A LIBRARY OF SCHEMA MATCHING ALGORITHMS FOR DATASPACE MANAGEMENT SYSTEMS

A dissertation submitted to the University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2011

By
Syed Zeeshanuddin
School of Computer Science

Contents

List of Tables

List of Figures

Abstract

A dataspace is a type of information management system that uses automated techniques and user feedback to improve the quality of the integration of heterogeneous, independently developed and maintained data sources over time. Current data integration platforms require significant human input and expertise, thus increasing cost and reducing responsiveness to changes in the underlying sources of data. Dataspaces aim to address these issues. Dataspaces is characterized by a *pay-as-you-go* approach, in which intensive automation is deployed to bootstrap the integration process in early stages, and user feedback is relied upon to improve the quality over the lifetime of the resource.

The functional components of dataspace management systems (*DSMSs*) bear similarity with model management operations. Hence the architecture of DSMSs consists of set of types and operations that extend, complement and refine classical model management operations. One of the most fundamental operations in a *DSMS* is that of postulating the schematic (i.e., semantic) relationships between autonomous, independently-designed resources that are assumed to be modeling the same concept at least partially. The process of postulating such relationships is called schema matching and builds upon algorithms called matchers, for example, one matcher might use edit distance to postulate a subsumption relationship between a relation with name *'STUDENT'* in one source and another with name *'MSC_STUDENT'* in another source. Matchers detect and rank the similarity of data sources at the schema level, Structure level and at the instance level. This dissertation developed a library of matchers along with detailed information that maybe derived from them in a principled way. In particular, This dissertation allows matchers algorithms to be combined in a flexible way and developed a uniform approach to assign concept similarity judgments to the output of the matchers in the library.

Deliverables for this project include connection wrappers to connect to data

sources, example data sources for testing and evaluation, a library of schema matching algorithms operating at the element level and structure level, wrappers to configure matcher combination, wrappers to configure matching operation and to aggregate final schema matching correspondences generated by matchers.

Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://www.campus.manchester.ac.uk/medialibrary/policies/intellectual-property.pdf`), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see `http://www.manchester.ac.uk/library/aboutus/regulations`) and in The University's policy on presentation of Theses

Acknowledgments

I would like to thank my Mother and Grandfather for all the their hard work and sacrifices they made for me.
Special thanks to my supervisor Dr. Alvaro A A Fernandes for the patient guidance, for enriching my understanding of database concepts and for providing concrete shape to my otherwise incoherent ideas.

List of Abbreviations

NBWoCon	Name based without context
NBWCon	Name based with context
DBWoCon	Domain based without context
NB	NBWoCon+NBWCon
DBWCon	Domain based with context
DB	DBWoCon+DBWCon
DBMS	Database management systems
DSMS	Dataspace management system

Word Count

Complete dissertation - 22089

Chapter 1

Introduction

Advancements in the database management area led to the extensive use of databases as the primary means for data storage. This triggered the development of independently designed heterogeneous databases by institutions and organizations to store their own organization specific data. With time the importance of reconciling these heterogeneous data sources has grown in importance and complexity. Schema matching is a crucial step in this data integration process. A schema matching operation takes two or more schemas as input and produces a mapping between the elements of the input schemas that correspond semantically to each other [RB01].

Schema matching is a fundamental operation in the application domains like data warehousing, E-commerce and querying data over multiple data sources. In dataspace management systems *(DSMSs)*, schema matching is used to postulate a semantic correspondence between the heterogeneous data sources [HBP$^+$]. Manual schema matching operations are performed by domain experts, and for large data sources, this can be a costly operation due to the high labour costs of domain experts. Manual schema matching is a labour intensive process specially for large data sources because domain expert need to manually search for possible element matches. The data sources involved are also susceptible to frequent changes; Propagating these new changes to already generated schema mapping is very important to avoid incorrect or even failed integration. Manual schema matching is, therefore, very brittle with respect to changes in the underlying data sources. For all these reasons, schema matching need to be fast and fairly accurate with little or no human intervention. Ideally, it should be possible to preform efficient schema matching using available information about schema elements, their

data types , schema structure and the data instance available.

The complexity of schema matching is due to the inherent differences between the participating data sources, such as the fact that different organizations have different preferred database management systems (DBMS) software. For example, one organization may use *MySQL* DBMS while another *Oracle*. This choice of DBMS can be different even for different applications within the same organization. Apart from diversity in the choice of DBMSs, data sources can opt to store data in different formats as dictated by application requirements. Some may store the data using structured representations, like relations and tables, others may opt for semi-structured representations like XML, or else use graphs, or trees. For example, one data source can store student information in the form of *XML* documents, while another may do so in the form of a relation table. Schema matching must grapple with this diversity in the data storage and representation.

Heterogeneous data sources are created by different persons. These creators can have different design for the same underlying concept and can choose to assign names to schema elements in different ways. The complexity of schema matching is further increased by this diversity of representations of the same concept. For example, the creator of one data source can choose to store student information in the form of relation with the table name *'STUDENT'* while the creator of another could name it *'STUDENT_INFO'* or *'ENROLLMENTS'*. Schema matching algorithms must identify this diversity in element names such as stems from the use of synonyms, hypernyms, hyponyms and generate similarity scores between schema elements by performing *name-based* schema matching.

Data sources can store the same information in more than one entity, For example, one data store can store student information in relation name *'STUDENT'* while another data store can split the same information in two relations *'MSC_STUDENT'* and *'PHD_STUDENT'* as dictated by application requirements. This is a case of horizontal partitioning of the data over multiple relations. Similar diversity in representation, naming and splitting of data over a number of entities can also be found in the attributes of information stored in different data sources. For example, the creator of one data source can choose to store student addresses in separate attributes such as *'HOUSE_NO', 'STREET', 'CITY', 'COUNTRY, 'POST_CODE'* while another creator can choose to have only two attributes *'ADDRESS'* and *'POSTAL CODE'*. This is an example of

one-to-many partitioning of data where one attribute in one data source is partitioned over multiple attributes in another.

Heterogeneous data sources also differ in representing the domain information. Domain information includes the data type assigned to a column, the length of the values and the constraints on them. Data types and constraints are system-specific and hence can differ from data source to data source. Schema matching algorithms should be able to tackle this diversity in domain information and use it to perform *domain-based* schema matching.

Information about a schema can be extracted from schema catalogs stored by a *DSMS*. Schema catalogs contain information, such as table structure, number of columns, column names, data types and lengths of the columns and the constraints on the columns. Structure-level schema matching uses the structure information of elements to generate correspondence between them.

Catalog information can be used to perform *element-level* and *structure-level* schema matching. The outcome of statements (such as SQL CREATE) in the data definition language (*DDL*) used can also serve as a source of information to perform *element-level* and *structure-level* schema matching. In data sources where neither catalog information nor *DDLs* statements are available, data instances can be used to extract information about the schema and this instance information can be used to perform *instance-level* schema matching.

Given the wide applicability of schema integration across many domains, the schema matching problem has been extensively studied and researched. A taxonomy of schema matching approaches was presented in [RB01] along with the characteristics of already published system prototypes for schema matching, For example, *Microsoft's 'CUPID'*[MBR01] system, which uses a hybrid matching technique combining element-level and structure-level schema matching.

Another system called *COMA*, solves the element-level name matching problem [DR02, RB01, DDH03] using various string matching and data type matching algorithms. The *iMAP* system [DLD+04] presents a semi-automatic approach for finding *1:1* and *1:n* correspondences between schema elements including complex matches involving mathematical operations.

Most of this previous work performs schema matching using either strings or graph matching techniques as covered in [RB01] and [DMR03]. The *LSD* [RB01] system uses machine learning techniques to develop instance-level matchers to

perform schema matching by exploiting information present in sampled data instances conformant to a schema.

The element-level name-matching problem is tackled by systems like *CUPID* and *COMA*. *CUPID* addresses diversity of abbreviations, acronyms, synonyms, problem associated with different element names, data types and structure using a linguistic matching technique[MBR01].

Linguistic matching techniques are hybrid techniques which offerers a combined solution for element-level schema matching, domain matching and structure-level matching. *COMA* performs element-level name and domain-based matching using a set of *simple matchers* which can be run independently. *COMA* also provides hybrid matchers containing predefined combination of simpler matchers with structure-level matchers. The *iMAP* system makes use of specialized algorithms that look for a specific attribute combinations to infer *1:1* and *1:n* correspondences. These techniques for schema matching either uses, single matcher that cannot be combined at all with other matchers in the system or only offer hybrid or composite matcher that contains a predefined combination of matchers. Hence, there is a lack of user control over which matchers can be used in a combination.

Systems like *COMA* and *CUPID* uses, different techniques to perform *name-based* and *domain-based* schema matching. For example, *COMA* uses string matching techniques to perform *name* matching but uses a data type compatibility table for *data type* matching. These systems also make the joint use of context information in *name* and *domain* matchers as compulsory and hence do not allow user to experiment with the finer-grained constraints.

Instance level schema matching involves extracting schema information from data instances present in the store. Extracted information can be used to infer a schema. This can then be used in schema matching scenarios where there is no schema catalog available. Commercial data sources contain large amounts of data. Extracting schema information can therefore be very processing intensive. To perform instance level schema matching, it is not practical to compare every instance in one schema with every other instance in another. The *LSD* system [DDL00] takes a machine learning approach to develop instance-level matchers. *LSD* relies on user supplied mappings to train instance-level matchers and create matching rules, which then can be used to match a new data source.

1.1 Aims and Objectives

The project behind this dissertation aimed to investigate the benefits of exposing matching capabilities as a flexible library that allows users full control on which algorithms to use and on how to combine and aggregate their outputs.

This dissertation would try to meet following objectives,

- This dissertation would develop a taxonomy of schema matching approaches for element-level, structure-level and instance-level.
- As part of this dissertation a library of schema matching algorithms instantiating the taxonomy to perform automatic schema matching would be implemented.
- Present flexible technique to configure schema matching algorithms combinations.
- Present techniques to aggregate schema matching results generated by the schema matching algorithms.
- Explore how different kinds of schema matcher combinations perform with respect to schema matching accuracy in controlled experiment.

A library would contain algorithms that can perform schema matching at element-level, structure-level and instance-level. In concrete terms, the objectives was to develop separate algorithms that can perform element-level name-based and domain-based schema matching with and without the use of context information. For structure-level schema matching, we decided to present every schema as graph with schema elements represented by is-a and has-a relationship, and then use graph comparison technique, like graph edit distance, to calculate the similarity between the structure of the two schemas.

The project had the goal of developing two algorithms for instance-level schema matching. Due to lack of time we present only the idea of how instance-level schema matching problem can be solved. One matcher can perform data instance comparison between elements of both the schemas. A second matcher can perform instance-level schema matching by exploring a novel approach based on grammatical inference [SG10].

1.2 Overview Of Approach

One of the early design decision was to use string comparison techniques as the fundamental operation for all levels of schema matching, namely element-level, structure-level and instance-level.

For element-level name-based schema matching, the goal was to develop two algorithms : name based matcher without the use of context information and one that used such information. Similarly, for domain-based, the library would contain two algorithms: domain based schema matcher without the use of context information and the one that used such information. Both name-based and domain-based schema matching would be founded on string comparison operations.

For structure-level schema matching, the goal was to represent every relational schema as a graph (having is-a and has-a edges between its elements). A graph represents a schema having nodes as elements and attributes. Structure-level schema matching would then use graph edit distance to calculate a similarity score between graphs. Set of strings representing node names and data types will be treated as one unit. On these units graph edit distance operations will be carried out to get the similarity measure between the structure of two schemas.

As already stated string comparison algorithms are used as the basic foundation for schema matching operations. Three string comparison algorithms are used, namely string edit distance, N-gram and stem based.

To achieve flexibility when combining matcher algorithms from library, a configuration document is used. A generic interface to call any algorithm and a generic output structure to allow the results generated by algorithms in combination to be used would be provided for aggregating results. Input configuration document is used to assign weights to individual schema matching algorithms.

1.3 Summary of Achievements

This project successfully achieved most of its goals. A taxonomy of schema matching algorithms was developed that is ground in string comparison. String matching algorithms based on string edit distance, N-gram based and stem based are used. A library comprising name and domain-based matcher with and without context as well as a structure-based matchers was developed. Mechanisms to

flexibly combine matchers, to configure them and to control their behavior were developed. However, due to lack of time instance-level schema matching was only discussed , implementation is left as future work.

Empirical experiments were carried out that provided evidence that the flexibility of the library can be used to achieve encouraging results that stand comparison with existing state-of-the-art platforms.

1.4 Dissertation Structure

The rest of this dissertation is structured as follows,

Chapter 2 discusses the technical background and presents the motivation for undertaking this project. An overview of applications of schema matching is discussed in this chapter. This chapter also presents previous work and existing researches in schema matching arena. The chapter presents existing prototypes, discusses their achievements and compare their techniques with those adopted in this project.

Chapter 3 presents an overview of element-level, structure-level and instance-level schema matching along with a taxonomy developed as part of this project. The chapter also contains an overview of the string matching techniques implemented in this project.

Chapter 4 presents the approach taken to develop the prototype implementation along with development methodology used. The chapter presents an architecture using unified modeling language (UML) diagrams. The chapter also explains how to configure schema matching algorithms in the library to form a matcher combination and discusses in detail the techniques to combine schema matching results generated by individual matchers.

Chapter 5 presents a detailed discussion of all the algorithms presented as part of schema matching algorithm library. This chapter also presents which string matching algorithms are used in an individual matcher.

Chapter 6 presents some of the critical implementation details. The chapter

presents the development environment used for this prototype implementation. The chapter also discuss application programming interfaces *APIs* used in the prototype.

Chapter 7 presents the results of experiments carried out to evaluate the performance of the library. The chapter evaluates the performance of individual matchers and of combinations thereof using different parameters and draws some observations from these results.

Chapter 8 summarizes the contents of this dissertation and draws broader conclusions from the performance evaluation. Chapter also presents possible future work to extend the capabilities implemented.

Chapter 2

Background

As stated in chapter 1, schema matching is the operation of creating semantic matching between the elements of heterogeneous data sources [RB01]. Schema matching is an important step in data integration operation and got wide application in domains like E-commerce, data warehousing and semantic query processing [RB01]. This chapter presents previous, related work in schema matching. This chapter also discusses existing prototypes and contrasts their approach with those taken in this dissertation.

2.1 Applications Of Schema Matching

To emphasize the importance of schema matching operation, this dissertation presents some of the important applications of schema matching as follows,

Dataspace Management systems

Heterogeneous data sources, though different, are assumed to be modeling same concept at least partially. *Pay-as-you-go* data integration of such heterogeneous data sources is one of the primary functions of dataspace management systems (*DSMS*) [HBP$^+$].*DSMS* is in response to the high upfront cost associated in the creation and maintenance of near perfect integrated sources and their low adoption rate to changes in data sources. *DSMS* provides agile data integration with low upfront and maintenance cost by extensive use of automation in bootstrapping stage and use of user feedback for improving the quality of data integration over time. Integration of independently developed, heterogeneous data

sources, require postulating schematic correspondence between elements of data sources and derivation of schema mappings from this schematic correspondence. Automatic schema mapping using model management techniques is a crucial step for *DSMS*. This process of postulating schematic relationship between data sources is called schema matching for *DSMS*.

Schema Integration

Schema integration is one the primary motivation to perform schema matching. Schema integration is the process of automatically creating a global view of multiple independently developed schemas[PS98, BLN86, RB01]. To create global view of independently developed schemas, integration process must identify relationships between schema elements of the data sources. Data sources as modeled independently by different people they will inherently have diversity in their schema representation; schema integration process must resolve this diversity in schema representation to create schema element mappings. Once schema mappings are created mapped elements of the data sources are combined to create a unified global view. For example, schema integration is required to combine the search results fetched from multiple data sources to provide a final unified search result.

Data Warehouse

Data warehouses are used for analytical processing to facilitate better decision making. Data warehouses are build by an extract transform and load process (*ETL*) which extracts data from multiple data sources, transforms data from these data sources into data warehouse format and then loads data in data warehouse. Transformation processes require matching elements from data source to elements in data warehouse. This matching operation is schema matching. Automatic schema matching can generate an initial elements mappings without any human input. Data warehouse designers can then confirm this initial mappings and reconciles the semantics of the data sources with that of the data warehouse [RB01].

E-commerce

Advancement in networking technologies and global connectivity facilitated by world wide web allowed trading partners to communicate with each other easily. Trading partners exchange messages that represent their business transactions. As each trading partner's system was developed independently, it is common for them to use different message formats. For example, having different *XML* message schemas or custom data structures [RB01]. To allow trading partners to exchange messages, application must be able to transform messages from one format to another. This transformation of messages require to perform matching between schemas of messages. This transformation is largely done by application designers manually. Thus, automating schema matching will reduce this manual mapping task by proving application designers with initial mappings, which application designers can validate or modify.

Semantic Query Processing

Semantic query processing is another valuable application of schema matching [RB01] operation. In semantic query processing user describes the output of a *SQL* query using the *SELECT* clause; this query out will be in terms of concepts familiar to user which may not be same as the elements of the database against which query need to be processed. To process such a query request, system need to first figure out correspondence between elements of the schema and user specified concepts. Similar semantic mapping need to be done for *WHERE* clause of the *SQL* query. For semantic query processing system need to perform schema matching at run time.

Given wide application of schema matching across many domains, schema matching operation is thus very critical and should be generic. Schema matching is a well studied field and many notable previous works exist. Extensive classification of schema matching approach is presented in [RB01] which classifies schema matching approaches as individual matchers and combined matchers. Combined matchers are fundamentally a preconfigured set of individual matchers. They are further classified into composite and hybrid matchers, which differs in how individual matchers and their results are combined. Choice of matcher is governed by application domain and schema type. In [RB01] Individual matchers are classified as element-level, structure-level and instance-level. This dissertation builds upon this classification of schema matching algorithms. Instead of using

hybrid or composite matchers this dissertation presents a more flexible approach to combine matchers.

Another classification of schema matching approach is presented in [SE05]. This paper classifies schema matching technique based on the granularity of schema or by the kind of input.The basic matcher techniques using granularity of schema are classified as element-level and structure-level. Instance level schema matching is not considered in [SE05]. This dissertation adopts a similar approach to classify the schema matching techniques based on the granularity of the schema at element-level and structure-level. This dissertation extends the classification approach to include instance-level schema matching.

Most of the previous approaches of schema matching [RB01, SE05, DR02, ADMR05, MBR01] makes use of external information in addition to using schema information. Use of external information is to improve the accuracy of schema matching process. External information can be fetched from sources such as generic or domain specific thesauri, manually entered schema mappings or other user supplied information that can be used to enhance schema matching accuracy. Approach presented in this dissertation make use of generic thesauri and domain specific mappings. In this prototype, generic thesauri *WordNet* is used as context information to help identify diversity in element level matching[wor11]. Diversity in the element name matching could be due to use of synonyms, hypernyms or hyponyms; Use of generic thesauri helps identify such relationships between element names. *WordNet* is also used to identify diversity in data types for domain-based matching. This dissertation also makes use of domain specific information stored in the form of manually entered mappings. Use of domain specific mappings enables integrating domain specific knowledge in the schema matching process. This prototype implementation uses domain specific mappings for recording diversity of data types associated with the use of different *DBMS* as data sources.

2.2 Review Of State-of-the-art Schema Matching Systems

A comparison of prototype implementations for schema matching approaches is presented in [RB01]. [RB01] compares prototypes based on the type of input it supports, schema matching search space covered, application domain and level

of automation. Some of the notable prototypes discussion along with the comparison of their schema matching approaches with the approaches taken in this dissertation are as follows,

COMA

COMA (COmbination of MAtching algorithms) is a schema matching system that allows combining matchers and evaluate their schema matching performance[DR02]. *COMA* systems present a wide array of individual matchers. It allow reusing of match results of the previous match operations. It is a generic match system which can be potentially used for many applications and supports *XMLs* and *relational* schemas as input types.

COMA presents a set of individual matchers that can perform name-based comparison of schema elements. Every individual matcher uses different string comparison technique like edit distance, N-gram or Affix. *COMA* also presents individual matchers for data type comparison, for use of synonym in string comparison, and for use of user feedback. It allows combining these individual matchers to form a hybrid schema matcher or a composite schema matcher. Hybrid schema matcher combines multiple individual matchers in a single matcher to generate one match result; while a composite schema matcher combines the output of multiple individual matchers. Composite schema matcher can also combine a hybrid matcher with individual matchers. *COMA* system performs schema matching at element-level and structure-level.

COMA++ is an extension of original *COMA* system, it supports a graphical user interface and support for ontologies [ADMR05]. *COMA++* system is intended to solve large real world problems. It uses a fragment based matching technique which is a divide and conquer approach to solve the large matching problems and it gives emphasis on reusing already performed matchings.

This dissertation builds upon techniques presented in *COMA* system. It classifies schema matching algorithms based on the schema level information the matcher uses to perform schema matching. For example, element-level schema matcher makes use of element names and element domain information, structure-level schema matcher uses structure information of the schema, and instance-level schema matcher make uses information in data instance of schema elements. This dissertation presents a more flexible approach to combine individual matchers and to control their contribution in a combination's match result by giving weight to

each matcher. Output of all matchers are combined by taking a weighted average of the similarity scores. This dissertation also provides a framework to evaluate schema matching performance for different schema matcher combinations. All element-level schema matchers make use of string comparison techniques; these string comparison techniques are not treated as individual matchers. Schema matching algorithms and string matching techniques are discussed in detail in chapter 3.

CUPID

'CUPID' uses a hybrid matcher which combines linguistic level matching and structure matching. To improve schema matching accuracy *'CUPID'* makes use of generic as well as domain specific thesauri. Linguistic schema matching is carried out in three steps, Normalization, categorization and then actual linguistic comparison making use of thesauri[MBR01]. Data type matching is done along with structure-level schema matching. Mapping results of *'CUPID'* can be a *1:1* mapping or *1:n* mapping between schema elements.

Major different between *'CUPID'* and the approach taken in this dissertation is in schema matcher combining technique. *'CUPID'* combines element-level and structure-level schema matching operations into a hybrid matcher, while this dissertation presents element-level and structure-level schema matchers as separate matchers. Moreover *'CUPID'* systems combines element-level name comparison and name comparison using external information as one operation, Structure-level schema matching and data type matching as one operation while this dissertation presents all matchers separately and leaves the choice of combining matchers to user.

'CUPID' systems represent a schema as graph with elements represented as nodes. Nodes can have edges reflecting that node is connected related to the connected element. Every relationship is represented by a distinct node name in this graph model. *'CUPID'* extends its tree matching algorithm to perform schema matching using this graph representation of the schema. *'CUPID'* system is capable of performing referential relationship mapping by representing referential constraint as a distinct element in the graph and then perform tree comparison to get the mappings.

This dissertation performs structure-level schema matching using graph comparison. We represent a schema as a graph having nodes as elements and leaves

as attributes. Elements and attributes are connected by edges representing is-a and has-a relation between elements and attributes. This dissertation uses graph comparison techniques such as graph edit distance [ZTW+09, ZS89] to get the similarity measure between the structure of two graphs. This dissertation do not present any matcher utilizing constraint information for schema matching, implementing such a matcher is deferred as a future extension.

LSD

LSD (Learning Source Descriptions) system uses a machine learning approach to solve schema integration problem[DDL00]. *LSD* system performs element level matching and instance level matching using a set of matchers called the base learners. It creates schematic mapping using a two phase semi-automatic process. Every base learning is first trained using user supplied mapping between the source schema and the mediated global schema. Learners discovers matching rules during the learning phase by analyzing the data instance in the source schema. These discovered matching rules and patterns are then used in the matching phase to create mappings between new source schema and the internal mediated global schema. *LSD* generates *1:1* mapping between elements of the two schemas. The learner can exploit schema information to perform *XML* element matching or perform matching based on data values in the schema elements. To further improve matching accuracy *LSD* system can make use of user supplied domain constraints.

LSD system do not perform structure level schema matching; This dissertation make use of schema structure to perform schema matching. The *LSD* system is intended for data integration purpose while techniques presented in this dissertation are generic and can be applied for any application domain. For instance level schema matching, *LSD* system uses machine learning technique that requires training phase for every base learner, this dissertation presents a novel idea for instance level schema matching. For instance-level schema matching, this dissertation suggests two algorithms, first algorithm performs a data instance comparison on reduced search space or on sampled data. Second algorithm constructs schema for a data source using grammatical inference technique [SG10]. Research direction suggested for instance-level schema matching is discussed in detail in chapter 3.

SemInt

SemInt prototype performs schema integration[LC94, LC95, RB01]. It utilizes both schema information fetched from catalog of a relational *DBMS* and data contents of an element to perform schema matching. *semInt* perform matching my a classifier method that trains neural networks to recognize categories of elements and generates *1:1* mapping. This project takes a different approach to create matchers but shares the concept of utilizing schema information from the system catalogs of relational *DBMS*.

ARTEMIS

Another notable prototype is *ARTEMIS* [CA99, RB01], which is a schema integration system. *ARTEMIS* calculate element-level and structure-level schema match affinity using a hybrid model and then uses clustering technique to cluster the attributes based on match affinity. *ARTEMIS* makes use of generic and domain specific thesauri. It allows user to assign adjustable weights to match operations, these operations are then used to calculate the matches. This dissertation adopts a similar approach and assigns weight to individual matcher to control the contribution of matcher in the final match result. This dissertation takes weighted average of similarity scores generated by individual matchers to calculate the final similarity score of the matcher combination; this is discussed in detail in chapter 3.

This dissertation make use of system catalog of *DBMS* to get information about a schema to perform schema matching. Most of the relational database management systems *DBMS* like *Oracle 11g* and *MySQL* stores schema descriptions of all the databases they manage in system catalogs.

Schema information is stored in a set of *DBMS* system tables and views, called system catalogs. System catalogs contain information like names of all the tables, column names in every table, data type of columns, length of columns and information about constraints and referential integrity. This dissertation makes use of system catalog to get schema information to perform element-level name matching, element-level domain matching and to construct a graph representing a schema. This graph is then used to perform structure level schema matching. Schema information can also be fetched from the data definition language's create statements *DDLs*. *DDL* statement contains all the information about a table, like table name, column name, column data type, length of column and constraints

on columns. *DDLs* can be parsed as a string to extract schema information. Similarly, schema information can be extracted from *XML schema* of an *XML* document by parsing *XML schema* file. Extracting information from *DDLs* and *XML schema* files are left as future extension of this dissertation.

Other pertinent prototype systems are *SKAT, TranScm* which uses hybrid model of schema matching[RB01, MWK00]. *SKAT* can perform element-level and structure-level schema matching while *TranScm* can only perform element-level schema matching and can have a fixed order of matcher in the hybrid model. A detailed comparative evaluation of schema matching prototypes is presented in [DMR03, RB01]. Most of the studied prototypes takes a hybrid approach to perform schema matching and generally lacks the ability to combine different individual schema matching techniques flexibly. This dissertation presents an approach that allows matchers from library to be combined flexibly to form different matcher combinations.

Most prototype systems in previous approaches [RB01, SE05] for schema matching develops separate matcher to tackle a specific problem, for example, specific matchers are developed to match string, date, numeric and currency values. This approach makes it difficult to combine matchers because of the differences in their input format. This dissertation takes the approach of expressing every schema matching operation as fundamental string matching operation. This dissertation also takes a more comprehensive approach towards matching technique, and combines string matching and graph matching techniques seamlessly to perform element-level and structure-level schema matching respectively. Previous work on schema matching combines outputs of the different matchers in a ad-hoc manner; this dissertation presents a generic composite approach which is similar to approach taken in *COMA* system [DR02]. Result aggregation method is discussed in more detail in chapter 3.

This chapter discussed what is schema matching and its application in dataspace management systems for postulating schematic relationship between different data sources. Chapter covered detailed discussion of schema matching application in data integration, data warehousing, E-commerce and in semantic query processing. This chapter also discussed how techniques presented in this chapter builds upon existing work for schema matching. This chapter discussed in detail relevant prototype implementation of schema matching and contrasted their approaches with the approach taken in this dissertation. Next chapter

presents an overview of performing schema matching at element-level, structure-level and instance-level along with the taxonomy of their schema matching approaches. Next chapter also presents string comparison techniques utilized in schema matchers.

Chapter 3

Overview

Chapters 1 and 2 explained the motivation for taking up this research in schema matching, application of schema matching in different domains specifically in *dataspace management systems*. Chapter 2 discussed important related works, upon which this dissertation is build upon. Previous chapter also presented existing schema matching prototypes along with the comparison of their schema matching techniques with the technique presented in this dissertation.

As discussed schema matching can be performed using three types of schema information namely information at the element-level, structure-level and instance-level. This chapter presents a taxonomy of schema matching approaches along with a detailed discussion for element-level, structure-level and instance-level schema matching approaches. The Schema matching approaches presented in this dissertation are build upon previous works and prototypes. This chapter also discuss string comparison techniques that form the building blocks for all the schema matching algorithms in the library.

3.1 Taxonomy Of Schema Matching Algorithms

This dissertation classifies schema matching operation into three broad categories, element-level schema matching, structure-level and instance-level schema matching. These categories represent the type of schema information utilized to perform schema matching operation, for example, if schema matching operations is using element level schema information such as element names or element's domain information then it is performing element-level schema matching. Schema matching operation can use schema structure information to perform structure-level schema

matching. Instance-level schema matching operation use information extracted from stored data instance in schema elements to find correspondence between the schema elements. As state in the background chapter, this dissertation make use of system catalogs of *DBMS* to get schema information, which are then used to perform element-level and structure-level schema matching.

Every schema matching operation in this dissertation awards a *similarity score* between *0-1*, to indicate the degree of similarity between two entities where *0* represents the least similarity.

3.1.1 Element-level Schema Matching

Element level schema matching operation is the process of finding correspondence between elements in the first schema with the elements in another schema. Schema elements can be at finest Granularity of schema like data type information of a column or can be coarser like table name or schema name[RB01]. Schema informations that are used for the element level schema matching includes table name, column name, column data type and column length. This information is fetched from system catalogs of *DBMS*. Based on the type of element used, element level matching is subdivided into name-based element-level schema matching and domain-based element-level schema matching. Taxonomy of element level schema matching approaches is as shown in figure 3.1.

Name-based Element Matching

Name-based element matching is the most intuitive matching technique. In this method, one element name in one schema will be compared with all the element names in another schema. Elements names represent the name of tables and names of column in the tables. Element names are essentially strings represented by a set of characters, hence, element name matching can be performed using string matching algorithms. Name-based element matching can be performed with or without use of context information. Hence, name-based schema matching is further classified into name-based element matching without context information and with context information.

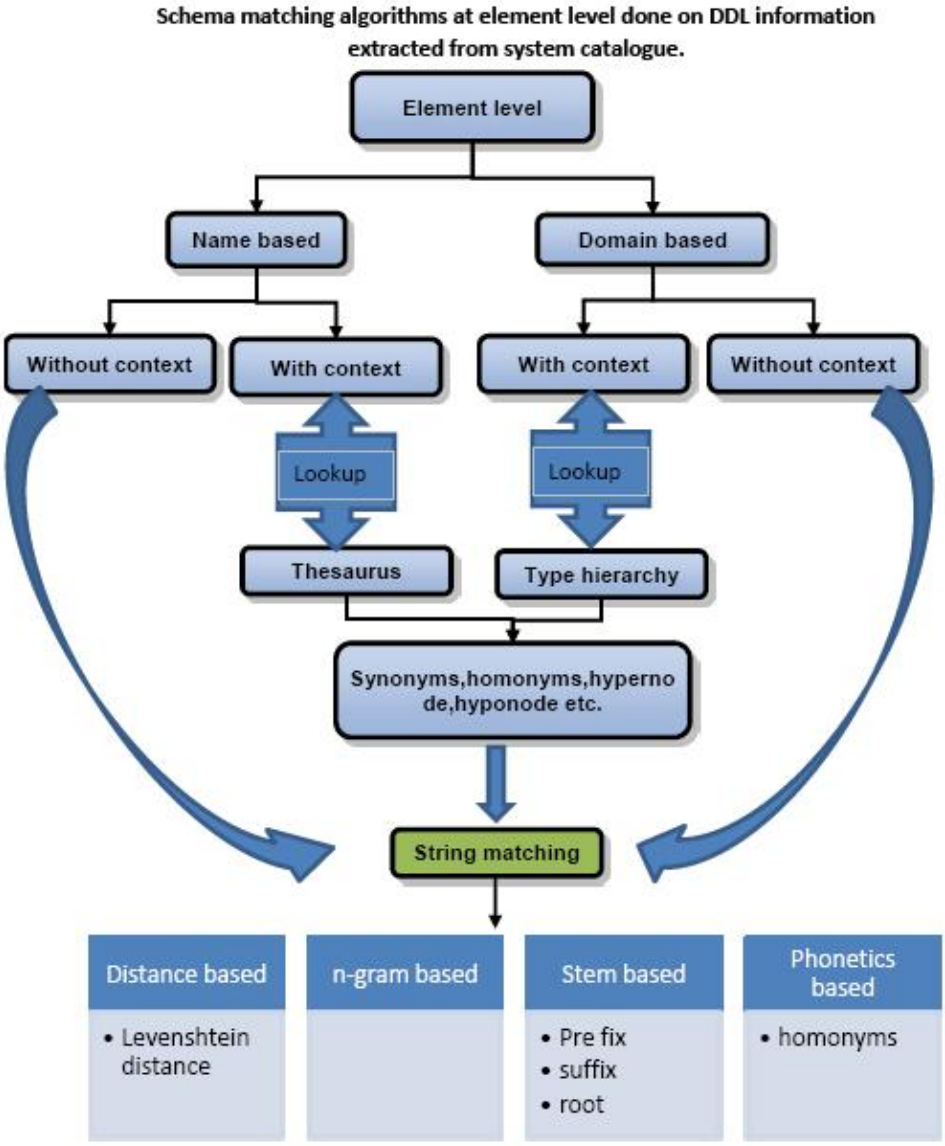

Figure 3.1: Taxonomy of Element Level Schema Matching.

Name-based Element Matching Without Context Information

Name-based element matching without use of context information is essentially a process of comparing two strings. Similarity between two names without the use of context information can be measured in following ways [RB01, SE05],

- Equal names
  If two element names are equal, they are considered to be representing same real world concepts and are awarded similarity score of *'1'*.

- Edit Distance
  This operation accepts two strings as input and calculates the number of edit operation required to transform one string into another.Number of edit operations are the number of character inserted, deleted or substituted.

- Canonical Name checking
  This operation checks for variation between input strings caused by use of prefix or suffix.

- N-gram checking
  This operation accepts two strings as input and checks for the number of common sequence of character of length *'N'* (*N-gram*). Large number of common set of characters between two strings represents greater similarity between two strings.

All above operations are string comparison operations and are discussed in more details in section 3.2.

Name-based Element Matching With Context Information

Use of context information means use of external information in the matching process. Name-based element matching with context information make use of generic thesauri *'WordNet'*. Checking similarity between two strings using context information involves following operations,

- Synonym checking
  Two string representing element names may be entirely different but can be synonyms of each other hence representing the same real world concepts.

For example, element names 'PROVINCE' and 'STATE' are entirely different strings but are synonyms of each other hence more likely to represent the same real wold concepts hence are awarded higher similarity score.

- Hyper name checking
  This operation takes in two input strings and checks for *is-a* relationship between the first and second string [RB01]. For example, strings 'SEA' and 'WATER BODY' are two different string but have *is-a* relationship between them and hence are more likely to be representing similar real world concepts.

- Hypo name checking
  This operation is similar to hyper name checking with order of input strings reversed.

- Homonym checking
  This operation takes in two strings as inputs and check if they are homonyms of each other. Homonyms are the same string but representing different real world concepts. External information source *'WordNet'* do no support homonyms as of now and hence this operation is deferred as future work.

Domain-based Element Matching

Schema elements are often associated with domain information. This information can be used to deduce mappings between elements of schemas. The process of deducing these mappings is called domain-based element-level schema matching.

Domain information represents the type of data that can be stored in an element of schema, for example, a column can store data of type integer, date, or string. Domain information also contains details about the constraints on an element in a schema, for example, element can contain unique data, not null data, or data from a defined range. Domain-based element matching finds elements in two schemas having similar domain information. For example, if an element in both schemas got same data range, then that elements are most likely to be representing similar real word entities. Domain based schema matching do not indicate strong correspondence between elements as many elements can have the same domain information. Hence, it will be used in combination with other matchers and will be assigned low weights.

Domain information of a schema is stored in system catalogs. This information can be fetch from system catalogs and can be used to perform domain-based element-level schema matching. This dissertation makes use of data type of a column and length of column for matching. Implementation of matchers, utilizing constraints of a column, and other domain information are deferred as future work. Domain-based element-level schema matching can be subdivided into one not using any context information and another using context information. Hence, this dissertation presents two matchers for domain-based element-level matching as follows,

Domain-based Element Matching Without Context Information

In this approach, domain information of an element in a schema is compared with domain information of all the elements of another schema. For example, data type of a column in a schema is matched with data type of every column in another schema. A similarity score is awarded when data types get matched. Each domain information is essentially a string represented by a set of characters. Hence, this dissertation uses string comparison techniques to perform domain-based element-level without context information schema matching. Domain information is often specific to *DBMSs* of data source. *DBMSs* limits the set of domain information that can be used to define a schema, hence, this context information can be very specific to the domain. This prototype uses edit distance string comparison technique to match domain information.

Domain-based Element Matching With Context Information

In domain-based with context element matching approach, an element in a schema is matched with elements in another schema having compatible data types. If a column in a schema with a data type can store data from a column in another schema having different data type, then the two data types are said to be compatible. For example, a column with *'string'* data type can store data of *'char'*; reverse is also true but with limitation of size of *'char'* data type column.

Data type compatibility can be represented in the form of data type hierarchy tree with nodes representing compatible data types. In data type hierarchy tree, a column with parent node data type can hold data of its own type, and of its, all child nodes types. For example, in *Oracle DBMS* data type hierarchy *VARCHAR* type can accommodate *NUMBER* type data, but *NUMBER* type node,

which is a child node, cannot accommodate *VARCHAR* type data. This type of context information can be exploited in domain-based element level matching, by looking up data type hierarchy tree and performing node name comparison. This dissertation presents use of following two types of context information for domain-based schema matching.

- Using of generic dictionary

  This dissertation makes use of generic dictionary *'WordNet'*, to check for data type compatibility. *'WordNet'* has limited support for domain specif data type compatibility.

- Use of domain specific mappings

  This dissertation stores data type compatibility tree and thus allow maintaining domain specific data type mappings. By traversing this tree, and doing node name comparison, domain specific data type compatibility is checked. Node name are strings; hence, node name comparison is a string comparison operation. This prototype only looks for equal node names.

Element level schema matching algorithms are explained in detail in chapter 5.

3.1.2 Structure-level Schema Matching

Structure-level schema matching is the process of finding correspondence between the elements of schemas with similar structure. Schema elements may differ in their structure, or may require some transformations such as, one element in a schema can be derived by combining two or more elements from another or by doing mathematical processing on the set of elements from one schema to obtain a single element of another[DLD+04]. For example, element 'Name' in one schema can be obtained by combining elements 'First name' and 'Last name' in another, or 'Gross salary' in one schema can be derived by performing mathematical addition operation on elements 'Basic salary' and 'income tax paid' of another. This constitutes *1:n* correspondence between one element of a schema to n elements of another with a function defining mathematical transformation [RB01, SE05].

Structure-level matching creates correspondence between elements by analyzing element structure. In ideal scenario, structure of two elements matches entirely producing *1:1* mapping. This project only checks for *1:1* mapping between structure of elements.

This project represents every schema in the form of graphs, with nodes representing elements of the schema (relation name and attributes names) and edges representing relationship between elements (is-a, has-a relationships). Graph comparison technique, graph edit distance, is used to calculate the similarity score between the structure of two elements. Graph edit distance calculates the number of node edit operation required to transform structure of one element into structure of another. Edit operations can be a node insert, node delete and node rename.

Calculating graph edit distance requires comparing node names. Node names are strings; Hence, node comparison is a string comparison operation. String equality operation is used to check for equal node names. A detailed discussion about structure-level matcher implementation appears in chapter 5.

3.1.3 Instance-level Schema Matching

Instance level schema matching aims at finding the schematic correspondence between elements of schemas by using information present in data instances of store. In real world scenario, element can contain millions of data instances. Presence of this large number of data instances makes schema matching operation at instance level different and more complex then schema matching at element or structure-level. For large data store, it is practically not possible to compare all data instance of an element in a schema with all the data instance of every element in another. Problem of instance-level matching also increasing in size and complexity with an increase in number of elements, due to the increase in search space[DLD$^+$04].

This project started with the an aim to implement matchers for all schema matching approaches, but due to time constraint on the dissertation we are not able to implement any matcher for instance-level schema matching. We only discuss an idea that can form the building block for instance-level schema matchers. To solve the hard problem of instance-level schema matching, this dissertation offers an idea to first find a sample of data instance which is conformant to the schema and probabilistically representing the actual data in the store. Sampling

can be done by randomly selecting data from the elements or we can employ different statistical techniques for sampling data like taking average, clustering or aggregation.

Instance-level schema matching operation can be divided into two algorithms. First algorithm can perform instance data comparison, and the second can use grammatical inference technique [SG10] to infer schema using sampled data and then perform element-level and structure-level schema matching to get similarity scores between elements of schemas.

Figure 3.2 shows the two approaches for performing instance-level schema matching. First algorithm does not make use of any context information and perform data instance comparison between two elements. This matcher can also treats every instance as a string; hence, instance comparison can be performed as a string comparison operation and thus can utilize all the string comparison operations discussed in section 3.1.1 to generate similarity scores. For example, for schemas representing student informations, an element *'COURSE'* can contain data like *'MSc Advanced Computer Science'* and another element *'COLUMN1'* can contain data like *'MSc Advanced Computer Sci'*.In this example though column names are different mapping can be found using data instance comparison which are similar. Elements sharing most number of similar data instances would be given higher similarity score.

Instance-level schema matching without context information can be inaccurate as different organizations often represent data in terminologies specific to their organization. Data sources can store the same information in different ways, for example, an element in a data sources can contain data as *'MSC ACS'* and another data source element can store the same information as *'Master in Science in Advance computer Science'*. In cases like this instance level schema matching without context information can have low element matching accuracy.

For second algorithm we present a novel idea of constructing a schema using data instance stored in the schema elements by applying grammatical inference technique [SG10]. [SG10] presents grammatical inference algorithms to merge states and developing k-testable machine from a reasonable set of data. [SG10] presents the implementation results of many grammatical inference algorithms like RPNI (Regular positive and negative inference) and EDSM (Evidence driven state merging). We can make use of grammatical inference algorithms to construct a finite state automaton (FSA) for every element. *FSAs* can be constructed

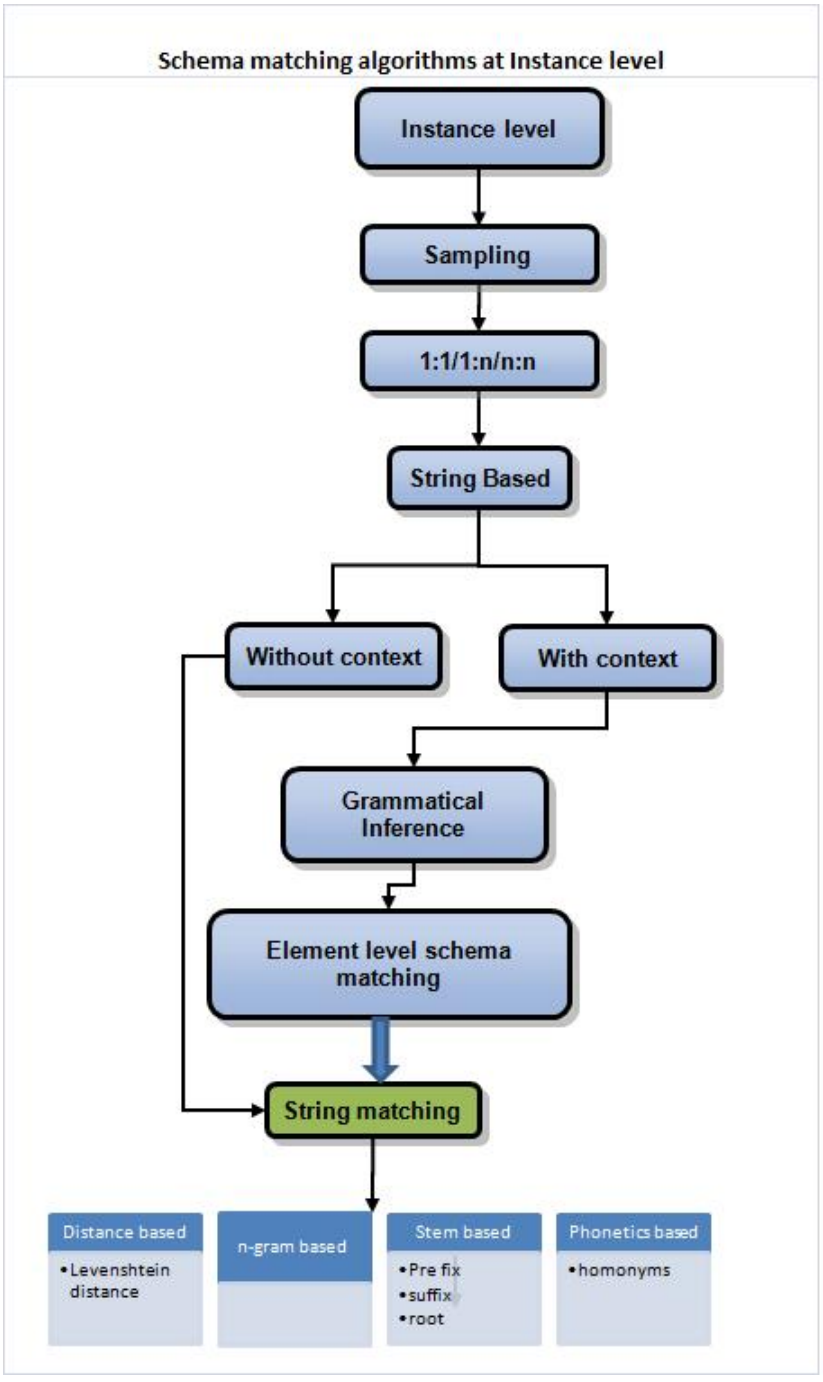

Figure 3.2: Taxonomy of Instance Level Schema Matching.

using data instances present in the elements; *FSA* represents an high-level description of the values that are legal in that column. FSAs can then be compared for equality, subsumption, difference, etc, and this can be used to postulate that two columns are equivalent, or one is a superset or subset of the other.

Grammatical inference can be carried out for all data sources under instance level schema matching process. Deterministic finite automaton given by grammatical inference can be treated as schemas which can accept data instance that are used to drive them. Instance-level schema matching can explore this idea of constructing internal (for processing purpose only) schemas. These constructed schema can then be be used to generate element correspondence by putting them through element-level and structure-level schema matching processes.

Data instances better represent an schema rather than element names which are prone to heterogeneity associated with the schema creation and thus can be unreliable to form the basis of the final schema matching results. Similarity score and correspondence generated by instance level schema matching using grammatical inference process should carry more weightage in the final matching result due to its inherent nature of using information in the actual data instances.

3.2 String Matching Algorithms

String matching algorithms are building blocks for element-level, structure-level and instance-level schema matching algorithms. All string matching operations take two strings as input and calculates their similarity score. In this project edit distance based, N-gram based and stem based string matching algorithms are used. A discussion about phonetics based string comparison technique is presented, but implementation is deferred as future works.

3.2.1 Distance Based

Distance based schema matching techniques takes two strings as inputs and calculates edit distance between them; This prototype uses Levenshtein edit distance technique of string comparison.

Levenshtein Distance

Levenshtein algorithm, also known as the edit distance algorithm, is a measure of the amount of difference between two strings. It is the least number of edit operations required to transform one string into another normalized by the length of the longest string [SE05]. Edit operations are character substitution, insertion and deletion. For example, the Levenshtein distance between *'raining'* and *'drain'* is 4. The following four edits operations transforms first string into another, and there is no way to do it with fewer than four edits.

- raining to draining (insertion of 'd' at the start)
- draining to drainin (deletion of g from end)
- drainin to draini (deletion of n from end)
- draini to drain (deletion of i from end)

similarity score is calculated using the following formula,

$$Similarity\ score = 1 - \frac{Edit\ distance\ between\ two\ strings}{Length\ of\ longest\ input\ string}$$

Two equal strings do not require any edit operation to transform one string into another, and will be assigned similarity score of *'1'*.

3.2.2 N-gram Based

This operation takes in two strings as inputs and calculates the number of common N-grams (N-gram is a sequence of *'N'* characters in a string) [SE05]. The number of N-grams common between the two strings, represents the degree of similarity between them; String with no overlapping N-grams are thus two different strings. Most commonly used N-grams are with value of N=1,2,3,4 and are called uni-gram, bi-gram, tri-gram and tetra-gram respectively. For example, there are *'9'* 2-grams common between strings *'INTERNATIONAL STUDENT'* and *'INT STUDENT'*. *COMA* system [DR02] discussed in chapter 2 uses bi-gram and tri-gram while this prototype uses tri-gram.

This project uses *Dice coefficient* [KMK03] to calculate the similarity score using N-gram string comparison technique.

$$Similarity\ score = \frac{2*Number\ of\ N\text{-}grams}{Sum\ of\ lengths\ of\ strings}$$

3.2.3 Stem Based String Comparison

Stem based string comparison technique takes two strings as inputs and checks if the first string is suffix or prefix of the second string [SE05]. If the first string is at the end of the second string then it is a suffix of the second string and if its at the start then it is a prefix. For example, string *'Name'* is suffix of string *'StudentName'*.

3.2.4 Phonetics Based String Comparison

In this dissertation we only present idea for phonetics based string comparison technique. Phonetics based string comparison operation takes two strings as inputs and checks if the two strings are homonyms of each other. Homonyms are the words having the same spelling but different meanings hence representing different real world concepts. Homonyms can mislead the distance based and N-gram based string matching algorithms, and thus can steals a higher similarity score. Phonetics based string comparison technique can make use of context information to check if two strings are homonyms. If two strings are the same, and also homonyms, then they may represent different real world concepts. In such cases, algorithm can give a negative similarity score thus lowering the overall aggregate similarity score.

This technique can look at the similarity scores generated by the instance level schema matching to filter out false positives. A high similarity score given by the instance level schema matchers underscores the fact that both the elements represents the same real world concepts as they have many similar data instances and hence are not homonyms. *COMA* systems use phonetics string comparison in one simple matcher called *Soundex* [DR02]. Phonetics based string comparison is left as a future work.

In this chapter, we discussed a taxonomy of schema matching approaches with detailed discussion of element-level, structure-level and instance-level schema matching techniques. The chapter also presented the string comparison algorithms which are used as building blocks for the schema matching algorithms. Next chapter presents the design approach adopted for prototype implementation of this project. Chapter also discusses the development methodology used to develop the prototype.

Chapter 4

Architecture

Chapter 3 presented and overview of schema matching approaches and string comparison algorithms. This chapter presents the design architecture of this dissertation's prototype implementation. The chapter presents the development methodology used to develop the prototype. Various design tasks are completed to develop the prototype; First the use case diagram is presented describing the available actions that user can perform. Chapter then discuss the flow of activities between components of prototype along with detailed discussion about each component. Sequence of activities are discussed using a activity sequence diagram. All diagrams are developed in unified modeling language (*UML*).

The primary objective of this project was to develop a library of schema matching algorithms for dataspace management system. To evaluate performance of schema matching operations, project developed a framework that allows to flexibly combine the individual schema matchers to perform schema matching. Developing prototype involved designing software architecture of prototype, developing a framework to evaluate performance of matcher combinations and implementing code. This project first developed an initial prototype with library containing element-level schema matchers. First performance of element-level schema matchers were evaluated using the evaluation framework and then library was extended to include structure-level schema matcher. This method of iterative development and evaluation fits the evolutionary model of software development. In following section evolutionary software development methodology is discussed.

4.1 Development Methodology

This dissertation followed an iterative evolutionary method of design, development and evaluation; this evolutionary model extended the capability of prototype in each iteration. Evolutionary model starts with the initial steps of design and development to come up with first prototype. This first prototype is evaluated of schema matching performance and set targets are validated, if targets are not met prototype is again put through design and development phase. If prototype performance is validated then prototype can be extended to develop even more requirements to come up with second prototype with enhanced capabilities by putting prototype through another design, development and testing phase. In this way prototype can be enhanced or evolved in an iterative evolutionary manner.

This project followed a model closely representing the evolutionary model with extra emphasis on evaluation phase. In every evaluation phase performance of schema matching library is checked. The matcher combination were reconfigured when the matching performance was not good. Once good results were achieved, the project was iterated through the design and development phase again to extend prototype's schema matching capability followed by another evaluation phase. This flow of project development is as represented in activity diagram of figure 4.2.

4.2 Prototype Design

4.2.1 Use Case

This project performs automatic schema matching but some user inputs are essential to facilitate schema matching. For example, user need to select matchers, if required reconfigure matcher combination, select schemas for matching, initiate schema matching and finally check the results. Available actions to the user are as shown in the use case diagram of figure 4.1.

As can be seen from figure 4.1; selecting schema includes accessing the data source details from data source configuration file. Selecting and configuring matchers would involve accessing matcher configuration file and performing actual schema matching would include using matchers from the library.

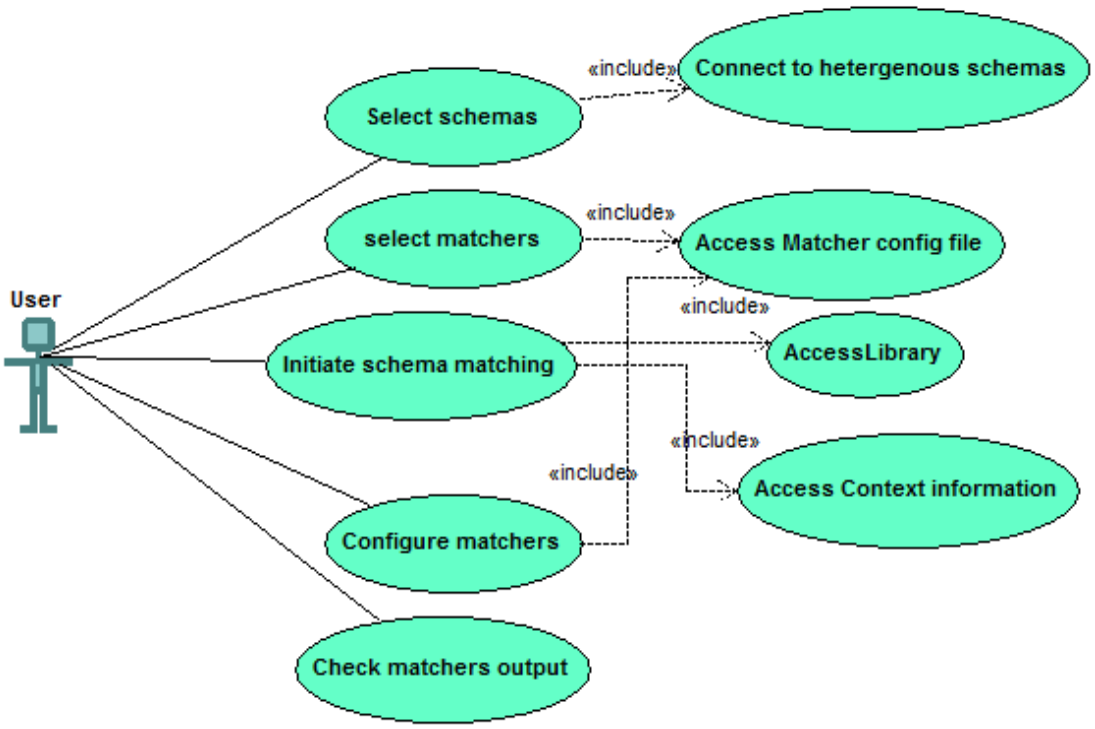

Figure 4.1: Actions Available to User

4.2.2 Flow Of Activities

Figure 4.2 represents the major activities of the prototype. As can be seen in the activity diagram, major activities of prototype to perform automatic schema matching are get schema details, get matchers, perform actual matching, store results of individual matchers and finally aggregate schema matching results.

4.2.3 Prototype Components

Every major activity is carried out by one or more components of the prototype. The components and the interaction between them is as shown in figure 4.3. Details of each component is as follows,

Main Wrapper

This component is the main driving component for all project activities. As shown in figure 4.3, this wrapper serves the user request to initiate schema matching process. Once request is received, main wrapper interacts with connection wrapper to get the details of the data sources from the data source configuration file. Connection wrapper establishes connection with the data sources and fetches all the available schema names in the data sources and returns them to main wrapper . This schema names are then displayed for user selection.

Main wrapper accepts the schema names from user and initiates schema matching. Main wrapper invokes *Run Matchers Wrapper(RunAlgos)* with schema names as inputs. RunAlgos fetches element information of the input schemas from system catalogs. RunAlgos then fetches matcher configuration details from matcher configuration file. Configured matchers algorithms are selected from the library and then executed individually to perform schema matching on input schemas. Results of schema matching operation performed by individual matchers are stored in *results data store*.

Main wrapper finally invokes *aggregator wrapper* to aggregate the results generated by individual matchers. Aggregator wrapper performs a weighted average of the similarity scores generated by matchers to get an aggregated schema matching result. Aggregated results are stored in *results data store* with a unique *run number* representing the complete end to end schema matching operation.

If user wants to further improve accuracy of the generated results, user can reconfigure matcher configuration file as seen in figure 4.1 and then re-trigger

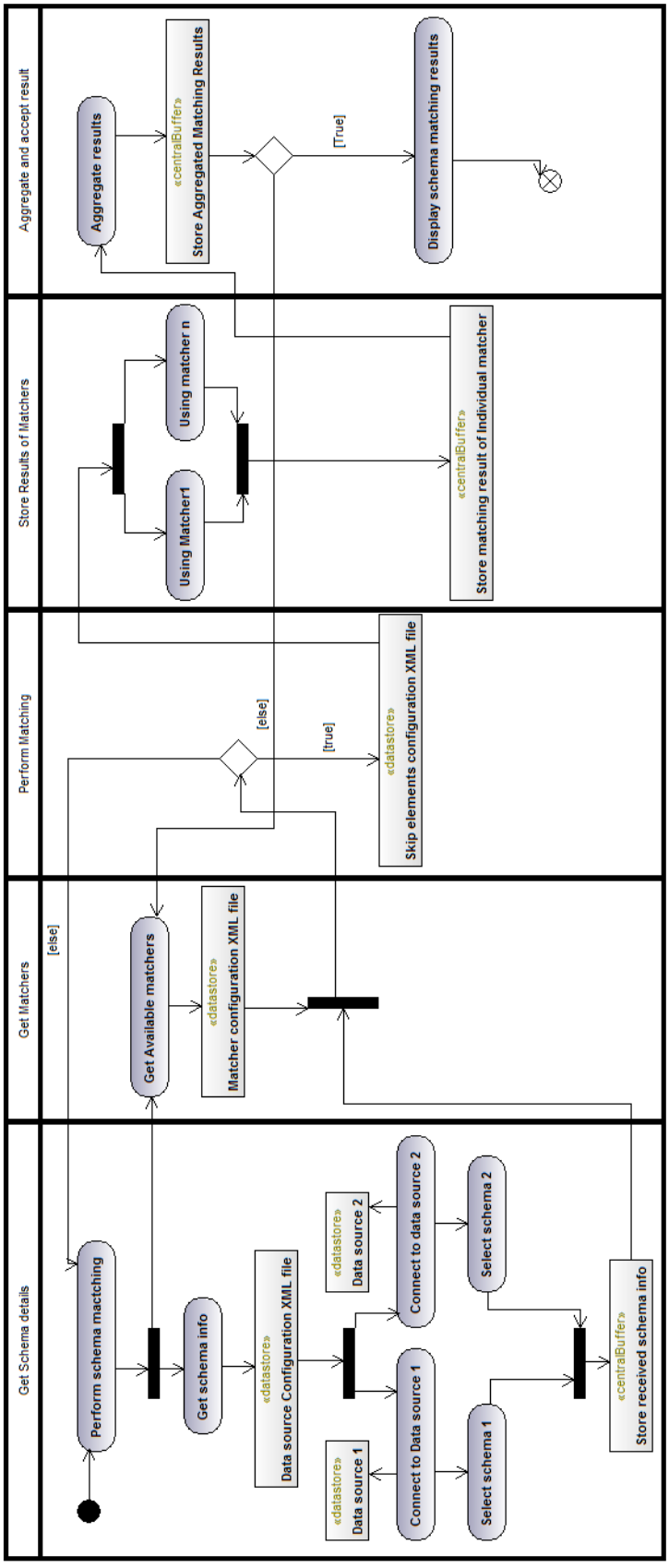

Figure 4.2: Project Activity Diagram.

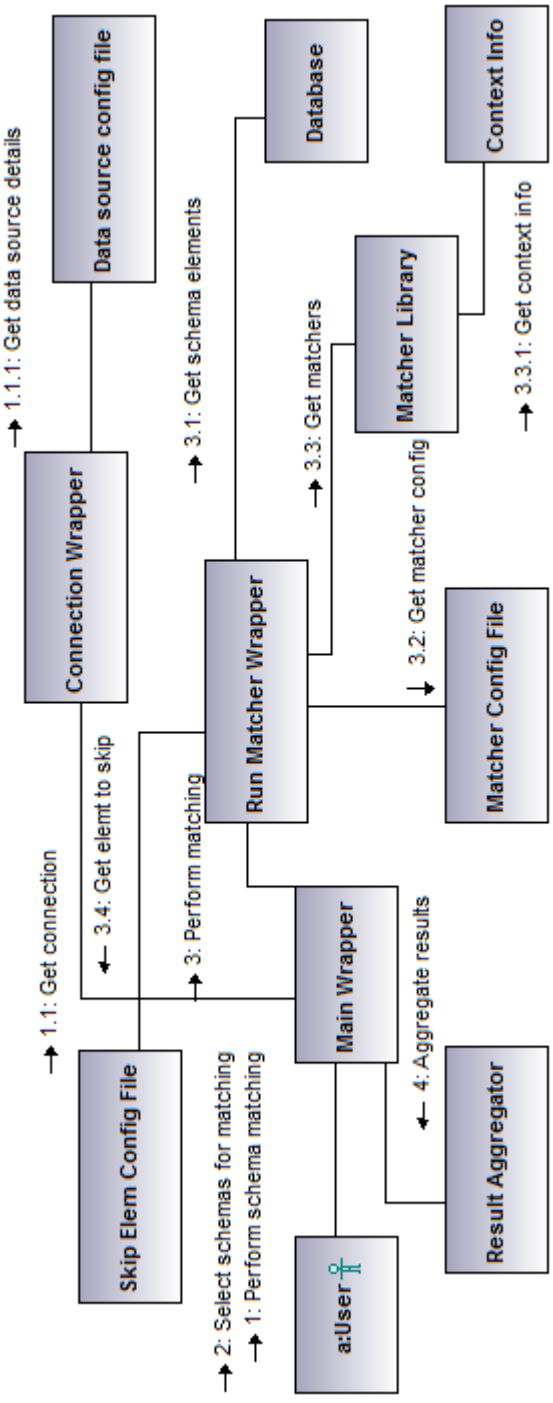

Figure 4.3: Project Component Interaction Diagram.

schema matching operation. This technique of improving the schema matching results is similar to incremental schema matching technique presented in [BMC06].

Connection Wrapper

Connection wrapper provides a generic interface to the main wrapper to connect to the heterogeneous data sources. Datasource informations are preconfigured in *data source configuration file*.This wrapper parses the configuration file to get the data source details. Connection wrapper uses data sources information in configuration file to establish connections to the data sources. Once connections are established available schema names are fetched from system catalogs of each data source and are then returned to the main wrapper. Connection wrapper also returns the connection session details to the main wrapper; this connection session details will be used by *RunAlgos* in further matching operation.

Run Matchers Wrapper

Run matchers Wrapper *RunAlgos* is the wrapper responsible for performing actual schema matching. Figure 4.3 shows the interaction of this component with other components in the system. This wrapper receives schema names and connection details as inputs from main wrapper. RunAlgos fetches the element details of each schema using schema name and connection details provided as inputs. RunAlgos then extracts details about elements to skip from the schema matching process by parsing the *skip element configuration file*. User can skip elements from schema matching process if mappings for those elements are already known. This project allow three fold control over skipping elements from matching process. User can choose to skip complete table including columns, only the table name, or skip one or more columns. Example *skip element configuration file* can be found in Appendix C.

RunAlgos then fetches the matcher configuration by parsing the *matcher configuration file*. This configuration files contains the name of the matchers to run, weights assigned to matchers and threshold values of the matchers. Threshold is an extra control provided in the prototype to control the behavior of matchers in the combination. If the element similarity score generated by a matcher is less the the assigned threshold value of that matcher, then only RunAlgos will execute the next matcher configured in the matcher configuration file. Threshold

control is not used in any evaluation experiments done in chapter 7. Example matcher configuration file is as as shown in appendix B.

RunAlgos stores the schema matching results of individual schema matcher in a data store. This data store is accessed by result aggregator wrapper to fetch results of individual matcher to perform aggregation of results. This data store acts as storage for all temporary and final schema matching results. Each user request to perform schema matching operation is given a unique *run number*. This number acts a unique key and is used to log final result of combination and matching result of individual matchers along with the matcher name.

Result Aggregator Wrapper

Each matcher in matcher combination generates different schema matchings and similarity scores for every schema element. These schema matchings results and similarity scores need to be combined and stored as an aggregated schema matching result for the entire matcher combination. As previously stated, individual matcher is assigned a weight in matcher configuration file; this weight is stored along with the matching results of the individual matcher. Result aggregator wrapper access the data store to get the results and weight of every matcher and aggregates them using weighted average method. Aggregated similarity score for each element is calculated as follows,

$$Weighted\ Average\ Similarity\ Score = \frac{SC\_m1 * W\_m1 + SC\_m2 * W\_m2 + ... + SC\_mn * W\_mn}{W\_m1 + W\_m2 + ... + W\_mn}$$

Where,
SC_mn is the similarity score of element generated by matcher n.
W_mn is the weight assigned to matcher n.

Library Of Schema Matchers

Library includes schema matchers that can be flexibly combined to form a schema matcher combination. User can configure a matcher combination using schema matcher configuration *XML* file. Library includes following matchers,

1. Element-level name-base without context matcher
2. Element-level name-base with context matcher

3. Element-level domain-base without context matcher

4. Element-level domain-base with context matcher

5. Structure-level schema matcher

Above schema matchers are discussed in detail in chapter 5.

Figure 4.4 describes the sequence of message flow between different components of prototype. It can be observed that *run matcher wrapper* component is the most processing intensive component of the whole prototype while *main wrapper* acts as a driving component for all the prototype activities.

This chapter described the prototype architecture with the help of *UML* diagrams. This Chapter also presented detailed working of major components. In the next chapter, individual schema matching algorithm is explained in detail.

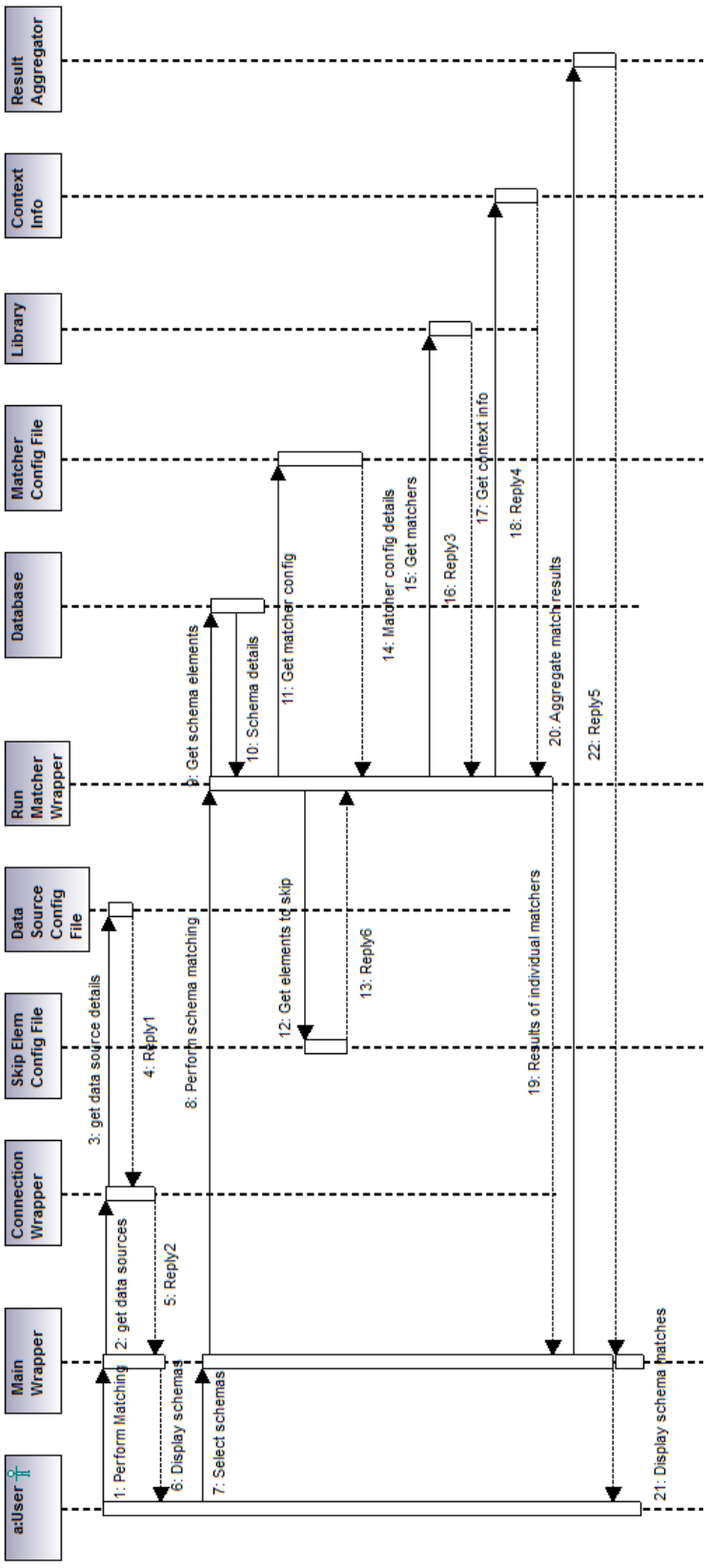

Figure 4.4: Project Message Sequence Diagram.

Chapter 5

Algorithms

Chapter 4 discussed the design of the prototype implementation. Chapter 4 presented the flow of activities, interaction between components and the sequence in which activities occur along with the detailed working of major components. This chapter presents a details discussion about schema matching algorithms in the library.

As discussed in chapter 3, this prototype implemented five individual matchers operating at element-level and structure-level. These matchers are independently configurable and have a generic input and output structure; Hence, they can be combined flexibly with each other. Following sections presents implementation details of these matchers.

5.1 Element-level Schema Matchers

This prototype implementation successfully instantiated the complete taxonomy of element-level schema matching approaches as presented in the figure 3.1. Library of schema matching algorithms contains following four element-level matchers.

5.1.1 Element-level Name-based Without Context

This matchers operates on inputs which represents element names. Element name can be a table name or a column name. This matcher calls all the string matching algorithms presented in this prototype to get the similarity scores. String

matching algorithms used in this matcher are levenshtein distance, N-gram, suffix and prefix stem based. Similarity scores generated by these string matching algorithms are then aggregated by taking an average of all similarity scores. Final similarity score is then returned to the *run matcher wrapper* which stores it to the result data store.

5.1.2 Element-level Name-based With Context

This matcher also takes element names as inputs. If input strings are not equal then the matcher calls the *APIs* to get the generic context information. This matcher checks if the first string is hypernym of the second string, if no relationship exists then it checks if first string is hyponym of second string.

WordNet is used to get the relationship between the element names. To check for the relationship between the two strings, WordNet constructs a tree representing relationship, with first string as the top node and if relationship exists then second string as leaf node. WordNet return an array containing this relationship with size of array equal to the height of the tree. If the two strings are distantly related to each other the size of returned array will be large.

Similarity score is calculated based on the depth of the array. For example, if relationship is found at depth of one then similarity score will be given 0.9 similarly for depth of 10 similarity score will be given as 0.09.

5.1.3 Element-level Domain-based Without Context

This matcher taken domain information as inputs. Data type of the column and the maximum allowed length of the values in the column are the two domain information this matcher operated on.

Levenshtein edit distance algorithm is used to calculate the difference between the domain information of two elements. For each column two separate similarity scores are calculated based on the edit distance of data type and length of the column. These similarity scores are then averaged to get the final similarity score for the column.

5.1.4 Element-level Domain-based With Context

This matcher taken only the data type of columns as inputs. As discussed in section 3.1.1 domain based with context matcher can use generic as well as

domain specific context information. This prototype implemented use of both the type of context information. Domain specific context information used in this prototype is as shown in appendix D.

This matcher checks if input data types are same, if not then checks if they are compatible using domain specific context information. Similarity score is calculated based on how far leaf is from the top node. Hence closely compatible data types are awarded higher similarity scores. For example, for *'char'* and *'string'* data types as inputs, a similarity score of 0.9 is awarded.

5.2 Structure-level Schema Matcher

As discussed in section 3.1.2, structure-level schema matching uses tree edit distance to calculate the similarity between the structure of two elements. In this prototype implementation, structure-level matchers operate at the table level.

This matcher accepts table names as inputs and constructs a three level tree with table name at top, columns at second level and data types at leaf level. The Goal of this matcher is to check only the structure of the table, hence, table name and column names in both tables are kept same to avoid node edit operations caused by difference in table and column names. Column having different data types are treated as having different structure; This approach is similar to approach taken in *CUPID* system for structure-level schema matching [MBR01].

This matcher calculates similarity score based on the number of node edit operations required to transform one table structure into another. The three supported node edit operations are as shown in figure 5.1. In figure 5.1 first table is transformed into another by carrying out one node edit operation, Hence, a similarity score of 0.9 will be awarded to such transformation.

This chapter presented a detailed discussion about the working of the schema matching algorithms. This chapter also presented which string matching techniques are used to build an individual matcher and how similarity scores are derived from string or node comparison operations. In the next chapter, implementation details of prototype is discussed along with the details about which application programming interfaces *APIs* are used in the prototype implementation of this dissertation.

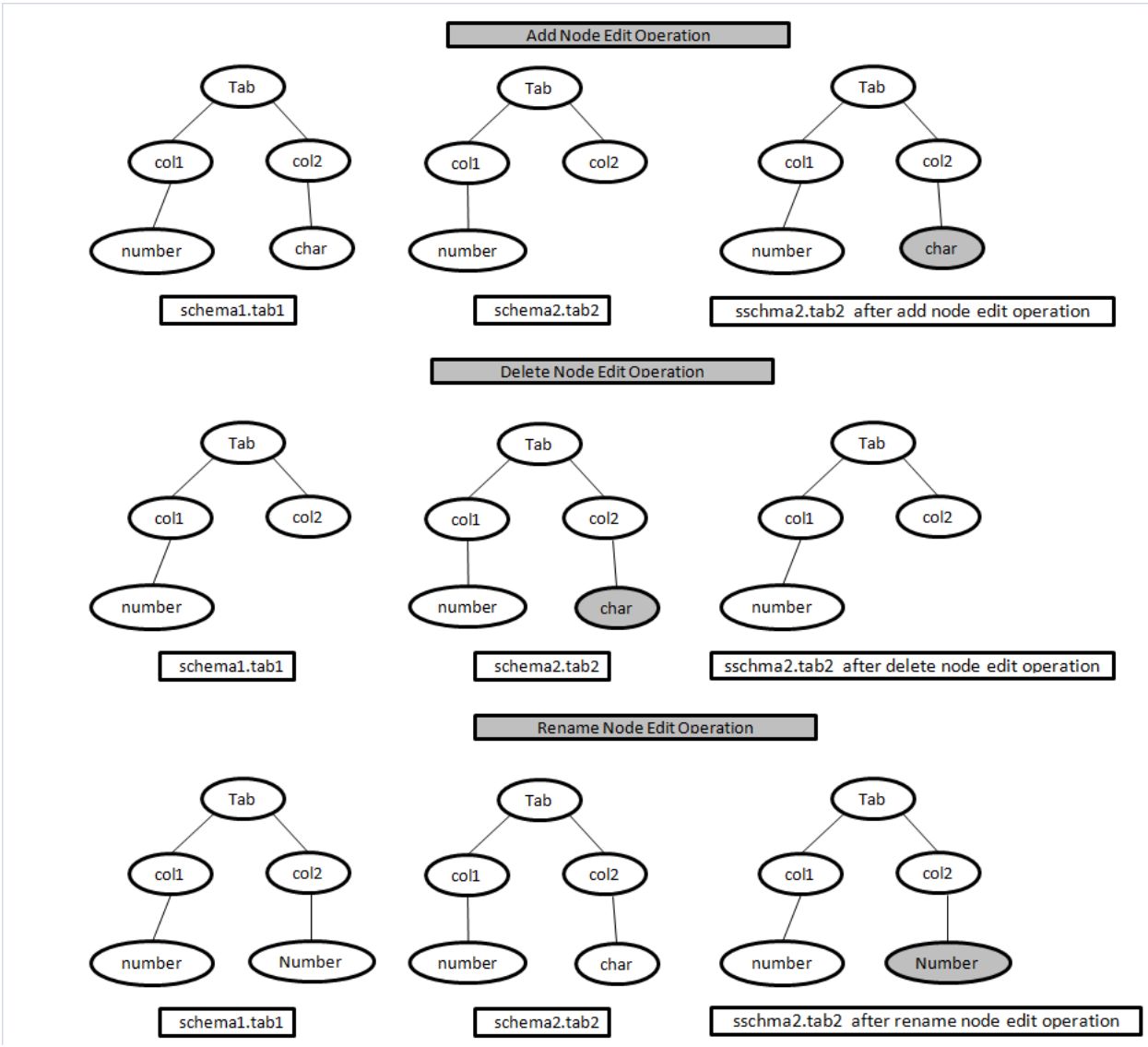

Figure 5.1: Tree Edit Operations

Chapter 6

Implementation

Chapter 5 presented the details about the matcher in the schema matching library. Previous chapter also discussed in details how similarity scores are calculated from string comparison operations. This chapter presents the development environment used for the implementation of this prototype. This chapter will also presents the *APIs* used in the prototype.

6.1 Development Environment

Development environment consists of hardware and software tools. This prototype is developed on Windows Vista 32-bit operating system running on a HP-Laptop. System is using a AMD-Turion X2-Dual core 2.10GHz processor and RAM used is 4GB in size.

Software tools used in the development of this prototype is summarized in the table 6.1 ,

Tool	***Description***
Eclips helios	IDE to develop applications in JAVA
Oracle 11g	*DBMS*
Java SE 6	Java development tool kit
SQL Developer	Tool to run *SQLs* and develop *PL/SQL* scripts

Table 6.1: Development Environment

6.2 APIs Used

6.2.1 SAX Parser API

Simple API for XML (SAX) is an event based API. SAX parses XML file sequentially. While parsing an XML file, SAX parser looks for specific events and executes the code specified for that particular event. This parser takes only one pass through the XML file and do not retain the XML file in the memory; Hence, this parser do not allow looking back in the XML file.

This API is used in the prototype because of its low memory requirement. This API is used to extract information from the configuration files in this prototype. Files parsed in this prototype using SAX parser are data source configuration file, matcher configuration file and skip element configuration file.

6.2.2 S-Match API

S-Match is a schematic matching framework [GSY04, sma11]. It takes input in the form a graphs and performs semantic matching on them. It find the relation between the nodes of graphs based on the information contained in the node labels and the structure of the graphs. S-Match contains a library of general purpose schematic matching algorithms which can be easily extended. S-Match can be used to perform schema matching and presents a hybrid matching technique. It supports element-level and structure-level schema matching.

This prototype implementation do not use S-Match to perform schema matching but only uses the string matching algorithms of the S-Match. String matching algorithms are modified to calculate the similarity scores rather then the relationship between the nodes. This API is not used in the prototype implementation as an external *JAR* library, rather only required string matching algorithms code is taken and modified as required.

6.2.3 JWNL API For WordNet

JWNL API facilitates accessing WordNet style dictionaries. JWNL provides procedures to discover relationships and morphological processing[jwn11] between string using WordNet dictionary. JWNL is developed as an open source project and can be used without restrictions.

This project use JWNL to access WordNet 2.0 dictionary data base. Using JWNL, this project finds relationships between strings and use them as context information. JWNL is also used to find hypernyms and hyponyms relationship of a string. This project also used JWNL in the mutation process of *'MONDIAL'* schema to create test schemas for evaluation. Details about test schemas can be found in chapter 7.

6.2.4 Graph Matching API

This API provides mechanism to perform tree edit distance operation[gra11]. This API implements tree edit distance algorithm presented in [ZS89]. API represents a tree as a string containing a series of ordered edges separated by semicolon. This API calculates the number of edit operation required to transform one tree into another. Edit operations supported are add node, delete node and rename node . Edit operations are performed based on the node names.

This project do not represent edges showing referential integrity; Hence, schemas represented as graphs would be basically trees. Hence, this project is able to uses this graph matching API to perform structure-level schema matching.

6.2.5 JDBC Driver

This prototype implementation uses the Java database connectivity *JDBC* API to connect to the Oracle 11g DBMS. This prototype used JDBC thin driver hence making this prototype platform independent [jdb11].

6.3 Context Information

As discussed in chapter 3, this project make use of both generic and domain specific context information. Domain specific context information is maintained in the form of a tree. This tree of compatible domain information is stored in a table which is populated manually. Example mappings for domain specific context information is as shown in appendix D.

6.3.1 WordNet

WordNet is a lexical database of English words. WorndNet supports nouns,adjectives,verbs and groups them as a set of cognitive synonyms [wor11]. WordNet resembles a thesauri as it groups words based on their meaning; However, WordNet also links words based on their sense. WorNet maintains records for semantic relationship between words. It supports discovering super-subordinate relationships such as hypernymy and hyponymy; discovering this type of relationship is crucial for this project. WordNet is easily accessible through JWNL API.

In this project *WordNet 2.1* is used as a source for generic context information. Context information fetched from WordNet is used in element-level name-based and domain-based schema matching.If a relationship is found using WordNet, a similarity score is awarded based on how closely the strings are related to each other.

This chapter presented the development environment used for this project. Chapter offered an overview of open source APIs used in this prototype and how APIs are modified to suit the needs of this prototype. Next chapter presents the results of experiments carried out to evaluate the performance of the library. The chapter also evaluates the performance of individual matchers and of combinations using different parameters and also draws some observations from these results.

Chapter 7

Evaluation

Chapter 6 presented the implementation details of this prototype implementation. It also presented details of *APIs* used in the development of the prototype. This chapter presents experimental results collected to support research hypothesis of this project. Experiments demonstrate how schema matching algorithms can be combined flexibly and how a particular matcher's impact can be controlled in a matcher combination.

Experiments show the impact of schema size on processing time, impact of schema size on matching accuracy of matcher algorithms, matching accuracy achieved by different matcher combinations. Impact of using context information on name-based and domain-based schema matcher performance is also presented. Impact of using structure level schema matcher on matching accuracy of name-based and domain-based schema matchers is also demonstrated. All experiments are also carried out on mutated schema to collect statistical evidence of the matcher and combination performance. These results demonstrate individual matcher algorithms ability to perform schema matching and what matcher combinations could produce good matching accuracy for a given schema having particular characteristics.

7.1 Experimental Design

All experiments are designed with the aim to get evidence of matching algorithms ability, ability of various algorithm combinations and to demonstrate flexibility of combining algorithms to get matching results depending on schema characteristics. Experiments are also designed to demonstrate the effect of schema size,

degree of dissimilarity and assigned weight to matchers on the matchers schema matching ability. Term matcher will be used interchangeably to refer to an individual schema matcher algorithm or combination of algorithms.

Matcher performance is calculated by comparing the matches generated by matcher algorithm or combination of algorithms (A) with the manually determined matches (M). True positive matches (TP) are matches correctly identified by the matcher algorithm. False positive matches (FP) are the ones that are not correct but are still identified as a potential match by the matcher.

As part of this evaluation precision, recall and F-measure values of schema matching performance will be calculated for each experiment when ever applicable. Precision, recall and F-measure, will have a value in the range from *0* to *1*. Precision of a matcher is defined as a proportion of the number of correct matches identified by matcher and the total matches returned by matcher i.e. TP+FP.

$$Precision(P) = \frac{TP}{A} = \frac{TP}{TP+FP}$$

Recall is the ration of correct matches identified and the real matches calculated manually that matchers are expected to find. Recall is calculated as,

$$Recall(R) = \frac{TP}{M}$$

F- Measure gives the accuracy of matcher's matching process. In all the experiments, balanced F-measure will be calculated which gives equal importance to precision and recall measures. Balanced F-measure can be calculated as,

$$F - measure(F) = \frac{2*P*R}{P+R}$$

7.2 Experimental Setup

All experiments for this project are executed on relation schema. Oracle 11g Database management system (DBMS) is used for creation and management of all test data source. Open source schema *'MONDIAL'*[mon11] which contains geographical, and general statistical information about all countries in the world is used as the primary reference test schema for all experiments. For the purpose of performing schema matching another relational schema *'MONDIAL_DUMMY'* is created which represents the same underlying concept of statistics about countries but using different element naming method, data types and structure. To create

'MONDIAL_DUMMY' schema table names, column names, domain information and structure of the tables in original *'MONDIAL'* schema are altered manually. Experiments conducted using *'MONDIAL'* and *'MONDIAL_DUMMY'* schemas represents a more realistic scenario that can arise in schema matching operations which calls for matching schemas whose elements are not identical but represents same real world concepts. Both schemas have 30 tables each with one to one correspondence done manually to form the reference baseline mapping, this manual matching of the schemas allows to measure the schema matching accuracy.

To collect the statistical evidence of every matcher's performance against different kind of mutations that can occur in a schema, five schemas are created by randomly mutating the reference *'MONDIAL'* schema. As the schema is successively mutated degree of similarity between the two schema decreases, and it becomes more and more difficult for the schema matching algorithms to identify the true base line element matches. Type of mutations done to *'MONDIAL'* schema are,

1. Character insertion
2. Character deletion
3. Random Character replace
4. Add element
5. Delete element
6. Replacing element

Element type in mutation action can be a table name, column name or domain information. Element type is selected randomly during schema mutation process. Replace element action randomly selects either table name, column name or domain information to replace. To replace element name with a new name, the new name is selected from *WordNet* dictionary. This action represents a scenario in which element names in schema matching operation are different but share a relationship that can be identified by use of context information facilitated by the use of WordNet dictionary. If action is to replace domain information then a random table is selected from the set of tables, from this table a random column is selected to change the domain information.

Changing domain information involves either changing data type or column length this is also done randomly. Compatible data types are fetched from WordNet or from a table of compatible data types. This project supports both method of fetching compatible data types, for the purpose of evaluation of schema matching performance for mutated schema compatible data type table method is used. If no related element name or domain information is found in WordNet or in compatible data type table, then the same string is returned without replacement.

For the purpose of evaluation ten tables of reference *'MONDIAL'* schema are mutated. Actual manual matching (M) is done between the original 10 tables of *'MONDIAL'* schema and table of schemas created after mutation, this manual mappings forms the reference base line matches against which schema matching performance of every matcher or matcher combination will be measured. Five schemas are created after 20 successive mutations of the *'MONDIAL'* schema as,

1. *'MONDIAL_S1'* = 20 mutation of *'MONDIAL'* schema.
2. *'MONDIAL_S2'* = 20 mutation of *'MONDIAL_S1'* schema.
3. *'MONDIAL_S3'* = 20 mutation of *'MONDIAL_S2'* schema.
4. *'MONDIAL_S4'* = 20 mutation of *'MONDIAL_S3'* schema.
5. *'MONDIAL_S5'* = 20 mutation of *'MONDIAL_S4'* schema.

Hence *'MONDIAL_S5'* schema is successively mutated 100 times from the original reference *'MONDIAL'* schema.

Evaluation is performed by executing various activities allowed by the developed system. Various actions available in the system are outlined by use case diagram of Figure 4.1.

Basic flow of each experimental evaluation is as shown in 7.1,

As shown in 7.1 to perform each experimental evaluation, details of schemas present in the data sources that are to be matched and configuration of schema matching algorithm are passed as inputs to the system developed as part of this project. Algorithm configuration includes details about which schema matching algorithms will be part of combination and weight assigned to them. For example in below example configuration file matchers Name-based element without context information with weight *'1'*, Name-based with context information with weight *'0.5'* and structure-level matcher with weight *'0.8'* are configured to be executed.

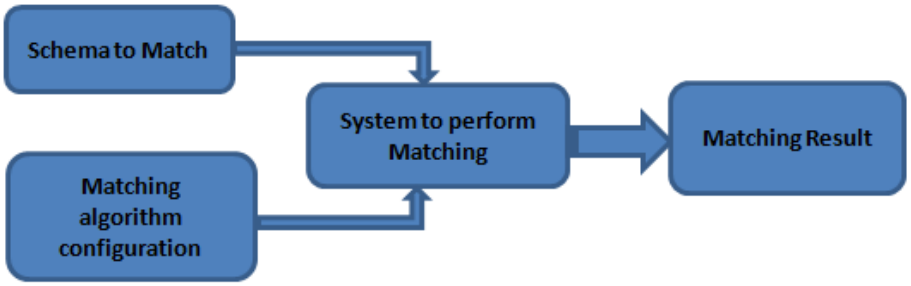

Figure 7.1: Evaluation Environment

```
<algoConfiguration>
     <elementLevel>
      <matchers>
       <elementLevelNBWoCon joinThreshold="1" weight="1" run="y">
       </elementLevelNBWoCon>
       <elementLevelNBWCon joinThreshold="1"  weight="0.5" run="y">
       </elementLevelNBWCon>
       <elementLevelDBWCon joinThreshold="1"  weight="1" run="n">
       </elementLevelDBWCon>
       <elementLevelDBWoCon joinThreshold="1"  weight="1" run="n">
       </elementLevelDBWoCon>
       <structureLevel joinThreshold="1"  weight="0.8" run="y">
       </structureLevel>
      </matchers>
     </elementLevel>
</algoConfiguration>
```

Prototype executes configured algorithms and performs matching of input schemas. Result of schema matching operation is produced and stored in temporary schema created for result aggregation and logging of results. Result of Schema matching operation includes mapping generated for each table in the first schema to its best match in second schema along with the column to column mapping between the matched tables.

7.3 Experiment 1: Impact of schema characteristics on matching accuracy of individual matchers

In this experiment, equal weight of *'1'* is assigned to all matchers. First three graphs represent evaluation done on test schema *'MONDIAL_DUMMY'*. Schema size is increased from 10 tables each to 30 tables for both schemas under matching operation. For final evaluation represented by graph 7.5 five mutated schemas are used.

Schema size is an important factor effecting matching results as it defines the search space for matcher algorithms. Increase in schema size means large search space for matchers resulting in greater choices available with matcher algorithm to match a particular element in one schema to a element in another schema. This increase in potential match candidate choices impacts accuracy of schema matching algorithms or combination of algorithms.

Figure 7.2 shows results for schema matcher precision with increasing schema size having fixed weight of *'1'* given to individual matcher.

In graph 7.2 Y axis represents the impact on precision of the matching algorithm with the increase in schema size represented on X-axis. Following can be observed from results,

- Precision of structure level schema matcher decreases with the increase in schema size. This decrease in precision is due to increase in number of potential candidate match choices having the same structure available to structure level schema matcher.

- Precision of matcher *Name based without context* information remains above 0.85 even with increasing schema size. This high matching precision value is due to the similarity in strings used as element names in schemas.

- Precision of matcher combination containing all the matchers remains at same high level of above 0.85 for all schema sizes.

Graph in figure 7.3 represents recall value for individual matcher algorithms and combined recall value of matcher combination containing all individual schema matchers.

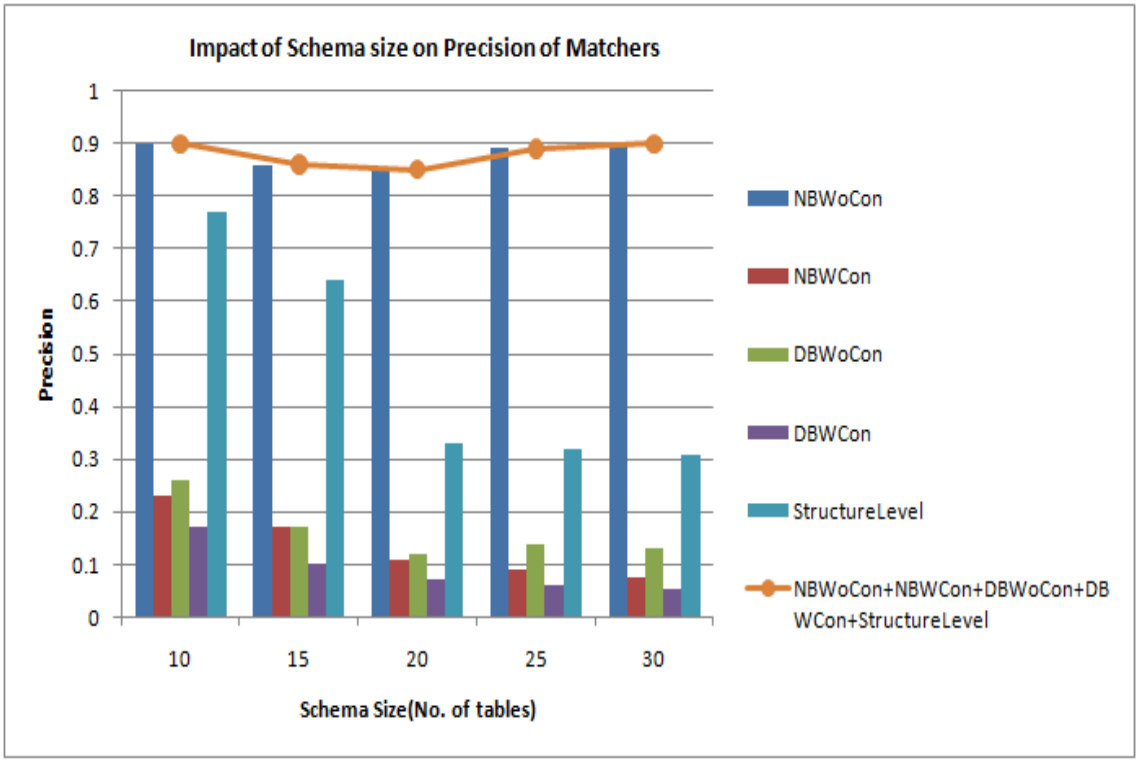

Figure 7.2: Impact of schema size on precision of matchers

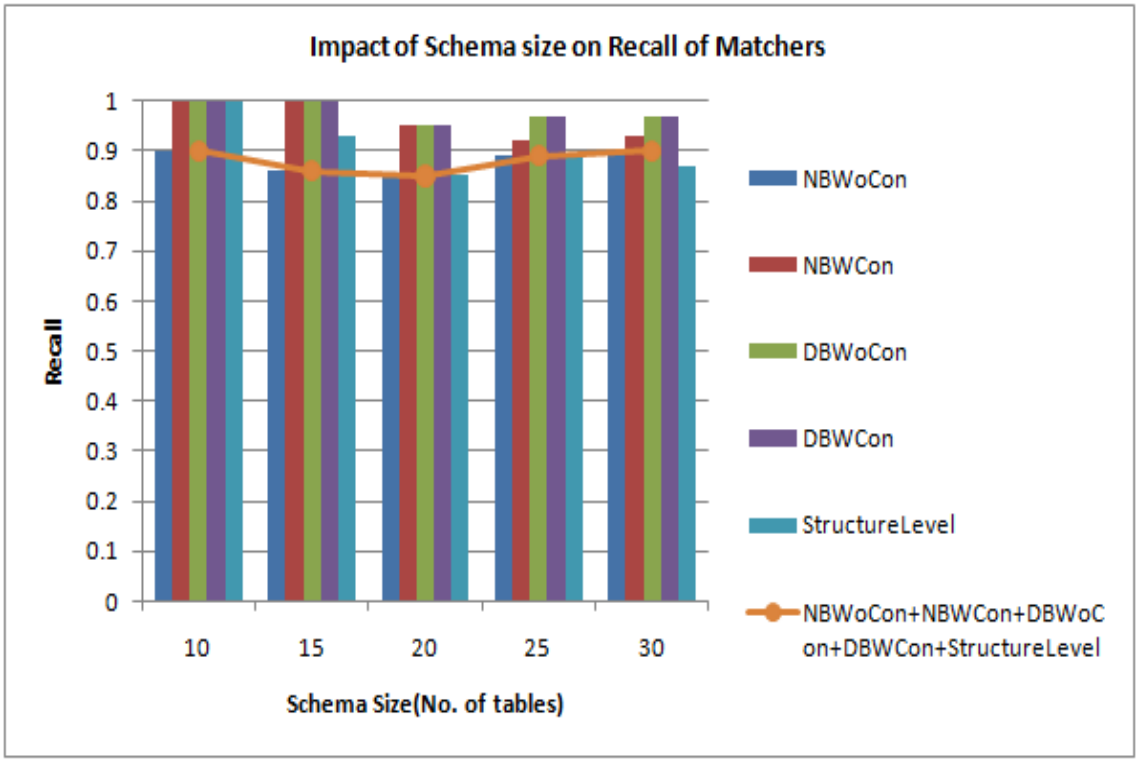

Figure 7.3: Impact of schema size on Recall value of matchers

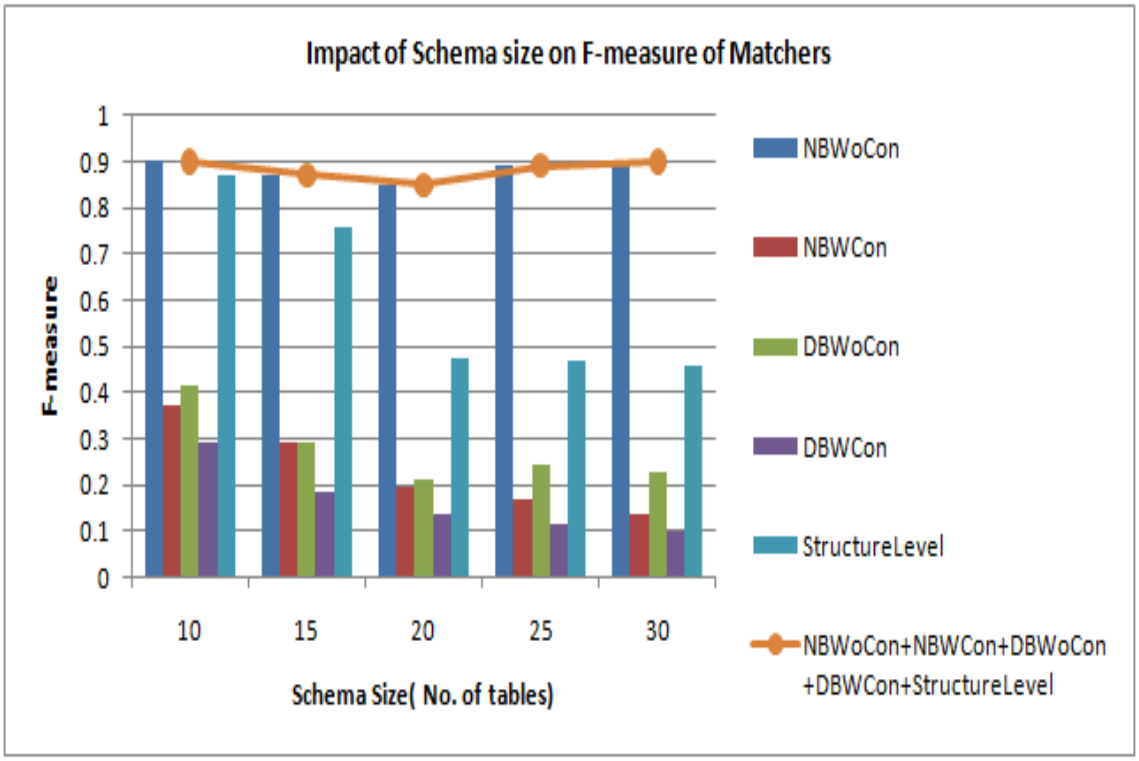

Figure 7.4: Impact of schema size on Accuracy of matchers

In graph 7.3 Y axis above represents the impact on recall of the matching algorithm with the increase in schema size on X-axis. Following can be observed.

- Share of correct matches identified remains consistently above 85 percent for every matcher.
- Domain base matchers return correct matches consistently even with increased search space. This consistently high recall value means both schemas have similar or compatible domain information.

Graph in figure 7.4 represents results for F-measure values for increasing schema size.

In graph 7.4 Y-axis represents the impact on the accuracy of the matching algorithm with the increase in schema size on X-axis. Following can be observed,

- Matching accuracy of Name-based without context matcher remains consistently high for all schema size.

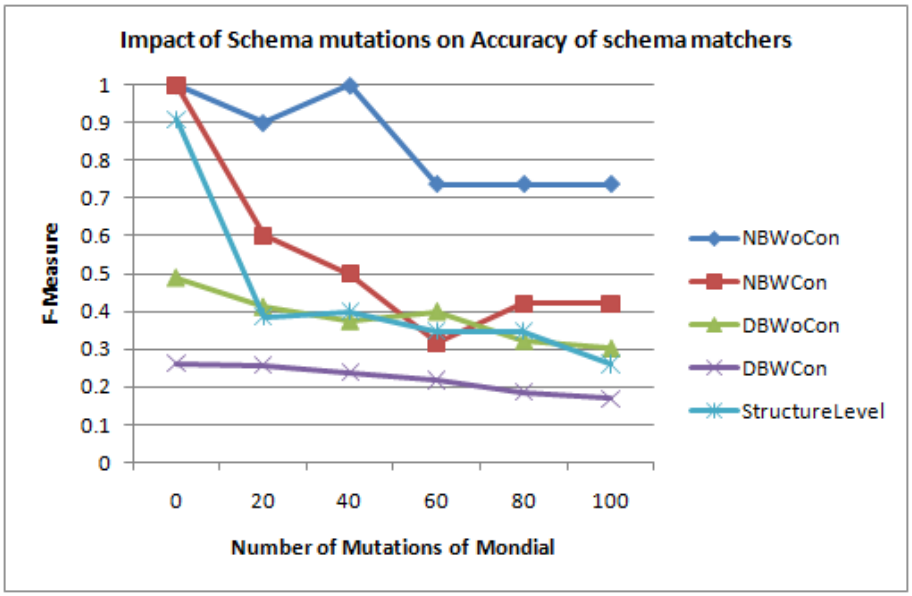

Figure 7.5: Impact of the number of schema mutations on matching accuracy of individual matchers

- Matching accuracy of structure-level schema matcher is above 0.7 for small schema size but plateaus to below 0.5 when search space becomes large.
- Matching accuracy of Domain based matchers remain below 0.4 due to the fact that many tables can have the same number of elements with similar domain properties. This cause domain based schema matchers to return many False Positive matches lowering accuracy of matcher.
- When matcher combination contains all individual matchers, combined matching accuracy remains consistently high at above 0.85.

Result for matching accuracy of individual matchers for randomly mutated schemas is represented in graph 7.5. It can be observed that matching accuracy of all the matchers decreases with the increase in number of mutation. This decrease in schema matching algorithms accuracy is due to decreasing similarity between reference *'MONDIAL'* schema and successive mutated schemas. It can be observed that accuracy of name-based without context information matcher plateaus after initial fall.

In graph 7.5 Y-axis represents the impact on the accuracy of the matching algorithm with the increase number of schema mutation as represented on X-axis.

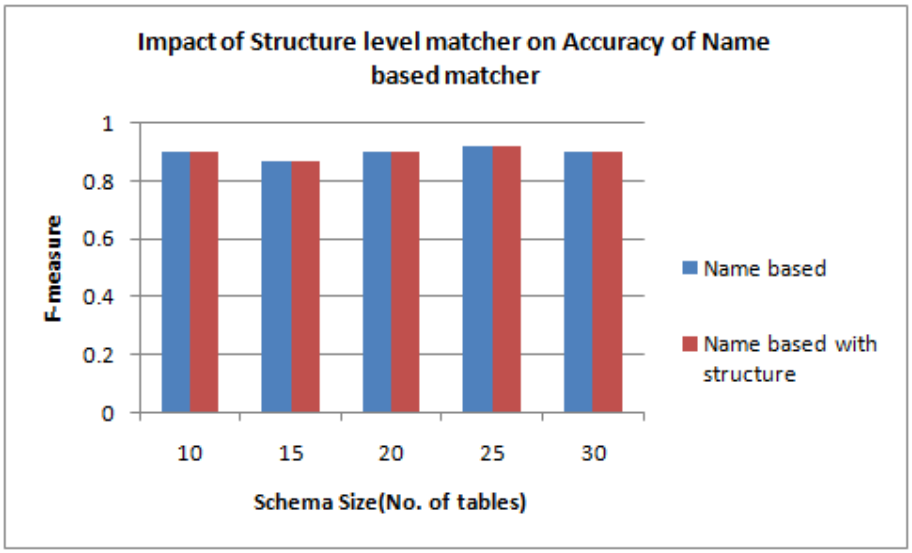

Figure 7.6: Impact of Structure level matcher on schema matching accuracy of Name based matcher

7.4 Experiment 2: Impact of combining structure level schema matcher with name-based schema matchers

In this experiment, weight assigned to all matchers is kept constant at *'1'* and schema size is increased from 10 tables each in both schemas to 30 tables each. Name based matcher in this experiment includes algorithms for name-based matcher without context information and name-based with context information. Results of impact of including structure information using structure-level schema matcher on schema matching accuracy of name-based matcher is represented in graph 7.6.

In graph 7.6 Y-axis represents the impact on schema matching accuracy of name-based matcher with and without structure level matcher. X-axis represents increasing schema size of *'MONDIAL_DUMMY'* and *'MONDIAL* schemas. It can be observed that there is no significant change in the matching accuracy of name based matcher after inclusion of structure-level schema matcher. This result is dependent on schema characteristics. Above results suggest that both schemas have a similar structure hence inclusion of structure-level schema matcher is not causing further increase in the accuracy of schema matching operation.

For example, structure level schema matcher will cause an increase in accuracy in the following scenario,
Consider table tab1 (col1 number, col2 number, col3 number) in *schema1*
and tab1 (col1 number, col2 number) and tab2 (col1 number, col2 number, col3 number) in *schema2*
When *schema1* is matched with *schema2* with only name based matcher, name based matcher will match tab1 of schema1 to both tab1 and tab2 of schema2 with col3 of schema1.tab1 matched to any one column of schema2.tab1 even after that column is already matched to one of the column of schema1.tab1. This is because there are only two columns in schema2.tab1, though the score of the match between schema1.tab1 and schema2.tab2 will be higher. When name based matcher is combined with structure level schema matcher then schema1.tab1 will be correctly matched to schema2.tab2 as both have the same number of columns hence the same graph structure.

7.5 Experiment 3: Impact of combining structure level schema matcher with domain based schema matchers

In this experiment, weight assigned to matchers is kept constant at *'1'*. First three graphs represent evaluation done on test schema *'MONDIAL_DUMMY'* for which schema size is increased from 10 tables each to 30 tables for both schemas under matching operation. For final evaluation represented by graph 7.10 five mutated schemas are used. In this experiment, domain based schema matcher includes *domain based without context* schema matching algorithm and *domain base with context* schema matching algorithm.

Columns in a table can be defined with different domain information. Domain information can differ in data type of data that can be stored in column or the maximum allowed size of data. Columns can be defined with limited combination of domain information restricted by *DBMS* in use, but in large data sources, one table can have many columns. This leads to schema containing many columns having similar or compatible domain information. This results in one column of a table in a schema potentially getting matched to many columns having similar domain information in another schema by *domain-based schema matchers*.

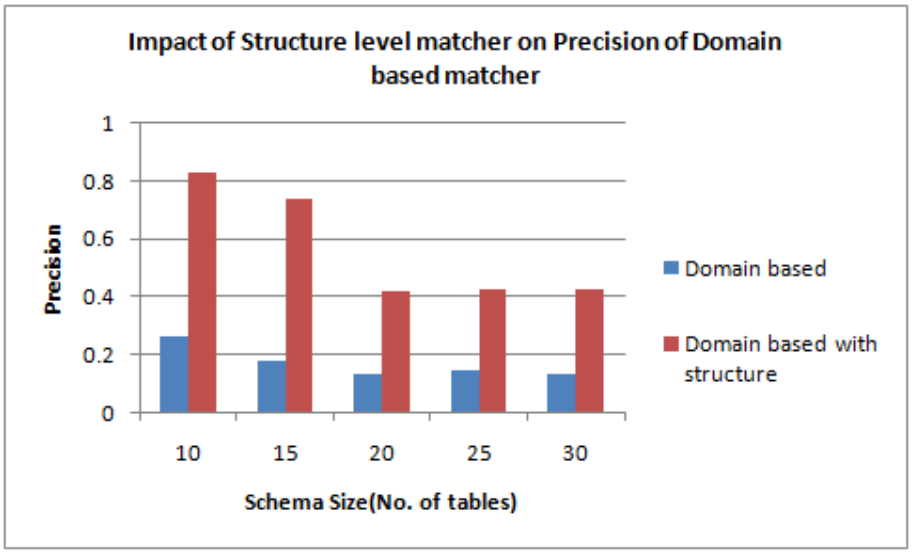

Figure 7.7: Impact of Structure level matcher on Precision of Domain based matcher

This wrong matching of column due to similar or compatible domain information causes domain based matchers to return many false positive matches. With the increase in schema size, this problem gets multiplied.

Figure 7.7 represents impact of structure level schema matcher on domain-based schema matcher precision. It can be observed that precision of domain based schema matcher is low due to many false positive matches caused by similar or compatible domain information of table columns. When domain based matcher is combined with structure level schema matcher, precision increases many folds because structure level schema matcher eliminates false positive matches having different structure. It can be observed that matcher combination's precision is high for small schema size but plateaus to above 0.4 with increasing schema size. Thus precision of domain based matcher is capped by precision of structure level matcher and confirms to the observed behavior of individual structure-level schema matcher as seen in figure 7.2.

In graph 7.7 Y-axis represents the impact on precision of domain based matcher with and without structure level matcher with increasing schema size represented by X-axis.

Graph 7.8 represents the impact of combining structure-level schema matcher with domain-based schema matcher's recall value. It can be observed that recall value for domain based schema matcher is consistently high to maximum possible

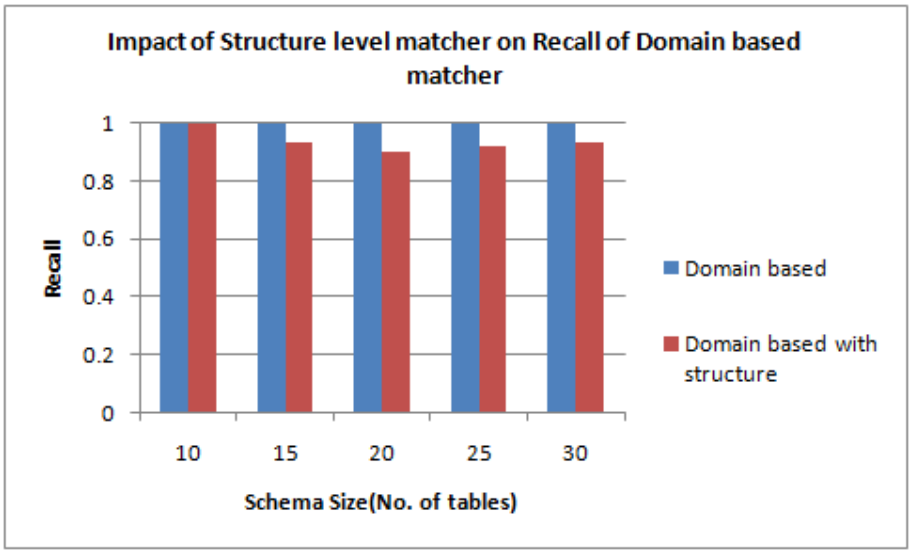

Figure 7.8: Impact of Structure level matcher on Recall of Domain based matcher

value of *'1'*. This is due to the fact that domain-based matcher returns many false positive matches because domain based matcher tends to match all table having at least one column with similar or compatible domain information.

For example, consider table tab1 (col1 number, col2 char,col3 char) in schema1 and tab1(col1 number,col2 char) and tab2(col1 number,col2 char,col3 char) in schema2
Domain based matcher will return both schema2.tab1 and schema2.tab2 as a potential match for schema1.tab1 as it matches column col2 of schema2.tab1 having *char* data type twice to both col2 and col3 of schema1.tab1 which is incorrect. When combined with structure level schema matcher, schema2.tab1 false positive match will be filter out as schma2.tab1 got a different structure having one less node when represented as a graph. This elimination of false positive matched due to the difference in structure increases precision of matcher algorithm combination. Recall value of domain based matcher with structure level matcher is less than recall value of domain based matcher without structure level matcher because there are instances in *'MONDIAL'* and *'MONDIAL_DUMMY'* schemas, where eliminated match due to structural difference was in fact, the actual intended match as confirmed by manually matched reference baseline matches.

In graph 7.8 Y-axis represents the impact on recall value of domain based matcher with and without structure level matcher with increasing schema size

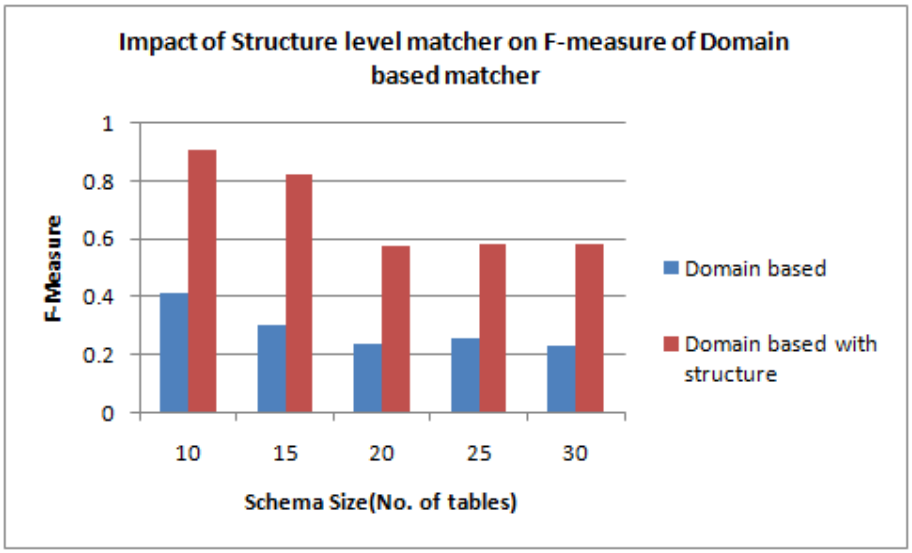

Figure 7.9: Impact of Structure level matcher on Accuracy of Domain based matcher

represented by X-axis.

Impact of structure level matcher on domain based matcher's schema matching accuracy is represented in figure 7.9. Following observations can be made,

- Accuracy increases when domain based matcher is combined with structure level matcher as false positive matches are filtered out by structure level schema matcher.
- Accuracy of matcher combination reduces with the increase in schema size but plateaus to above 0.57. This confirms to behavior of structure level schema matcher observed in figure 7.2.

In graph 7.9 Y-axis represents the impact on Accuracy value of domain based matcher with and without structure level matcher with increasing schema size represented by X-axis.

Impact of combining *structure level* schema matcher with domain based matcher is evaluated on randomly mutated schema in this experiment. Impact of the number of mutations on schema matching accuracy is represented in graph 7.10. It can be observed that schema matching accuracy increases when domain based matcher is combined with *structure level* matcher.

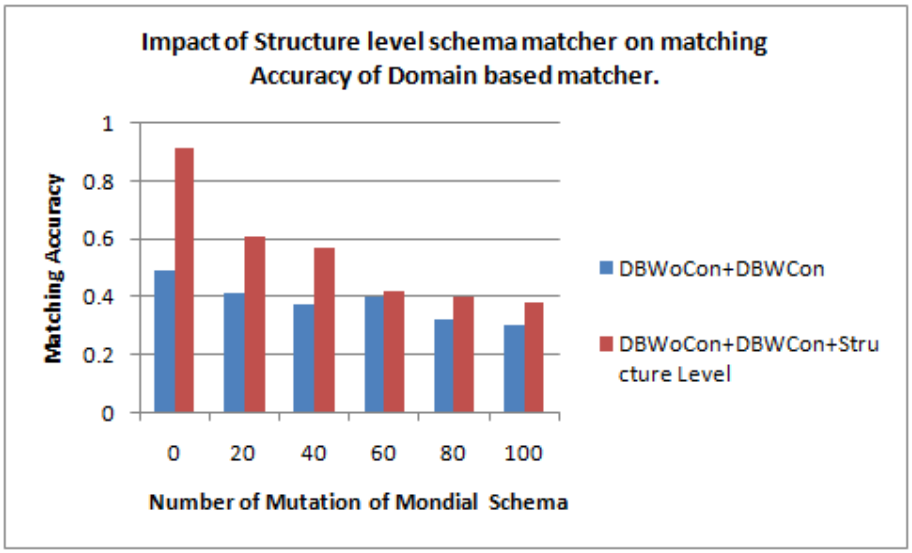

Figure 7.10: Impact of Structure level schema matcher on matching Accuracy of Domain based matcher for mutated schemas.

In graph 7.10 Y-axis represents the impact on Accuracy value of domain based matcher with and without structure level matcher with increasing number of mutation of *'MONDIAL'* schema represented by X-axis.

7.6 Experiment 4: Impact of using context information with name-based and domain-based schema matchers

In this experiment , weight assigned to all matchers is kept constant at *'1'*. For first five evaluation *'MONDIAL_DUMMY'* schema is used with increasing schema size from 10 tables each in both schemas to 30 tables each. Impact caused by inclusion of context information or auxiliary information on matching performance depends on characteristics of schemas under consideration. If two schemas represent same real world concept but employ diverse element naming techniques, then use of context information will result in an increase of schema matching accuracy only when element names or domain information have some relationship that can be identified by exploring context information.

Below figure represents the impact of use of context information on schema matching accuracy of namebased matcher. It can be observed that use of context

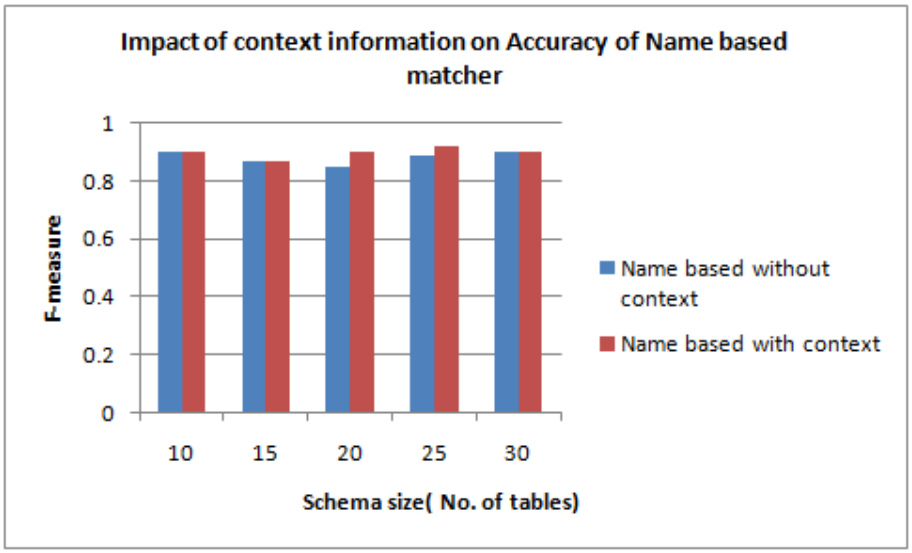

Figure 7.11: Impact of Context information on Name based matcher

information always either increases matching accuracy or keeps the accuracy at the same level as without the use of context information. It can be observed that there is an improvement in schema matching accuracy for schema size 20 and 25 as relationships are identified between element names by use of context information obtained from *WordNet* dictionary.

In graph 7.11 Y-axis represents the impact of use of context information on Accuracy of name based matcher with increasing schema size represented by X-axis.

Graphs 7.12, 7.13, 7.14 represents the impact of combining domain based schema matcher with context information. As explained before use of context information in domain based matcher allows matchers to match elements with compatible data types, for example *'char'* data type can be matched with *'varchar2'* or *'string'* data type. Hence domain based schema matcher with context information will cause an increase in matching accuracy if actual match involves matching compatible data types. Improvement of schema matching accuracy using context information with domain based schema matcher is dependent on schema characteristics. User need to make an informed choice whether to use domain based matcher with context information or domain based matcher without context information in matching operation so as to avoid increased overhead associated with exploring context information.

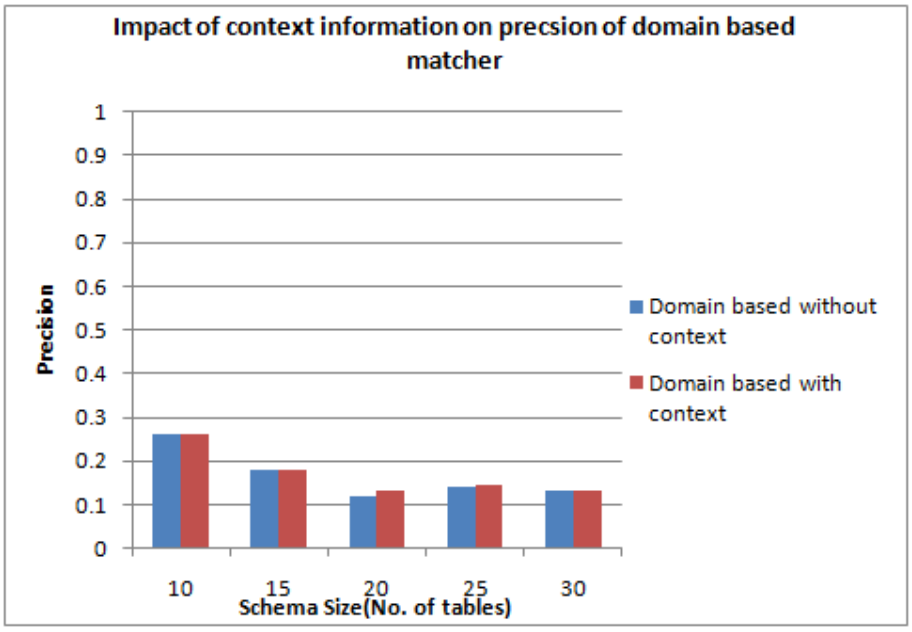

Figure 7.12: Impact of Context information on Precision of Domain based matcher

It can be observed from graphs 7.12, 7.13, 7.14 that use of context information do not have a significant impact on precision but improves recall value. Increase in recall value is because use of context information allows identification of matches with compatible domain information. It can also be observed that accuracy of domain based matcher improves with a small amount due to inclusion of context information.

In graph 7.12 Y-axis represents the impact of use of context information on precision of domain based matcher with increasing schema size represented by X-axis.

In graph 7.13 Y-axis represents the impact of use of context information on recall value of domain based matcher with increasing schema size represented by X-axis.

In graph 7.14 Y-axis represents the impact of use of context information on schema matching of accuracy of domain based matcher with increasing schema size represented by X-axis.

Graph 7.15 represents results of evaluating the impact of combining *Name based with context matcher* with *Name based matcher* on schema matching accuracy for mutated schemas of *'MONDIAL'* schema. It can be observed that using

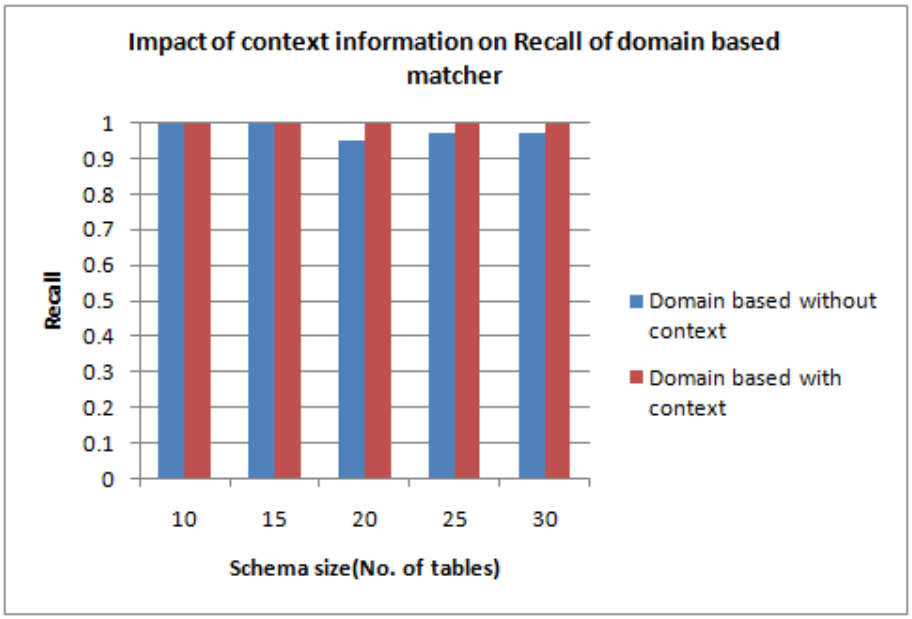

Figure 7.13: Impact of Context information on Recall of Domain based matcher

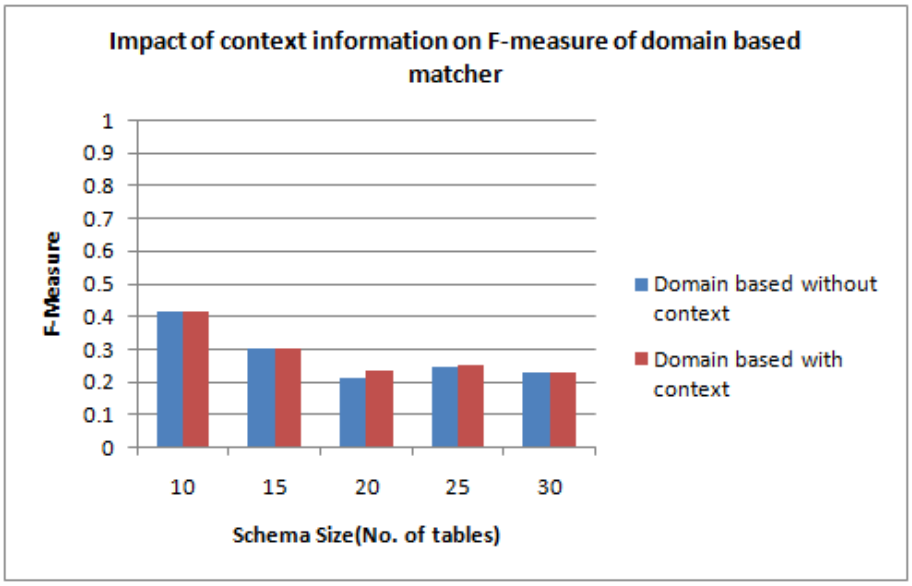

Figure 7.14: Impact of Context information on schema matching accuracy of Domain based matcher

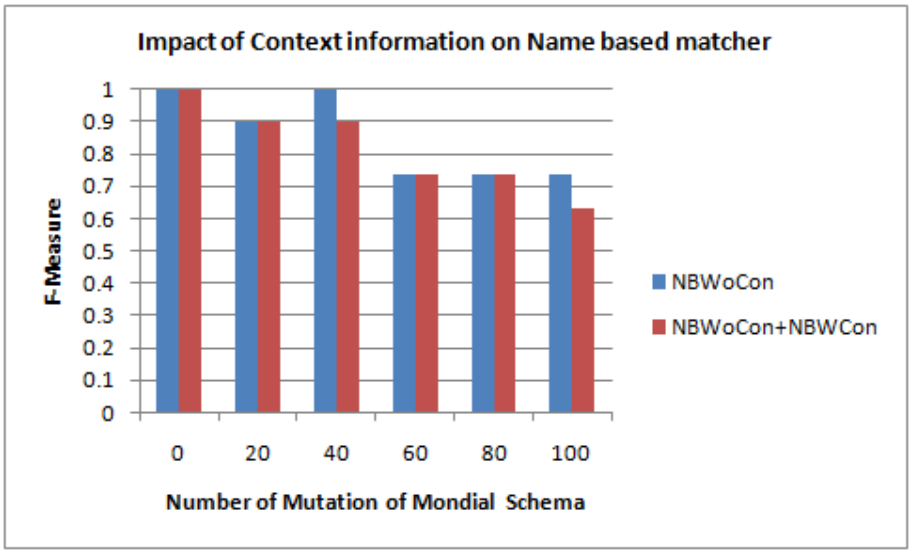

Figure 7.15: Impact of Context information on schema matching accuracy of Name based matcher for mutated schemas.

context information with *Name based matcher* do not improve schema matching accuracy of combination. This is mainly due to the nature of the mutation process used to created mutated schemas, for example, if an element name got replaced by element replacement operation during initial mutation step and later mutation steps replaced the characters from the new replaced element name. In such cases during schema matching process no relationship will be uncovered by use of context information as replaced element name got further mutated.

In graph 7.14 Y-axis represents the impact of use of context information on schema matching accuracy of domain based matcher with increasing number of mutation of *'MONDIAL'* schema represented by X-axis.

7.7 Experiment 5: Impact of matcher combinations on matching accuracy

In this experiment first evaluation of performance of different schema matching algorithm configuration is done on *'MONDIAL'* and *'MONDIAL_DUMMY'* test schemas. Schema size is kept constant at 20 tables each in both schema and matcher combinations are changed to evaluate the impact on precision, recall and over all schema matching accuracy. Equal weight of *'1'* is assigned to all

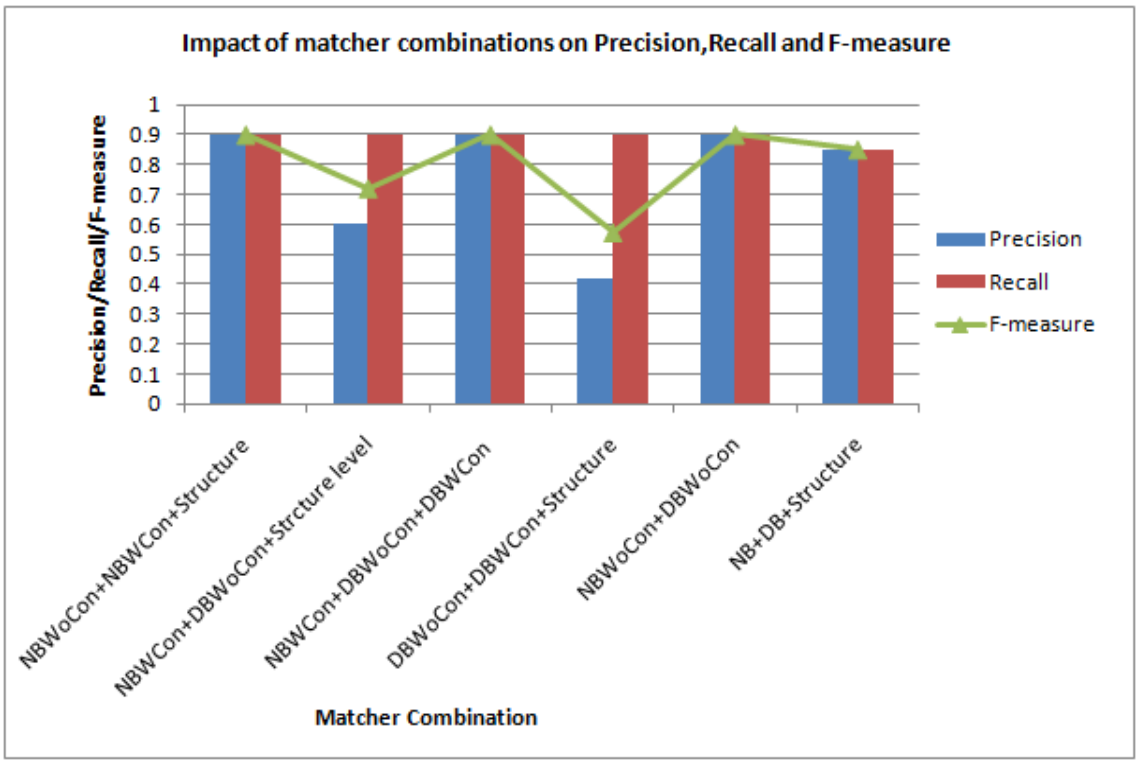

Figure 7.16: Impact of different matcher combinations on schema matching performance

matcher algorithms.

Matching accuracy of individual matcher depends on characteristics of the schema under consideration. For example if matching tables in both schemas have the same structure then structure level schema matcher algorithm will improve the accuracy; if elements names are different but share a relationship such as element names are synonyms of each other or is either a hyper node or hypo node in a relationship hierarchy then name based matcher with context information will contribute more to improve the accuracy of combination.

Results of different combinations are presented in graph 7.16, it can be observed that combinations in which name-based without context matcher is a part gives high accuracy. This in line with the behavior of individual name-based without context matcher for schema matching operation between *'MONDIAL'* and *'MONDIAL_DUMMY'* test schemas as seen in figure 7.4.

In graph 7.16 Y-axis represents schema matching performance measures and X-axis represents different schema matching algorithms combinations.

Results of evaluation of schema matching algorithm performance on mutated schema are represented in graph 7.17. Constant weight of *'1'* is assigned to all individual algorithms, and number of mutations are increased from 0 to 100. It can be observed that,

- schema matching accuracy initially decreases with the increase in number of mutations but gets stable after 60 mutations for most of the matcher combinations.
- It can be observed that matchers can be flexibly combined to form a matcher combination.
- different matchers give different matching accuracy for the same schema. Hence choice of appropriate schema matching algorithm combination is crucial to achieve good matching results. Effectiveness of a particular schema matching combination on schema matching accuracy also depends on schema characteristics.

In graph 7.17 Y-axis represents accuracy measures and X-axis represents the number of successive mutations of *'MONDIAL'* schema.

7.8 Experiment 6: Impact of Matcher weights on Accuracy of schema matcher combination

This experiment's aim is to demonstrate the effect of weights assigned to individual matchers in a matcher combination on over all matching accuracy of a schema matcher combination. Schema matcher combination used in this experiment is *Name based without context, Name based with context and structure level* schema matchers. Weight assigned to individual matcher is changed, and its impact on matching accuracy of combination is calculated. Weight of (0.5,0.3,1) represents that *NBWoCon* matcher is assigned weight '0.5', matcher *NBWCon* is assigned weight '0.3' and matcher *structure-level* is assigned weight as '1'. In this experiment *'MONDIAL_S5'* schema having 100 successive mutations of original *'MONDIAL'* schema is used.

Results are represented in graph 7.18. It can be observed that when highest weight of '1' is assigned to all matchers in the matcher combination, matching

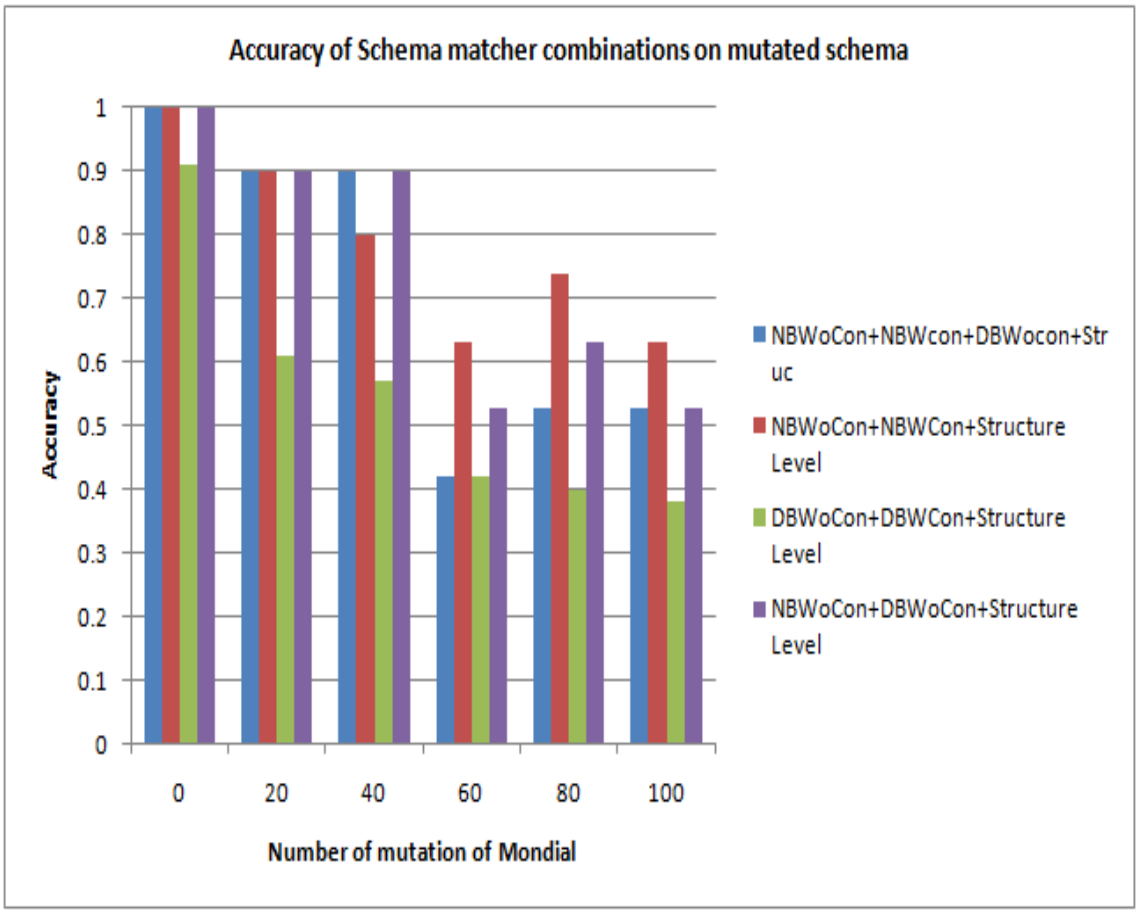

Figure 7.17: Schema matching Accuracy of Schema matcher combinations for mutated schemas.

accuracy achieved is not highest. It can be observed that matching accuracy increases when weight assigned to *NBWCon* schema matcher is reduced gradually and reaches to *'0.842'*. When weight assigned to matcher *NBWCon* is reduced, share of similarity score contributed by *NBWCon* reduces, and share contributed by *NBWoCon* and *structure-level* matchers increases in final similarity score which is a weighted average of similarity score of individual matcher. Matcher *NBWoCon* achieves accuracy of '0.737' for mutated schema *'MONDIAL_S5'* when run independently as can be seen in graph 7.5, but when run in combination with other matchers with highest assigned weight of '1' combination achieves matching accuracy of '0.632'. Contribution of a schema matcher need to be balanced with other matcher when in combination by changing its assigned weight to achieve better matching accuracy as can be observed in graph 7.18. Thus, this project provides control over behavior of an individual matcher in a schema matcher combination. Hence, achieves its set objective to allow flexible configuration of matchers from library to form a matcher combination.

In graph 7.18 Y-axis represents the impact of assigned weights on accuracy of *NBWoCon+NBWCon+Structure level* matcher, X-axis represents weights assigned to matchers in matcher combination.

7.9 Experiment 7: Impact of schema size on processing time of individual matchers

In this experiment weight of *'1'* is assigned to individual matcher and schema size of *'MONDIAL'* and *'MONDIAL_DUMMY'* is increased from 10 tables each to 30 tables each.

Results are represented in graph 7.18. It can be observed that the time taken by matchers to perform schema matching increases with the increase in schema size. Increase in schema size increases search space for matcher to find a potential match for a particular element. This increase in the search space increases processing that need to be done to find a potential matching there by increasing processing time. It can be observed that processing time for *name based with context matcher* and *domain based with context matcher* is higher as compared to other matchers and increases more rapidly with the increase in schema size. This Rapid increase in processing time of matchers using context information is due to overhead of accessing dictionary information to search for

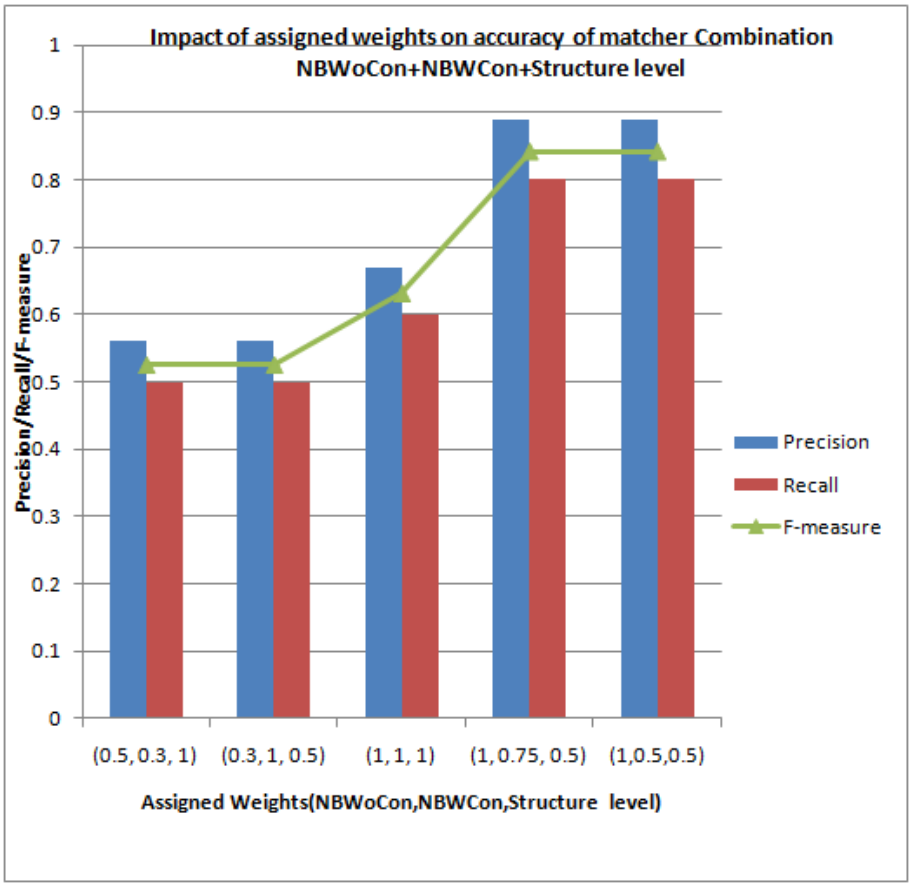

Figure 7.18: Impact of assigned weights on the accuracy of matcher Combination *NBWoCon+NBWCon+Structure level*.

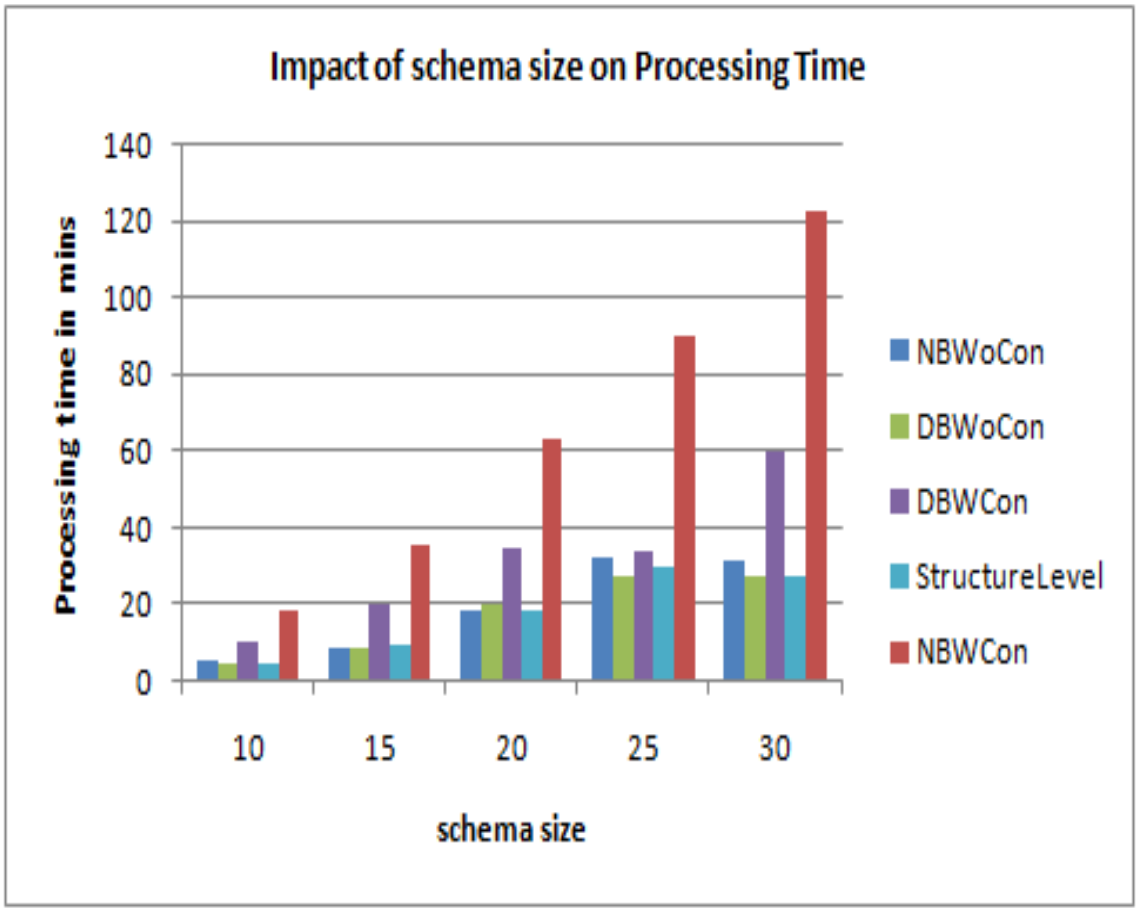

Figure 7.19: Impact of schema size on processing time of schema matching algorithms

synonyms, hypernyms and hyponyms and then extra processing of working out relationships depth between strings under matching. In any schema matcher combination, if matcher is making use of context information then that matcher will dominate the procession time of the matcher combination.

In graph 7.19 Y-axis represents the impact on processing time of matcher algorithms with increasing schema size represented by X-axis.

This chapter described the experimental design and setup to do the performance evaluation of the schema matching algorithms. This chapter presented evaluation results of 7 experiments along with observations drawn from their results. Next chapter presents the conclusion for this dissertation along with the possible future work that can be done to extend this research idea and prototype capabilities.

Chapter 8

Conclusion

Chapter 7 presented the experimental setup and design. The chapter presents the evaluation results of matching performance of schema matching algorithms and combinations along with observations about the matching results. This chapter summarizes the achievements of this dissertation. This chapter discuss about the objectives achieved and draws a broader conclusion based on the performance evaluation done in chapter 7. Chapter presents the possible future works and research directions to extended the capability of this implementation.

The primary goal of investigating the benefits of exposing schema matching capability as a library that allows user the flexibility of which algorithm to use and how to aggregate their outputs is met. To achieve this goal, this dissertation developed a taxonomy of schema matching approaches. Taxonomy classifies schema matching operations as schema-level and instance-level operations. This dissertation developed a library of schema matching algorithms that instantiates most of this taxonomy of matchers.

Library consists of five schema matching algorithms, namely, element-level name-based with and without context information, element-level domain-based with and without context information and structure-level schema matcher. This dissertation presented an approach to combine this algorithms flexibly to form schema matcher combinations. Project allows user to configure matchers to form a combination using a *XML* configuration file as shown in appendix B. This project prototype combines results generated by individual matchers to generate final matching results.

This project aggregates results of matchers in a combination using weighted average method. This aggregation of matching results is made possible because

of the flexibility to combine matchers and the generic output format of individual matchers. Dissertation developed a simple mechanism to skip the elements from schema matching process if the mappings for that element is known. Schema elements that are to be skipped can be easily configured in a *XML* configuration file.

The flexibility of combining matchers to form combinations, and aggregating matching results in a composite way, provides user with better control over creating matcher combinations than the existing systems like *COMA* and *CUPID* [DR02, MBR01].

This dissertation successfully demonstrated that string comparison algorithms can form the building blocks for schema matching algorithms. This dissertation treats every matching task as string comparison operation, This approach is in contrast to the previous approaches that emphasize developing specific matcher for specific matching task [DR02, MBR01, RB01, SE05]. Three string comparison techniques are implemented in this project, namely, distance based, stem based and N-gram based.

Thus the framework presented in this project facilitates to flexibly combine matchers, combine results of individual matchers, skip elements with known mappings and assigned weights to matchers. This framework allows user to have greater control over matching process and facilitates for better evaluation of performance of the matchers combinations.

Performance evaluations as explained in chapter 7, highlights the clear benefits of exposing schema matching capabilities using a library that can be easy configured. It can be observed that schema matching performance of a matcher combination depends on the schema characteristics. Matcher combination *NBWoCon+NBWCon+StructureLevel* showed promising schema matching performance results for schemas with different characteristics and high mutation. Experiments also showed that matcher behavior in a combination can be successfully influenced by altering its assigned weights. It is observed that schema matching accuracy of a matcher combination can improve if weights are assigned to the matchers based on schema characteristics. It is also observed that the processing time of matching operations increases with the increase in schema size. Matchers using context information takes more processing time, hence, user need to configure matchers in a combination based on schema characteristics and keeping in mind the acceptable response time.

8.1 Future Works

There are number of avenues where future research and capability of this prototype can be extended,

- Extending schema matching library to include instance-level matcher will certainly increase the capability of this prototype.
- Structure-level schema matching or instance-level schema matching operation can include capability to find *1:n* correspondences. Such matcher can be further extended to discover mathematical and string based transformations between columns.
- Element-level domain-based schema matching currently performs matching using data type and length of data in the column. Domain-based schema matching can be extended to include matchers using constraints information.
- Structure-level schema matching can be extended to include referential integrity constraint. Tree edit distance technique used to perform structure-level schema matching need to be extended to calculate graph edit distance once integrity constraint is considered.
- Prototype capability can be extended to extract schema information by parsing *DDL* statements and *XML* schema files.
- Set of string comparison algorithms can be extended to include phonetics based string comparison technique. Implementing phonetics based string comparison algorithm will help to filter out false positive matches caused by homonyms.
- To reduce the search space for instance-level schema matching, one possible research direction could be to explore use of element mappings generated by element-level and structure-level schema matcher. Elements having similarity score less then a certain threshold can be ignored from instance-level schema matching operation. Using such a technique can also help to further reinforce the element mappings generated by element-level and structure-level schema matching operations.

Bibliography

[ADMR05] David Aumueller, Hong Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with coma++. In *SIGMOD Conference*, pages 906–908, 2005.

[BLN86] Carlo Batini, Maurizio Lenzerini, and Shamkant B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[BMC06] Philip A. Bernstein, Sergey Melnik, and John E. Churchill. Incremental schema matching. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 1167–1170. ACM, 2006.

[CA99] Silvana Castano and Valeria De Antonellis. A schema analysis and reconciliation tool environment for heterogeneous databases. In *IDEAS*, pages 53–62, 1999.

[DDH03] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *VLDB Journal*, 50:279–301, 2003.

[DDL00] AnHai Doan, Pedro Domingos, and Alon Y. Levy. Learning source description for data integration. In *WebDB (Informal Proceedings)*, pages 81–86, 2000.

[DLD+04] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Y. Halevy, and Pedro Domingos. imap: Discovering complex mappings between database schemas. In Gerhard Weikum, Arnd Christian Knig, and Stefan Deloch, editors, *SIGMOD Conference*, pages 383–394. ACM, 2004.

[DMR03] Hong Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems*, pages 221–237. Springer-Verlag, 2003.

[DR02] Hong Hai Do and Erhard Rahm. Coma - a system for flexible combination of schema matching approaches. In *VLDB*, pages 610–621, 2002.

[gra11] `http://web.science.mq.edu.au/~swan/howtos/treedistance/package.html`, September 2011.

[GSY04] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *ESWS*, pages 61–75, 2004.

[HBP+] Cornelia Hedeler, Khalid Belhajjame, Norman W. Paton, Lu Mao, Chenjuan Guo, Alvaro A. A. Fernandes, and Suzanne M. Embury. A functional model for dataspace management systems. Under Publication, made available through supervisor.

[jdb11] `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136101.html`, September 2011.

[jwn11] `http://nlp.stanford.edu/nlp/javadoc/jwnl-docs/overview-summary.html`, September 2011.

[KMK03] Grzegorz Kondrak, Daniel Marcu, and Kevin Knight. Cognates can improve statistical translation models. In *HLT-NAACL*, 2003.

[LC94] Wen-Syan Li and Chris Clifton. Semantic integration in heterogeneous databases using neural networks. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *VLDB*, pages 1–12. Morgan Kaufmann, 1994.

[LC95] Wen-Syan Li and Chris Clifton. Semint: A system prototype for semantic integration in heterogeneous databases. In *SIGMOD Conference*, page 484, 1995.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[mon11] `http://www.dbis.informatik.uni-goettingen.de/Mondial/`, August 2011.

[MWK00] Prasenjit Mitra, Gio Wiederhold, and Martin L. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *EDBT*, pages 86–100, 2000.

[PS98] Christine Parent and Stefano Spaccapietra. Issues and approaches of database integration. *Commun. ACM*, 41(5):166–178, 1998.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001. 10.1007/s007780100057.

[SE05] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. pages 146–171, 2005.

[SG10] José M. Sempere and Pedro García, editors. *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*. Springer, 2010.

[sma11] `http://semanticmatching.org/s-match.html`, September 2011.

[wor11] `http://wordnet.princeton.edu/wordnet/related-projects/#Java`, August 2011.

[ZS89] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[ZTW+09] Zhiping Zeng, Anthony K. H. Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

Appendix A

Example of Input Schemas

MONDIAL schema and *MONDIAL_DUMMY* schema have 33 tables each. In this chapter we mention schema information for 10 tables each along with the reference base line mapping used to calculate the schema matching accuracy. We also present complete schema information for schema after 100 successive random mutation of mentioned 10 tables of *MONDIAL* schema along with the reference base line mapping used to calculate the schema matching accuracy.

TABLE NAME	COLUMN NAME	DATA TYPE	LENGTH
ECONOMY	COUNTRY	VARCHAR2	4
ECONOMY	GDP	NUMBER	22
ECONOMY	AGRICULTURE	NUMBER	22
ECONOMY	SERVICE	NUMBER	22
ECONOMY	INDUSTRY	NUMBER	22
ECONOMY	INFLATION	NUMBER	22
LOCATED	CITY	VARCHAR2	35
LOCATED	PROVINCE	VARCHAR2	35
LOCATED	SEA	VARCHAR2	35
LOCATED	LAKE	VARCHAR2	35
LOCATED	RIVER	VARCHAR2	35
LOCATED	COUNTRY	VARCHAR2	4
LOCATEDON	PROVINCE	VARCHAR2	35
LOCATEDON	CITY	VARCHAR2	35
LOCATEDON	COUNTRY	VARCHAR2	4
LOCATEDON	ISLAND	VARCHAR2	35
MERGESWITH	SEA1	VARCHAR2	35
MERGESWITH	SEA2	VARCHAR2	35
MOUNTAINONISLAND	ISLAND	VARCHAR2	35
MOUNTAINONISLAND	MOUNTAIN	VARCHAR2	35
POLITICS	COUNTRY	VARCHAR2	4
POLITICS	INDEPENDENCE	DATE	7
POLITICS	DEPENDENT	VARCHAR2	4
POLITICS	GOVERNMENT	VARCHAR2	120
POPULATION	POPULATION_GROWTH	NUMBER	22
POPULATION	COUNTRY	VARCHAR2	4
POPULATION	INFANT_MORTALITY	NUMBER	22
PROVINCE	NAME	VARCHAR2	35
PROVINCE	COUNTRY	VARCHAR2	4
PROVINCE	POPULATION	NUMBER	22
PROVINCE	AREA	NUMBER	22
PROVINCE	CAPITAL	VARCHAR2	35
PROVINCE	CAPPROV	VARCHAR2	35
SEA	NAME	VARCHAR2	35
SEA	DEPTH	NUMBER	22

Table A.1: 10 tables of *'MONDIAL'* schema

A.0.1 Schema - *MONDIAL*

A.0.2 Schema - MONDIAL_DUMMY

TABLE NAME	COLUMN NAME	DATA TYPE	LENGTH
COUNTRY_ECONOMY	COUNTRY_NAME	VARCHAR2	4
COUNTRY_ECONOMY	COUNTRY_GDP	NUMBER	22
COUNTRY_ECONOMY	AGRICULTURE_PART	NUMBER	22
COUNTRY_ECONOMY	SERVICE_PART	NUMBER	22
COUNTRY_ECONOMY	INDUSTRY_PART	NUMBER	22
COUNTRY_ECONOMY	INFLATION_RATE	NUMBER	22
COUNTRY_POPULATION	COUNTRY_NAME	VARCHAR2	4
COUNTRY_POPULATION	POPULATION_GROWTH	NUMBER	22
COUNTRY_POPULATION	INFANT_MORTALITY	NUMBER	22
ETHNICGROUPS	COUNTRY_NAME	VARCHAR2	4
ETHNICGROUPS	POPULATION_PERCENTAGE	NUMBER	22
ETHNICGROUPS	ETHNICGROUPS_NAME	VARCHAR2	50
LOCATED	CITY	VARCHAR2	35
LOCATED	PROVINCE	VARCHAR2	35
LOCATED	COUNTRY	VARCHAR2	4
LOCATED	RIVER	VARCHAR2	35
LOCATED	SEA	VARCHAR2	35
LOCATED	LAKE	VARCHAR2	35
LOCATED	CONTINENT	VARCHAR2	35
LOCATEDON	CITY	VARCHAR2	35
LOCATEDON	ISLAND	VARCHAR2	35
LOCATEDON	COUNTRY	VARCHAR2	4
LOCATEDON	PROVINCE	VARCHAR2	35
MOUNTAIN_ON_ISLAND	MOUNTAIN_NAME	VARCHAR2	35
MOUNTAIN_ON_ISLAND	ISLAND_NAME	VARCHAR2	35
POLITICS	COUNTRY_NAME	VARCHAR2	4
POLITICS	INDEPENDENCE	DATE	7
POLITICS	DEPENDENT	VARCHAR2	4
POLITICS	GOVERNMENT	VARCHAR2	120
SEA	S_NAME	VARCHAR2	35
SEA	S_DEPTH	NUMBER	22
SEA_JOINS	SEA_1	VARCHAR2	35
SEA_JOINS	SEA_2	VARCHAR2	35
STATE	STATE_NAME	VARCHAR2	35
STATE	STATE_COUNTRY	VARCHAR2	4
STATE	STATE_POPULATION	NUMBER	22
STATE	STATE_CAPPROV	VARCHAR2	35
STATE	STATE_CAPITAL	VARCHAR2	35
STATE	STATE_AREA	NUMBER	22
STATE	STATE_GOVERNOR	VARCHAR2	50

'MONDIAL'	*'MONDIAL_DUMMY'*
PROVINCE	STATE
ECONOMY	COUNTRY_ECONOMY
POPULATION	COUNTRY_POPULATION
POLITICS	POLITICS
ETHNICGROUP	ETHNICGROUPS
SEA	SEA
MERGESWITH	SEA_JOINS
LOCATED	LOCATED
LOCATEDON	LOCATEDON
MOUNTAINONISLAND	MOUNTAIN_ON_ISLAND

Table A.2: Base line reference mapping between 10 tables of *'MONDIAL'* schema and *'MONDIAL_DUMMY'* schema.

Base line reference mapping between tables of *'MONDIAL'* schema and *'MONDIAL_DUMMY'* schema for 10 tables.

Table A.2 shows the reference base line table mappings between *'MONDIAL'* schema and *'MONDIAL_DUMMY'* schema. This base line mapping are used to calculate the accuracy of schema matching operations.

A.0.3 Schema - MONDIAL after 100 successive random mutations

Base line reference mapping between tables of *'MONDIAL'* schema and schema after 100 successive mutation of *'MONDIAL'* schema.

Table A.4 shows the reference base line table mappings between *'MONDIAL'* schema and schema with 100 mutation of *'MONDIAL'*. This base line mapping is used to calculate the accuracy of schema matching operations.

TABLE NAME	COLUMN NAME	DATA TYPE	LENGTH
DOUNTANONILANDY	LADUJ	VARCHAR2	7
GERGESWITHWFM	IEA1	VARCHAR2	35
GERGESWITHWFM	SEA2	VARCHAR2	35
IDMINISTATIVE_DIST	CRA	NUMBER	22
IDMINISTATIVE_DIST	APPCOL18	CHAR	10
KEAB	VPTH	NUMBER	22
LOCTDONN	PROVINCE	VARCHAR2	35
LOCTDONN	ISLAND	VARCHAR2	35
POPULATIONKD	NOPULATON_GROWTHJ	NUMBER	22
SOLITICSAM	COUNY	VARCHAR2	4
SOLITICSAM	INDPENDENCE	DATE	7
SOLITICSAM	APPCOL8	NUMBER	22
SOLITICSAM	DEPENDNT	VARCHAR2	4
SYSTEM	COUNTRY	VARCHAR2	4
SYSTEM	GDP	NUMBER	22
SYSTEM	AGRICULTURE	NUMBER	22
SYSTEM	SRVICE	NUMBER	22
SYSTEM	INDUSTRY	NUMBER	22
SYSTEM	INFLATION	NUMBER	22
UOCATED	RIVER	VARCHAR2	35
UOCATED	PROVINCE	VARCHAR2	35
UOCATED	SEA	VARCHAR2	35

Table A.3: Schema with 100 mutation of *'MONDIAL'* schema.

'MONDIAL' TABLE	100 MUTATIONS
ECONOMY	SYSTEM
LOCATED	UOCATED
LOCATEDON	LOCTDONN
MERGESWITH	GERGESWITHWFM
MOUNTAINONISLAND	DOUNTANONILANDY
POLITICS	SOLITICSAM
POPULATION	POPULATIONKD
PROVINCE	IDMINISTATIVE_DIST
SEA	KEAB

Table A.4: Base line reference mapping between *'MONDIAL'* and its 100 mutation schema.

Appendix B

Example Matcher Configuration File

```
<algoConfiguration>
     <elementLevel>
      <matchers>
       <elementLevelNBWoCon joinThreshold="1" weight="1" run="y">
       </elementLevelNBWoCon>
       <elementLevelNBWCon joinThreshold="1"  weight="0.5" run="y">
       </elementLevelNBWCon>
       <elementLevelDBWCon joinThreshold="1"  weight="1" run="n">
       </elementLevelDBWCon>
       <elementLevelDBWoCon joinThreshold="1"  weight="1" run="n">
       </elementLevelDBWoCon>
       <structureLevel joinThreshold="1"  weight="0.8" run="y">
       </structureLevel>
      </matchers>
     </elementLevel>
</algoConfiguration>
```

Appendix C

Example Schema Elements Configuration File

```
<skipElements>
  <schema name="MONDIAL_DUMMY">
  <skipTab name="LAKE" skip="n"/>
  <skipTab name="LOCATED" skip="n"/>
  <skipTab name="LOCATEDON" skip="n"/>
  <skipTab name="MOUNTA" skip="n"/>
  <skipTab name="MOUNTAIN_ON_ISLAND" skip="n"/>
  <skipTab name="POLITICS" skip="n"/>
  <skipTab name="RIVER" skip="n"/>
  <skipTab name="SEA" skip="n"/>
  <skipTab name="SEA_JOINS" skip="n"/>
  <skipTab name="STATE" skip="n"/>
</schema>
<schema name="MONDIAL">
  <skipTab name="ECONOMY" skip="y">
    <col name="COUNTRY" >
</col>
</skipTab>
  <skipTab name="LOCATED" skip="n"/>
  <skipTab name="LOCATEDON" skip="n"/>
  <skipTab name="MERGESWITH" skip="n"/>
  <skipTab name="ECONOMY" skip="y"/>
```

```
  <skipTab name="MOUNTAINONISLAND" skip="n"/>
  <skipTab name="POLITICS" skip="n"/>
  <skipTab name="PROVINCE" skip="n"/>
  <skipTab name="POPULATION" skip="y"/>
  <skipTab name="SEA" skip="n">
      <col name="NAME" >
</col>
</skipTab>
</schema>
</skipElements>
```

Appendix D

Domain Specific Context Information

Table D.1 shows the domain specific context information used in the element-level schema matching. This context information is specific to Oracle DBMS. Each row in the table represent a tree with first column value representing the top node; Thus nodes nearer to the top node are more compatible to it.

varchar2	string	char	number
string	vharchar2	char	number
char	string	varchar2	number
number	long	float	double

Table D.1: Domain Specific Context Information

Appendix E

How to Configure Project

Following is the step by step guide to configure the development environment to configure this project.

1. Install Oracle 11g *DSMS*.
2. create temporary schema and temporary user. Scripts for creating temporary schema and user can be found in the attached CD containing source code.
3. Grant select privilege on catalog tables to the temporary user. Grant scripts are included in the CD.
4. Download following external library JAR files. All JARs are in the provided CD also.
    - odbc5
    - $log4j - 1.2.16$
    - jwnl
    - $sdl\_v1.3.0$
    - $orbital - core$
    - $orbital - ext$
    - $sat4j - 1.7$
    - source code of s-match project for reference only as this project only uses modified string comparison algorithms of s-match which are included in provided source code CD.

5. Download and configure JDK 1.6 or later.

6. Install *eclips helios*, create new project and import the file structure as provided in CD.

7. `\DissertationLSMA\LSMA\src\LSMA\index.java` is the starting point.

8. To create the test schemas, SQL scripts for test schemas can be found in the CD. PL/SQL Scripts for the procedures used to calculate the accuracy of schema matching operation can also be found in the CD.