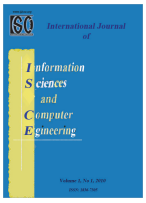
Information Sciences and Computer Engineering, Vol. 1, No. 2, (2010) 61–67

PHARUS

International Journal of
Information Sciences and Computer Engineering

journal homepage: http://www.ijisce.org



A Role based Address Cleaner

Md. Abul Bashar*, Md. Mashiur Rahman, Abdullah Al Rahed, Md. Misbahul Alam Chowdhury, Md. Pervez Sajjad

Department of Computer Science and Engineering, Shah Jalal University of Science and Technology, Sylhet, Bangladesh

***Abstract*– About 80% to 90% of governmental and business data collections contain address information. In many cases, address records are captured and/or stored in a free-form or inconsistent manner. There are many causes to dirty data: misuse of abbreviations, data entry mistakes, control information hiding, missing fields, spelling, outdated codes etc. Due to the 'garbage in, garbage out' principle, dirty data will distort information obtained from it. The purpose of address cleaning is to maximize the value of address data and ensure that every address is spelt correctly and properly structured. This improves accuracy and standardization in mailing, boosts company image, reduces mailing costs, and through geocoding opens up a number of opportunities to support strategic decisions through accurate spatial analysis. We report the implementation of a role based address cleaner. In this address cleaner we define indicators and grammars for address cleaning and thus make the cleaning process configurable.**

Keyword: Address cleaning, address standardization, data integration, natural language processing.

1. Introduction

Most real world address collections contain noisy, incomplete and incorrectly formatted information [1, 2, 3, 4]. Address data is often captured and stored with typographical and phonetical variations. Moreover, such data may be recorded or captured in various, possibly obsolete formats, and data items may be missing or contain errors [5]. For address data to be useful and valuable, it needs to be cleaned and standardized into a well defined format [6, 7]. For example various abbreviations should be converted into standardized forms, and postcodes should be validated using official postcode lists. Address cleaning and standardization are important first steps in many applications [1, 2, 8]. The cleaning and standardization of address is especially important for data integration, to make sure that no misleading or redundant information is introduced (e.g. duplicate records). Besides elimination of duplicates [9], the integration process contains the transformation of data into a form desired by the intended application and the enforcement of domain dependent constraints on the data [10]. The main task of address cleaning and standardization is the conversion of the raw input address into well defined, consistent forms and the resolution of inconsistencies in the way information is represented or encoded [11].

*Corresponding author:
Email address: basharcse@gmail.com, Ph: +88 01735112020

The main objective for collecting and processing address data is to fulfill different tasks in (i) administration, e.g., to keep track of the employees in a company, the customers of our company, or the sales volume of our companies branches, (ii) supporting business process, e.g., using the customers address list for direct mailing to advertise new products. Consider a company using a list of consumers with their addresses and buying habits and preferences to advertise a new product by direct mailing. Invalid addresses cause the letters to be returned as undeliverable. People duplicated in the mailing list account for multiple letters being send to the same person, leading to unnecessary expenses and frustration [10].

Usually the process of address data cleaning cannot be performed without the involvement of a domain expert, because the detection and correction of anomalies requires detailed domain knowledge. The ability for comprehensive and successful address cleansing is limited by the available knowledge and information necessary to detect and correct anomalies in address [10].

The existence of anomalies in real-world address data motivates the development and application of address cleansing methods [10, 12]. With our implemented role based address cleaner we are now able clean address data more correctly. About 1.58 Million addresses across South Africa were cleaned with this cleaner and success rate was very high. Whenever cleaner failed to handle any row address, we did necessary configuration in xml files where we defined indicators and grammars and thus our cleaner was optimized most.

2. Existing Approaches for Data Cleaning

Muller H et al. showed different existing approaches for data cleaning in [10]. Here in the following we summarize these cleaning techniques.

1. AJAX: AJAX [5, 12] is an extensible and flexible framework attempting to separate the logical and physical levels of data cleansing. The logical level supports the design of the data cleansing workflow and specification of cleansing operations performed, while the physical level regards their implementation. AJAX major concern is transforming existing data from one or more data collections into a target schema and eliminating duplicates within this process. For

this purpose a declarative language based on a set of five transformation operations is defined. The transformations are mapping, view, matching, clustering, and merging.

2. FraQL: FraQL [13, 14] is another declarative language supporting the specification of a data cleansing process. The language is an extension to SQL based on an object-relational data model. It supports the specification of schema transformations as well as data transformations. at the instance level, i.e., standardization and normalization of values. This can be done using user-defined functions. The implementation of the user defined function has to be done for the domain specific requirements within the individual data cleansing process.

3. Potter's Wheel: Potter's Wheel [3] is an interactive data cleansing system that integrates data transformation and error detection using spreadsheet-like interface. The effects of the performed operations are shown immediately on tuples visible on screen. Error detection for the whole data collection is done automatically in the background. A set of operations, called transforms, are specified that support common schema transformations without explicit programming. These are value translations, that apply a function to every value in a column, One-to-one mappings that are column operations transforming individual rows, and Many-to-many mappings of rows solving schematic heterogeneities where information is stored partly in data values, and partly in the schema. The anomalies handled by this approach are syntax errors and irregularities.

4. ARKTOS: ARKTOS [15] is a framework capable of modelling and executing the Extraction-Transformation-Load process (ETL process) for data warehouse creation. Data cleansing is an integral part of this ETL process which consists of single steps that extract relevant data from the sources, transform it to the target format and cleanse it, then loading it into the data warehouse. A meta-model is specified allowing the modelling of the complete ETL process. The single steps (cleansing operations) within the process are called activities. Each activity is linked to input and output relations. The logic performed by an activity is declaratively described by a SQL-statement. Each statement is associated with a particular error type and a policy which specifies the behaviour (the action to be performed) in case of error occurrence.

5. IntelliClean: IntelliClean [16, 17] is a rule based approach to data cleansing [18] with the main focus on duplicate elimination [19]. The proposed framework consists of three stages. In the Pre-processing stage syntactical errors are eliminated and the values are standardized in format and consistency of used abbreviations. It is not specified in detail, how this is accomplished. The Processing stage represents the evaluation of cleansing rules on the conditioned data items that specify actions to be taken under certain circumstances. There are four different classes of rules. Duplicate identification rules specify the conditions under which tuples are classified as duplicates. Merge/Purge rules specify how duplicate tuples are to be handled. It is not specified how the merging is to be performed or how its functionality can be declared. If no merge/purge rule has been specified, duplicate tuples can also manually be merged at the next stage. Update rules specify the way data is to be updated in a particular situation. This enables the specification of integrity constraint enforcing rules. For each integrity constraint an Update rule defines how to modify the tuple in order to satisfy the constraint. Update rules can also be used to specify how missing values ought to be filled-in. Alert rules specify conditions under which the user is notified allowing for certain actions.

There is no cleaning technique that specially focuses on address cleaning. As about 80% to 90% of governmental and business data collections contain address information [8] so accuracy in address cleaning is vital and should be handled specially. Our technique gives special focus on address cleaning and its accuracy is high comparing to existing techniques for data cleaning. Also as our technique is role based, any kind of address cleaning is possible just defining grammars there is no need to change the s/w system. We are hope our technique will play an important role in data cleaning technology.

3. Description of Our Role Based Address Cleaning Process

We define indicators and grammars [20, 21, 22, 23] for address cleaning process and make the cleaning process configurable. There are two supporting files:

a. **Indicator List File:** An 'indicator' is a type of address that helps to retrieve meaningful information from an unformatted raw address using grammars. 'Street', 'Building', 'PO Box' etc are few example of common indicators used. An indicator may have synonyms. For example, 'Avenue' and 'Street' are synonym indicators whereas 'Building' and 'House' are also synonym. The xml file 'IndicatorList.xml' attached with GAC package contains all the indicator names and their synonyms.

b. **Grammar File:** Grammars are rules for address cleaning. These are defined in an xml file named "AddressCleaning-Grammar.xml". The file is attached with GAC package.

The address cleaning process goes through two major steps:

a. **Preprocessing:** Some preprocessing is done on unformatted raw address so that a number of 'smart' tokens can be constructed from raw address. Each token have some 'attribute (Token type, Token value etc)'.

b. **Cleaning:** Grammars are applied on constructed smart tokens.

Let us discuss the whole process with a sample input string.

Sample Input String: Unit 10 Pink garden building

a. **Preprocessing:** Preprocessing of an unformatted raw address consists of several sub steps. The steps are described below with our sample input string.

I. Alphanumeric/Numeric Tagging: Tokenize the space delimited input string & mark alphanumeric/numeric tokens.

Before this step, the input string is: *Unit 10 Pink garden building*

After this step, the input string is: *Unit, [N]10, Pink, garden, building*

Here comma is put between two space-delimited token. "**[N]**" stands for alphanumeric/numeric value and precedes that token.

II. Indicator Tagging: Presence of any indicator in input string is identified and tagged.
Before this step, the input string is: *Unit, [N]10, Pink, garden, building*
After this step, the input string is: *[I=UnitNoName]Unit, [N]10, Pink, [I=Building]garden, [I=Building]building*
Here if the token value matches with 'IndicatorTitle' type indicator or with its synonym then "[I=IndicatorTitle]" is put in front of that token.
III. Construct Smart Tokens: Input string processed at step-I & step-II is tokenized [24]. Each token represents a smart object having information of its own. From these tokens smart tokens are constructed as bellow - Input string to this step: *[I=UnitNoName]Unit, [N]10, Pink, [I=Building]garden, [I=Building]building*
Generated smart tokens:

Element-1:
TokenValue = *Unit*
TokenType = **Indicator**
IndicatorTitle = **UnitNoName**

Element-2:
TokenValue = 10
TokenType = **Numeric**
IndicatorTitle = ""

Element-3:
TokenValue = *Pink*
TokenType = **String**
IndicatorTitle = **""**

Element-4:
TokenValue = *garden*
TokenType = **Indicator**
IndicatorTitle = **Building**

Element-5:
TokenValue = *building*
TokenType = **Indicator**
IndicatorTitle = **Building**

Here (I) The order of the constructed smart tokens is significant. This order is preserved as same as they appeared in unformatted raw address. (II) Tokens are called 'smart' as each of the tokens contain information of what type of token it is. Token types may be – Number, Indicator, String, Undefined etc.
IV. Change of Token Type if needed (Preprocessing special cases): Change **TokenType** of one or more smart token (if needed) depending on the presence of some other particular token in particular direction in the input string.

Consider our sample input string again: *[I=UnitNoName]Unit, [N]10, Pink, [I=Building]garden, [I=Building]building*
Here both "garden" and "building" is Building Type indicator. But it's clear that 'garden' actually stands here as part of building name, not indicator. That is, desired output is [Building]="Pink garden". So TokenType of "garden" should be changed from **Indicator** to **String** before grammar templates are applied.
Following grammar of preprocessing does this task:

<**ProcessSpecialInput**>
<**RootToken** TokenType="Indicator">**Building** </**RootToken**>
<**TargetToken** TokenType="Indicator" SearchDirection= "Backward" Span="SingleLine" Position="Adjacent"> **Building**</**TargetToken**>
<**HandleTargetToken**>
<**ChangeTo**>**String**</**ChangeTo**>
</**HandleTargetToken**>
</**ProcessSpecialInput**>

According to the above grammar –

(1) If a building type indicator is found [Root Token]
(2) And if, on backward direction of root token, a second building type indicator is found [Target Token]
(3) And if they are adjacent and in same single line (they are not separated by comma or other delimiter in user string)

Then, change TokenType of that second indicator from Indicator Type to String Type. In our sample example, according to the grammar, RootToken = "building" (Element-5) and TargetToken = "garden" (Element-4). The constraints are fulfilled. So, **TokenType** of "garden" should be changed from **TokenType** Indicator to **TokenType** String. Thus the smart tokens now look like:

Element-1:
TokenValue = *Unit*
TokenType = **Indicator**
IndicatorTitle = **UnitNoName**

Element-2:
TokenValue = 10
TokenType = **Numeric**
IndicatorTitle = ""

Element-3:
TokenValue = *Pink*
TokenType = **String**
IndicatorTitle = **""**

Element-4:
TokenValue = *garden*
TokenType = **String**
IndicatorTitle = **Building**

Element-5:
TokenValue = *building*
TokenType = **Indicator**
IndicatorTitle = **Building**

After the above (I) to (IV) tasks of preprocessing are performed on unformatted raw input address, we now have a number of formatted smart tokens. These smart tokens are now suitable for cleaning grammars to be applied on them in next step of role based address cleaning.

Cleaning: We have a number of cleaning grammars defined in "AddressCleaningGrammar.xml" file. Each of the grammar is applied on the set of constructed smart tokens one by one. Thus the raw address is gradually filtered and at the end, we find the cleaned outputs of the raw address.
The set of smart tokens we've got already from our input string after preprocessing are:

Element-1:
TokenValue = *Unit*
TokenType = **Indicator**
IndicatorTitle = **UnitNoName**

Element-2:
TokenValue = 10
TokenType = **Numeric**
IndicatorTitle = ""

Element-3:
TokenValue = *Pink*
TokenType = **String**
IndicatorTitle = **""**

Element-4:
TokenValue = *garden*
TokenType = **String**
IndicatorTitle = **Building**

Element-5:
TokenValue = *building*
TokenType = **Indicator**
IndicatorTitle = **Building**

The following two grammars would be responsible and would clean our sample input:

GRAMMER NO – **G1:**
<**ProcessField**>
<**BaseToken** TokenType="Indicator">**UnitNoName**</**BaseToken**>
<!– Searching will always start at "BaseToken" –>
<**RequiredTokens**>
<RequiredToken TokenType="Indicator" SearchDirection="Any">**Building**</RequiredToken>
<!– Tokens that are needed to be presented in the set of smart tokens –>
</**RequiredTokens**>
<**ProcessFieldInfo**>
<TokenType>**Number**</TokenType>
<!– Type of token to be read –>
<ReadDirection>**Any**</ReadDirection>
<!–The direction, in which reading will occur (starting at "BaseToken") –>
<ReadNumberOfTokens>**1**</ReadNumberOfTokens>
<!– How many token will be read in that particular direction –>
<OutputFieldName>**UnitNo**</OutputFieldName>
<!– Name of the output field where cleaned token value will go –>
</**ProcessFieldInfo**>
<**ProcessFieldInfo**>
<TokenType>**Indicator**</TokenType>
<ReadDirection>**NA**</ReadDirection>
<!–'ReadDirection' values can be Backward / Forward / Any / NA etc. –>
<!–ReadDirection 'NA' means value of BaseToken itself will go to Output Field –>
<OutputFieldName>**UnitNoName**</OutputFieldName>
</**ProcessFieldInfo**>
</**ProcessField**>

GRAMMER NO – **G2:**
<**ProcessField**>
<**BaseToken** TokenType="Indicator"> **Building**</**BaseToken**>
<**ProcessFieldInfo**>
<TokenType> **String** </TokenType>
<ReadDirection>**Any**</ReadDirection>
<ReadNumberOfTokens>**5**</ReadNumberOfTokens>
<!– Allow to read maximum no of 5 string type token if found –>
<OutputFieldName> **Building** </OutputFieldName>
</**ProcessFieldInfo**>
</**ProcessField**>

Let us discuss how these cleaning grammars do the tasks:

When the set of smart tokens will go through grammar no **G1**, BaseToken is matched with Element-1 of intelligent vector. Defined required token (BuildingType Indicator) is also found in the set (Element-4). Thus conditions are fulfilled and this grammar is applicable for our intelligent vector. Each block under the tag "ProcessFieldInfo" corresponds to reading token values for a particular output field. Reading will starts at BaseToken. However, reading in a particular direction continues until –

I. Desired Read Token Type is found and read successfully
II. Another Indicator type token is found in the reading direction
III. Begin or end of the set of tokens is reached

Cleaned values will be passed to proper output fields:

[UnitNo] = 10
[UnitNoName] = *Unit*

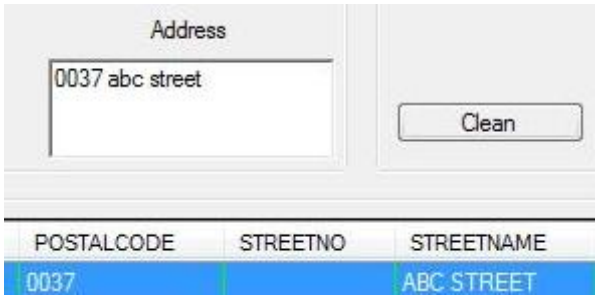
Secondly, when the set of smart tokens will be filtered by grammar no **G2**, BaseToken value is "Building" type indicator and it is matched with Element-4. No other "Required" or "NotRequired" token is defined here. Thus this Grammar is also applicable. When set of smart tokens goes through this template, we will get the cleaned output:

[Building] = *Pink garden building*

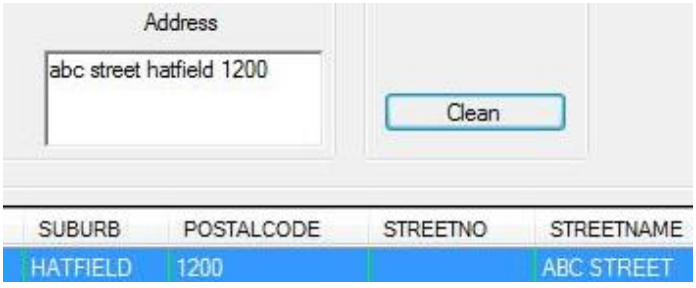
Thus our raw unformatted address is cleaned. Following is the screen shot of our system -

Postal Code Computation: It is assumed that a postal code consists of 4-digit number. The raw unformatted address is searched for the presence of any valid postal code. When a 4-digit number is found, its validity as a postal code is checked using postal code hierarchy. If it is found in hierarchy, then the decision finally depends on few cases:

Case-1: If any leading zero is found in the 4-digit number, then it is certainly a valid postal code. The number is retrieved as postal code and '**erased**' from the raw address prior to indicators processing. Following is the screen shot of our system with this case -

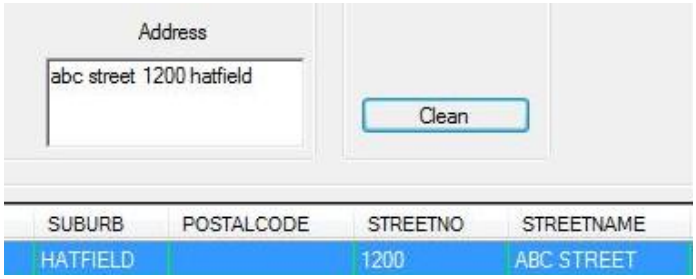


Case-2: If no leading zero is found but the 4-digit number appears after suburb name in the input string, then it is also a valid postal code. The number is retrieved as postal code and '**erased**' from the raw address prior to indicators processing. Following is the screen shot of our system with this case –



Case-3: If no leading zero is found and the 4-digit number doesn't appear after suburb name, then the decision gets pending. The number is retrieved as postal code but "**left**" in the raw address prior to indicators processing assuming that it also could be a street number etc. After indicator processing is done, it is checked again whether that 4-digit number still remains in Rest

Field. If it still remains in the Rest Field, then it is actually a postal code, otherwise not. Following two are the screen shots of our system with this case –



"1200" is considered here as Street Number as no other street number is found

"1400" is postal code as Street Number =1026 is found in input string

Town & Suburb Retrieval:

Before indicators processing is done, presence of any valid suburb or town is checked in the input string using the hierarchy. If any valid town or suburb found, they are retrieved but '**left**' in the raw address prior to indicators processing assuming that the retrieved town or suburb actually might be part of building name, street name etc. For example, "Hatfield Avenue" could be a valid street name though 'Hatfield' itself a valid suburb.

After indicator processing is done, it is checked again if already retrieved town or suburb name still remains in Rest Field. If they still remain in the Rest Field, they are actually valid town or suburb, otherwise not.

4. Output of our System

We have tested our system for 130 different formatted raw addresses and found it giving accurate result. Following are some of them –

Input	***Output***						
	Postal Code	***Street No***	***Street Name***	***Building***	***Building No***	***Unit Name***	***Unit***
unit no 99A abc building				*abc building*		*unit no*	*99A*
unit no 99 & 100 6th wasa building				*6th wasa building*		*unit no*	*99 & 100*
pqr building no 99 Laan street no 77 zzz		77	*Laan street*	*pqr building*			99
99 abc building no 77 xyz street		77	*xyz street*	*abc building*			99
77 street xyz 99 building abc		77	*street xyz*	*building abc*			99
12 central plaza road no 77		12	*central plaza road*				
12 central plaza building no 99				*central plaza building*	12	*no*	99
6 street 77		77	*6 street*				
6 street 77, no 99 abc building		77	*6 street*	*abc building*	99		
6 street, unit 99 abc building			*6 street*	*abc building*	99		
abc building 99, xyz street			*xyz street*	*abc building*			99
xyz street abc building no 99			*xyz street*	*abc building*		*no*	99
abc building, 12 street 99		99	*12 street*	*abc building*			
0018 xyz stree	0018		*xyz street*				
xyz street hatfield 1200	1200		*xyz street*				
xyz street no 99 abc building		99	*xyz street*	*abc building*			
77 12th way		77	*12th way*				
unit no 2345678 abc building	1200		*xyz street*	*abc building*		*unit no*	2345678
no 99 abc building 77 xyz street			*xyz street*	*abc building*	77	*no*	99
no 77 xyz street no 99 abc building		77	*xyz street*	*abc building*	99		

5. Conclusion

Addresses are often captured in noisy, incomplete and incorrect format. Therefore it is important that such data is transformed into a clean and standardized format before further processing. Correct and proper formatted addresses are important for governmental and businesses. Our address cleaner can clean and properly format raw addresses efficiently and effectively. Also our cleaner is reconfigurable by changing the xml files, so it can be optimized more and more as time goes without any change in the software code.

References

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publisher, 2001.

[2] E. Rahman and H. H. Do, "Data cleaning: Problems and current approaches," *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 23, no. 4, pp. 3247–3259, 2000.

[3] V. Raman and J. M. Hellerstein, "Potter's wheel: An interactive data cleaning system," in *Proceedings of the 27th International Conference on Very Large Data Bases*, pp. 381–390, 2001.

[4] J. I. Maletic and A. Marcus, "Data cleansing: Beyond integrity analysis," in *Proceedings of the Conference on Information Quality*, pp. 200–209, MIT, 2000.

[5] H. Galhardas, D. Florescu, D. Shasha, and E. Simon, "Ajax: An extensible data cleaning tool," in *Proceedings of the ACM SIGMOD on Management of data*, p. 590, 2000.

[6] P. Christen, T. Churches, and J. X. Zhu, "Probabilistic name and address cleaning and standardisation," *Proceedings of the Australasian Data Mining Workshop*, December 2002.

[7] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita, "Improving data cleaning quality using a data lineage facility," in *Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses*, p. 3, 2001.

[8] P. Christen, T. Churches, and A. Willmore, "A probabilistic geocoding system based on a national address file," in *In Proc. 3rd Australasian Data Mining Conf.*, 2004.

[9] M. L. Lee, H. Lu, T. W. Ling, and Y. T. Ko, "Cleansing data for mining and warehousing," in *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, pp. 751–760, 1999.

[10] J. Han and M. Kamber, *Problems, Methods, and Challenges in Comprehensive Data Cleansing*. Humboldt-Universitat zu Berlin zu Berlin, 2003.

[11] P. Christen and D. Belacic, "Automated probabilistic address standardization and verification," in *Australasian Data Mining Conference*, pp. 53–67, 2005.

[12] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita, "Declarative data cleaning: Language, model, and algorithms," in *Proceedings of the 27th VLDB Conference*, pp. 371–380, 2001.

[13] K. Sattler, S. Conrad, and G. Saake, "Adding conflict resolution features to a query language for database federations," *Australian Journal of Information Systems*, vol. 8, no. 1, pp. 116–125, 2000.

[14] K.-U. Sattler and E. Schallehn, "A data preparation framework based on a multidatabase language," 2001.

[15] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis, "Arktos: towards the modeling, design, control and execution of etl processes," *Information Systems*, vol. 26, pp. 537–561, 2001.

[16] M. L. Lee, T. W. Ling, and W. L. Low, "Intelliclean: A knowledge-based intelligent data cleaner," in *Proceedings of the ACM SIGKDD*, pp. 290–294, 2000.

[17] W. L. Low, M. L. Lee, and T. W. Ling, "A knowledge-based approach for duplicate elimination in data cleaning," *Information Systems*, vol. 26, pp. 585–606, 2001.

[18] A. Sleit, M. Al-Akhras, I. Juma, and M. Alian, "Applying ordinal association rules for cleansing data with missing values," *Journal of American Science*, vol. 5, no. 3, pp. 52–62, 2009.

[19] J. J. Tamilselvi and V. Saravanan, "Detection and elimination of duplicate data using token-based method for a data warehouse: A clustering based approach," *International Journal of Dynamics of Fluids*, vol. 5, no. 2, pp. 145–164, 2009.

[20] C. Gardent, B. Guillaume, G. Perrier, and I. Falk, "Maurice gross' grammar lexicon and natural language processing," in *Language and Technology Conference*, pp. 120–123, 2005.

[21] D. Chiang, *Evaluating grammar formalisms for applications to natural language processing and biological sequence analysis*. PhD thesis, University of Pennsylvania, 2004.

[22] R. Hwa, *Learning Probabilistic Lexicalized Grammars for Natural Language Processing*. PhD thesis, Harvard University, 2000.

[23] A. Arasu, S. Chaudhuri, and R. Kaushik, "Learning string transformations from examples," tech. rep., 2009.

[24] "Cs 164: Programming languages and compilers (class notes #2: Lexical)." http://www.cs.berkeley.edu/ hilfingr/cs164/public_html/lectures/note2.pdf.

M. A. Bashar received the B.Sc. (Eng) degree in Computer Science and Engineering from Shah Jalal University of Science and Technology, Sylhet, Bangladesh, in 2008. He worked as Software Engineer in SDSL (It is the local development branch of AfriGIS, South Africa (http://www.afrigis.co.za)) from August, 2008 to April, 2010. He has recently joined to the Department of Computer Science and Engineering at Comilla University as a Lecturer. His research interests are robotics, data analysis and pattern recognition, data mining, artificial intelligence, information retrieval, web intelligence, e-health, searching, web map, location based service, ubiquitous advertisement, and quantum computation.

M. Mashiur Rahman received the B.Sc. (Eng) degree in Computer Science and Engineering from Shah Jalal University of Science and Technology, Sylhet, Bangladesh. He worked as Software Engineer in SDSL (It is the local development branch of AfriGIS, South Africa (http://www.afrigis.co.za)). He has recently joined to Asenic (a software firm). His research interests are data analysis and pattern recognition, data mining, information retrieval, and ubiquitous advertisement.

Abdullah Al Rahed received the B.Sc. (Eng) degree in Computer Science and Engineering from Shah Jalal University of Science and Technology, Sylhet, Bangladesh. He is working as Software Engineer in SDSL (It is the local development branch of AfriGIS, South Africa (http://www.afrigis.co.za)). His research interests are data analysis and pattern recognition, data mining, information retrieval, and ubiquitous advertisement.

M. A. Chowdhury received the B.Sc. (Eng.) degree in Computer Science and Engineering from Shah Jalal University of Science and Technology, Sylhet, Bangladesh, in 2008. He worked as Software Engineer in SDSL (It is the local development branch of AfriGIS, South Africa (http://www.afrigis.co.za)). He has recently joined as software developer to Therap BD Ltd. (It is the technical brunch of Therap Services LLC, USA (http://www.therapservices.net)). His research interests are robotics, data analysis and pattern recognition, data mining, artificial intelligence, information retrieval, web intelligence, e-health, searching, web map, location based service, ubiquitous advertisement, and quantum computation.

M. P. Sajjad received the B.Sc. degree in Computer Science and Engineering from Shahjalal University of Science and Technology, Sylhet, Bangladesh, in 2007. Currently he is working as Software Engineer in SDSL (It is the local development branch of AfriGIS, South Africa (http://www.afrigis.co.za)). His research interests include information retrieval, data analysis, pattern recognition, building and managing distributed system.