal notation of ontology is as given below (Iroju *et al.*,2012):

   Ontology is a tuple;

   $$O = < C; A; I; R > \quad (2.1)$$

   Such that:

   $$C = \{c_1, c_2 \ldots c_k\} \quad (2.2)$$

   is the set of concepts.

   $$A = \{A(c_1), A(c_2), \ldots A(c_k)\} \quad (2.3)$$

With

$$A(c_1) = \{a_{11}, a_{12} \dots a_{1n}\} \quad (2.4)$$

being a set of attributes assigned to a concept c_1.

$$R = \{r_1, r_2 \dots r_n\} \quad (2.5)$$

R is a set of relations

$$I = \{i_1, i_2 \dots i_n\} \quad (2.6)$$

is a set of instances assigned to a concept C.

2.1.2 Benefits of developing an ontology

There are several reasons provided to justify the benefits of developing ontology. Basically, they can be summarized as follows (Julita, 2007):

1. To share common understanding of the domain and the related information among humans (the initial purpose) and among software agents.
2. To be able to reuse knowledge: how many times projects, developments and so forth start from scratch?
3. To separate domain knowledge from operational one.
4. To analyze domain knowledge: the thorough analysis usually encompassed in an ontology development, provides an excellent knowledge of the domain.

2.1.3 Ontology Languages

In the Semantic Web, software agents should be able to understand the content of web pages. Ontologies are one possibility of representing knowledge and correlations between different pieces of information. To be automatically processable for software agents, the knowledge has to be expressed as simple as possible in an adequate format.

For this purpose several languages have been developed. The most commonly used languages are:

2.1.3.1 RDF

RDF (Resource Description Framework): is a language used for describing resources recommended by the W3C. RDF uses a simple data model which is understandable for humans and machines, the RDF Model. The information is presented as a statement, which always consists of three parts: subject, predicate and object. The predicate represents a binary relation between the subject and the object (Antonio and Hermelen, 2008).

2.1.3.2 RDFS

RDFS (Resource Description Framework Schema): RDF is only usable for describing resource and relations between them; it is not possible to create new properties and classes or to create taxonomy. For this reason, an extension of RDF, RDF Schema (RDFS), has been developed which allows the definition of classes. A class denotes a group of resources; in other words, a resource is an instance of a class (Antonio and Hermelen, 2008).

2.1.3.3 OWL

OWL (Web Ontology Language) is another extension of RDFS that provides additional vocabulary to describe more semantics. It incorporates some features that are missing in RDF/RDFS such as: Range restriction of properties cannot be applied to some classes in RDFS e.g we cannot say paper is reviewed only by a reviewer. Disjointness between classes can be specified in OWL e.g it can be said that 'Author' and 'Reviewer' are disjoint though they belong to same superclass 'Person' while in RDFS it can only be stated that Author is a

subclass of Person. Cardinality Restrictions on properties can be expressed using OWL, for example stating that 'Paper' must have only one 'title' (Antonio and Hermelen, 2008).

2.1.4 Ontology interoperability techniques

Ontologies provide the best way to describe and share data among different organizations. Interoperability between ontologies involves the following processes (Amrouch and Mostefai, 2012):

Ontology Mapping (Matching): defines a formal expression that describes a semantic relationship between two (or more) concepts belonging to two (or more) different ontologies.

Ontology Alignment: describes a semantic bridge between two (or more) ontologies through a set of matches between them. The source ontologies have to describe the same domain or related domains. These matches are called "*Mappings*". In summary, Mapping is the process of aligning two ontologies; a similar process used in data base schema is called "*Matching*".

Ontology Merging: once the mappings between the two ontologies are identified, the mapped or aligned concepts are merged into a single one.

2.2 Ontology Alignment

Ontology alignment, or **ontology matching**, the two terms are being used interchangeably in most cases. The ontology alignment problem can be described in one sentence: given two ontologies which describe each 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 (Euzenat, *et al.*, 2004).

An alignment is a set of correspondences between entities belonging to the matched ontologies. Alignments can be of various cardinalities: 1:1 (one-to-one), 1:m (one-to-many), n:1 (many-to-one) or n:m (many-to-many).

Given two ontologies, a correspondence is a 4-tuple:

, (Shvaiko and Euzenat, 2013)

such that:

id: is a unique identifier of the given correspondence;

e_1 and e_2 are entities (e.g. properties, classes) of the first and the second ontology, respectively

r: is a relation (e.g., equivalence (=), more general , disjointness ( ),holding between e_1 and e_2.

The correspondence asserts that the relation *r* holds between the ontology entities e_1 and e_2. A confidence is frequently used in the correspondence typically in the [0, 1] range. The higher the confidence, the higher is the likelihood of the relation to hold. A graphical illustration of ontology alignment is shown in Figure 2.1 with mappings shown in red.

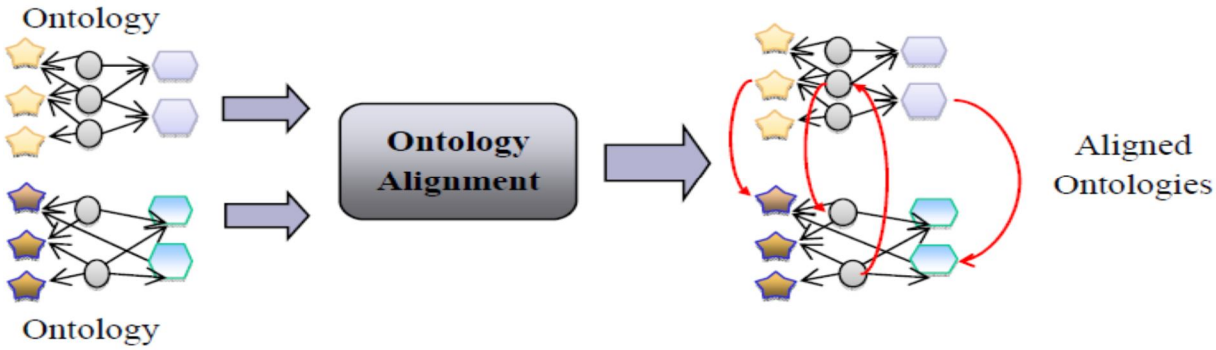


Figure 2.1: An illustration of ontology alignment (Granitzer *et al., 2010*)

2.2.1 Applications of Ontology Alignment

Ontology alignment is an important operation in different applications. Some of the applications include (Shvaico and Euzenat, 2005):

Information Integration: a very important application scenario for ontology alignment systems is information integration. When many heterogeneous information sources are needed to perform a common task such as query answering, the sources have to be matched and integrated. A matching system produces the connections between the sources by finding similar terms, concepts or attributes which provide the basis for the integration process. This is to ensure semantic interoperability among the systems to be integrated.

Agent Communication: Agents are pieces of software characterized by autonomy and capacity of interaction. An agent on the semantic web receives tasks from users, seeks information from different web sources, communicates with other agents, compares information with user requirement, makes some choices and then gives answers to the users. Semantic web agents make use of ontologies to assist in web searches, interpretation of retrieved information, and communication with other agents. Since agents often refer to some ontology, two autonomous and independently designed agents have the possibility to communicate but only little chance to understand each other if they do not share the same content language and ontology. Ontology alignment therefore helps to provide the possibility for software agents to match their ontologies in order to either translate their messages or bridge axioms in their own models.

Web Service Integration: Web services are processes that expose their interface to the web so that users can invoke them. Semantic web services provide a richer and more precise way to describe the services through the use of knowledge representation languages and ontologies. Web service discovery and integration is the process of finding a web service

able to deliver a particular service and composing several services in order to achieve a particular goal. However, semantic web services descriptions have no reasons to be expressed by reference to exactly the same ontologies. Henceforth, both for finding the adequate service and for interfacing services it will be necessary to establish the correspondences between the terms of the descriptions. This can be provided through matching the corresponding ontologies

Query Answering on the Web: The answering of queries is an important issue for the web. Search engines are used very often and they work with ontologies to refine queries. Each question that is posed by a user is translated into terms of the local ontology which is done by a matching process. Meta search engines translate a query into the terms of the different ontologies that are provided by the underlying search engines, collect the results and translate them back according to the original ontology.

The need for matching is not limited to one particular application; it exists in any application involving more than one party. Thus, it is reasonable to consider ontology matching as a unified object of study. However, there are notable differences in the way these applications use matching (Gal and Shvaico, 2008).

2.3 Ontology matching Techniques

Different matching techniques are being applied by matchers in an alignment system to calculate similarities between the terms (entities) from the different source ontologies. The techniques can be generally classified into two (Shvaico and Euzenat , 2005):

1. *Element level* matching techniques: Element-level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with other entities.

2. *Structure-level* techniques compute mapping elements by analyzing how entities appear together in a structure.

2.3.1 Element level techniques

Techniques in this category include:

1. String based techniques
2. Language based techniques
3. Constraint based techniques
4. Linguistic Resources
5. Alignment Reuse
6. Upper-level Dolce

2.3.1.1 String based techniques

These techniques consider names of entities as strings (sequences of characters). The intuition is that, the more similar the strings, the more likely they denote the same concepts. Some examples of string-based techniques which are mostly used in matching systems are ((Shvaico and Euzenat , 2005) :

1. **Prefix:** takes as input two strings and checks whether the first string starts with the second one (e.g., chair and chairman)
2. **Suffix:** takes as input two strings and checks whether the first string ends with the second one (e.g., phone and telephone),
3. **Edit distance:** takes as input two strings and computes the edit distance between the strings. That is, the number of *insertions*, *deletions*, and *substitutions* of characters required to transform one string into another, normalized by the length of the longest string.etc

2.3.1.2 Language based Techniques

These techniques do not consider names as strings but as words of a natural language. The techniques involve the following (Shvaico and Euzenat , 2005):

1. Tokenization: where Names of entities are parsed into sequences of *tokens* by a tokenizer which recognizes punctuation, cases, blank characters, digits, etc. (e.g, Conference_Events → Confrence, Events)
2. Lemmatization: where tokens are analyzed in order to find their entire possible basic forms i.e singular instead of plural (e.g, Events →Event).
3. Elimination of stop words: where tokens that are articles, prepositions, conjunctions, etc are discarded. The mentioned language based techniques are applied to names of entities before running string based techniques in order to improve result.

2.3.1.3 Constraint based Techniques

These techniques consider internal constraints applied to the definitions of entities such as data types, cardinality of attributes, sub concepts or keys. The similarity of data types or the cardinality of attributes (of entities) determines similarity between the entities.

2.3.1.4 Linguistic Resource

In this situation names of entities are considered as words of natural language, thus external resources such as common knowledge or domain specific thesauri are used in order to match words based on linguistic relations (synonyms, hyponyms and so on) between them (Shvaico and Euzenat , 2005).

2.3.1.5 Alignment Reuse

This also involves using an external resource which contains alignments of previously matched ontologies. The idea here is that ontologies to be matched may be similar to already matched ones. This can speed up alignment process.

2.3.1.6 Upper level formal ontologies

These are also external resources. Examples of such ontologies include: Suggested Upper Merged Ontology (SUMO) and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE).

2.3.2 Structure level Techniques

Techniques that consider the structure of ontology include:

1. Graph-based Techniques
2. Taxonomy-based techniques
3. Model-based techniques

2.3.2.1 Graph-based Techniques

These techniques use graph algorithms. The input ontologies are considered as labelled graphs. The ontologies are viewed as graph-like structures having entities and their inter-relationships. Usually, the similarity comparison between a pair of nodes from the two ontologies is based on the analysis of their positions within the graphs. The intuition behind is that, if two nodes from two ontologies are similar, their neighbours might also be somehow similar. Examples of matchers with such intuition are: graph matching, children, leaves, and relation.

2.3.2.2 Taxonomy-based techniques

These techniques also use graph structure but only specialization relation i.e. through "is-a" link is considered. The intuition behind taxonomic techniques is that *is-a* links connect terms that are already similar (being a subset or superset of each other), therefore their neighbours might be also somehow similar.

2.3.2.3 Model-based techniques

These techniques make use of the semantic interpretation of the input ontologies. Major techniques used are the propositional Satisfiability (SAT) description logics reasoning techniques.

2.3.3 Other Matching Techniques

Other forms of ontology matching techniques include (Iroju *et al.*, 2012):

2.3.3.1 Rule-Based Techniques

Rule-based techniques work with schema-level information, such as entity names, data types and structures. Some examples of rules are that two entities match if their names are similar or if they have the same number of neighbor entities.

2.3.3.2 Learning-based Techniques

Learning-based approaches can be used at both the instance-level as well as the schema level. The learning based approaches perform ontology matching by comparing value formats and distributions of data instances underlying the entities under consideration.

2.3.3.3 Instance-Based Techniques

Instance based approaches are considered whenever the schemas to be matched use different terms to describe the same real-world-concept. Instance-based techniques can be used to improve the effectiveness of the schema based approaches. Instance based approaches are classified into two. These include the linguistic and the constraint-based methods. Linguistic-based approaches use information retrieval techniques to gain information about word frequencies and combination of words. The use of value ranges, data types or the appearances of character patterns characterize the constraint-based approaches.

2.4 Limitations of Some Matching Techniques

1. **String-based techniques** rely only on entities' name (label) to compute similarity. It allows the successful matching of entities with similar strings. The limitations of this technique are that homonymous entities (different entities described by same name) are detected as equivalent, while synonymous entities (the same entities described by different names) cannot be detected. Again the techniques perform poorly on complex strings such as sentences or descriptions (Granitzer *et al.,* 2010).
2. **Language-based techniques** try to improve on some limitations of string-based techniques by considering names of entities as words of natural language not as string. Techniques used include: tokenization, elimination etc. though the techniques show improvement, but still do not match concepts on semantic level, and will fail when, for example, correct handling of synonyms or homonyms is required (Granitzer *et al.,* 2010).

3. **Linguistic resources:** perform matching on a semantic level, as opposed to matching on a syntactic level. Linguistic resources used in a matching process include: Wordnet or domain thesauri. It improves matching quality because relations such as synonyms, antonyms, etc are detected and also relationships such as equivalence and generalization are recognized. Its major drawback is that domain specific thesauri will be required for specific application domains. Again, thesauri for languages other than English are often of lesser quality or not available at all (Granitzer *et al.,* 2010).

2.5 Review of Ontology Alignment Systems

1. ONION (ONtology compositION system)

ONION (Choi *et al.,* 2006) resolves terminological heterogeneity in ontologies and produces articulation rules for mappings. The linguistic matcher identifies all possible pairs of terms in ontologies and assigns a similarity score to each pair. If the similarity score is above the threshold, then the match is accepted and an articulation rule is generated. After the matches generated by a linguistic matcher are available, a structure-based matcher looks for further matches. An inference-based matcher generates matches based on rules available with ontologies or any seed rules provided by experts. The major drawback of this system is that the linguistic matcher fails when semantics should be considered.

2. AOAS (Anatomical Ontology Alignment System)

AOAS (Zhang *et al.,* 2007) (Jean-Mary *et al.,* 2009) is a domain-specific ontology matching system developed by the U.S. National Library of Medicine specifically to investigate the alignment of anatomical ontologies, uses the concept of "anchors" and implements a structural validation that seeks to find correspondences in relationships between anchors. Its lexical component compares concept names using a model of lexical resemblance developed

for biomedical terms and exploits additional synonyms from an external resource: the Unified Medical Language System (UMLS). The presence of shared hierarchical paths among concepts across ontologies is then used as positive evidence for the mappings identified lexically.

The anatomy track of OAEI proved that domain specific alignment systems perform better than domain independent systems. AOAS system performed the best on the anatomy track on the OAEI 2007 but did not produce result in any other track as the system was fine tuned for the medical domain (Vargas-Vera and Nagy, 2010). Another drawback is that AOAS only identifies mappings between concepts, Falcon-AO++ finds mapping both between concepts and relations.

3. **CIDER** (Context and Inference baseD alignER)

CIDER (Gracia and Mena, 2008) is a system for ontology alignment that performs semantic similarity computations between entities of two ontologies expressed in OWL format. It extracts the ontological context of each ontology term, up to a certain depth. That is (depending on the type of term), their synonyms, textual descriptions, hypernyms, hyponyms, properties, domains, roles, associated concepts, etc. This process is enriched by applying a lightweight inference mechanism, this helps add more semantic information that is not explicit in the asserted ontologies. Several similarities between each pair of compared terms are computed: Linguistic similarity between the labels of terms (based on Levenhstein similarity) as well as structural similarities, by exploiting the ontological context of the terms and using vector space modelling in the comparisons. This comprises comparison of taxonomies and relationships among terms. In comparison, this system and the system implemented in this resarch are similar because they both do not rely on instances to

perform ontology matching. A major difference is that CIDER works well with small and medium sized ontologies but not with large ones while Falcon-AO++ has a good way of handling large ontologies.

4. RiMOM (*Risk Minimization based Ontology Mapping)*

RiMOM (Saruladha *et al.,* 2011) is an automatic ontology alignment system developed with dynamic selection of matchers for ontology alignment tasks. The input ontology should be in OWL format. It considers lexical, structural and instance similarities. Appropriate matchers for the matching tasks are chosen based on the features of the input ontology. RiMOM consist of six steps as follows:

Ontology Preprocessing and Feature Factors Estimation: generate the features (e.g name, label, children, etc) of the entities from both ontologies.

Matcher Selection: if two ontologies have some feature in common, then matcher based on these feature information are employed, and if some feature factors are two low, then these matcher may not be employed. The entities are first linguistically matched; structural matching is only applied if the schemas exhibit sufficient structural similarity.

Single strategy execution: The selected matchers are executed to find alignment independently.

Alignment combination: RiMOM combines the alignment results obtained by the selected strategies.

Similarity propagation (optional): If the two ontologies have high structure similarity factor, RiMOM finds new alignment according to the structural information.

Alignment refinement: refines the generated alignment results from the previous steps. Several heuristic rules are applied to remove the unreliable alignments.

The drawback of RiMOM is its inefficiency in handling large scale ontologies. Though it shows a very good effectiveness for large scale ontologies, it consumes long time and large amount of memory since it does not incorporate search space reduction techniques. Falcon-AO++ uses partitioning technique for large scale ontologies.

5. OMReasoner

OMReasoner (Shen *et al.,* 2011) is an ontology alignment system designed with an extensible framework for combination of multiple individual matchers, and reasons about ontology matching. It handles ontology matching in semantic level and makes full use of the semantic part instead of structure. OMReasoner performs semantic matching based on external dictionary: WordNet to achieve semantic correspondences. Still, this approach includes string matcher which computes the similarity between entities based on syntax. The results provided by the different matchers are first normalised then combined.The system also employs DL (Discription Logic) reasoner provided by Jena and also includes external rules to reason about the ontology matching.

The system does not consider structural information among ontologies. It also does not consider comment and label information which is very useful especially when name of concept is meaningless. Falcon-AO++ considers both structural, comment and label information.

6. AgreementMaker

AgreementMaker (Shvaiko and Euzenat, 2013) is an alignment system that comprises of a range of automatic matchers, an extensible and modular architecture, a multi-purpose user interface, a set of evaluation strategies. It handles ontologies in XML, RDFS, OWL, N3 and outputs 1:1, 1: m, n:1, n:m alignments. In general, the matching process is organized into two modules: similarity computation and alignment selection. The system combines

matchers using three layers; **First layer** compares concept features, such as labels, comments, instances, etc string based techniques such as edit distance, are used. **Second layer** uses structural ontology properties to obtain similarities. **Third layer** combines the results computed by the first two layers. Because of the combination of different techniques and matching layers, agreementMaker gives good result. A major weakness is that all the concepts used in the AGREEMENTMAKER are common strategies which are there in ontology matching. What the size is and what happens if the size of the ontology is very big is not mentioned (Muzaffar and Srivatsa, 2012). AgreementMaker does not have special technique for large ontologies.

7. Falcon-AO (Falcon- Aligning Ontologies)

Falcon-AO (Jian *et al.,* 2005) is a fully automatic ontology alignment system which aims at finding alignments between web ontologies that are expressed in RDFS and OWL format. Falcon-AO has five basic components (see Figure 2.2); *Model pool*: parses and creates model for received ontologies*, Matcher library*: manages a collection of matchers. There are four matchers incorporated; *V-doc*, *I-Sub*, *PBM*, and *GMO*, *Alignment Set*: Generates alignments and also evaluates the generated alignments against a reference alignment. *Central Controller*: executes matchers and performs similarity combination based on linguistic and structural comparability. *Repository*: stores any reusable data for the matching process.

The system first parses input ontologies by *Jena* and creates model (an internal representation). The matchers are then executed and their results are combined using similarity combination method. Falcon-AO uses both structural and string techniques and also considers comment and label information. It has a special method for large ontologies based on divide and conquer approach (Hu and Qu, 2008).

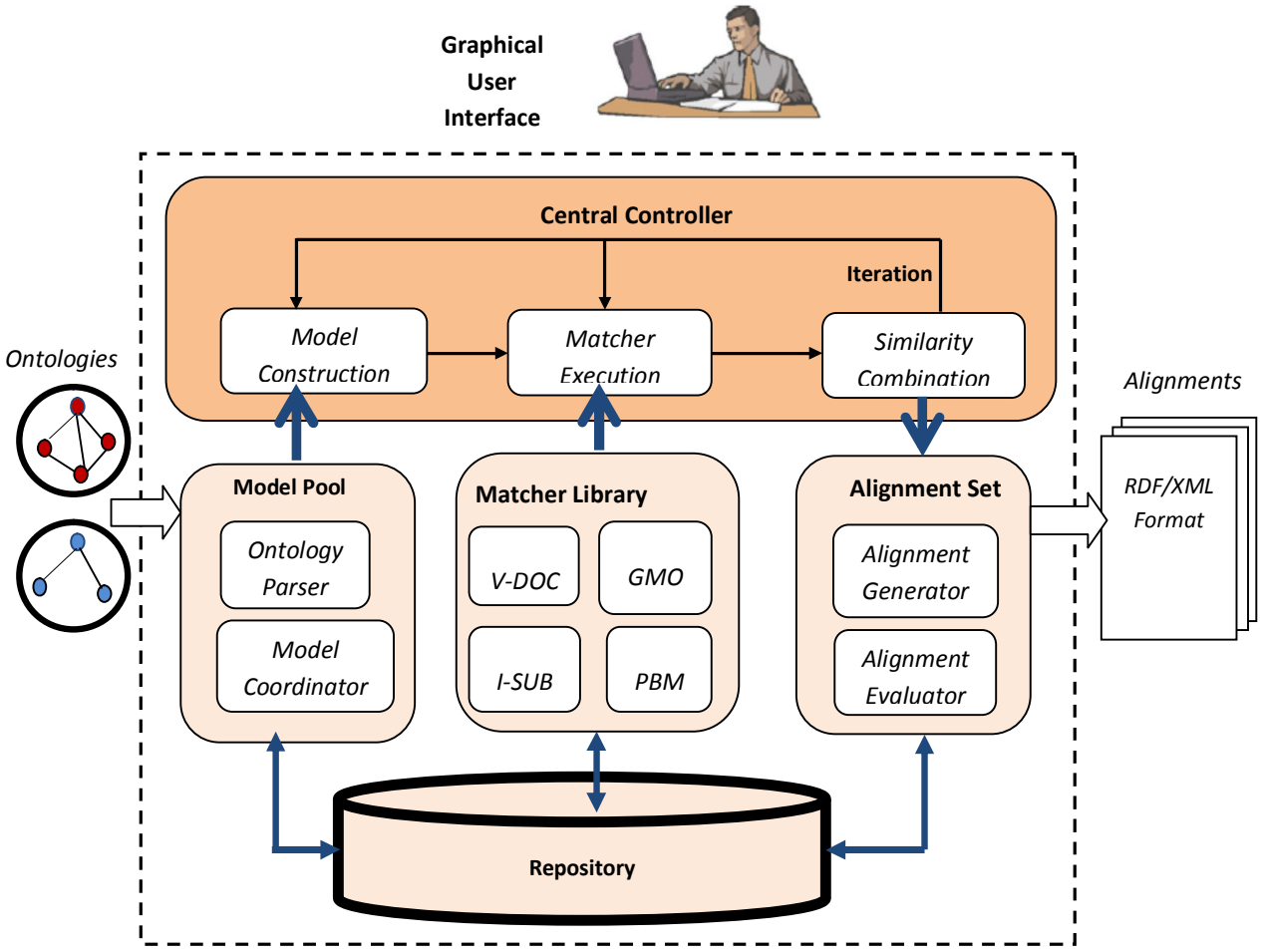


Figure 2.2: Architecture of Falcon-AO (Hu and Qu, 2008)

Limitations of Falcon-AO

Falcon-AO works well with ontologies that are having much lexical and structural similarity. The following weaknesses were observed:

1. When the structures of given ontologies are not similar, GMO and V-DOC cannot easily detect alignments because they both exploit the graph structure. As a result, some alignments are lost.
2. The system accepts results with high similarities from a string matcher. In some cases, same words may have different information content; therefore results provided by string matchers are not always reliable regardless of the similarity value.

In this research, Falcon-AO was extended with user assisted strategy so that domain experts can assist in the matching process by providing some input information. Falcon-AO was chosen because it is an open source system. A summary of reviewed system is given in Table 2.1.

Table 2.1: Summary of reviewed systems

Alignment System	**Input Language supported**	**Output Mapping**	**Matching techniques**	**Limitations**
RiMOM	OWL	1:1	String technique, structural technique.	Incompetent in dealing with large scale ontologies.
AOAS	OWL	1:1	String and structural based techniques, external resource; UMLS	Domain specific, considers only name of concepts.
CIDER	OWL	1:1	String based technique and thesauri look-up, taxonomic structural technique.	Works well only when matching small or medium size ontologies.
OMReasoner	OWL	1:1	String based technique and thesauri look-up; Wordnet.	Does not consider structural information. Also does not consider comments and labels.
AgreementMaker	RDF/RDFS OWL, N3	n:m	String and structural techniques	No specific technique for large ontologies
ONION	-		String and structural technique, WordNet.	The linguistic matcher fails when semantics should be considered
Falcon-AO	RDFS/OWL	1:1	String and structural. Partitioning technique for large ontologies.	Alignments are lost when structures are not similar, accepts matching with high similarity from string mathcer

2.5 Summary

In this chapter, an overview of the basic concepts of ontology and ontology alignment was presented. The chapter also presented different matching techniques employed by alignment systems which can be classified into *element level techniques* and *structure level techniques*. Element level techniques consider the names of entities ignoring the structure; string techniques, language techniques and linguistic resources fall in this category. Structure level techniques consider the structure of ontologies. Different alignment systems were reviewed considering both their strengths and weaknesses. Among the reviewed systems, AOAS is domain specific. Other systems are domain independent; Falcon-AO, OMReasoner, RiMOM, AgreementMaker, ONION. Among the reviewed systems Falcon-AO considers a special technique for aligning large ontologies. The architecture of the base system (Falcon-AO) was also presented.

CHAPTER THREE

DEVELOPMENT OF FALCON-AO++

This chapter gives detailed description of the system, Falcon-AO++. It illustrates major components of the system and how they interact with one another. It also describes the different matchers integrated in the system and how the system combines results obtained by the matchers. Other tools used are also discussed.

3.1 System Requirement

When given two ontologies, an alignment system tries to get a set of matches between the entities (classes, properties etc) of the given ontologies in term of a relationship (only equivalence relation is considered here) based on a degree of confidence in the range [0,1]. In this system (Falcon-AO++), user input is considered to be highly reliable with a confidence of 1.0. Thus, the requirement for any alignment system is to accept input ontologies and discover correspondences between the ontologies by applying different matching techniques. Performing ontology alignment with fully automatic systems does not provide accurate results in most cases.

The main requirement was to develop an ontology alignment system which combines the matching ability of matchers and the knowledge of a domain expert using Falcon-AO as base system. Thus, the hardware and software requirements for the system are given as follows:

Hardware Requirement

1. Intel (R) Core (TM) i3 CPU
2. 4 GB RAM
3. Hard disk space: 400 GB

Software Requirement

1. Windows 7 Operating System
2. JDK 1.6: The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets.
3. Java NetBeans IDE 7.2: An integrated development environment (IDE) is a software application which consists of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder.
4. Jena API (detail of Jena is given in section 3.3)

3.2 Architecture of Falcon-AO++

Falcon-AO++, extends Falcon-AO with user input component which allows domain experts to assist the system by providing some input information. The system Architecture (Figure 3.1) is therefore an extended architecture of Falcon-AO. The integrated component is highlighted in green. The detail of user input and all other components are provided in subsequent sections.

3.2.1 User Input

Integrating user input strategy within an alignment system assists in improving alignment result. Users can provide input in different ways. Some of the ways include:

1. Providing some input information to the system such as example mappings.

2. Providing feedback; given a set of alignment generated by a system, user provides feedback by accepting correct alignments and rejecting the incorrect ones.

3. Selecting appropriate matchers to be executed depending on input ontologies

The user input strategy is based on the first option. A user provides input information by selecting different pairs of entities (entities in each pair are selected from the two input

ontologies) and specifying some restrictions between the entities by selecting a constraint from a list of provided constraints.

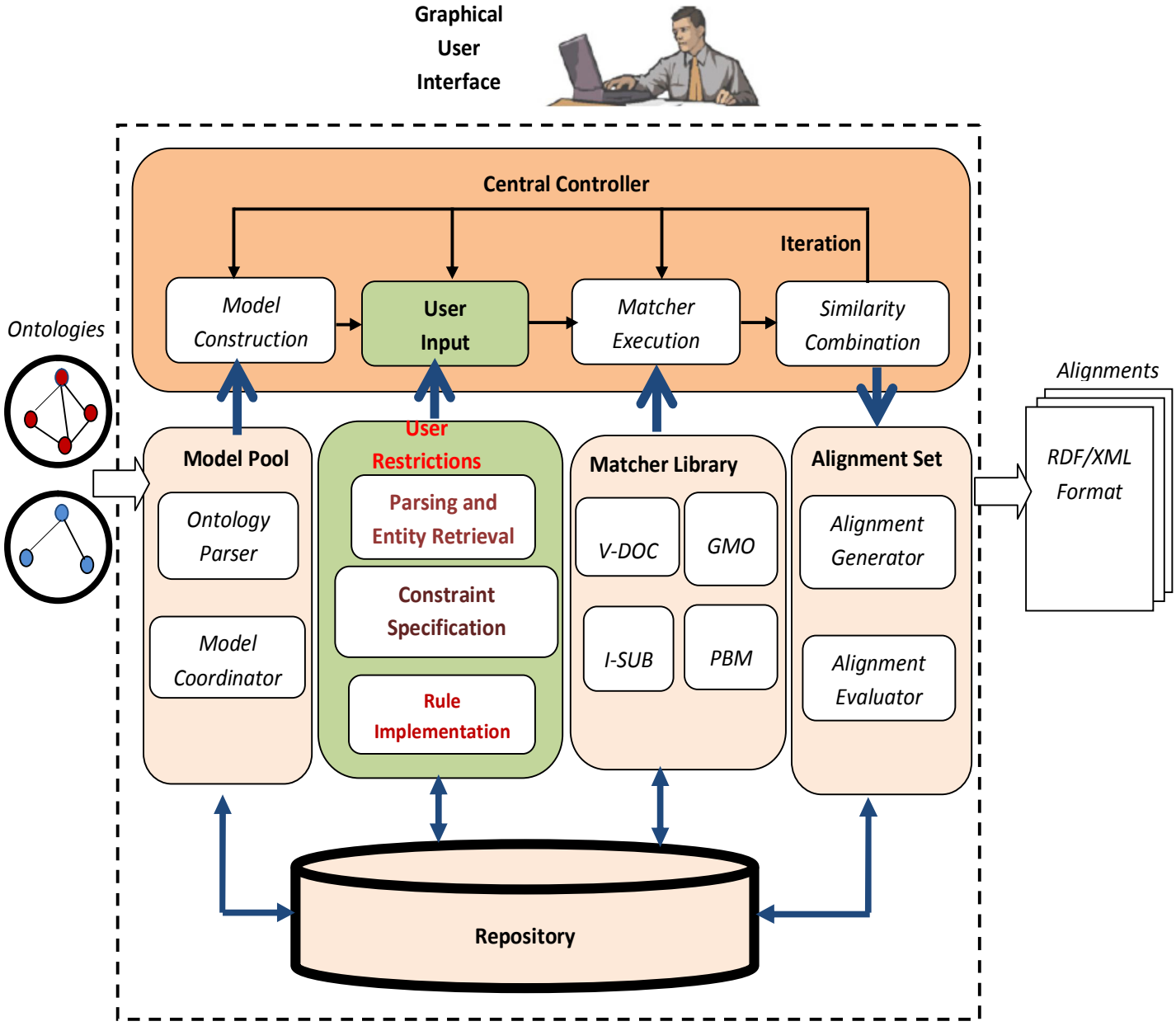


Figure 3.1 Architecture of Falcon-AO++

3.2.2 User Restrictions

User input is generated by this component. It allows a user to select two entities and add a constraint to define the relation that exists between the two entities. A user restriction is based on the following:

1. Parsing and Entity Retrieval
2. Constraint Specification
3. Rule Implementation

3.2.2.1 Parsing and Entity Retrieval

This component parses input ontologies and automatically retrieves all domain entities contained in the input ontologies. Retrieved entities are displayed on the GUI to provide easy means for users to select pairs of entities and add constraint

3.2.2.2 Constraint Specification

The Constraint specification allows a user to select a constraint for each selected pair of entities. The constraint defines the kind of relation that exists between the entities. User input is processed based on selected constraint. Two constraints have been implemented; Equivalent and Disjoint Constraints.

1. Equivalent Constraint

The equivalent constraint allows the selected entities to be seen as equivalent. When this constraint is selected by the user, a rule automatically defines the two entities as *SameAs* and stores in a separate and empty ontology file. By reading the content of the file the system can be able to recognize user input. The idea is that all possible alignment lost can be minimized by this constraint.

2. Disjoint Constraint

This constraint does not imply that the system would return a disjoint relation as part of alignment result. By selecting disjoint constraint, the two entities are seen as entirely different even if their strings are exactly the same. The constraint takes care of probable error alignments.

3.2.2.3 Rule Implementation

The *rule implementation* implements a rule based on selected constraints. The rule is generated by a *Generic Rule Reasoner* provided by *Jena*.

3.2.2.4 User Input Process Flow Diagram

The flow diagram of Figure 3.2 summarizes the entire development process of the user input functionality. The *parsing and entity retrieval process* is first invoked to parse input ontologies and retrieve domain entities from the ontologies. All retrieved entities are presented on the GUI. During *entity selection*, a user selects pairs of entities and a constraint (constraint specification) for each pair. In the *class creation* and *rule specification process*, each selected entity is added as a class in a temporary ontology file, a rule is generated based on selected constraint; if the constraint is equivalent (=) the entities are defined as *SameAs,* if it is disjoint (!=), the entities are defined to be *differrentFrom*. User specified matched entity stores user input in a data structure which is of the same type as Falcon-AO's data structure. The two data structures are finally merged together before final alignment.

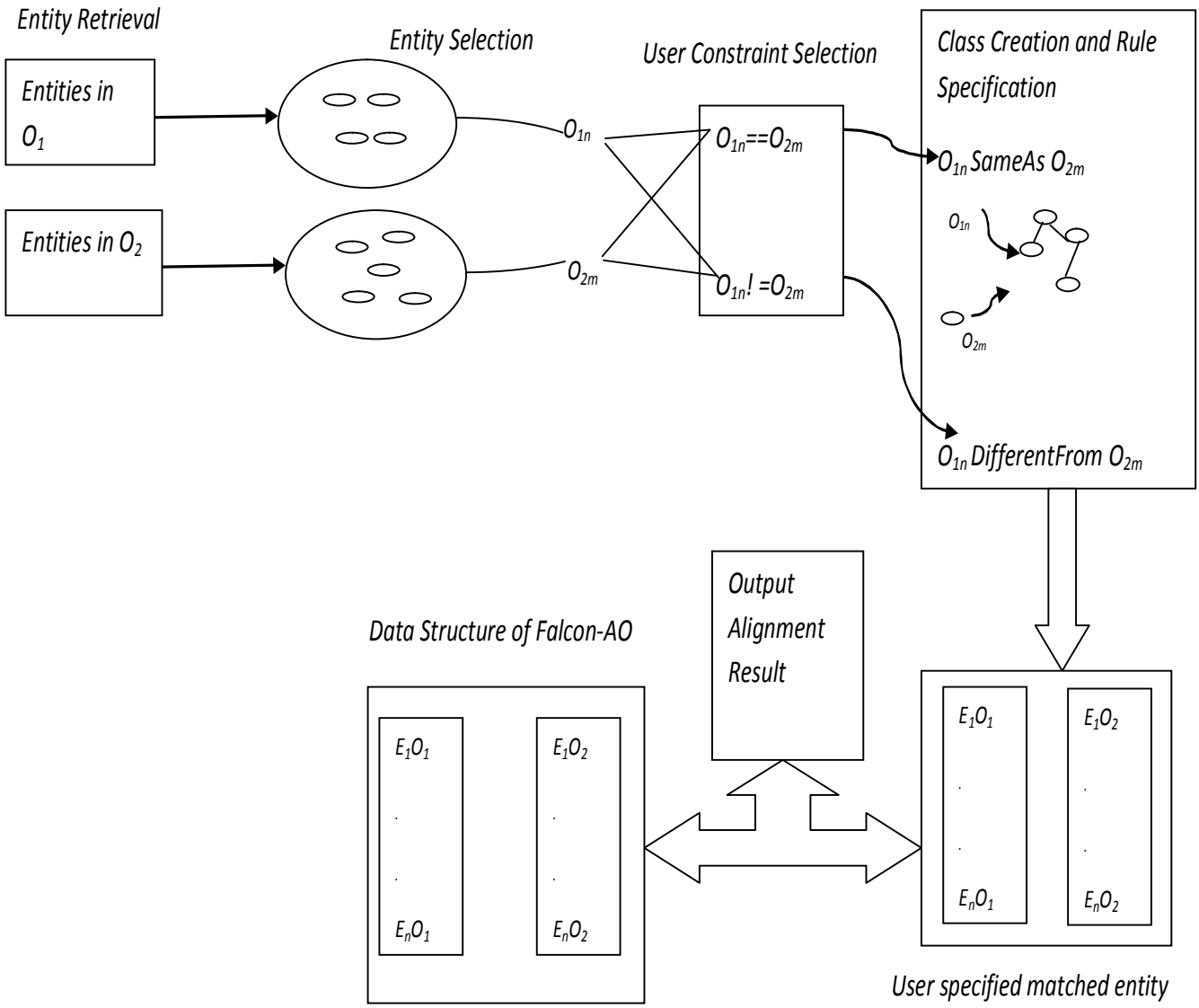


Figure 3.2: Process Flow Diagram

3.2.3 The Matchers of Falcon-AO++

Falcon-AO++ has four matchers within its matcher library; *V-Doc* and *I-Sub* are based on linguistic matching, *GMO* is a structural matcher, *PBM* uses a partitioning strategy to find alignments in large ontologies.

3.2.3.1 PBM -Partition Block Matching

PBM (Hu *et al.,* 2008) is executed only if any or both the ontologies to be matched are large. The specified size for large ontology is ≥ 5000 statements. The main approach is

based on the divide-and-conquer philosophy. An illustration of the approach is shown in Figure 3.3. Specifically, the input is two large ontologies to be matched. **Phase 1;**The approach starts by partitioning the entities (i.e., classes and properties) of each ontology into a set of small clusters by measuring their structural proximities, and then constructs blocks via assigning RDF statements. **Phase 2;** Those blocks are matched based on pre-calculated anchors (i.e., matched entities found by string matcher), and block mappings (i.e., matched blocks with high similarities) are selected. **Phase3;** Entities in mapped blocks are further matched by two powerful matchers, V-DOC and GMO for outputting final alignments.

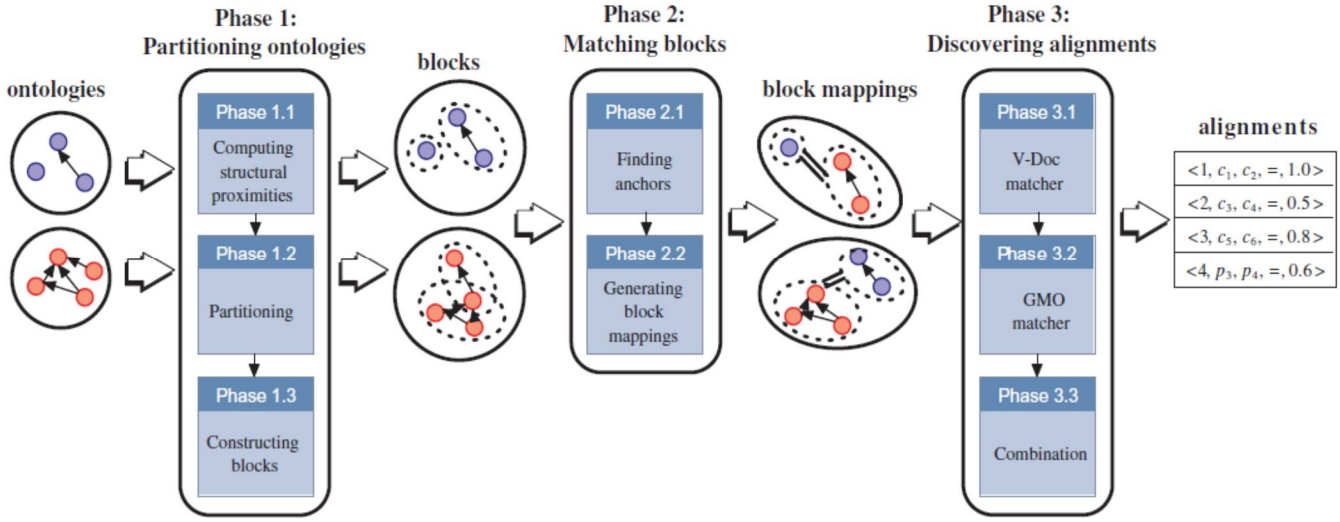


Figure 3.3: PBM Process (Hu *et al*., 2008)

Phase1: Partitioning: The portioning process starts by getting all the entities (Classes and Properties) in given ontologies. The process takes place in the following steps:

Phase1.1: After getting all the entities, structural proximities i.e the hierarchical distance between the entities is computed. Structural proximities between classes are defined based on how closely they are related in the hierarchies of the rdfs:sub-ClassOf relationships. For example, let C_i, C_j be two classes in a given ontology O, the structural proximity between C_i and C_j is (Hu *et.al,* 2008)

$$\mathrm{Prox}(\mathrm{C_i}, \mathrm{C_j}) = \frac{2*\mathrm{depth}(\mathrm{C_{ij}})}{\mathrm{depth}(\mathrm{C_i})+\mathrm{depth}(\mathrm{C_j})} \quad (3.1)$$

where;

C_{ij} is the common superclass of C_i and C_j , $depth(C_i)$ and $depth(C_j)$ are the depths of C_i and C_j in the graph respectively.

Example: Consider the ontologies Figure 3.4 (describing part of conference domain), if we assume that the depth of Participant in O_1 is 1, the proximity between Author and Chairman is

$$Prox(Author, Chairman) = \frac{2*1}{2+2} = 0.5$$

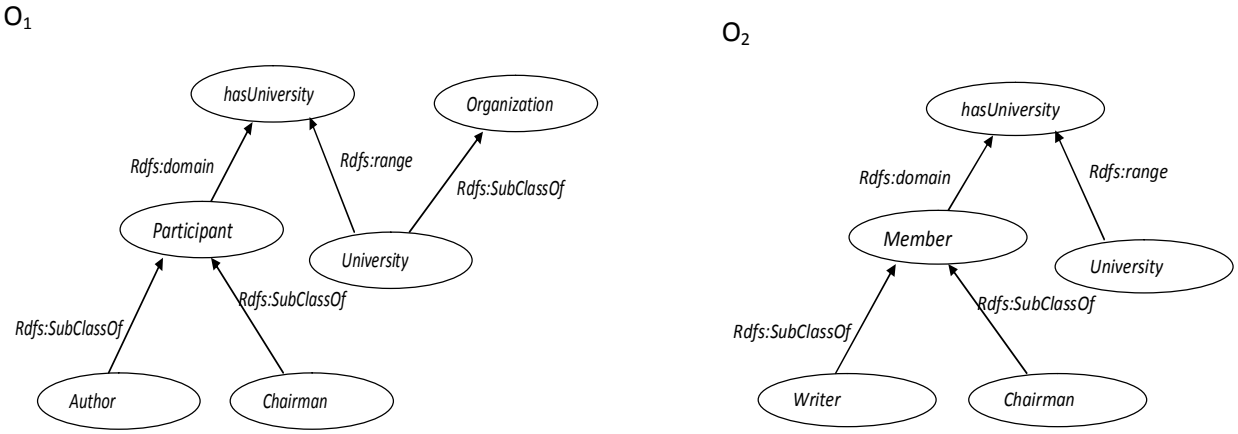


Figure 3.4: Sample Ontologies

The proximities between the classes in O_1 are given as:

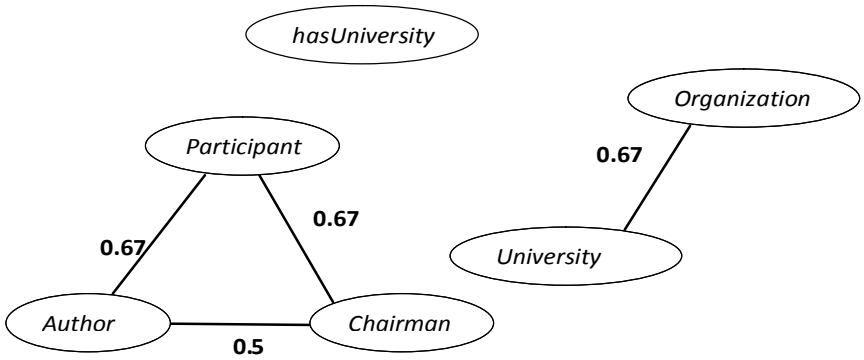


Figure 3.5: Structural Proximities of O_1

Phase1.2: Entities are grouped into a set of disjoint clusters based on their structural proximities.

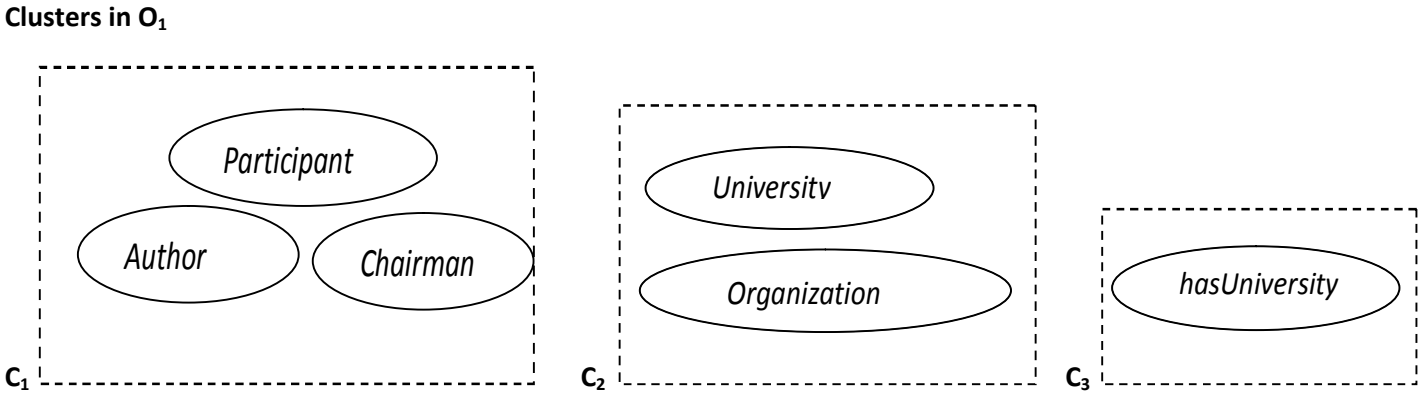


Figure 3.6: Sample of Clusters in O_1

Phase1.3: All the entities in a given ontology are partitioned into a set of disjoint clusters. But the RDF triples connecting the entities are missing and hence cannot be used for other matching approaches. At these step, existing triples within the cluster are recovered i.e blocks are constructed from the clusters by assigning each RDF triple to a cluster in which at least two entities are contained. By recovering the triples that exist between the entities in C_1 and C_2 of Figure3.5 above, the following blocks can be obtained:

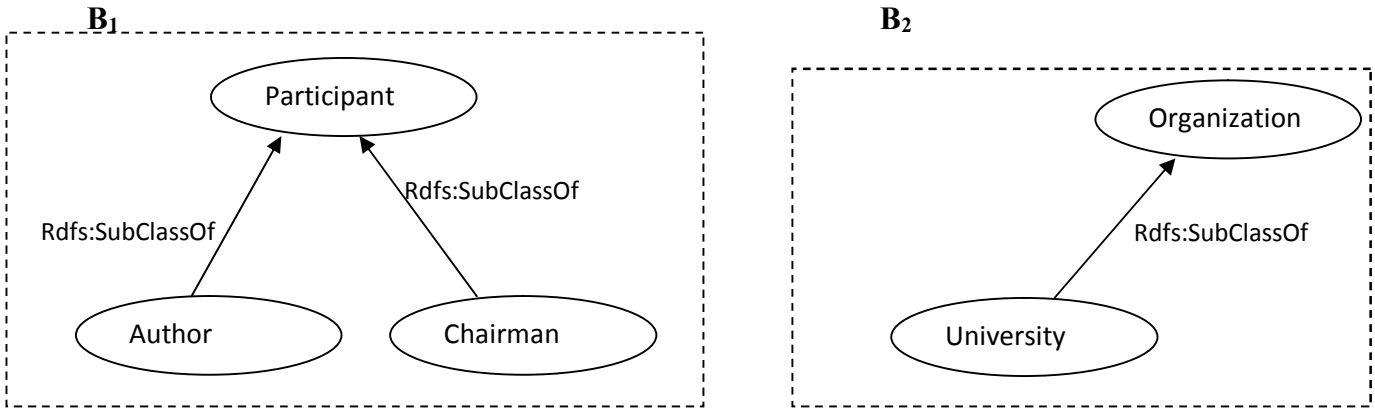


Figure 3.7: Sample of Blocks

A **cluster** is a set of entities, and the entities in each cluster are distinct from those of the other clusters. The union of all the clusters equals the entire set of entities.

A **block** is a set of triples and the triples in one block are distinct from the triples in other blocks. The union of blocks is not equal to entire ontology.

Phase 2: Matching Blocks: at the previous stage, ontologies to be matched have been partitioned into two groups of small blocks. At this stage similar blocks are matched by:

Phase 2.1: Finding Anchors. **Anchors** are alignments with high similarity. A string matcher (I-SUB) is employed to exploit anchors between two full ontologies.

Example: Given all the entities in O_1 and O_2, the string matcher (by comparing names of entities) would detect the following anchors (alignments) with high similarities; {hasUniversity, hasUniversity}, {Chairman, Chairman}, and {University, University}.

Phase 2.2: the blocks from the two ontologies are matched based on the distribution of the anchors. The idea here is that the more anchors can be found between two blocks, the more similar the two blocks are.

Example: After going through all the processes above, block $\mathbf{D_1}$ shown in Figure 3.8 is obtained from the second Ontology O_2. Mapping between block $\mathbf{D_1}$ of $\mathbf{O_2}$ and block $\mathbf{B_1}$ of $\mathbf{O_1}$ can be detected by the existence of the anchor {Chairman, Chairman} between the two blocks. Figure 3.8 shows the two blocks and the mapping;

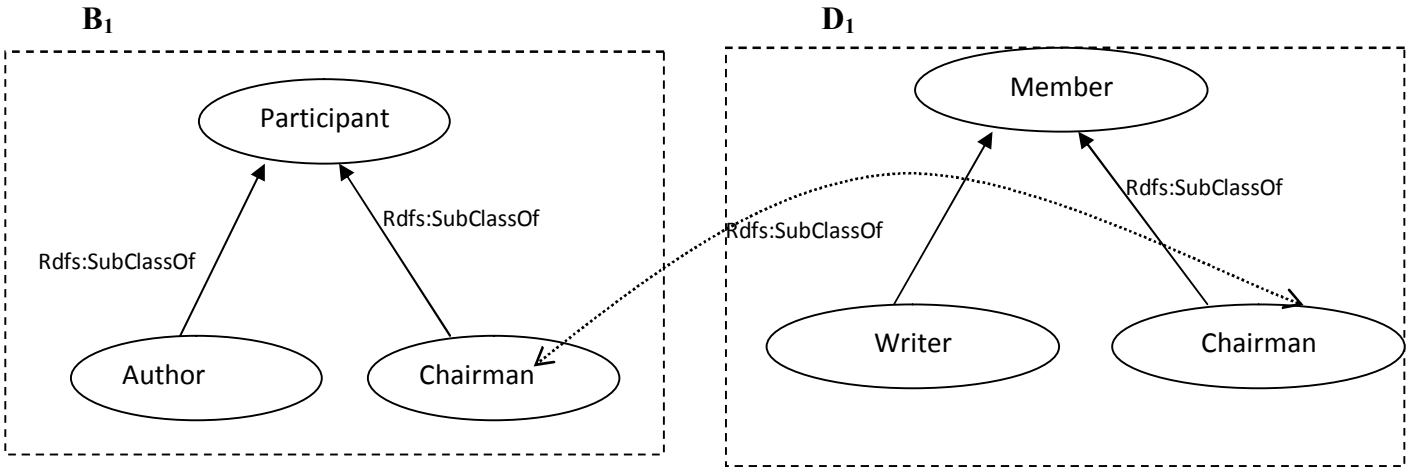


Figure 3.8: Block Matching

Phase 3: Discovering alignments: Each block in a block mapping is seen as a small sub-ontology. At this stage other existing matching techniques i.e V-DOC and GMO are employed to discover alignments between the blocks in each block mapping.

PBM approach has two major advantages:

1. It prevents the matching system suffering from lack of memory; (e.g., the ‘‘out of memory” error, which is caused by not enough memory to be allocated) and

2. It significantly decreases the execution time because it is likely that large portions of one or both input ontologies have no matching counterparts.

3.2.3.2 I-SUB

I-SUB (Hu *et al*., 2008) adopts the string comparison techniques. It improves the edit distance string similarity based technique by not only considering the commonalities in entities’ names but also their differences.

3.2.3.3 V-DOC (Virtual Document)

V-DOC (Qu *et al*., 2006) implements a linguistic approach to ontology matching. Its idea lies in the construction of virtual documents. A virtual document is simply a document for which no persistent state exists. In this system, a virtual document represented by a

collection of weighted words, is constructed for each entity declared in OWL/RDF ontology. The virtual document of an entity declared in ontology contains both the local descriptions and the neighbouring information to reflect the intended meaning of the entity, and a weighting scheme is incorporated to reflect the importance of information appeared in different categories. Particularly, the RDF graph model is exploited to extract the neighboring information (i.e subject, predicate, and object neighbours). The main idea of V-DOC is; the meaning of an entity is encoded within its documentation. Due to this, the V-DOC matcher performs matching by:

1. Parsing and collecting all the entities in each given ontology
2. Getting the description of each entity which consists of local name, label and comment information, and any other annotations of an entity. The description of an entity is given as (Qu *et al*., 2006):

$$\begin{aligned} Des(e) = \; & \alpha_1 * collection\ of\ words\ in\ the\ local\ name\ of\ e \\ & + \alpha_2 * collection\ of\ words\ in\ the\ rdfs{:}label\ of\ e \\ & + \alpha_3 * collection\ of\ words\ in\ the\ rdfs{:}comment\ of\ e \\ & + \alpha_4 * collection\ of\ words\ in\ other\ annotations\ of\ e \end{aligned} \quad (3.2)$$

   where; α_1, α_2, α_3 and α_4 are fixed rational numbers in [0, 1], which indicate the weights for each category.

3. Creating virtual document for each entity. The virtual document of an entity consists of the description of the entity and the description of its neighbours. The virtual document of an entity is given as (Qu *et al*., 2006):

$$VD(e) = Des(e) + \gamma_1 * \sum_{e' \in SN(e)} Des(e')$$

$$+\gamma_2 * \sum_{e' \in PN(e)} Des(e')$$

$$+\gamma_3 * \sum_{e' \in ON(e)} Des(e') \quad (3.3)$$

where; *SN*(*e*): subject neighbouring; The nodes that occur in triples with e as the subject, *PN*(*e*):predicate neighbouring, *ON*(*e*): object neighbouring and γ_1, γ_2 and γ_3 are fixed rational numbers in [0, 1].

4. The similarities between virtual documents in the two ontologies are computed. If two virtual documents don't share any words, the similarity will be 0.0; hence the entities with the documents are not similar. If all the word scores equal completely, it will be 1.0.

Example: Consider the two ontologies in Figure 3.4. When trying to match the entities 'Participant' and 'Member' from O_1 and O_2 respectively, the virtual documents for the two entities will contain the following words:

VD (Participant) = {Chairman, Author, hasUniversity}

VD (Member) = {Chairman, Writer, hasUniversity}

Therefore the similarity between the two entities is obtained through the shared words *'Chairman'* and *'hasUniversity'* in their virtual documents.

3.2.3.4 GMO (Graph Matching for Ontologies)

GMO (Hu *et al.*, 2005) takes the structural matching approach to match ontologies. It uses RDF bipartite graph to represent ontologies and computes structural similarities between domain entities and between statements (triples) in ontologies by recursively propagating similarities in the bipartite graphs. GMO also takes a set of matched pairs i.e alignments with high similarities already found by V-DOC and I-SUB as external input, and incrementally generates extra alignments as output. The Directed bipartite Graph

representation of ontology is obtained by assigning statement node (a kind of triple connector) between triples and using directed edges.

Example: The directed bipartite graph of O_2 in Figure 3.4 is;

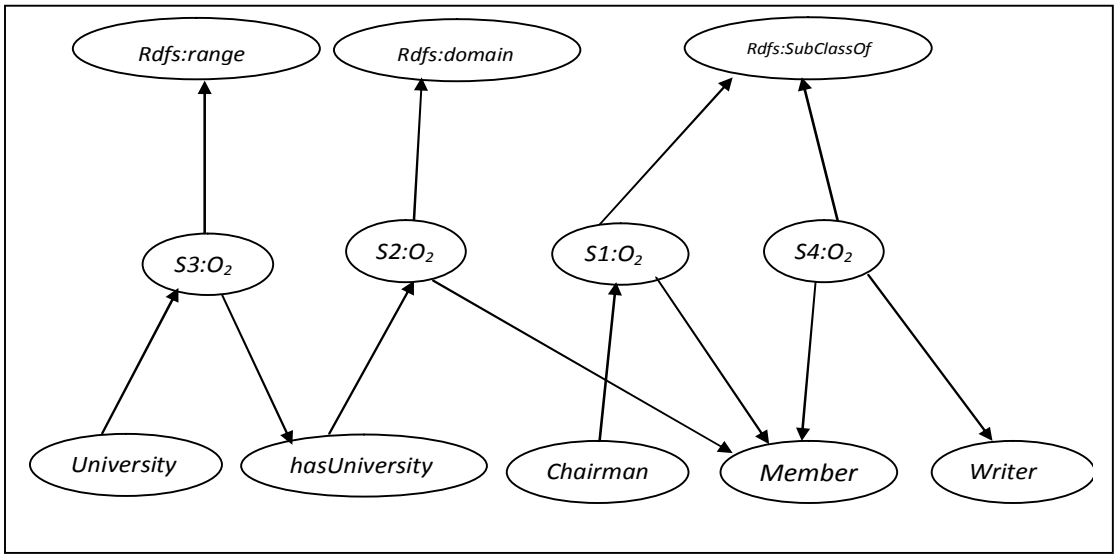


Figure 3.9: Directed Bipartite Graph of O_2

The idea of GMO is:

The similarity of two entities from two ontologies comes from the accumulation of similarities of their involved statements (RDF triples) taking the two entities as playing the same role (subject, predicate or object) in the triples, while the similarity of two triples comes from the accumulation of similarities from the involved entities as the same role in these two triples being compared (Hu *et al.,2005*).

3.2.4 Parsing and Model Construction

Ontology parsing is done by Jena API. The process starts by validating the input ontologies and creating a model (an internal representation for each of the two ontologies) for each matcher. The *model coordinator* adjusts models by applying some *coordination rules*. Ontologies are expressed heterogeneously therefore coordination rules are applied to adjust

the ontologies before passing to matchers in the matcher library. Some of the coordination rules applied by the system include:

1. Ontology header rule: this rule finds and removes all ontology headers
2. List rule: reconstructs list structures by using rdfs:member property to describe the relationship between a list and each of its members, instead of RDF list collection vocabularies *rdf:first*, *rdf:rest* and *rdf:nil*.
3. Deprecated rule : gets and removes all undesirable entities

The main essence of the coordination is to extract the core entities needed for matching

3.2.5 Similarity Combination

Similarity combination is an important and difficult issue in building ontology matching systems. Basically, three matchers (I-SUB, V-DOC, and GMO) perform the matching. The question here is how do we integrate these matchers with best performance? The alignments generated by the different matchers are integrated by observing the measure of linguistic and Structural comparability.

3.2.5.1 Linguistic Comparability

Measure of linguistic comparability is calculated by examining the proportion of the candidate alignments against the minimum number of domain entities in the ontologies. The intuition is that if the number of alignments is close to the number of domain entities in the smallest ontology, then we are almost done with matching, and it is not necessary to execute GMO.

3.2.5.2 Structural Comparability

The structural comparability is determined using a measure of cosine similarity which compares the occurrences of built-in properties used in the two ontologies to be aligned. The

built-in properties are RDF, RDFS and OWL built-in vocabularies used as properties in triples (e.g. rdf:type,rdfs:subClassOf and owl:onProperty).

The alignments generated by I-SUB, V-DOC, and GMO are integrated by observing the linguistic and structural comparability following the rules below:

1. If the linguistic comparability is higher than the structural comparability, then the threshold of V-DOC is lowered so that more alignments are accepted from V-DOC and I-SUB, while only alignments with high similarities are accepted from GMO.
2. If the linguistic comparability is lower than the structural comparability, then most of the alignments are derived from GMO.
3. If the linguistic comparability and the structural comparability are both low, then fewer alignments from V-DOC and GMO are combined to the final result. But in this case more alignments are accepted from V-DOC than GMO because it is assumed that alignments from V-DOC are more reliable than those from GMO.

From our example, both LC and SC are high. This implies that all the alignments are combined to the final result.

3.2.6 Alignment Generation

To select the final alignments, a greedy selection method is implemented; alignments are chosen by performing multiple iterations. In each iteration, candidate alignment with the highest similarity is selected and all its conflicting alignments are removed (the system performs 1 to 1 mapping therefore an entity is only permitted to appear in one alignment). If an alignment containing entity ‘paper’ for instance is selected, then all other alignments that include paper are deleted. The iteration process is terminated when no candidate is left.

3.3 Jena

Jena is a Java based framework which contains extensive java libraries for developing Semantic Web applications. It assists programmers in developing codes that handle *RDF, RDFS, OWL* and *SPARQL*. Some of the tools provided by Jena to help programmers in developing semantic web applications include:

1. Ontology API for handling OWL and RDFS ontologies;
2. Rule-based inference engine to perform reasoning with RDF and OWL data sources;
3. Storage strategies to allow large numbers of RDF triples to be efficiently stored on disk;
4. Query engine compliant with the latest SPARQL specification

Jena converts all information contained in ontology into a collection of RDF triples and stores the information in a data structure called a **Model**. The model is a representation of an *RDF graph*, consisting of a collection of *RDF nodes*, connected to each other by relations. Jena consists of different packages, classes and interfaces; some are listed in Table 3.1.

Table 3.1: Some Jena Packages and Classes

Packages, Classes, and Interfaces	**Description**
Chh.Jena.rdf.model	This package has the Jena core for creating and manipulating rdf graphs
Chh.Jena.reasoner	The reasoner package supports a range of inference engines which perform inferences on rdf model
OntDocumentManger	This is a class that provides services for managing the ontology document
OntModelSpec	A class that provides description of the components of an ontology model, including the storage scheme, reasoner and language specification
ModelFactory	A class that provides methods for creating standard kinds of Model
InfModel	An interface that supports inference capability by a reasoned
OntModel	Interface that provides enhanced view of a Jena model that is known to contain ontology data, under a given ontology vocabulary (such as OWL)
OntClass	Interface that represents an ontology node characterising a class description.

3.3.1 Jena Reasoners and Rule Engines

Jena reasoning package contains a range of *rule engines* and *reasoners* which can be used to infer additional statements based on some ontology information and specified rules. The package basically supports the use of languages such as RDFS and OWL which allow additional facts to be inferred from instance data and class descriptions.

Jena includes different kinds of *Reasoners*. For the purpose of this work, only *Generic Rule Reasoner* is considered. This is because it is used to implement both RDFS and OWL *Reasoners* and also supports user specified rules. The behavior of the *GenericRuleReasoner* is defined by a set of specified rules.

3.4 Summary

An alignment system called Falcon-AO++ that supports user input from experts to align ontologies was designed. The system is an extension of Falcon-AO; a fully automated ontology alignment system. It uses the five basic components of the base system and an additional user input component; 1. *Model pool:*parses ontologies and creates models (an internal representation) for ontologies, 2. *Matcher library:* controls a set of four matchers; **I-SUB** is a string matcher that considers names of entities as strings of characters, **V-DOC** creates virtual document for each entity and compares the similarity between the virtual documents to determine similar entities, **PBM** partitions entities into clusters based on structural proximities and further creates blocks by assigning triples within clusters, blocks are mapped by distributing anchors previously found by I-SUB across blocks, and finally mapped blocks are passed to V-DOC and GMO for final output. PBM is executed only when the ontologies are large. **GMO** is based on structural technique 3. *Alignment Set:* generates alignments using RDF format, 4.*Central Controlller:* executes matchers and perform similarity combination; Alignments from different matchers are combined based on observation of values of linguistic and Structural comparability. 5. *Repository:* stores reusable data. 6. *User Restriction* component generates the *user input* by the following three stages: 1. *Parsing and Entity retrieval*: parses input ontologies and all the entities in input ontologies are retrieved and displayed for easy user selection, 2. *Constraint selection*: allows a user to select a constraint that defines the relation that holds between each pair of selected

entities. Two constraints were provided; **Equivalent** which implies that the entities are equal and should be matched and **Disjoint** which implies that the two entities are different and should not be matched. 3. *Rule implementation* generates and implements rule depending on selected constraints. Finally, user input is stored in *Data Structure* same as that of Falcon-AO and further integrated with data structure of Falcon-AO for final output of alignments.

CHAPTER FOUR

IMPLEMENTATION OF THE FALCON-AO++

This chapter mainly focuses on the implementation of the developed Falcon-AO++ system. It gives an explanation of implemented processes and codes. Snapshots of the GUI for Falcon-AO and Falcon-AO++ are presented. Finally the evaluation process and results are presented.

4.1 Graphical User Interface of Falcon-AO

Falcon-AO provides a GUI to facilitate ontology alignment. Figure 4.1 shows a snapshot of the GUI. The Browse buttons are used to input the two ontologies to be aligned. When a user clicks on the Match button, the system executes different processes and matching techniques, and finally the alignment result is displayed on the interface. The alignment result consists of a number of rows, each row displays the ID, the two matched entities (entity from ontology 1 and entity from ontology 2), Similarity; calculated similarity value, and the relation; in this case the relation is always '=' (equivalent) because the system only considers equivalent relation. A user can save the alignment result by clicking on the save button. The system also provides an evaluation process for users to evaluate the system's result using a reference alignment.

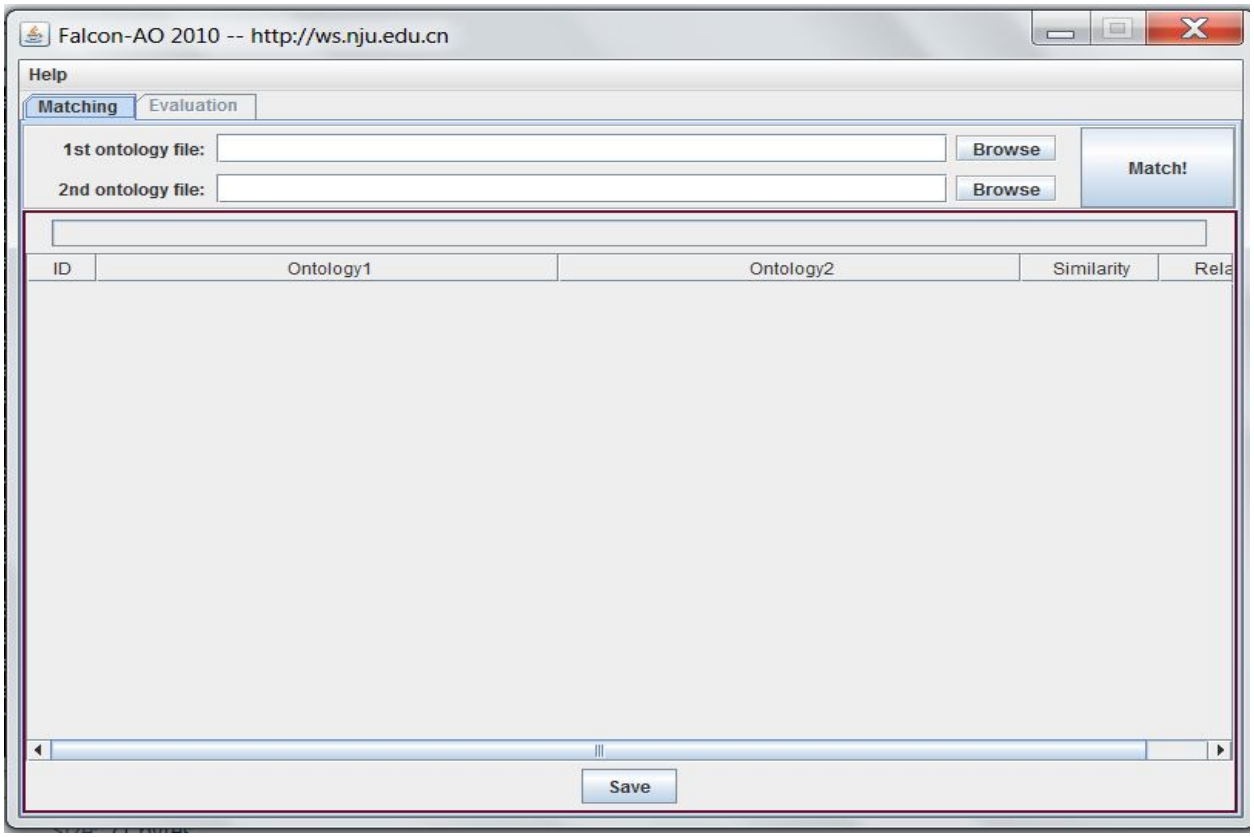


Figure 4.1: GUI of Falcon-AO

4.2 Falcon-AO++

Falcon-AO++ is implemented using java ***(Netbeans IDE7.2***). This is because the base system is implemented in java. Also ***Jena*** was used to parse and create models. The ***Jena Rule Engine*** consists of a ***Generic Rule Reasoner*** which was used in generating and implementing rules. The following processes were implemented:

1. Parsing and Entity Retrieval Process
2. User constraint Specification
3. Rule Implementation
4. User input data structure

4.2.1 Parsing and Entity Retrieval Process

The entity retrieval process is invoked as soon as a user inputs the second ontology file. The system scans through the two files and retrieves all the entities in the given ontologies. Entities retrieved from each ontology file are displayed in a combo box on the GUI to provide easy access to users. This differentiates our GUI (modified GUI) from Falcon-AO's GUI. User constraints are also presented in a combo box. Figure 4.2 shows the modified GUI with two selected ontology files, the list of retrieved entities, and the list of constraints.

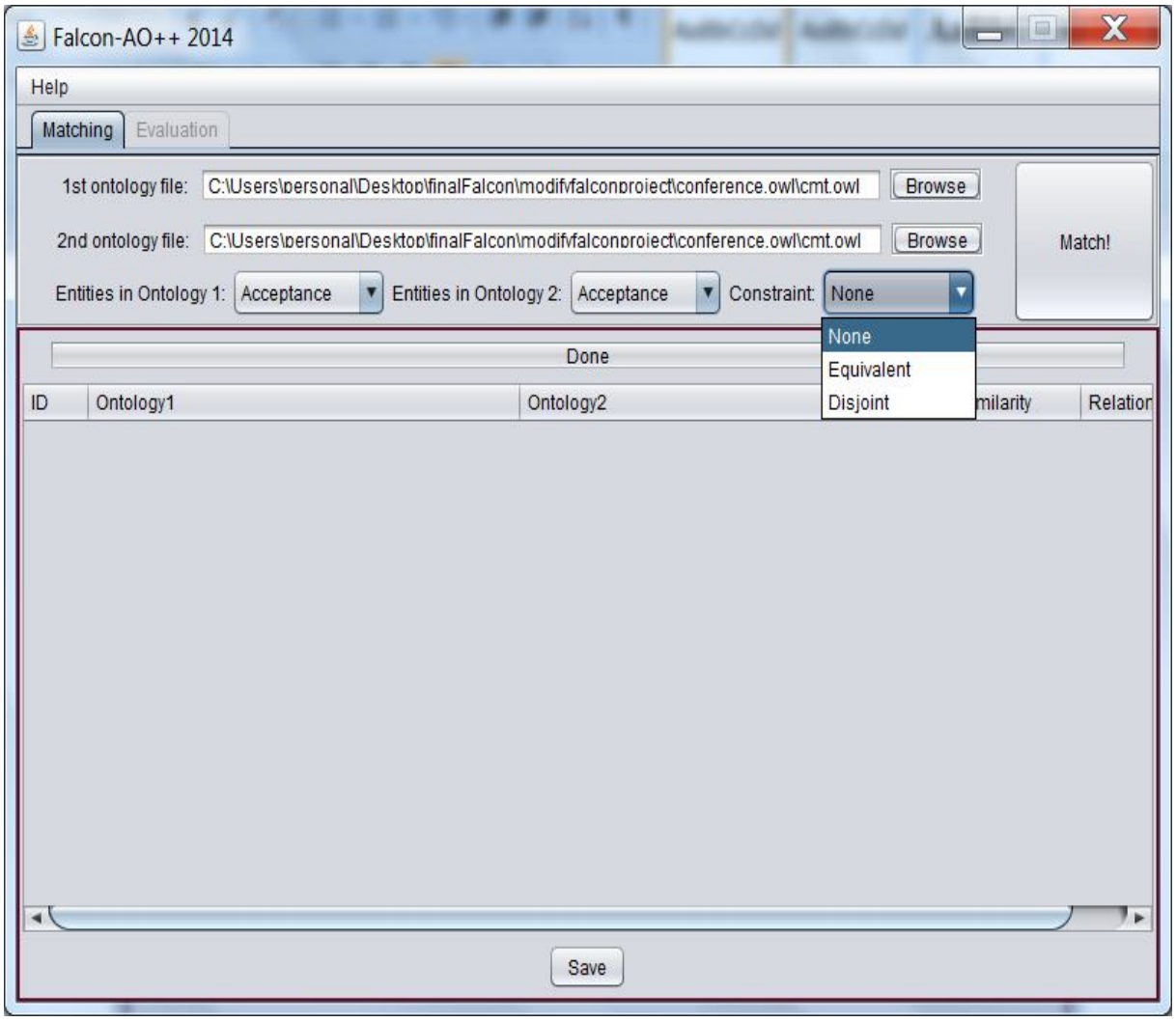


Figure 4.2: GUI of Falcon-AO++

4.2.2 Model Construction and Constraint Specification

The model construction and Constraint Specification processes are contained in the file *TestRule.java.* Once the two entities and constraint have been selected, *TestRule.java* is invoked automatically. First, an empty temporary ontology file is created in order to simplify rule generation and implementation. Figure 4.3 shows how the system first reads the temporary ontology file and creates an empty model for it. The model specification is given by the Jena class *OntModelSpec. OntModelSpec.OWL_MEM_RULE_INF* implies that the ontology is in OWL format, should be stored in memory and inferences are performed on the ontology.

```
private void readOntology(String file)
{
    //Model m=FileManager.get().loadModel(file1);
    file1=file;
    OntDocumentManager mgr = new OntDocumentManager();
    mgr.setProcessImports(false);
    OntModelSpec s = new OntModelSpec( OntModelSpec.OWL_MEM_RULE_INF); //
    //s.setBaseModelMaker(getMaker());
    s.setDocumentManager( mgr );
    m1 = ModelFactory.createOntologyModel(s, null);
    model1=m1.read(file1);
```

Figure 4.3: Creating an Empty Model

Each entity selected by a user is added as a Class into the temporary ontology file Prefix1 as shown in Figure 4.4. A single ontology file is used for all selected entities from both ontologies because a rule can only be implemented in a single ontology file. After selecting two entities, and a constraint, the rule implementation process is executed.

```
        OntClass classA=m1.createClass(prefix1+entity1);      //prefix1+
    }
if(type != 2){
        OntClass classB=m1.createClass(prefix1+entity2);
}
```

Figure 4.4: Adding User Selected Entities as Classes into Temporary Ontology File

4.2.3 Rule Implementation

Figure 4.5 shows how user specified input is coded as a rule. The rule is implemented based on specified constraint. If the selected constraint is 'equivalent', then the rule *classA.addEquivalentClass(classB)* implies that the two entities (classA and classB) should be added as equivalent. But if the constraint is 'disjoint', then the rule *classA.addDisjointWith(classB)* specifies that the two entities should be added as disjoint in our temporary ontology file.

```
//applyRules(rule1,(OntModel)m1);
if(constraint.equals("Equivalent"))
{classA.addEquivalentClass(classB); //.addSameAs(classB);
    System.out.println("Equi");
}
else
{   classA.addDisjointWith(classB);
    System.out.println("Disj");
}
getEquivalentEntities(constraint);
}
```

Figure 4.5: Rule Implementation

After rule implementation, all equivalent entities in the temporary ontology file are further stored in *Constraint_pairs1* and *Constraint_pairs2* which are data structures of type array list storing objects of type *Node*. *Constraint_pairs1* and *Constraint_pairs2* are variables that are used to store user specified constraints which will subsequently be used to form aligned entity (based on user specification) that Falcon-AO will later display as matched entity.

```
String name2=classname3.asNode().getLocalName();
if(!name.equals(name2) && !constraint_pairs1.contains(name) && !constraint_pairs2.contains(name2))
  {
      String nameA=classname.asNode().getLocalName();
      name2=classname3.asNode().getLocalName();

      OntModel modelA=(OntModel)classname.getModel(); //.getNamedClassNodes();
      RBGModel rbgm1 = RBGModelFactory.createModel("");
      rbgm1.setOntModel(modelA);
      //rbgm1.getNode(classname);

      OntModel modelB=(OntModel)classname3.getModel(); //.getNamedClassNodes();
      RBGModel rbgm2 = RBGModelFactory.createModel("");
      rbgm2.setOntModel(modelB);
      //rbgm2.getNode(classname3);
    constraint_pairs1.add((Node)rbgm1.getNode(classname3));
    constraint_pairs2.add((Node)rbgm2.getNode(classname));
  }
 }
}
```

Figure 4.6: Storing Equivalent Entities in Array List Data Structures

Similar to equivalent entities, entities declared as disjoint are stored in *Constraint_pairsd1* and *constraint_pairsd2* which are also array list data structures that store objects of type *Node* as in Figure 4.7

```
if(cons.equalsIgnoreCase("Disjoint"))
{
  if( ( !constraint_pairsd1.contains(n1) && !constraint_pairsd2.contains(n2)
  && !constraint_pairsd1.contains(n2) && !constraint_pairsd2.contains(n2)))
  {
      constraint_pairsd1.add(n1);
      constraint_pairsd2.add(n2);
  }
```

Figure 4.7: Storing Disjoint Entities in Array List Data Structures

4.2.4 User Input Data Structure and Integration with Falcon-AO

When user input is specified, the first file that is executed is *TestRule.java* which consists of all the previously discussed methods. After all the processes in the file have executed, user inputs are stored in data structures *constraint_pairs1* and *constraint_pairs2* for equivalent entities, and *constraint_pairsd1* and *constraint_pairsd2* for disjoint entities.

To implement disjoint constraint, *constraint_pairsd1* and *constraint_pairsd2* are searched. If there are any specified entities, *alignment.removeMapping(n)* (Figure 4.8) implies that any possible mapping between the entities should be removed.

```
for(int i=0; i < TestRule.constraint_pairsd1.size(); i++)
{
 Node node1=TestRule.constraint_pairsd1.get(i);
 Node node2=TestRule.constraint_pairsd2.get(i);

    for(int n=0; n<alignment.size(); n++)
    {
        Mapping m=alignment.getMapping(n);
        Node n1=m.getEntity1();
        Node n2=m.getEntity2();
        if(n1.getLocalName().equalsIgnoreCase(node1.getLocalName()) &&
           n2.getLocalName().equalsIgnoreCase(node2.getLocalName()))
           { System.out.println(n1.getLocalName()+"="+node1.getLocalName()+"  differentFrom  1  "+n2.getLocalName()
             alignment.removeMapping(n);
```

Figure 4.8: Implementing Disjoint Constraint

Result of Falcon-AO is stored in a variable *alignment* which is an instance of a class *Alignment.* In the same way, a variable *align* an instance of *Alignment* was created to store

user matched entities. In figure 4.9 *align.addMapping(new Mapping (node1, node2, 1.00, “”))* implies that the contents of constraint_pairs1 and constraint_pairs2 are passed to data structure of type *Mapping* and finally stored in the variable *align*. The variable *align* stores values of type Mapping. Data structure of type *Mapping* is used in order to avoid conflict when integrating with the data structures of Falcon-AO because *alignment* also stores values of type *Mapping*.

```
for(int i=0; i < TestRule.constraint_pairs1.size(); i++)
 {
    Node node1=TestRule.constraint_pairs1.get(i);
    Node node2=TestRule.constraint_pairs2.get(i);
    align.addMapping(new Mapping(node1, node2, 1.00,""));
    System.out.println(node1.getLocalName()+"="+node2.getLocalName());
 }
```

Figure 4.9: Creating Data Structures for Equivalent Entities

The content of *align* (user input data structure) is integrated with the content of *alignment* (data structure of Falcon-AO) in figure 4.10, before the alignment result is finally displayed.

```
 private void addUserSpecifiedConstraint()
{

   for(int k=0; k < MatchPanel.align.size(); k++){        //JOptionPane.showMessageDialog( null,
       alignment.addMapping(MatchPanel.align.getMapping(k));
   }
}
```

Figure 4.10: Integrating User Input Data Structures with the Data Structure of Falcon-AO

4.3 Evaluation of Alignment Systems

Alignment systems need to be evaluated in order to ascertain the quality of their performance. Due to this, it is good to use standard test data. Ontology Alignment Evaluation Initiative (OAEI) provides standard datasets for evaluation of alignment systems. In the following subsections, we discuss detail of evaluation.

4.3.1 OAEI (Ontology Alignment Evaluation Initiative)

The OAEI is a coordinated international initiative, which organizes the evaluation of the increasing number of ontology matching systems. Its major goals include; assessing strengths and weaknesses of alignment/matching systems, comparing performance of techniques, increase communication among algorithm developers, improve evaluation techniques, most of all, helping improving the work on ontology alignment/matching. The campaign consists of different tracks such as: **Anatomy track**; about matching the Adult Mouse Anatomy (2744 classes) and a part of the NCI Thesaurus (3304 classes) describing the human anatomy, **Conference track**; about finding all correct alignments within a collection of ontologies describing the domain of organizing conferences, **library track**; a real-world task to match two thesauri results etc. For each track, OAEI provides data set and reference alignment.

4.3.2 Data Set

To evaluate the effect of user input on the quality of alignment result, the OAEI conference track dataset and reference alignments have been used. This is because the data set is standard from a widely accepted organization, the domain of organizing conferences is familiar to people in different fields of educational sector, and reference alignments are provided.

Reference Alignment: Is a manually generated alignment which is agreed to be the most correct solution to the matching problem. The reference alignment is not always available, it is generated to be used only for the purpose of evaluation. The goal of alignment systems is to produce an alignment which gets as close to a reference alignment as possible. The results of alignment systems are compared with the reference alignment.

4.3.3 Evaluation Measures

Data Set and evaluation measures are the two essential things in any evaluation. To examine the impact of user input on the quality of alignment result, six pairs of ontologies from OAEI conference track dataset and reference alignments were used. The alignment results of the system were compared with alignments provided in the reference alignment. For the purpose of evaluation, the standard measures of *Precision, Recall, and F-measure* for evaluating ontology alignment systems have been used.

4.3.3.1 Precision

Precision calculates the ratio of correct found mappings with respect to total number of found mappings. i.e among the found mappings, how many are correct? Precision is expressded as (Hu *et al., 2005*):

$$Precison = \frac{Number\ of\ correct\ found\ alignments}{total\ number\ of\ found\ alignments} \quad (4.1)$$

Precision P can also be expressed as:

$$P = \frac{AS \cap RAS}{AS} \quad (4.2)$$

Where; *AS* is the alignment set found by the system and *RAS* is the reference alignment set. Having precision value of 1 does not imply that all alignments have been found but rather it implies that all the found alignments are correct but there may still be some lost alignments.

4.3.3.2 Recall

Recall calculates ratio of correct found mappings with respect to total number of correct mappings, i.e. `How many of the mappings in reference alignment were found? Recall is expressed as (Hu *et. al., 2005*):

$$Recall = \frac{Number\ of\ correct\ found\ alignments}{total\ number\ of\ existing\ alignments} \quad (4.3)$$

Given *AS* as alignment set generated by the system, *RAS* as the reference alignment set, Recall *R* can be expressed as:

$$R = \frac{AS \cap RAS}{RAS} \quad (4.4)$$

4.3.3.3 F-Measure

F-Measure combines the measures of precision and recall as single efficiency measure i.e it measures how close the mappings found are to the correct mappings. F-measure is given as (Hu *et al., 2005*):

$$F - measure = \frac{2 * Precision * Recall}{Precision + Recall} \quad (4.5)$$

4.3.4 Results and discussion

As mentioned earlier, the OAEI conference track dataset was used as test data. The experiments were performed on a system with Intel (R) Core (TM) i3 CPU, 2.13GHz, 4 GB RAM and Windows 7. Tables 4.1 and 4.2 give the comparative results for Falcon-AO and Falcon-AO++. Table 4.1 gives the alignment result; For each pair of input ontologies, the number of existing alignment in the reference alignment, the number of alignments found by each system, and the number of correct alignments found i.e the number of found alignments that exist in the reference alignment are given. The input ontologies are the ontologies needed to be matched. For Falcon-AO++, the number of inputs (**E** for equivalent

and **D** for disjoint) provided by the user is specified. Example, in the first input (Table 4.1) the user passes one input as equivalent and one input as disjoint, this increased the number of correct found alignments from 9 (in Falcon-AO) to 10 (in Falcon-AO++), though equivalent input is suppose to increase the number of found alignments from 18 (in Falcon-AO) to 19 (in Falcon-AO++), in this case it is still 18 because the disjoint input reduced the number of found alignments by 1.

Table 4.1: Alignment Results

			Falcon-AO		Falcon-AO++			
S/No	**Input Ontologies**	**No. of existing alignments in reference alignment**	**No. of found alignments**	**No. of correct found alignments**	**No. Of input**		**No. of found alignments**	**No. of correct found alignments**
					E	**D**		
1	cmt conference	15	18	9	1	1	18	10
2	cmt confOf	16	11	6	2	1	12	8
3	cmt edas	13	13	9	2	2	13	11
4	cmt ekaw	11	11	6	3	2	12	9
5	cmt iasted	4	8	4	0	3	5	4
6	cmt sigkdd	12	13	10	1	2	12	11

The values in Table 4.1 were used to obtain values for precision, recall, and f-measure for Falcon- AO and Falcon-AO++ using the specified formulas. The values are shown in table 4.2. Six pairs of input ontologies were used. For each pair, the precision, recall, and f-measure were computed including the average values. From the results in Table 4.2, it is clear that even the minimum input provided by a user improves values for precision, recall, and F-Measure. This shows that user input has positive impact on alignment result.

Table 4.2: Evaluation Results

		Falcon-AO			Falcon-AO++		
S/No	**Input Ontologies**	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
1	cmt conference	0.5	0.6	0.54545456	0.5555556	0.6666667	0.6060606
2	cmt confOf	0.54545456	0.375	0.44444445	0.6666667	0.5	0.5714286
3	cmt edas	0.6923077	0.6923077	0.6923077	0.84615386	0.84615386	0.84615386
4	cmt ekaw	0.54545456	0.54545456	0.54545456	0.75	0.8181818	0.7826087
5	cmt iasted	0.5	1.0	0.6666667	0.8	1.0	0.8888889
6	cmt sigkdd	0.7692308	0.83333333	0.8	0.9166667	0.9166667	0.9166667
Average		**0.5920746**	**0.674372133**	**0.61572133**	**0.75584048**	**0.79127818**	**0.76863456**

Figure 4.11 shows a comparative result of Falcon-AO and Falcon-AO++ in terms of averages in precision, recall, and f-measure shown in Table 4.2.

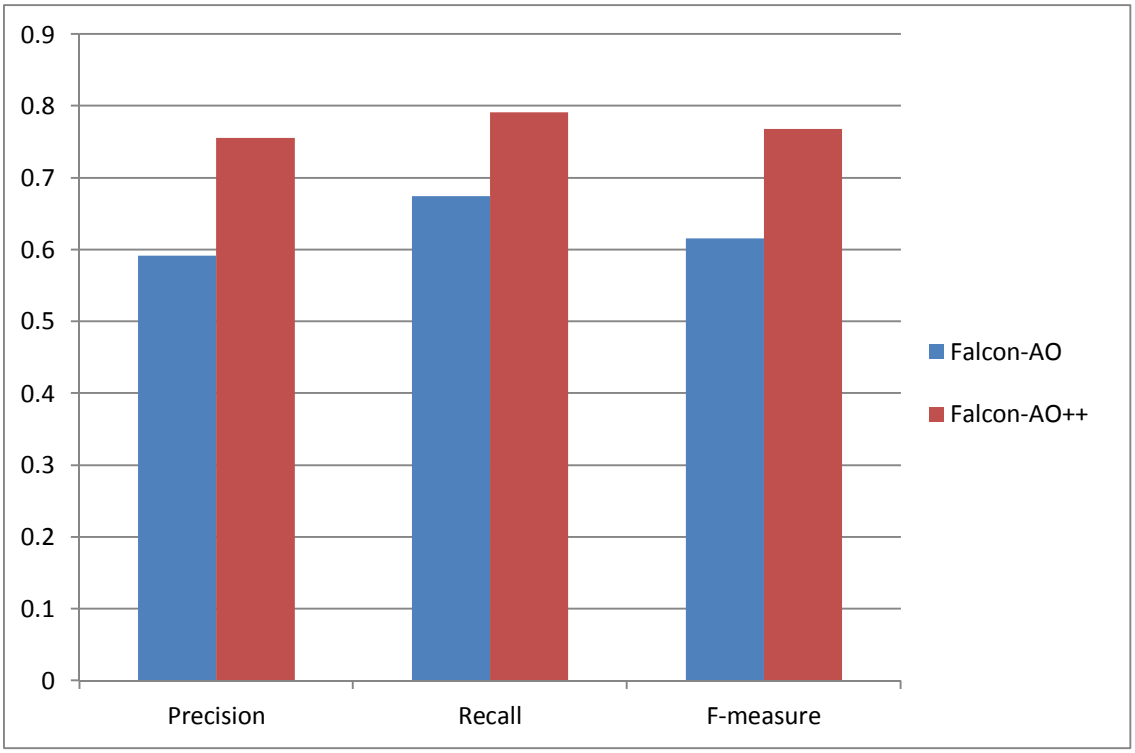


Figure 4.11: Comparative Results of Falcon-AO and Falcon-AO++

From Figure 4.11, it is clear that Falcon-AO++ shows better performance than Falcon-AO. This implies that the contribution of an expert in alignment process is highly beneficial.

4.4 Summary

Four basic processes have been implemented to handle user provided input; *Parsing and Entity Retrieval Process* checks the validity of input ontologies and automatically retrieves all entities from input ontologies. Based on this, the interface has been modified to display retrieved entities. *Model Construction and Constraint Specification* reads a temporary ontology file and creates a model of it, all selected entities are created as classes in the temporary file and later used to generate and implement rules based on specified constraint. *Rule Implementation Process* first checks if the selected entities are in the temporary file, then if the selected constraint is 'equivalent' the entities are defined as '*SameAs'* otherwise they are '*DisjointWith*'. *User Input Data Structure* stores user input in data structures similar to that of the system and are subsequently merged together for final output. OAEI

conference track dataset and reference alignment were used for testing and evaluation. The standard evaluation measures of *Precision, Recall, and F-measure* for evaluating ontology alignment systems were used.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATION

This chapter gives the summary of the work. The conclusion and directions for future works are all mentioned.

5.1 Summary

The WWW impressively contributes to the development of an increasing global information database. It is being used to support delivery and access of information in different domains such as educational, transportation, healthcare and so on. However, the Web has limitations that can be remedied by adding semantics to published information. Adding semantics to the Web requires developing and aligning ontologies, a process that is hard to automate fully. In this thesis, an ontology alignment system Falcon-AO++, based on Falcon-AO was designed and implemented. The system combines the matching capability of matchers and domain knowledge of experts to align ontologies.

5.2 Conclusion

The major goal of this research was to develop and implement a semi-automatic ontology alignment system Falcon-AO++; an extension of Falcon-AO which supports the interactive contribution of domain experts to align ontologies. The semi-automatic approach implemented in this research allows an expert to provide input information to the alignment system. Expert's input is provided by selecting pairs of entities from list of retrieved entities from the input ontologies and selecting a constraint to define the relation between the entities in each pair. Two constraints have been used; Equivalent and Disjoint constraint. Equivalent constraint allows two entities to be matched by the system while Disjoint constraint tells the system that the entities are not the same and should not be matched.

The two systems were evaluated using standard test data from OAEI and evaluation measures of Precision, Recall, and F-measure. The evaluation results obtained have shown that user interaction has positive impact on alignment results.

5.3 Recommendation

This research has shown clearly that the contribution of domain expert in alignment process improves the results of alignment systems. It is therefore recommended to consider the involvement of domain experts in alignment processes. This can be achieved by designing ways in which domain experts can assist without much overhead.

5.4 Limitations and Future work

Although user interaction improves the results of an alignment system, providing input information is still a bottleneck since experts have to be manually involved especially when many inputs are passed. As future improvement, it would be good to integrate more matching techniques that would reduce the level of expert interaction. Presently, the system supports only one to one mapping and considers only equivalent relation. As another direction for future work, the system can be improved to support different kinds of mappings and also different relations. Also, *V-doc* can be extended to consider further neighbours rather than only one-step neighbours.

REFERENCES

Amrouch, S., and Mostefai, S. (2012). Ontology Interoperability Techniques, The State of the Art. *Journal of Information Organization* , 2(1), 20-27.

Antoniou, G., and Harmelen, F. (2008). *A Semantic Web Primer. Second Edition.* Massachusetts London, England: The MIT Press Cambridge, 1-15.

Berners-Lee, T. H. (2001). The Semantic Web. *A Scientific American Magazine*, *284*(5), 28-37.

Choi, N., Song, I., and Han, H. (2006). A Survey on Ontology Mapping. *SIGMOD Record*, 35 (3), 34-41.

Euzenat, J., Le, T., Barrasa, J., Bouquet, P., De, J., Dieng, R., ... Zaihrayeu, I. (2004). State of the art on ontology alignment. *Knowledge web Consortium.* Author manuscript, published in "IEEE Transactions on Knowledge and Data Engineering 158-176.

Gal, A., and Shvaiko, P. (2009). Advances in ontology matching. In *Advances in web semantics i,* Springer, Berlin Heidelberg 176-198.

Gracia, J., and Mena, E. (2008). Ontology matching with CIDER: Evaluation report for the OAEI 2008. *In Proc. of 3rd Ontology Matching Workshop (OM'08), at 7th International Semantic Web Conference (ISWC'08), Karlsruhe, Germany 140-146.*

Granitzer, M., Sabol, V., Weng, K. O., Lukose, D., and Tochtermann, K. (2010). Ontology Alignment—A Survey with Focus on Visually Supported Semi-Automatic Techniques. *Future Internet* , *2*(3), 238-258

Gruber, T. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* , *5*(2), 199-220.

Hu, W., and Qu, Y. (2008). Falcon-AO: A practical ontology matching system. *Web Semantics: Science, Services and Agents on theWorldWideWeb*, *6*(3), 237-239.

Hu, W., Jian, N., Qu, Y., and Wang, Y. (2005). GMO: A Graph Matching for Ontologies. *In Proceedings of K-CAP Workshop on Integrating Ontologies,* 41–48.

Hu, W., Qu, Y., and Cheng, G. (2008). Matching large ontologies: A divide-and-conquer approach. *Data and Knowledge Engineering* , *67*(1), 140-160.

Iroju, O., Soriyan, A., Gambo, I., and Ikono, R. (2012). State-of-the Art: A Comparative Analysis of Ontology Matching Systems. *African Journal of Computing and ICT, 5(4),* 79-81.

Jean-Mary, Y., Shironoshita, E., and Kabuka, M. (2009). Ontology Matching with Semantic Verification. *Journal of Web Semantics.* 7(3), 235–251

Jian, N., Hu, W., Cheng, G., and Qu, Y. (2005). Falcon-AO: aligning ontologies with Falcon. *In Proceedings of K-CAP Workshop on Integrating Ontologies.* 85–91.

Julita, B. (2007). A Simplified Guide to Create an Ontology. *The Autonomous Systems Laboratory ASLab.org. R-2007-004* Universidad Politecnica de Madrid. http://tierra.aslab.upm.es/documents/controlled/ASLAB-R-2007-004.pdf Retrieved on 10th May, 2013.

Lambrix, P., and Tan, H. (2007). A method for recommending ontology alignment strategies. *Proceedings of the 6th International Semantic Web Conference, LNCS 4825*, 494-507.

Maedche, A. (2002). Ontology Learning for the Semantic Web. *The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Kluwer Academic Publishers,* 665: 3-10 .

Morant, A. (2009). *Extending and Optimizing an Ontology Matching System.* MSc dissertation, Oxford University Department of Computer Science 1-10.

Muzaffar, H. M., and Srivatsa, S. K. (2012). A Study of Different Ontology Matching Systems. *International Journal of Computer Applications 37(12),* 10-16.

Paulheim, H., Hertling, S., and and Ritze, D. (2013). Towards Evaluating Interactive Ontology Matching Tools. *The semantic web: semantics and big data, Lecture Notes in Computer Science LNSC*, 31-45.

Qu, Y., Hu, W., and Cheng, G. (2006). Constructing virtual documents for ontology matching. *In Proceedings of the 15th International World Wide Web Conference,* 23–31.

Sarasua, C., Simperl, E., and Noy, N. F. (2012). CROWDMAP: Crowdsourcing Ontology Alignment with Microtasks. *International Semantic Web Conference ISWC,* 525-541. Springer.

Saruladha, K., Aghila, G., and Sathiya, B. (2011). A Comparative Analysis of Ontology and Schema Matching Systems. *International Journal of Computer Application*. *34*(8), 14-21.

Shen, G., Jin, L., Zhao, Z., Jia, Z., He, W., and Huang, Z. (2011). OMReasoner: Using Reasoner for Ontology Matching: results for OAEI 2011. http://disi.unitn.it/~p2p/OM-2011/oaei11_paper13.pdf. Retrieved on 27th June, 2013.

Shvaiko, P., and Euzenat, J. (2005). A survey of schema-based matching approaches. *Journal on Data Semantics IV, Lecture Notes in Computer Science LNCS,* 146-171.

Shvaiko, P., and Euzenat, J. (2013). Ontology Matching: State art and Future Challenges. *Knowledge and Data Engineering, IEEE Transactions on*, *25*(1), 158-176

Vargas-vera, M., and Nagy, M. (2010). Towards Intelligent Ontology Alignment Systems for Question Answering: Challenges and Roadblocks. *Journal of Emerging technologies in web intelligence* , 2(3), 244-257

Zhang, S., and Bodenreider, O. (2007). Lessons Learned from Cross-Validating Alignments between Large Anatomical Ontologies. *Studies in health technology and informatics* 129(pt 1), 822-826.

APPENDIX ONE: Sample of Codes

```
//Entity Retrieval
public void initiateComboRetrievalSequence()
   {
      if (textFieldOnto1.getText().length() > 0 && textFieldOnto2.getText().length() > 0)
      {
         Thread retrieve =  new Thread()
         {
            @Override
            public void run()
            {
               progressBar.setStringPainted(true);
               progressBar.setString("Entity Retrieval Procedure Initiated, Please Wait...");
               progressBar.setIndeterminate(true);
               String fp1 = textFieldOnto1.getText().trim(), fp2 =
textFieldOnto2.getText().trim();
               readFile(fp1, word1);
               readFile(fp2, word2);
               word1.removeDuplicates();
               word2.removeDuplicates();
               word1.sort();
               word2.sort();
               Configuration config = new Configuration();
```

```
config.init();

String s1 = textFieldOnto1.getText().trim();

String s2 = textFieldOnto2.getText().trim();

fp1 = s1;

fp2 = s2;

if (!fp1.startsWith("file:")) {

   fp1 = "file:" + fp1;

}

if (!fp2.startsWith("file:")) {

   fp2 = "file:" + fp2;

}
```

```
//Processing user input
import com.hp.hpl.Jena.rdf.model.Model;
import com.hp.hpl.Jena.ontology.*;

import com.hp.hpl.Jena.ontology.OntClass;

import com.hp.hpl.Jena.rdf.model.*;

import com.hp.hpl.Jena.reasoner.Reasoner;

import com.hp.hpl.Jena.reasoner.ValidityReport;

import com.hp.hpl.Jena.reasoner.rulesys.GenericRuleReasoner;

import com.hp.hpl.Jena.reasoner.rulesys.Rule;

import com.hp.hpl.Jena.util.FileManager;

import com.hp.hpl.Jena.util.PrintUtil;
```

```
//import com.hp.hpl.Jena.graph.Node;
import nju.websoft.falcon.model.Node;
import com.hp.hpl.Jena.rdf.model.StmtIterator;
import nju.websoft.falcon.model.RBGModel;
import nju.websoft.falcon.model.RBGModelFactory;
//import edu.smu.tspell.wordnet.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
public class TestRule {
   private Model model1, model2;
   private String f1, f2, ent1, ent2, consta, rule1, rule2,file1,
         dest="./temp/myOWL2.owl"; //testing2   tempOWL  myOWL
   private OntModel m1, m2;
   private Vector<String> keywords1, keywords2;
   public static InfModel  inf1, inf2;
   static String usr_const;
   String prefix1="http://www.owl-ontologies.com/myOWL2.owl#";
   String prefix2="http://www.owl-ontologies.com/myOWL2.owl#";
   String browseFile1, browseFile2;
   public static ArrayList<Node> constraint_pairs1=new ArrayList<Node>(),
constraint_pairs2=new ArrayList<Node>();
```

```
public static ArrayList<Node> constraint_pairsd1=new ArrayList<Node>(),
constraint_pairsd2=new ArrayList<Node>();


public TestRule()
{
  browseFile1=MatchPanel.textFieldOnto1.getText().trim();
  browseFile2=MatchPanel.textFieldOnto1.getText().trim();
  this.f1="file:"+browseFile1;
  this.f2="file:"+browseFile2;
    }
public void init(String ent1, String ent2, String option)
{
  this.ent1=ent1;
  this.ent2=ent2;
  readOntology("file:"+dest);//reads temporary ontology file.
  clearOntology();
  this.usr_const=option;
  addUserConstraints(this.ent1, this.ent2, option);
}
private void clearOntology()
{
   model1=m1.removeAll();
```

```
    }


//Reading an empty ontology file and creating a model
    private void readOntology(String file)
    {
      //Model m=FileManager.get().loadModel(file1);
      file1=file;

      OntDocumentManager mgr = new OntDocumentManager();

      mgr.setProcessImports(false);

      OntModelSpec s = new OntModelSpec( OntModelSpec.OWL_MEM_RULE_INF); //

      //s.setBaseModelMaker(getMaker());

      s.setDocumentManager( mgr );

      m1 = ModelFactory.createOntologyModel(s, null);

      model1=m1.read(file1);

    }

//Implementing user provided Constraint
private void addUserConstraints(String entity1, String entity2, String constraint)
    {

       boolean found=false;

       int type=0;

       ent1=entity1;

       ent2=entity2;

       for(Iterator i=m1.listClasses(); i.hasNext();)

       {
```

```
 String classname=((OntClass) i.next()).getLocalName();
 if(classname.equals(entity1) || classname.equals(entity2))
    found=true;
 else if(classname.equals(entity1))
       type=1;
 else if(classname.equals(entity2))
       type=2;
}
    if(!found)
{
     OntClass classA=m1.createClass(prefix1+entity1);     //prefix1+
      OntClass classB=m1.createClass(prefix2+entity2);
 if(constraint.equals("Equivalent"))
 {classA.addEquivalentClass(classB); //.addSameAs(classB);
    System.out.println("Equi");
 }
 else
 {   classA.addDisjointWith(classB);
    System.out.println("Disj");
 }
 getEquivalentEntities(constraint);
}
```

```
    else
     {
      JOptionPane.showMessageDialog(null,  "This two entities exist already", "Error",
JOptionPane.ERROR_MESSAGE);
     }
   }

//Saving user input in data structures
    private void getEquivalentEntities(String cons)//Property prop
      {
        model1=m1.read(file1);
       for(Iterator i=m1.listClasses(); i.hasNext();)
        {
          OntClass classname=((OntClass) i.next());
          String name=classname.getLocalName();

          if(constraint_pairs2.contains(name) || constraint_pairs1.contains(name))
            continue;
          if(name.indexOf("_") > 0)
            continue;
          OntResource classname3=(cons.equals("Equivalent") ?
    classname.getEquivalentClass() : classname.getDisjointWith());
          //OntResource classname3=classname.getSameAs(); //.getEquivalentClass(); //
                if(classname3 != null && classname != null) //&&
    classname3.hasLiteral(prop, "constraints") && classname.hasLiteral(prop,
    "constraints"))
          {
             int index=dest.lastIndexOf("/")+1;
```

```
        if(classname.getURI().contains(dest.substring(index))                    &&
classname3.getURI().contains(dest.substring(index)))
        {
         String name2=classname3.asNode().getLocalName();
         System.out.println(name+" >>>> "+name2);
         String nameA=classname.asNode().getLocalName();
         name2=classname3.asNode().getLocalName();

         OntModel                          modelA=(OntModel)classname.getModel();
//.getNamedClassNodes();
         RBGModel rbgm1 = RBGModelFactory.createModel("");
         rbgm1.setOntModel(modelA);
             //rbgm1.getNode(classname);

         OntModelmodelB=(OntModel)classname3.getModel();
//.getNamedClassNodes();
         RBGModel rbgm2 = RBGModelFactory.createModel("");
         rbgm2.setOntModel(modelB);
         Node n1=(Node)rbgm1.getNode(classname); //prefix1+name;
         Node n2=(Node)rbgm2.getNode(classname3); //prefix1+name2;
         String p1=n1.getLocalName();
         String p2=n2.getLocalName();
         boolean duplicate=false;
         if(cons.equalsIgnoreCase("Equivalent"))
            {
              for(int b=0; b < constraint_pairs1.size(); b++)
                {
                  String v1=constraint_pairs1.get(b).getLocalName();
                  String v2=constraint_pairs2.get(b).getLocalName();

                  if( (( !p1.equalsIgnoreCase(v1) && !p2.equalsIgnoreCase(v2))
```

```
                    && (!p1.equalsIgnoreCase(v2) && !p2.equalsIgnoreCase(v1))))
//!name.equals(name2) &&
                        continue;
                    else
                        duplicate=true;
                }
                if( !duplicate)  //!name.equals(name2) &&
                {
                    if(!p1.equals(p2)       &&       (ent1.equalsIgnoreCase(p1)       &&
ent2.equalsIgnoreCase(p2)))
                    {System.out.println(n1+" == "+p1+" loop "+n2+" == "+p2);
                        constraint_pairs1.add(n1);
                        constraint_pairs2.add(n2);
                    }
                }
            }

            if(cons.equalsIgnoreCase("Disjoint"))
            {
             if(            (            !constraint_pairsd1.contains(n1)            &&
!constraint_pairsd2.contains(n2)
             &&                  !constraint_pairsd1.contains(n2)                  &&
!constraint_pairsd2.contains(n2)))  //!name.equals(name2) &&
             {
                constraint_pairsd1.add(n1);
                constraint_pairsd2.add(n2);
             }
            }

        }
    }
}
```

```
    }

//Integrating user input data structures with data structures of Falcon-AO
private void addUserSpecifiedConstraint()
    {

      for(int k=0; k < MatchPanel.align.size(); k++){
//JOptionPane.showMessageDialog( null, "Inside user constraint adder!" );
          alignment.addMapping(MatchPanel.align.getMapping(k));
      }
    }
```