

Data warehouse screening and personalizing agent

A. H. Mohamed, N. L. M. Noor & N. M. Noh

*Faculty of Information Technology and Quantitative Sciences,
MARU University of Technology, Malaysia*

Abstract

Intelligent software agents have been developed for many purposes these days. Software agents have been developed in many types of applications such as Internet search engines etc. This article is all about implementing the intelligent software agent in a data warehouse implementation. The first purpose of this software agent is to screen data in order to avoid data duplication in the data warehouse. The second objective of this research is to personalize the user requirements of data marts in data warehouse environment. The software agent consists of two basic modules. The first module is the information screening module. This module is implemented as an addition to the data staging process, before the data is loaded into the data warehouse. It functions as a pre-loading step, where the data to be loaded into the data warehouse has been cleaned, extracted and transformed. This module uses an enhanced Fuzzy String Matching to check the existence of data in the data warehouse. The second module involved is the personalization module. This is a module where the agent will study the history queries sent by users to the system. The queries may represent user requirements to find any pattern of time of the queries being sent to the system. The technique used by the agent to study users' queries is grouping the queries which are similar to each other. The largest group of queries is considered as the most needed query. However, this agent may be enhanced to produce a more robust intelligent agent.

Keywords: intelligent software agent, data warehouse, information screening, personalization.



1 Introduction

The objective of this research is to develop two agents which involved in data warehouse implementation. The first objective is to develop a screening agent to screen the data to avoid data duplication in the data warehouse. Besides that, the second purpose of the research is to develop a personalizing agent to study the pattern of queries sent by users according the table, fields related to the queries. Information filtering is a popular task in information retrieval. The term information filtering means the process of filtering a large amount of information in order to get relevant answers during searching process [2]. Therefore, the term information filtering is not suitable to be used in this research development.

The information filtering involved in this research is to check there is no duplication of data when the data is loaded into the data warehouse. This task seems simple because checking a data to match another data can be done directly. However, the concern of this research is the data which means the same but written in slightly different words. For example misspelled or abbreviation words. This problem always happen because a direct string matching always ignores misspelled and abbreviation words. These two cases always cause data duplication in data warehouse environment. Therefore, the term information filtering is best replaced by information screening, which involves screening for misspelled and abbreviated words to avoid data duplication. The focus of the screening process is the duplicate records due to inconsistent standards of data. Normally, problems occur when user of an operational database uploaded their data with the exact same data, but in different syntax, for example, the usage of abbreviation words, or misspelled words. It may cause the data replication in the data warehouse. Therefore, the cleaning process needs a tool that can identify the abbreviation or misspelled words. For information screening purposes, the prototype is developed using Fuzzy String Matching [3, 4].

2 Fuzzy String Matching

Fuzzy matching attempts to improve recall by being less strict but without sacrificing relevance [1]. The fuzzy matching algorithm is previously designed to find documents containing terms related to the terms used in the query. The assumption is that related words (in the English language) are likely to have the same core and differ at the beginning and/or end.

An algorithm created by Gonzales Cenelia [4] is used in this research. This algorithm performs well in order to match two strings approximately. It solves the problem of misspelled words (wrong letter position, left out letters, etc). However, it still needs some addition to make it more effective.

Enhancement on this algorithm involves the case of abbreviation words. Previous algorithm performs its matching process by counting the matching letters. Each time it founds unmatched letters, the increment will be in a lower rate. It takes care the problem of misplaced letters. The problem occurs when the compared word is the abbreviation of the other word. For example, the first word to be compared is “jalan” and the other word is “jln”. Both of these words give



the same meaning but its degree of membership returned is 0.6, using previous algorithm. It will not be a match if the user sets the fuzzy accuracy larger than 0.6. This is because the number of matched letters is too small, compared to the original word, which is, normally long words.

To cater this problem, we should study the abbreviation characteristics. In this research, the case studied is using Malay language abbreviation words. For example, “jalan” and “jln”, “sungai” and “sg”, “kampung” and “kg”, etc. The obvious characteristic is the number of the letters in the abbreviation is usually more or less the number of syllables in the original word.

This is shown clearly by the word “jalan” which has two syllables. “jln” has three letters in it. “kampung” contains two syllables as well, and its abbreviation “kg” contains two letters. Taking into consideration of other words that may be abbreviated, some addition is done onto the original algorithm.

The addition involved is checking the length of the words. In the original algorithm, if the distance of the length of the words is too big, it will totally being mismatched. This happens to be a problem when “kampung” which contains seven letters to be matched with “kg” which only contains two letters. Therefore, if the length of the words is obviously too far, the algorithm used only checked matched letters compared to the length of the abbreviation word. In other word, if the abbreviation is a subset of the original word, it will be assumed matched.

The suitable technique to solve the problem is to design an algorithm that represents the abbreviated words itself. Before the fuzzy match process being done onto the words, the length of the words should be studied. If the distance of both the two length of two compared words are too large, we should assume that the shorter word is the abbreviation word of another. But still fuzzy match process need to be apply, but it differs in the membership function applied.

The formula to calculate the degree of membership for the previous algorithm is:

$$A = \sum a(1.00) \text{ where } a \text{ is number of matched letters in sequence}$$

$$B = \sum b(0.75) \text{ where } b \text{ is number of matched letters not in sequence}$$

$$f(\text{match}) = (A + B) / \text{length of the longer word}$$

The formula to calculate the degree of membership for the enhanced algorithm is:

$$A = \sum a(1.00) \text{ where } a \text{ is number of matched letters in sequence}$$

$$B = \sum b(0.75) \text{ where } b \text{ is number of matched letters not in sequence}$$

$$g(\text{match}) = (A + B) / \text{length of the abbreviation word}$$

As a result of this algorithm implementation, the degree of membership of some sample words is shown in Table 1. As seen in the table, every abbreviation words are found as a match (degree of membership = 1.000) by using the enhanced algorithm. The differences are too big for certain words. For example, the second sample words “kampung” and “kg”. The result from the original algorithm found too low to be declared as a match.



Table 1: Comparison of results from two fuzzy string matching algorithms.

First word	Second Word	$f(x)$	$g(x)$
Jalan	Jln	0.600	1.000
Kampung	Kg	0.286	1.000
Bangunan	Bggn	0.500	1.000

$f(x)$: degree of membership using original algorithm

$g(x)$: degree of membership using enhanced algorithm

2.1 Personalization

The queries sent by user need to be personalized to find a pattern of queries in order to achieve the objectives of this research. These queries are kept in the form of Standard Query Language (SQL) [6]. The pattern discovered may represent a need of a data mart for particular user. Finding a pattern of a group of data involves clustering the data into some groups and studying the pattern of the group. In order to cluster the records, it needs to be classified into certain groups of records.

2.2 Clustering method

In general there are two types of hierarchical clustering methods; divisive and agglomerative hierarchical clustering [5, 7]. Divisive hierarchical clustering is a top-down strategy of clustering. All data objects (queries) are assigned to one large cluster before it being divided into smaller clusters. It will split the clusters until it achieve the level of clusters identified.

In the table of `sql_history`, the records basically represent the queries sent by user. These queries are actually being kept in a text file. Then they are parsed into the table in the database. After being parsed, they will become records in the table. It will contain the date it being parsed, the table name, fields name, conditions, orders field and group by fields.

The parsing algorithm is created to parse the queries sentence by sentence. The parsing process started with a sentence of query. For example: "SELECT FIELD_A FROM TABLE_A WHERE FIELD_A = 'A'". This sentence will be treated word by word. Next, the identifier in the SQL statement such as "SELECT", "FROM" and etc. is identified. The parser algorithm continues with following steps:

- Step 1: Assign the word to a variable 'word'.
- Step 2: Check if 'word' is between identifier "SELECT" and "FROM". If true, the word is the table name. Repeat step 1.
- Step 3: Check if 'word' is between identifier "FROM" and "WHERE". If true, the word is the field name. Repeat step 1.
- Step 2: Check if 'word' is between identifier "WHERE" and "ORDER" or "GROUP". If true, the word is the conditions. Repeat step 1.



- Step 4: Check if 'word' is after identifier "GROUP BY". If true, the word is the group. Repeat step 1.
- Step 5: Check if 'word' is after identifier "ORDER BY". If yes, the word is the order field of the query. Repeat step 1.

As mentioned earlier, after the query parsing process completed, the queries will be kept in a database. After a certain amount of time, an agent will automatically study the queries (which have been kept in database). The agent will cluster the query records according the user identification, table name, field name and conditions. After the four level of clustering, the node sets of records are assumed to be a cluster or a pattern of user queries. The pattern with largest size will be selected as the most needed query pattern. The selected pattern will then be studied further by the agent to find a pattern of the date. If there is any pattern of date, it is assumed that there is a need for a specific data mart for the particular user.

The clustering agent involves an algorithm that clusters the query records into sets of records. Clustering is done by classifying the records according the user identification, table names, field names and conditions of each records. Each record in a final node clusters need to be similar to each other. The following steps are the algorithm involved in clustering the query records.

- Step 1: Find the names of users. Each name will be a set name for user clusters.
- Step 2: Divide the large group of queries. Classify the records according to the user identification. The result will be the groups of records according to users.
- Step 3: Find the names of tables in each user group. Each table name will be a set name for table clusters.
- Step 4: For each user group, divide the group according to the table names. Repeat step 4 until no more user group left.
- Step 5: Find the names of fields in each table group. Each field name will be a set name for field clusters.
- Step 6: For each table group, divide the group according to the field names. Repeat step 6 until no more table group left.
- Step 7: Find the conditions of queries for each field group. Each condition will be a set name for condition clusters.
- Step 8: For each field group, divide the group according to the conditions of the query. Repeat step 8 until no more field group left.
- Step 9: The clustering process completed.
- Step 10: Find the largest group of records.
- Step 11: Pattern the date of the largest group of records.

The clustering agent will sent the cluster to date pattern agent. This agent will study the pattern of the dates. The algorithm involves in finding the date pattern is as follows. The object mentioned in the algorithm will represent a query.



- Step 1: Sort the objects according to the date in ascending order.
- Step 2: Find the difference of day, month and year between object i and object $i + 1$. Repeat step 2 until no more object left.
- Step 3: Analyze the difference of day, month and year. If the differences are all the same, a pattern found. For example, if the difference between object 0 and object 1 is 1 day, 1 month and 1 year. The difference between object 1 and object 2 is also 1 day, 1 month and 1 year. If the difference is the same for all object, the pattern is 1 day, 1 month and 1 year.
- Step 4: Send a notification to mailer agent.

Even though the agent may see the need on behalf of the user, the decision to create a data mart still depends on the system developer / analyst. This is because the creation of data mart requires much more analysis to be done. And the most important thing, it needs human judgment to make the decision.

3 Multiagent system design

The multiagent system consists of screening and personalizing agent is designed as shown in figure 2. The location of the screening agent is before the data / records from operational database being loaded into the data warehouse. The screening agent acts as a pre-loading process, after the data being extracted, cleaned and transformed in data staging phase. This is shown in figure 1.

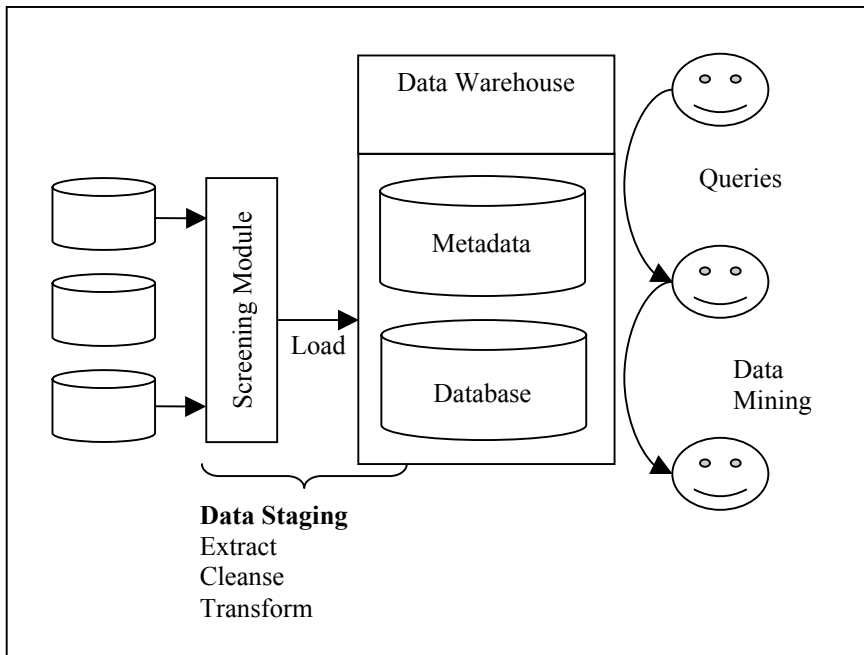


Figure 1: Data warehouse architecture with screening module.



As mentioned earlier, after the query parsing process completed, the queries will be kept in a database. After a certain amount of time, an agent will automatically study the queries (which have been kept in database). The agent will cluster the query records according the user identification, table name, field name and conditions. After the four level of clustering, the node sets of records are assumed to be a cluster or a pattern of user queries. The pattern with largest size will be selected as the most needed query pattern. The selected pattern will be studied further by the agent to find a pattern of the date. If there is any pattern of date, it is assumed that there is a need for a specific data mart for particular user. The information will be sent to a mailing agent, which will send a notification e-mail to the knowledge engineer, or system analyst who is responsible on developing the data warehouse. The location of the clustering module is as viewed in Figure 2.

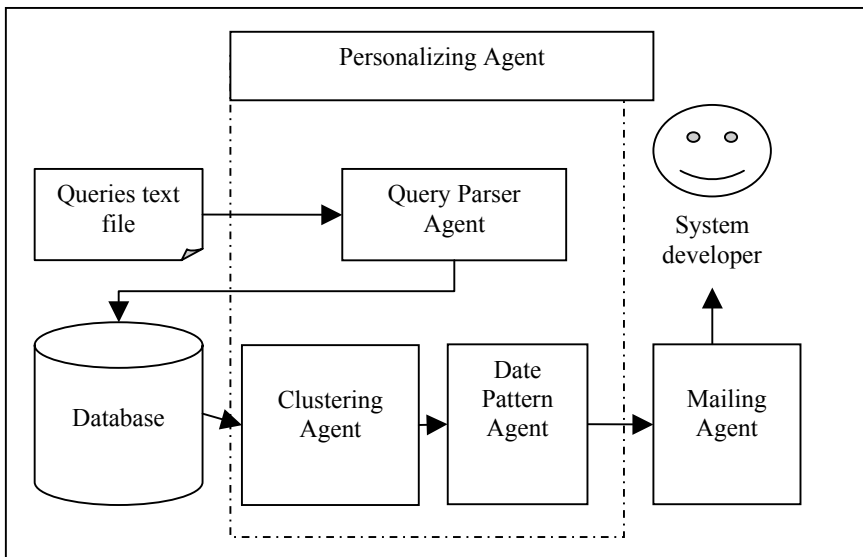


Figure 2: Personalizing agent.

The final product of this research is a prototype of an intelligent software agent. The prototype architecture of the multiagent system for this research is shown in the dotted box in Figure 3.

In this research, the focus is on the screening and personalizing agent. The query text file is assumed has been created by an interface agent. It is also assumed that a mailer agent will receive the output from the personalization agent.

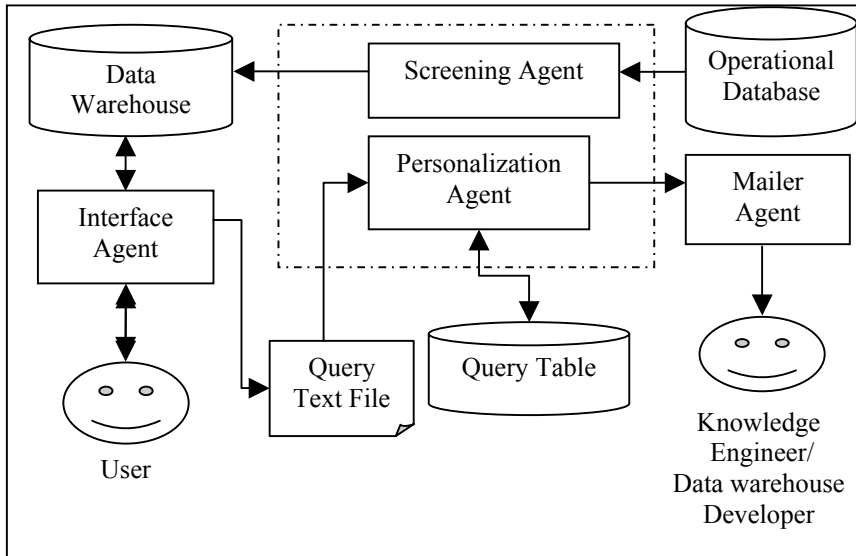


Figure 3: Multiagent system architecture.

4 Conclusion

As a conclusion, the software agent developed uses Java language programming does fulfill its objectives. Both objectives have been successfully handled by the software agent. The software agent is being developed to be robust and reliable.

The first objective, to develop an intelligent agent to find a popular pattern of user queries is handled by using a personalization agent. In this research, the agent clusters the queries; which are kept in database, according its user, table, field, and conditions. Then, the agent looks for the most popular cluster among all the clusters. The most popular clusters are being studied by the agent to check is there any pattern of the date for each queries in the clusters. If there is any pattern of the date, the cluster will represent one user's need. The agent will inform the database developer / administrator about the need for further action.

The next objective is to develop an intelligent agent to screen data to avoid redundancy in the data warehouse. The redundancies are always occurring because of obsolete data in source database, without being updated. The source data is matched with the related data in the destination data warehouse using fuzzy string matching technique. If the data is matched, it means the data doesn't need to be inserted into the destination table. Or else, the data is assumed to be a new data and need to be inserted into the destination table.

References

- [1] Christopher Le Bret. 2002. Fuzzy text search and robust string matching. <http://www.pmsi.fr/fuzstrng.htm>. (September 30, 2003).



- [2] Detlor, B. & Arsenault, C. 2002. Web information seeking and retrieval in digital library contexts: towards an intelligent agent solution. Volume 26. Number 6. <http://www.emeraldinsight.com/1468-4527.htm>. (August 20, 2003).
- [3] Girill, T.H. & Luk, C.H. 1996. Fuzzy matching as a retrieval-enabling technique for digital libraries. <http://www.asis.org/mid-year96/girillpaper.html>. (June 2, 2003).
- [4] Gonzales Cenelia. Fuzzy String Matching. <http://www.Planet-Source-Code.com>. (September 1, 2003).
- [5] Han, J. & Kamber M. 2001. *Data mining: concepts and techniques*. San Francisco: Morgan Kaufmann Pub.
- [6] Hawthorne, R. 2001. *Microsoft® SQL Server 2000™ database development from scratch*. USA: Que Corp.
- [7] Inmon, W. H. 2002. *Building the data warehouse*. 3rd Ed. New York: John Wiley & Sons.

