MATCHING SCHEMAS OF HETEROGENEOUS XML DATABASES

A PROJECT REPORT

Submitted by

JANAGANANDHINI.T (0810108011)

THILAGAVATHY.S (0810108055)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

(An Autonomous Institution Affiliated To Anna University Of Technology,Coimbatore)

COIMBATORE - 641049

APRIL 2012

BONAFIDE CERTIFICATE

Certified that this project report **"MATCHING SCHEMAS OF HETEROGENEOUS XML DATABASES"** is the bonafide work of **Ms. T.JANAGANANDHINI (0810108011)** and **Ms.S.THILAGAVATHY (0810108055)** who carried out the project work under my supervision.

GUIDE	**HEAD OF THE DEPARTMENT**
(Ms.S.Rajini ,M.S.,**)**	**(Prof.N.Jayapathi,M.Tech.,)**
Associate Professor	Professor
Department of Computer Science & Engineering	Department of Computer Science & Engineering
Kumaraguru College Of Technology	Kumaraguru College Of Technology
Coimbatore – 641 049	Coimbatore – 641 049

The candidates with **University Register Numbers 0810108011 and 0810108055** were examined by us in project examination held on _____

Internal Examiner **External Examiner**

DECLARATION

We

T.JANAGANANDHINI 0810108011

S.THILAGAVATHY 0810108055

hereby declare that the project entitled **"MATCHING SCHEMAS OF HETEROGENEOUS XML DATABASES"** is done by us and to the best of our knowledge a similar work has not been submitted to any other institution , for the fulfillment of the required course of study.

This report is submitted on the partial fulfillment of the requirements for all awards of the Degree of Bachelor of Computer Science and Engineering of Kumaraguru College Of Technology, Coimbatore.

Place: Coimbatore

Date:

JANAGANANDHINI.T

THILAGAVATHY.S

ACKNOWLEDGEMENT

We express our profound gratitude to our **Chairman Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E** for giving this great opportunity to pursue this course.

We extend our gratefulness to **Dr. S. Ramachandran, Ph.D.,** *Principal* for providing the necessary facilities to complete our project.

We are deeply obliged to **Prof.N.Jayapathi,M.Tech.,** *Head of the Department* for his precious suggestions.

We express our heartfelt and deep sense of gratitude to our project Co-ordinator **Ms.Devaki.P,M.E.,M.S.,** *Associate Professor, Department of Computer Science and Engineering*,, for her continuing support, patience and words of expertise during the project development work.

We are incident to express our heartiest thanks to our guide **Ms.S.Rajini ,M.S.,***Asso.Professor*, for her incredible support, valuable ideas and suggestions for our toil regarding the project.

We articulate our thankfulness to our class advisor, **Ms.L.Latha M.E.,***Asso.Professor*. We thank her for support, encouragement and ideas.

We would like to convey our honest thanks to all **Faculty members** and **non teaching staffs** of the department for their support. We would like to thank all our classmates who gave us a proper light moments and study breaks apart from extending some technical support whenever needed them most.

We dedicate this project work to our **parents** and **friends** for many reasons and feeling from bottom of the heart, without their love this work would'nt be possible.

TABLE OF CONTENTS

ABSTRACT

Schema matching is a basic problem in many database application domains, such as data integration, data warehousing. The problem of schema matching can be formulated as follows, “given two schemas, find the most plausible correspondences between the elements of the two schemas exploiting all available information, such as the schemas, instance data, and auxiliary sources” .Given the rapidly increasing number of data sources to integrate and due to database heterogeneities, manually identifying schema matches is a tedious, time consuming, error-prone, and therefore expensive process. As systems become able to handle more complex databases and applications, their schemas become large, further increasing the number of matches to be performed. Thus, automating this process, which attempts to achieve faster and less labor-intensive, has been one of the main tasks in data integration. However, it is not possible to determine fully automatically the different correspondences between schemas, primarily because of the differing and often not explicated or documented semantics of the schemas. This project presents an approach for matching schemas of heterogeneous databases that utilizes most of the information related to schemas, which explores the implicit semantics of the schemas, that further improves the results of the integration.

ABSTRACT

LIST OF SYMBOLS

XML eXtensible Markup Language

SQL Structured Query Language

SF Similarity Flooding

COMA COmbinig MAtcher

1. INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 SCHEMA MATCHING

Schema matching is an approach which was in research since late 1970s. Many researchers and scientists have focused on schema matching as its primary research interest in order to find a fully automatic schema matching system which is domain independent. Indeed, it didn't succeed till now due to various factors such as heterogeneities of schema and structures. Schema matching is considered important in almost all the fields like data warehousing, data integration, e-commerce and query processing which interacts with other schemas of several different systems. In today's world, an individual cannot exist without a computer; they have to always depend on the internet for getting any information available in large junks which is incomparable with the other systems. Hence, the basic need of schema matching comes into picture.

A match among the various systems needs to be performed in order for the end-user to easily co-relate the data from one system to another system and make his decisions. These schema matching systems cannot judge on its own without the help of a human intervention. Hence, semi-automatic schema matching techniques are currently being used nowadays which needs lesser percentage of human interaction. Many semiautomatic schema matching systems primarily focus on only one techniques rather than leveraging several individual matching techniques. Schema matching is the process of finding semantic correspondences between the objects. It can be formulated as follows:" Given two schemas S1 and S2, we have to find a plausible correspondence between elements of S1 and S2 exploiting all the information such as schemas, instance data". The main goal of the schema matching is to reduce the amount of manual effort and avoid human intervention in solving the matching problems from scratch. This process has gained profound interest in both research and practice. Matching the components either in database schemas or XML or Ontologies has become necessary for almost all the applications like peer to peer databases, data migration, data integration, and query processing.

Schema is the metadata of the database design. It contains the entire information of the attributes present in the database and instances that can be stored. Apart from the characteristics mentioned, it has some concerns relating to cardinality, integrity, referential constraints etc. Schemas may be developed using different schema definition languages. For example, Structured Query Language (SQL) is for relational model, Document type definition (DTD) and XML Schema Definition (XSD) for XML schemas and Web Ontology Language (OWL) for Ontologies. Each of these different types of schemas is developed for a particular application, but they all contribute to the use of schemas in their applications.

1.2 APPLICATIONS OF SCHEMA MATCHING

Schema matching is of ultimate help for database applications such as data integration, data warehousing.

1.2.1 DATA INTEGRATION:

This can be considered as the most important motivation for schema matching. Data integration needs to integrate the data from several different sources which tend to have different schematic structure and it is a huge

difficulty to find a match between the sources. Hence, schema matching came into picture in this context. Since the schemas, ontologies and the XML formats are developed by different individuals who don't coincide with each other, there is no possibility to find a match between them. Thus, schema matching will convert the local schemas i.e. the schema structure of each of the sources into a generic format accessible to everyone called global schema. Apart from matching the schemas of each of the sources, schema matching is also involved in matching the instance data in these schemas. Semi-automatic methods and approaches are currently in place which still needs human effort.

1.2.2. DATA WAREHOUSING:

Data warehousing integrates the data from several heterogeneous data sources. These data sources are developed by individuals from different organizations which follow different standards. Data warehousing is the key practice of maintaining the data from organizations across the world and generate reports to analyze and spread the business. Hence, when the actual part of maintaining the data warehousing comes into picture, there will be all sorts of problems in finding a match among the schemas. This is where schema matching will play its vital role. The schema matching will match the data from several sources and then it wouldn't be of enough trouble in maintaining a data warehouse.

1.3.XML

XML is a markup language that defines a set of rules for encoding documents in a format that is both readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.

The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.

Many application programming interfaces (APIs) have been developed for software developers to use to process XML data, and several systems exist to aid in the definition of XML-based languages.

An XML database is a data persistence software system that allows data to be stored in XML format. This data can then be queried, exported and serialized into the desired format. XML databases are usually associated with document-oriented databases.

Two major classes of XML database exist:

1. XML-enabled: these map all XML to a traditional database (such as a relational database), accepting XML as input and rendering XML as output. This term implies that the database does the conversion itself.
2. Native XML (NXD): the internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage, which is, however, not necessarily stored in the form of text files.

2. SYSTEM ANALYSIS

CHAPTER 2

SYSTEM ANALYSIS

2.1 EXISTING SYSTEMS

This chapter discusses about some of the existing approaches for schema matching.

2.1.1 SIMILARITY FLOODING(SF) ALGORITHM

This schema matching converts the schemas into directed graphs and uses fix point computation to analyze the correspondences between the nodes of the graph. The algorithm just implements a simple name matcher which performs basic string level matching techniques like prefix, suffix, postfix comparisons. It does not rely on any thesaurus like the other matching algorithms rather than just making use of the string matching comparisons. It converts all the other types of schemas like SQL, DDL, XSD, and OWL into directed graphs and then performs the match criteria. It usually takes two schemas as input and produces a mapping as an output which in turn will be filtered based on the matching criteria and then a part of the mapping will be considered as the final output.

2.1.2. COMA (Combining Matcher):

COMA is a generic match system which supports different applications and a variety of schemas and XML which combines all the matchers in a flexible way. There are a lot of efforts that have been made by the researchers in developing a fully automatic schema matching which had to combine several matching algorithms. The main issue arises while trying to combine the various matching algorithms. Thus, COMA will act as an intermediate to perform this schema matching by using all the available

matching algorithms in the library. Schemas are represented as directed acyclic graph and the attributes are represented as graph nodes connected by directed links. The schemas are imported and then a match algorithm will be performed on the inputs available. This system has a library which contains a huge list of all the matching algorithms and there is always an option to add new algorithms in the library so that new matching's can also be performed.

2.1.3ARTEMIS (Analysis of Requirements; Tool Environment for Multiple Information Systems)

It clusters the schema attributes based on name and structure affinities. This is a hybrid schema matcher which utilizes both the element level and structure level information of the input schemas. The name and structural affinities are calculated based on inter-schema relationships or intra-schema relationships using a common thesaurus. Global affinity coefficients is the measure of the level of matching between the schema elements based on their names, characteristics defined between them. Based on these coefficients, a clustering algorithm groups the similar classes at varied levels of affinity, which in turn creates a cluster with a set of global attributes called global class. Then, logical correspondences between the global class and other schema attributes are determined using a mapping table.

2.2 PROPOSED SYSTEM

Exploring and using as much as possible the available information during the process of matching the schemas is important. This is because, the more information is used, the more accurate the results of the schema matching process. Our approach attempts to explore the various elements of XML database schemas during the matching process before the integration process can be realized. Hence, we proposed an approach for matching XML database schemas by utilizing five matchers namely: schemas matcher, attribute name matcher, data type matcher, constraint matcher, and instance data matcher. Each matcher calculates and records the percentage of similarities between the elements being compared. Matching the databases' schemas based on the name of schemas and the name of attributes (element level) are accomplished using the following methods:

(i) *n-gram*: strings compared according to their set of *n*-grams, i.e. sequences of *n* characters. An *n*-gram is a sub-sequence of *n* items from a given sequence. The items in question can be letters, words or base pairs according to the application. The percentage of similarities between two elements, *E*1 and *E*2, is calculated.

(ii) *synonym*: words in different spelling but have the same meaning are known as synonym, which is part of an ontology. To search for the synonyms of words, a relative word table is used .The percentage of similarities is 100 if the terms compared exist as synonyms in the relative word table, 0 if the terms compared exist in the relative word table but not as synonyms, and '-' if one or both of the terms are not in the relative word table.

After applying the *n*-gram method, the percentage of similarities between two words possibly will be improved using the synonym method, which is the main idea of this process. Blending both methods produce correct results as opposed to applying either method as suggested by previous works.

3. SYSTEM SPECIFICATION

CHAPTER 3

SYSTEM SPECIFICATION

3.1 HARDWARE REQUIREMENTS

Table 3.1 Hardware Requirements

S NO	HARDWARE	REQUIREMENT
1	Processor Type	Pentium 4
2	Processor Speed	3.4 GHz
3	RAM	2 GB RAM
4	Hard Disk Capacity	160 GB
5	Monitor	14/17"
6	Mouse	Optical
7	Keyboard	101/105 Keys

3.2 SOFTWARE REQUIREMENTS

Operating System: Windows XP, 2003

Programming Tools:
IDE : Net Beans IDE 7.1
Front end : Java
Back end : SQL server

4. SOFTWARE DESCRIPTION

A class hierarchy always has a root class; this is a class with very general behavior. Java comes with an extensive set of classes, arranged in packages that you can use in your programs. For example, Java provides classes that create graphical user interface components, classes that handle input and output and classes that support networking functionality (the java.net package).

- **Interpreted**

Java is an interpreted language: the Java compiler generates byte-codes for the Java Virtual Machine (JVM), rather than a native machine code. To actually run a Java program, you use the Java interpreter to execute the compiled byte-codes. Because Java byte-codes are platform-independent, Java programs can run only on any platform that the JVM (the interpreter and the run-time system has been ported to. In an interpreted environment, the standard "link" phase at all, it is only the process of loading new classes into the environment, which is incremental, lightweight process that occurs at run-time. This is in contrast with the slower and more cumbersome compile-link-run cycle of languages like C and C++.

- **Dynamic and Distributed**

Java is a dynamic language. Any java class can be loaded into a running Java interpreter at any time. These dynamically loaded classes can then be dynamically instantiated. Native code libraries can also be dynamically loaded. Classes in java are represented by the class; you can dynamically obtain information about a class at run-time. This is especially true in Java 1.1, with the addition of the Reflection API. Java is also called a distributed language. This means, simply, that it provides a lot of high-level support for networking. For example, the URL class and other related classes in the java.net package make it almost as easy to read a remote file or resource as it is to read a local file. Similarly, in Java 1.1, the Remote Method Invocation (RMI) API allows a Java program to invoke methods of remote Java objects, as if they were local objects.

- **Robust**

Java has been designed for writing highly reliable or robust software. Java certainly doesn't eliminate the need for software quality assurance; it's still quite possible to write buggy software in Java. However, Java does eliminate certain types of programming errors, which makes it considerably easier to write reliable software. Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compare to other programming languages. Compiler checks the program whether there any error and interpreter checks any run time error and makes the system secure from crash. The robustness of Java is due in large part to its automatic memory management and strict compile-time and run-time checking. In C++, memory is addressed by the use of pointers, or variables that hold the address of the memory range used. A pointer could be misdirected to the again as it is interpreted at run time. Exception Handling is another feature in Java. An exception is a signal that some sort of exceptional condition, such as a wrong location. Java eliminates the pointer and encapsulates memory usage into classes. The syntax is strictly checked for errors upon compilation and "file not found" error, has occurred. Using the try/catch/finally statement, you can group all of your error handling code in one place, which greatly simplifies the task of error handling and recovery.

CHAPTER 4

SOFTWARE DESCRIPTION

4.1 FRONTEND: JAVA

4.1.1 FEATURES OF JAVA

Java is a computer programming language created by Sun Microsystems. Java is used mainly on the Internet and uses a virtual machine which has been implemented in most browsers to translate Java into a specific application on different computer system. With most programming languages, you either compile or interpret a program so that it can be run on the computer. The Java programming language is unusual in that a program is both compiled and interpreted. The most common Java programs are applications and applets. Applications are standalone programs. Applets are similar to applications, but they don't run standalone. Instead, applets adhere to a set of conventions that lets them run within a Java-compatible browser. Sun described Java as follows: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded and dynamic language.

- **Object-Oriented**

Java is an object-oriented programming language. The focus is on the data in the application and methods that manipulate that data, rather than thinking strictly in terms of procedures. In an object-oriented system, a class is a collection of data and methods that operate on that data. Classes are arranged in a hierarchy, so that a subclass can inherit behaviour from its super class.

- **High-Performance**

Interpretation of byte codes slowed performance in early versions, but advanced virtual machines with adaptive and just-in-time compilation and other techniques now typically provide performance up to 50% to 100% the speed of C++ programs. While the performance of interpreted byte codes is usually more than adequate, there are situations where higher performance is required. The byte codes can be translated on the fly (at runtime) into machine code for the particular CPU the application is running on. For those accustomed to the normal design of a compiler and dynamic loader, this is somewhat like putting the final machine code generator in the dynamic loader. The byte code format was designed with generating machine codes in mind, so the actual process of generating machine code is generally simple. Efficient code is produced: the compiler does automatic register allocation and some optimization when it produces the byte codes.

4.1.2 JAVA SWING JARGON

Swing is a widget toolkit, in JAVA, for generating Graphical User Interfaces (GUIs). In this project we have used Net Beans IDE which is a free and open source environment which supports the execution of Java programs. Other alternatives are JCreator, Eclipse etc. The GUI can be built in an easier method in Net Beans by using drag and drop options. This will generate the code automatically which allows us to focus more on the application rather than the look of the application. The main characteristics of Swing are look and feel, accessibility, drag and drop etc. It is known as a library extension of AWT which is much better when compared to AWT and it can also be used to build standalone desktop applications in a much simpler way. The main components of Swing that are used in the application are as follows:

1. JFrame:

It is a window which contains all user-defined components like buttons, toggles, title, border etc. Any component that has to be included in the window should be included in the content pane.

2. JLabel:

It is a subclass of JComponent. It is mainly used to display the text instructions on the GUI. In our desktop application, we have used JLabel to provide the status of the matching.

3. JButton:

These are simple buttons. These can be used to allow user to trigger different events. In our application, by clicking on different buttons, the user can select files and get the matching results displayed.

4. JDialog:

It is basically used to display a pop-message which indicates a temporary notice.

4.2 BACKEND: SQLServer

SQL server is one of the leading database management systems (DBMS) because it is the only Database that meets the uncompromising requirements of today's most demanding information systems. From complex decision support systems (DSS) to the most rigorous online transaction processing (OLTP) application, even application that require simultaneous DSS and OLTP access to the same critical data, SQL Server leads the industry in both performance and capability.

A database management, or DBMS, gives the user access to their data and helps them transform the data into information. Such database management systems include dBase, paradox, IMS and SQL Server. These systems allow users to create, update and extract information from their database.

A database is a structured collection of data. Data refers to the characteristics of people, things and events. SQL Server stores each data item in its own fields. In SQL Server, the fields relating to a particular person, thing or event are bundled together to form a single complete unit of data, called a record. Each record is made up of a number of fields. No two fields in a record can have the same field name.

SQL Server stores records relating to each other in a table. Different tables are created for the various groups of information. Related tables are grouped together to form a database.

Sometimes all the information of interest to a business operation can be stored in one table. SQL Server makes it very easy to link the data in multiple tables. Matching an employee to the department in which they work is one example. This is what makes SQL Server a relational database management system, or RDBMS. It stores data in two or more tables and enables you to define relationships between the table and enables you to define relationships between the tables.

5. PROJECT DESCRIPTION

CHAPTER 5

PROJECT DESCRIPTION

5.1 PROBLEM DEFINITION

An organization may contain many branches which may be located at different places and which may use different databases. It is difficult for a user to manually integrate these databases by analyzing their structures. This system takes two schemas of two different databases, say, XML and SQL to create a global unified schema by matching their schema names, attributes, data types and constraints.

5.2 OVERVIEW OF THE PROJECT

This approach combines four matching techniques, namely schema, attribute, data type and constraint matcher. To compare two strings n-gram approach and synonym table approach is used. In n-gram, strings are split into substrings and percentage of similarity is computed. In synonym table, relative word table is used to match the strings.

5.3 OBJECTIVE

The main objective of the project is to resolve schematic heterogeneity while integrating different database schemas.

5.4 MODULES

5.4.1 Schema matcher

5.4.2 Attribute matcher

5.4.3 Data type and constraint matcher

5.4.4 Global schema creation

5.4.1 SCHEMA MATCHER

The schemas matcher compares two schemas *Si* and *Sj* of XML database *Di* and SQL database *Dj* to identify similarities between these schemas. The comparison is based on the name of the schemas. The *n*-gram and synonym methods are used in measuring the percentage of similarities between these schemas and if there exists similarities between them, then they are considered as candidate for integration. Once two schemas are identified as similar by the schemas matcher, identifying the correspondences between these schemas at the attribute level is then performed. This level is responsible to identify whether two attributes from two different schemas are similar or not. If they are similar, they should be merged as a single attribute. This is important as it reduces the number of repeated attributes as well as the number of null values in the global schemas. Three types of matchers are then applied, namely: attribute name matcher, data type matcher, and constraint matcher.

5.4.2 ATTRIBUTE MATCHER

The attribute name matcher takes two attributes at a time and calculates the percentage of similarities by utilizing the *n*-gram and synonym methods. The following example illustrates this process. Once a pair of attributes has been identified to have some similarities based on the attribute name matcher, it is not convinced enough to conclude that the pair of attributes should be merged as a single attribute. This is due to the fact that the attributes might have different data types or even different roles which prohibit both attributes to be merged as a single attribute even if the percentage of similarities identified by the attribute name matcher is 100. For this reason, the data type matcher is then applied to the pair of attributes followed by the constraint matcher.

5.4.3 DATA TYPE AND CONSTRAINT MATCHER

Data type matcher uses a synonym table specifying the degree of compatibility between a set of predefined generic data types.It matches attributes returned by the attribute matcher.If data types are matched then it is considered as a candidate for constraint matcher.This reduces the number of pairs to be verified by the constraint matcher.

The constraint matcher compares the roles of two similar attributes to check whether these attributes share the same constraints or not. Similar to the previous matcher, a pair of attributes is matched if they have the same role (100% similarities) and not otherwise (0% similarities). As a result, the pairs that are identified as matched by the constraint matcher are the pairs that will be merged into a single attribute, while other attributes will remain as they are.

5.4.4 GLOBAL SCHEMA CREATION

The attributes that are returned as matched from the constraint matcher are merged as single attribute in the global schema. Other attributes that are not matched are also added from both the local schemas into the global schema.XML is used to create the global schema.

5.5 PROCESS FLOW

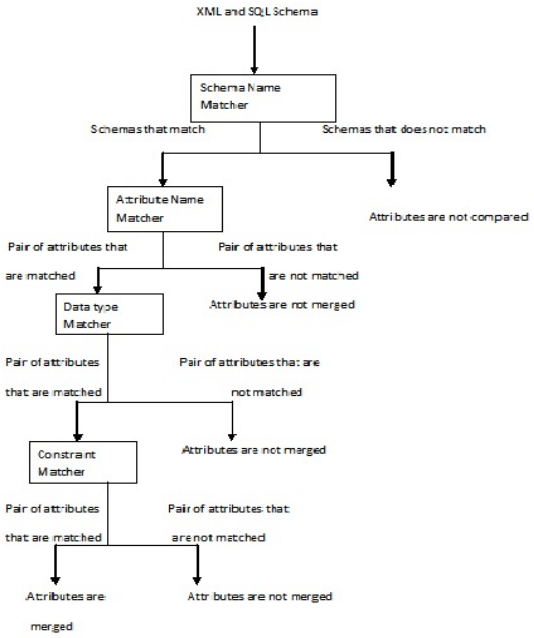


Fig 5.1 Process flow diagram for different matchers

6. RESULTS & DISCUSSION

It is important to note also that utilizing more elements does not mean that the percentage of matched attributes will increase. This is because two attributes might have some similarities in terms of their names but they might have different data types or even roles that caused the percentage of matched attributes to decrease.

6.1 RESULTS

From the figure it is observed that in most cases the percentage of similarities changed whenever a different element is considered. For example the percentage of similarities between Deposit and Balance when *n*-gram applied is 27% and when synonym is applied it is 100%.

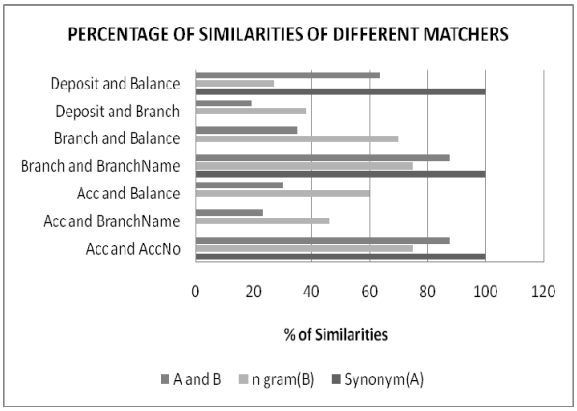


Fig 6.1.Percentage of Similarities of different matchers

7. CONCLUSION AND FUTURE WORK

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

This project presents an approach for matching heterogeneous database schemas, which consists of four matchers namely: schema matcher, attribute name matcher, data type matcher and constraint matcher. This approach, which exploits the various elements of heterogeneous database schemas during the matching process, has achieved higher percentage of similarities and percentage of matched attributes compared to the other approaches, which produces better global schemas during integration.

7.2 FUTURE WORK

Matching schemas of heterogeneous XML database matched two schemas of XML and SQL databases and produced a global schema for the two schemas. The future enhancements of the project is to

- Increase the number of schemas to be matched to more than 3.
- Currently, we are performing matching only between XML and SQL; hence it could be enhanced to OWL matching as well.

8. APPENDIX-1

CHAPTER 8

APPENDIX-1

8.1 SOURCE CODE

Main program:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;

public class SchemaMatching extends JFrame implements ActionListener
{
	JMenuBar mbMenu;
	JMenu meFile, meMaching;
	JMenuItem miExit, miXmlCreation,
			miSchemaMatching;
	JDesktopPane desktop;
	public Object object[]=new Object[20];
	public SchemaMatching()
	{
		super("Schema matching for heterogenous xml database");
		mbMenu = new JMenuBar();
		meFile = new JMenu("File");
		meMaching = new JMenu("Schema Matching Process");
		miExit = new JMenuItem("Exit");
		meFile.add(miExit);
		mbMenu.add(meFile);
		miSchemaMatching = new JMenuItem("Schema Matching");
		meMaching.add(miSchemaMatching);
		setJMenuBar(mbMenu);
		desktop=new JDesktopPane();
		setContentPane(desktop);
		miExit.addActionListener(this);
		miXmlCreation.addActionListener(this);
		miSchemaMatching.addActionListener(this);
		Dimension ss=Toolkit.getDefaultToolkit().getScreenSize();
		setSize(ss.width,ss.height);
		setVisible(true);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
	}
	public void actionPerformed(ActionEvent ae)
	{
		Object source=ae.getSource();
		if(source.equals(miExit))
		{
			System.exit(0);
		}

		if(source.equals(miSchemaMatching))
		{
			display(new Matcher());
```

```
        }
    }

    void display(JInternalFrame obj)
    {
        new CenterFrame(obj);
        obj.setVisible(true);
        desktop.add(obj);
        try
        {
            obj.setSelected(true);
        }
        catch(java.beans.PropertyVetoException e2){}
    }



    public ArrayList dataTypeMatcher(ArrayList mList)
    {
        ColumnDeatilsBean xcd, rcd;
        MatchedAttributes ma, newma;
        ArrayList dmList = new ArrayList();

        for(int i = 0; i < mList.size(); i++)
        {
            ma = (MatchedAttributes)mList.get(i);









            newma = new MatchedAttributes(ma.attr1, ma.attr2, 100);
                    dmList.add(newma);
                    continue;
                }else if(xcd == null && rcd != null){
                    continue;
                }else if(xcd != null && rcd == null){
                    continue;
                }else if(xcd.key.trim().equals(rcd.key.trim())) //||
                      //xcd.key.trim().startsWith("PK_")||
                      xcd.key.trim().startsWith("FK_") ||
                    //rcd.key.trim().startsWith("PK_")||
                    rcd.key.trim().startsWith("FK_")
                {
            newma = new MatchedAttributes(ma.attr1, ma.attr2, 100);
                    dmList.add(newma);
                                        }
        }
        return dmList;
    }
    public static void main(String args[])
    {

        new SchemaMatching().setVisible(true);

    }
}
```

```
            xcd = getColumnDetails(ma.attr1, "xml");
            rcd = getColumnDetails(ma.attr2, "");
            if(xcd.datatype.trim().equals(rcd.datatype.trim()))
            {
                    if(xcd.length == rcd.length)
                {
        newma = new MatchedAttributes(ma.attr1, ma.attr2, 100);
        dmList.add(newma);
                }
            }
        }
        return dmList;
    }
    public ArrayList constraintMatcher(ArrayList mList)
    {
        ConstraintDetailsBean xcd, rcd;
        MatchedAttributes ma, newma;
        ArrayList dmList = new ArrayList();

        for(int i = 0; i < mList.size(); i++)
        {
            ma = (MatchedAttributes)mList.get(i);
            xcd = getConstraintDetails(ma.attr1, "xml");
            rcd = getConstraintDetails(ma.attr2, "");
            if(xcd == null && rcd == null)
            {
```

9. APPENDIX-2

CHAPTER 9

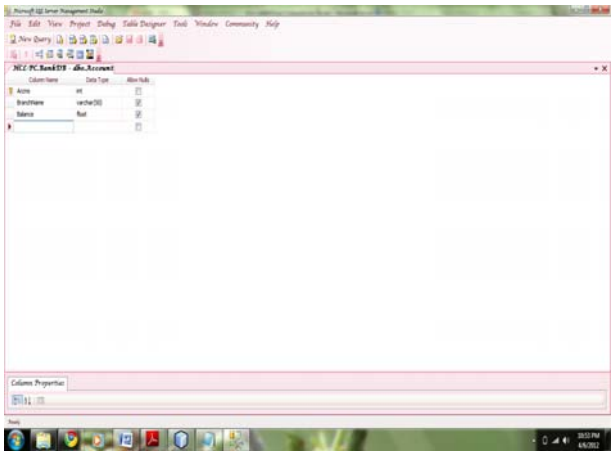
APPENDIX-2

9.1 XML SCHEMA:

```
<?xml version='1.0'?>
<Database>
    <Database name="Bank"/>
<tables>
    <table name="Acc">
    <constraints>
    </constraints>
    <metadata>
    <column name="Acc" typename="int" size="10"/>
    <column name="Branch" typename="varchar" size="50"/>
    <column name="Deposit" typename="float" size="15"/>
    <column name="Custn" typename="varchar" size="50"/>
    </metadata>
    <rowset>
    </rowset>
    </table>
    <table name="Br">
    <constraints>
    <column name="BID" key="PK_Branch1"/>
    </constraints>
    <metadata>
    <column name="ID" typename="varchar" size="50"/>
    <column name="Branch" typename="varchar" size="50"/>
    <column name="Asserts" typename="varchar" size="50"/>
    </metadata>
    <rowset>
    </rowset>
    </table>
</tables>
</Database>
```

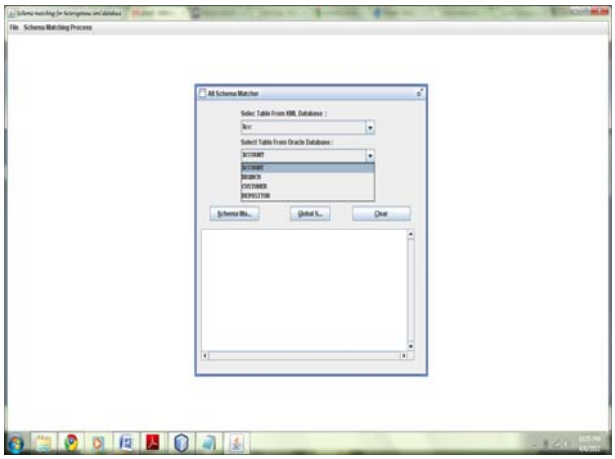
9.2 SCREEN SHOTS:

SQL SCHEMA



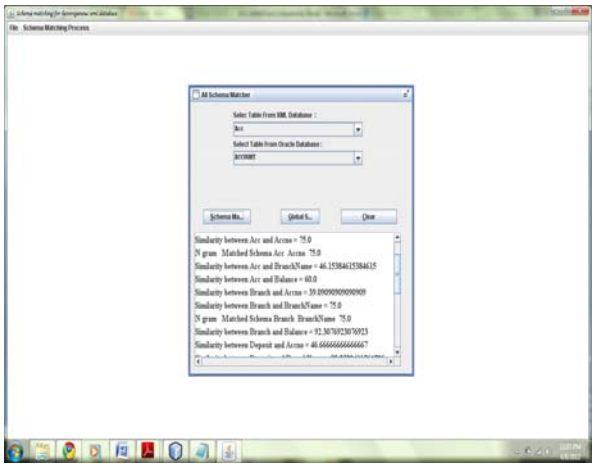
SCHEMA SELECTION

Schemas of XML and SQL are selected for matching.



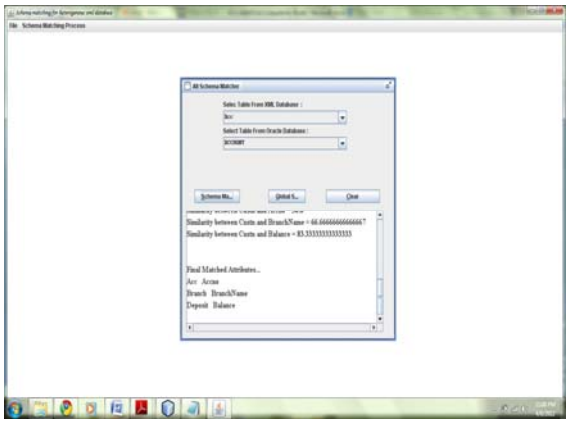
SCHEMA MATCHING

Percentage of similarities of all attributes compared by n-gram approach and synonym table approach are displayed.



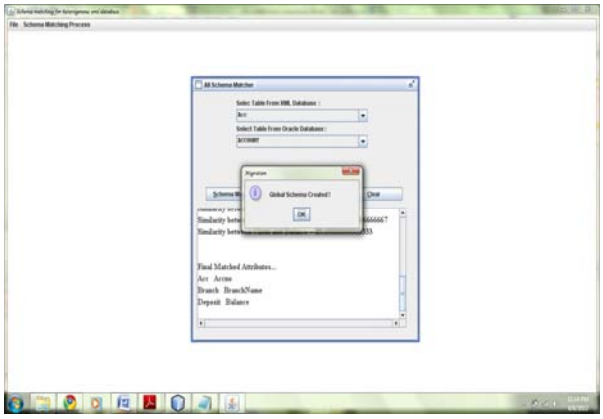
MERGED ATTRIBUTES

Attributes that are to be merged after comparing their data types and constraints are displayed.



GLOBAL SCHEMA CREATION

Global schema is created after matching the two schemas.

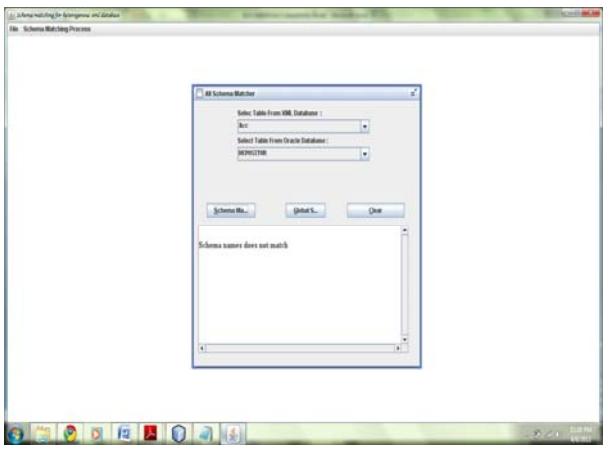


GLOBAL SCHEMA

<?xml version='1.0'?>
<GlobalSchema>
<schema name="Acc_ACCOUNT">
<constraints>
<column name="Accno" key="PK_Account"/>
</constraints>
<metadata>
<column name="Accno" typename="int" size="10"/>
<column name="BranchName" typename="varchar" size="50"/>
<column name="Balance" typename="float" size="15"/>
<column name="Custn" typename="varchar" size="50"/>
</metadata>
</schema>
</GlobalSchema>

SCHEMA MATCHER

Attributes are not compared if schemas does not match.



10. REFERENCES

CHAPTER 10

REFERENCES

[1]Yaser Karasneh, Hamidah Ibrahim, Mohamed Othman, Razali Yaakob, Matching Schemas of Heterogeneous Relational Databases, *Proceedings of the second international Conference on Applications of Digital Information and Web Technologies, 2009, pp.1-7.*

[2]Madhavan, J., Bernstein, P., Doan, A., and Halevy, A., Corpus-based Schema Matching, *Proceedings of the Twenty-first International Conference on Data Engineering, 2005, pp. 75-68.*

[3]Madhavan, J., Bernstein, P., and Rahm, E., Generic Schema Matching with Cupid, *Proceedings of the 27th International Conference on Very Large DataBases, 2001, pp. 49-58.*

[4]Bilke, A. and Naumann, F., Schema Matching using Duplicates, *Proceedings of the Twenty-first International Conference on Data Engineering, 2005,* pp. 69-80.

[5]Lu, J., J. Wang, and S. Wang, An Experiment on the Matching and Reuse of XML Schemas, *Proceedings of the International Conference on Web Engineering (ICWE), LNCS 3579, 2005, pp. 273-284.*

[6] Xu, L. and Embley, D., Discovering Direct and Indirect Matches for Schema Elements, *Proceedings of the Eight International Conference on Database Systems for Advanced Applications*, 2003, pp. 39-46.

[7]Ram, S. and Park, J., Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts, *IEEE Transactions on Knowledge and Data Engineering*, 16(2), 2004, pp. 189-202

[8]Rahm, E. and Bernstein, P., A Survey of Approaches to Automatic Schema Matching, *The VLDB Journal*, 2001, pp. 335-350.

[9] http://db.cs.washington.edu/papers/CupidTechReport.pdf

[10] http://disi.unitn.it/~p2p/RelatedWork/Matching/Paper_81_CIDR.pdf

[11] http://en.wikipedia.org/wiki/Schema_matching

[12]http://www.isi.edu/info-agents/workshops/ijcai03/papers/Madhavan-ijca-corpus2.pdf

[13] http://www.w3schools.com/xml/

[14] http://www.w3.org/XML/

[15]http://www.codeproject.com/Articles/11157/An-improvement-on-capturing-similarity-between-str