Data Warehouse Schema Evolution and Adaptation Framework Using Ontology

M.Thenmozhi
Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, India
thenmozhi@pec.edu

K.Vivekanandan
Department of Computer Science and Engineering
Pondicherry Engineering College
Puducherry, India
k.vivekanandan@pec.edu

***Abstract*— Data Warehouse systems aim at integrating data from multiple heterogeneous, distributed, autonomous data sources. Due to changing business needs the data warehouse systems are never meant to be static. Changes in the data source structure or business requirements would result in the evolution of data warehouse schema structure. When data warehouse schema evolves the dependent modules such as its mappings, queries and views gets affected. The existing works on data warehouse evolution focus only on schema evolution at the physical level. As ontology seems to be a promising solution in data warehouse research, the proposed framework handles data warehouse schema evolution at ontological level. Moreover, it analyses the impact of the dependent modules and proposes methods to automatically adapt to changes.**

Keywords-Datawarehouse schema evolution; Multidimensional schema evolution,;Impact of data warehouse evolution

I. INTRODUCTION

The main idea of a data warehouse is the integration of large amounts of data gathered from different heterogeneous sources throughout an enterprise. The data within the data warehouse is arranged in the form of multidimensional model in order facilitate business analysis. A multidimensional model consists of entities such as fact, measures, dimensions and levels [5]. A fact is said to be the subject by which a business is analyzed and measure represents the how the business transactions are measured. The different perspective by which business is measure is called as dimensions. The granularity of the dimension is defined by the levels. For example, in a retail business, sales is said to be the fact, the measures are total sales, profit etc., product, customer, branch and time are the different dimensions. The product dimension can have product category as a level.

The design of a multidimensional model is carried by means of analyzing the business requirements and utilizing the knowledge from the data sources of the data warehouse. The data warehouse design process typically consists of conceptual, logical and physical design stages [11]. The data warehouse multidimensional model or schema may evolve during the design or at later stage of implementation. One of the reasons for evolution is due to changes in business requirements such as ambiguous or insufficient requirements, changes in requirements in later stages of data warehouse environments, generation of new requirements due to technological advances [14]. Another important reason is the evolving data sources which not only changes its data but as well its structure. These changes need to be incorporated in the data warehouse schema to make it valid. As the data warehouse is a complex environment, any change in the data warehouse schema structure affects various dependent modules such as data source to data warehouse mapping and its ETL (extract, transform and load) operations by which the warehouse is populated, queries and views. The existing works such as schema evolution or schema versioning mainly concentrated on data warehouse schema evolution at physical level [15]. The impact that the data warehouse schema changes has brought on the dependent modules has not been addressed. The main objective of this paper is to provide an data warehouse evolution and adaptation framework to verify the impact and automatically adjust the dependent modules. As ontology [4] seems to be a promising solution for data warehouse research, we use ontological supported framework to automate the evolution process.

II. RELATED WORK

[1] In this paper the authors present a formal model of a multi-version data warehouse and the set of operators with their formal semantics that support a DW evolution. They also study the impact analysis of the operators on DW data and user analytical queries. [10] They proposed a approach based on versioning called MVTDW which is composed of real versions and alternative versions. They defined some constraints to assure

integrity between versions and some algorithms to be applied for schema and instance changes on the TDW versions. Here the user has to define which version should use to answer queries. [16] In this paper they introduce an approach which enables the user to integrate his knowledge for analysis capabilities of the warehouse. They also present some elements that are necessary for its implementation i.e. management model for the data warehouse progression, algorithms, etc. Here the user knowledge is expressed in terms of "if-then" type rules. which are used to create granularity levels in dimension hierarchies. [12] The evolution framework is able to handle changes in data sources and also direct changes in a data warehouse schema. In the evolution framework the data warehouse versions are supported in the development environment as well as in reports in the user environment. [13] In this paper they handle schema evolution of certain extended hierarchies prevailing in the data warehouse. They take into account three hierarchies namely multiple alternative, parallel dependent and parallel independent hierarchies and defined constraints for it that need to be satisfied for enforcing semantics and schema correctness. They also proposed an algorithm for the evolution operator for parallel dependent hierarchy.

III. PROPOSED WORK

In this section we propose a framework for managing the data warehouse schema evolution and adaptation using ontology. When data source and requirement evolves the data warehouse schema need to be updated in order to provide up-to-date information to the users. Before making structural changes to the existing data warehouse physical schema our proposed work provides a method for updating the ontological representation of the schema. Given information about a data source, requirements and data warehouse as well as the changes in the data source or requirements, this method produces the updated version of the data warehouse schema at the ontological level. Figure1. represents the components of our framework. By making use of the ontological representation of our inputs we facilitate the automation (semi- automation) of the evolution task. First the changes occurred at data source or requirements are extracted from the corresponding ontology representation. Next the type of change and the entity affected by the change are derived and the change is propagated to the DW schema. The updated data warehouse ontology is validated by verifying its consistency. Finally the dependent entities such as mapping, queries and views are adapted automatically and impact of evolution is analyzed. We illustrate our approach using TPC-H [2] which is a decision support benchmark. The following sections describe our approach in detail.

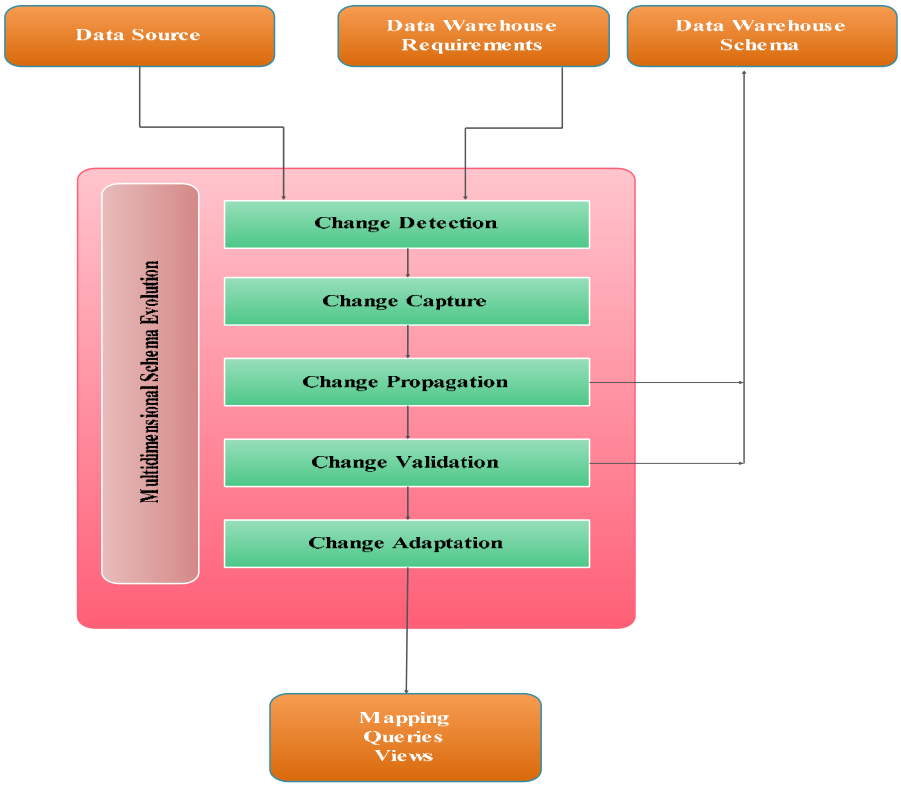

Figure 1. Data Warehouse Schema Evolution and Adaptation Framework

A. Input Representation

In order to automate the data warehouse schema evolution process we represent our input such as data source schema, data warehouse schema and data warehouse requirements in ontology format. Applying the reverse-engineering approach we define the ontological model of existing data sources and data warehouse system. A semantic mapping method is adapted to facilitate the transformation of data sources and data warehouse system from a relational model into an OWL [8] based structure. The ontology can be constructed using protégé tool [9]. The mapping from database meta-data such as E-R model or UML model to ontology could be done using the mapping rules as given below:

- The database table is mapped to an ontology class.
- If a database table is related to another, then the two tables are mapped to classes with parent-child relationship.
- If a database table is related to two tables, then the table is divided into two transferred classes.
- The primary key is mapped to a data type property of the ontology.
- The foreign key is mapped to an object property of the ontology.

The data source ontology (DSO) is represented as DSO = {C, DP, OP}. Where, C is the set of classes, DP is the set of data property and OP is the set of object properties of the ontology. Figure 2 represents the ontology for the TPC-H schema.

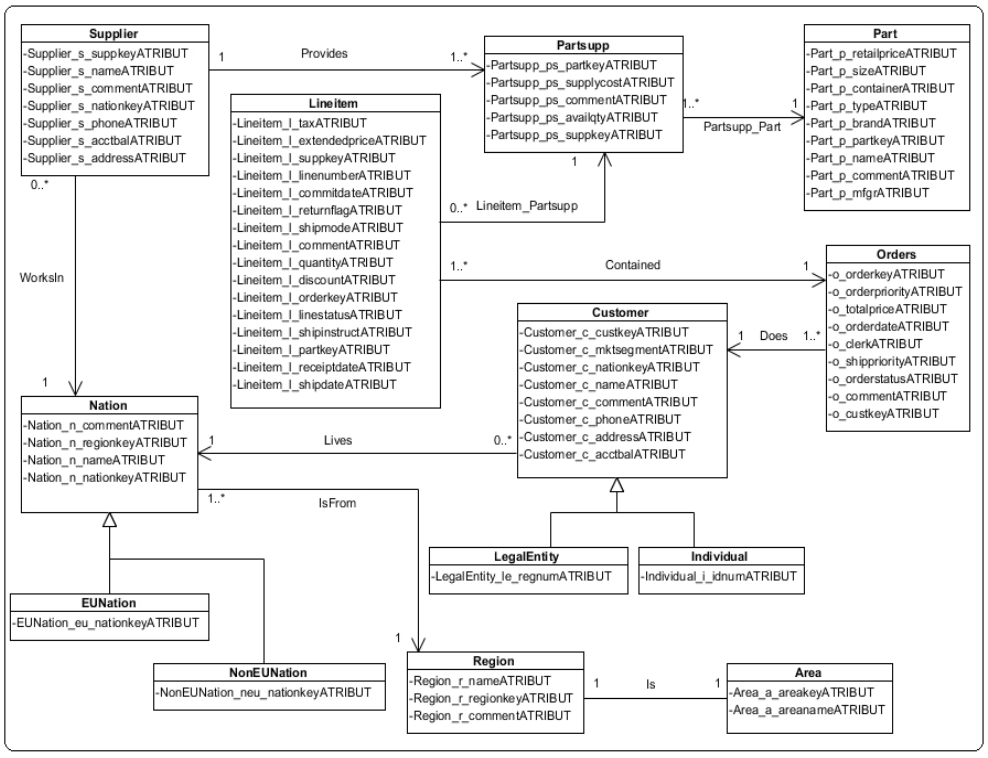

Figure 2. DSO representation of TPC-H data source

We assume that a formal requirement analysis for the given domain has been carried out earlier and the requirements based on i* modeling framework [3] is available. In order to capture the changes in requirements we represent it in a formal way using requirement ontology called as data warehouse requirements ontology (DWRO). Formally, DWRO = {SG, IG, DG, IR, M, C} where, SG is a set of OWL classes representing the strategic goals, IG is a set of OWL classes representing the information goals, DG is a set of OWL classes representing the decision goals, IR is a set of OWL classes representing the information requirements, M is a set of data type properties representing measures, C is a set of OWL classes representing the contexts. Figure 3 represents the ontology for the TPC-H requirements for data warehouse.

.

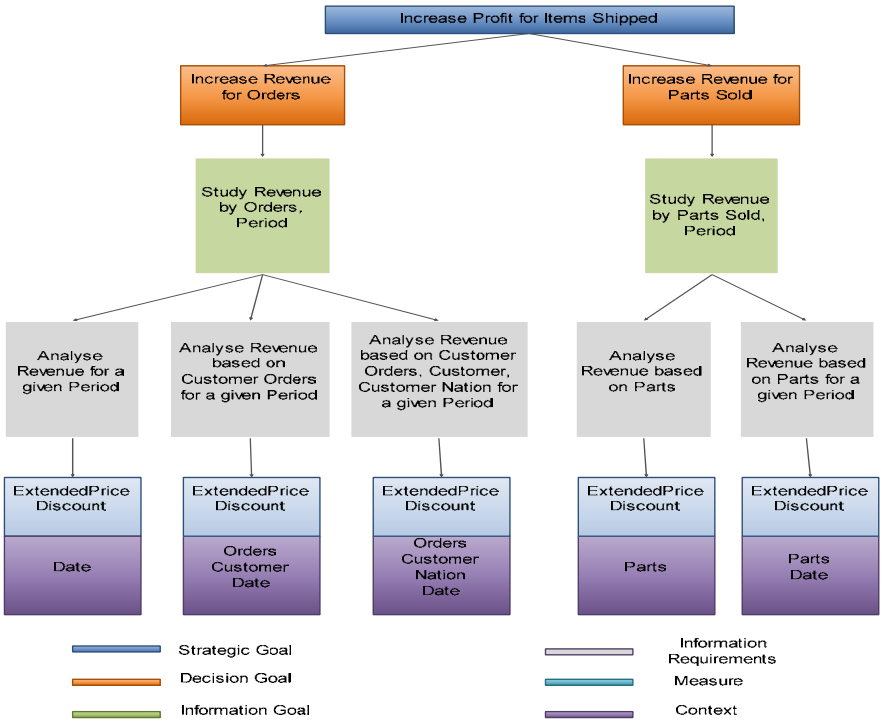

Figure 3. DWRO representation of TPC-H data warehouse requirements

Data warehouse schema can be formally defined as DWO = {F,FP,M,D,DP,RP} where, F is a set of OWL classes representing the fact, FP is a set of OWL classes representing the fact properties, M is a set of data type properties representing the measures of the fact, D is a set of OWL classes representing the dimensions, DP is a set of data type properties representing the dimension properties, RP is a set of object properties representing the relationship between facts and dimensions. Figure 4 represents the ontology for the TPC-H data warehouse.

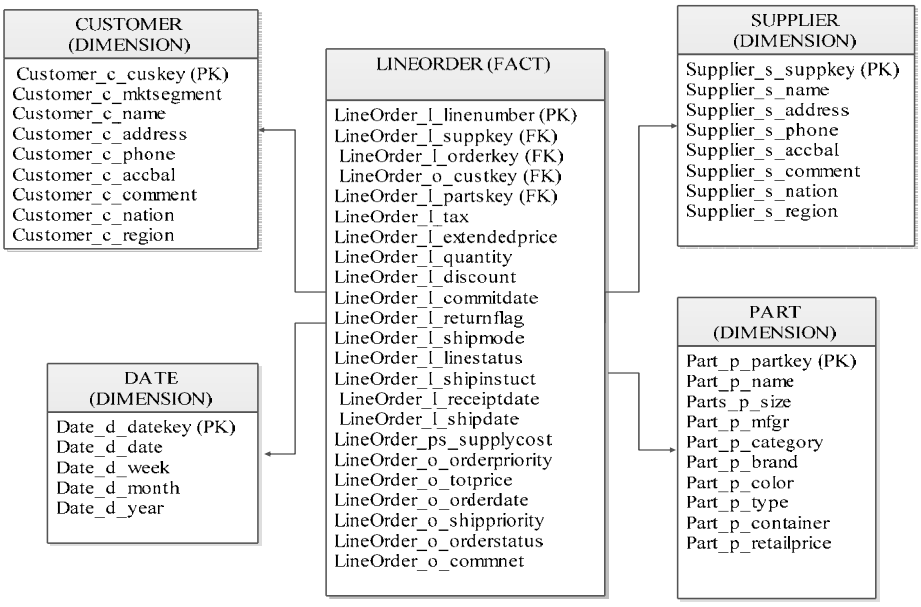

Figure.4. DWO representation of TPC-H data warehouse

B. Defining Evolution Operators

The possible changes that occur over the data warehouse schema are addition, deletion and rename. The set of evolution operators to represent the type of change and the concept changed are given in Table 1. The data warehouse elements such as Fact, Dimension, Measures etc., are subject to change hence the DWO need to be changed accordingly. Performing a change over the DWO may require additional changes to be executed over the ontology. For example, addition of a new dimension i.e., class to the DWO requires addition of its data property and object property.

TABLE 1. EVOLUTION OPERATORS

Type of Change	DW Schema Elements	Equivalent Ontology Concept Changed	Elementary Changes
Addition	Table (Fact, Dimension)	Class	Add Data Property Add Object Property
	Attributes (Measures, Descriptive)	Data Property	Add Property Domain Add Property Range
	Relationship (Primary Key, Foreign Key)	Object Property	Add Property Domain Add Property Range
Deletion	Table (Fact, Dimension)	Class	Delete Data Property Delete Object Property
	Attributes (Measures, Descriptive)	Data Property	Delete Property Domain Delete Property Range
	Relationship (Primary Key, Foreign Key)	Object Property	Delete Property Domain Delete Property Range
Rename	Table (Fact, Dimension)	Class	Rename Class (If required)
	Attributes (Measures, Descriptive)	Data Property	Rename Data Property (If required)
	Relationship (Primary Key, Foreign Key)	Object Property	Rename Object Property (If required)

C. Compute Mapping

The possible changes that occur over the data warehouse schema are addition, deletion and rename. The set of evolution operators to represent the type of change and the concept changed are given in Table 1. The data warehouse elements such as Fact, Dimension

In this step, we define the relationship between the data source and data warehouse. As the DW is populated with data obtained from several data sources through ETL (extract, load and transform) operations an ETL mapping exists for the DW under operation. Hence a mapping is produced between the attributes of the data source and data warehouse schema. Based on this mapping the ETL operations are identified. Since our approach uses ontology representation of the data source and data warehouse schema, we can automatically compute the mapping.

Based on the input ontologies DSO and DWO, an ontology matching algorithm first performs concept matching by measuring the similarity between concepts. Next, property matching is carried out by measuring the similarity between properties. The logical similarity measure is then performed based on the results of previous concept matching and property matching. The final matching results are produced after passing through the refinement process. The matching process is given in Algorithm 1. Here we use WordNet [7] matcher to perform the matching. Steps 1-6 compute the similarity between the classes in DSO and DWO. Steps 7-12 compute the similarity between the data properties in DSO and DWO. Here ci and cj represents the classes belonging to DSO and DWO respectively. And dpi and dpj represents the data properties belonging to DSO and DWO respectively.

1 *Algorithm ComputeMapping(DSO,DWO)*
2 **for all** $c_i \in DSO$ **do**
3 **for all** $c_j \in DWO$ **do**
4 similarity_score=**wordnet.getDistance(**c_i, c_j, pos)
5 **print(**c_i, c_j, similarity_score)
6 **end for**
7 **end for**
8 **for all** $dp_i \in DSO$ **do**
9 **for all** $dp_j \in DWO$ **do**
10 similarity_score=**wordnet.getDistance(**dp_i, dp_j, pos)
11 **print(**dp_i, dp_j, similarity_score)
12 **end for**

13 **end for**

D. Change Capture

Our next step is to extract information related to the changes which need to be propagated to the DWO. A number of changes, ranging from classes to properties, can affect the ontology which is captured using log. To address this we use Change Annotation Ontology (ChAO) [6] which acts as a log to capture the changes happening in the ontology. Table 2 represents a sample change set extracted from DSO. The following steps represents the methods use to extract property changed in ontology:

```
Algorithm ExtractChange(DSO,DWRO)
for each change in changes
   if(change.getAction().equals("Name_Changed"))
        if (change.getApplyTo().getComponentType().equals("Property"))
        if (change.getApplyTo().getInternalStatus().name().equals("CHANGED"))
                print(change.getAuthor());
                print(change.getAction());
                print(change.getTimestamp().getDate());
                print(change.getApplyTo().getComponentType());
                print (change.getContext());
                print (change.getApplyTo().getInternalStatus().name());
        end if
        end if
    end if
end for
```

TABLE 2. CHANGE SET

Data Source Change	Data Source Ontology Change	Entity Changed
ADDITION		
Table	Class	Promotion
Attribute	Data Property	Promotion _p_id
Attribute	Data Property	Promotion _p_name
Attribute	Data Property	Promotion _p_category
Attribute	Data Property	Promotion _p_subcategory
Attribute	Data Property	Promotion _p_cost
Attribute	Data Property	Promotion _p_begdate
Attribute	Data Property	Promotion _p_enddate
Attribute	Data Property	Promotion _p_total
Relationship	Object Property	hasPromo
RENAME		
Attribute	Data Property	OldName:Customer_c_comment ,
Attribute	Data Property	OldName:Part_p_category,
DELETION		
Attribute	Data Property	Customer_c_mktsegment
Attribute	Data Property	Part_p_container

E. Change Propagation

To apply the changes over the DWO, we use different algorithms depending on the type of change. For applying addition change to DWO, if the concept type ci is a class and it has 1:n relationship with existing fact class then new class ci is added as dimension class to DWO. If the concept type ci has n:1 relationship with existing dimension class then new class ci is added as fact class to DWO. If the concept type ci has 1:n relationship or 1:1 relationship with existing dimension class then new class ci is added as level to DWO (steps

2-11). If the concept type dpi is a data property, the domain class ci of dpi is obtained. The data property is added to ci (steps 12-14). If the concept type opi is a object property, the domain class ci and range class cj of opi is obtained. The object property is added to DWO (steps 12-14).

If change to be applied to DWO is deletion change deletion algorithm is used. If the concept type ci is a class then the class is deleted from DWO. All data properties dpi and object properties opi of ci are deleted. If the concept dpi is data property to be deleted from DWO then its corresponding domain and range is also deleted. If the concept opi is object property to be deleted from DWO then its corresponding domain and range is also deleted (steps 1-13).

For the change rename change rename algorithm is applied. The corresponding concept type i.e class/ data property / object property is obtained from DWO and it is replaced with new name (steps 1-10). Figure 5 represents the DWO after change set given in Table 2 is propagated. Following are the steps for addition, deletion and rename algorithms:

```
1   ChangeAddition(DWOntology,Concept,Concept _Type)
2   if Concept _Type ==Class then
3       if ci has 1:n relationship with existing fact f then
4           Add dimension ci to DWO
5       else if ci has n:1 relationship with existing dimension d then
6           Add fact ci to DWO
7       else if ci has n:1 or 1:1 relationship with existing dimension d then
8           Add level ci to DWO
9       end if
10      end if
11      end if
12  else if Concept _Type ==DataProperty then
13      ci = Domain(dpi)
14      Add dpi to ci in DWO
15  else if Concept _Type ==ObjectProperty then
16      ci = Domain(opi)
17      cj = Range(opi)
18      Add opi to ci and cj in DWO
19  end if
20  end if
21  end if
```

```
1   ChangeDeletion (DWOntology,Concept,Concept _Type)
2   if Concept _Type ==Class then
3       Delete ci from DWO
4       Delete dpi and opi from DWO
5   if Concept _Type ==DataProperty then
6       Delete dpi from DWO
7       Delete   Domain(dpi) and Range(dpi) from DWO
8   if Concept _Type ==ObjectProperty then
9       Delete opi from DWO
10      Delete   Domain(opi) and Range(opi) from DWO
11  end if
12  end if
13  end if
```

```
1   ChangeRename(DWOntology, OldConcept,NewConcept, Concept _Type)
2   if Concept _Type ==Class then
3       Rename ci in DWO
4   else if Concept _Type ==DataProperty then
```

5 **Rename** dp_i *in* DWO
6 **else if** *Concept _Type ==ObjectProperty* **then**
7 **Rename** op_i *in* DWO
8 **end if**
9 **end if**
10 **end if**

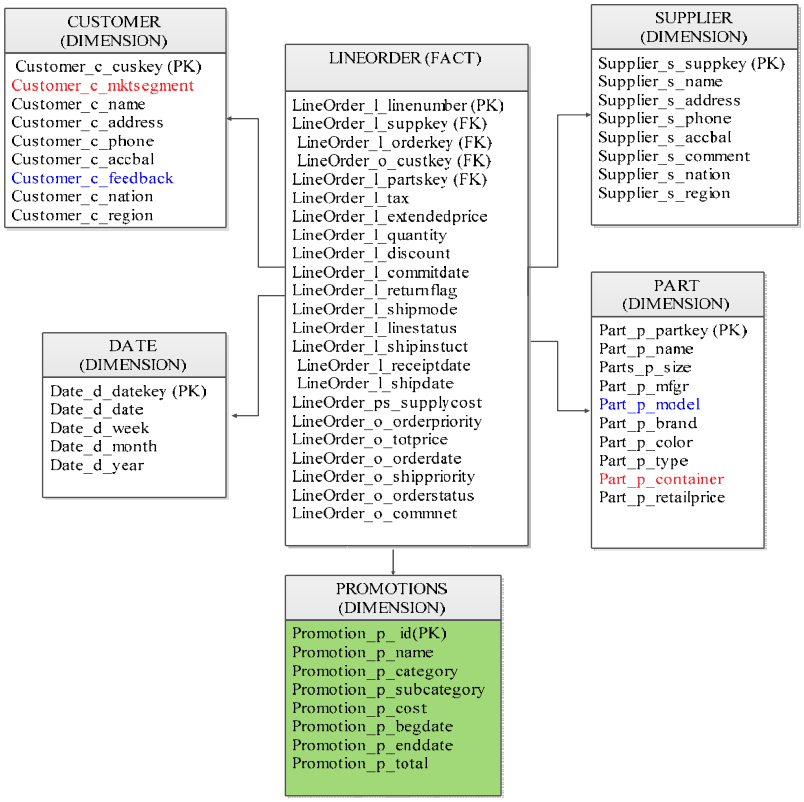

Figure 5. DWO after change propagation

F. Change Validation

In this step the DWO is validated after the required changes are propagated. Using ontology reasoner any inconsistency in the DWO is validated and resolved with help of data warehouse designer suggestion. Finally the data warehouse schema is constructed in the underlying database as a evolved schema or as a new version using DWO.

G. Automatic Adaptation

The important step of our approach is find the dependent entities that are affected and adapt then automatically after the changes are propagated to DWO. The DWO has a mapping with the DSO, due to recent changes the mapping becomes invalid. Hence it is required to make mapping adjustments between them. The queries which worked over the previous DW schema may not work for the new DW schema. Hence it is required to perform a query rewriting. Finally the views maintained for the DW schema also becomes invalid hence it is necessary to update the views. For each of the dependent entities discussed, we need to calculate the cost of updating each.

1) Mapping adjustments

The mapping between DSO and DWO was automatically obtained using wordnet. To update the mappings we use the CHAO log entries for both ontologies by identifying the changed resources in both ontologies. Mappings are then established only for the changed resources and the existing mappings are updated. The previous mappings between these two ontologies are updated at the completion of the Algorithm. In Algorithm first the changed concepts are obtained from log and read into CH for DSO and DWO (steps 1-6). Next the similarities between the changed resources are computed if the type of change is addition in DSO and DWO (steps 6-10). If the change type is deletion the concepts are searched in the mapping file and the corresponding mapping is removed (steps 11-13). For a renamed concept the mapping entity is obtained and the concepts are renamed using information from the log (steps 14-16). Finally the mapping file is updated with new mapping information. The

total no. of entities affected and corrected is computed. Figure 6 represents the diagrammatical representation of mapping updation for customer class.

Algorithm UpdateMapping

Input: Ontologies DSO and DWO for mapping reconciliation, Ontology change information (i.e., CH1and CH2) from ChAO of both ontologies, i.e., CH1 DSO and CH2 DWO

Output: Number of mapping affected and corrected.

```
1       if  CH ∩ CH. DSO.ChAO.NewChange then
2               CH1 = CH.ChAO
3       end if
4       if  CH ∩ CH.DWO.ChAO.NewChange then
5               CH2 = CH.ChAO
6       end if
7       if ChAO.NewChange.ChangeType = ADDITION  then
8               NewMap ←Similarity(CH1,CH2)
9               Execute.update(MappingsFile, NewMap)
10              Count=Count+1
11      else if ChAO.NewChange.ChangeType = DELETION  then
12              Execute.update(MappingsFile, DeleteMap(CH1,CH2))
13              Count=Count+1
14      else if ChAO.NewChange.ChangeType = RENAME  then
15              Execute.update(MappingsFile, RenameMap(CH1,CH2))
16              Count=Count+1
17      else
18      Print("No Change")
19      end if
20      end if
21      end if
```

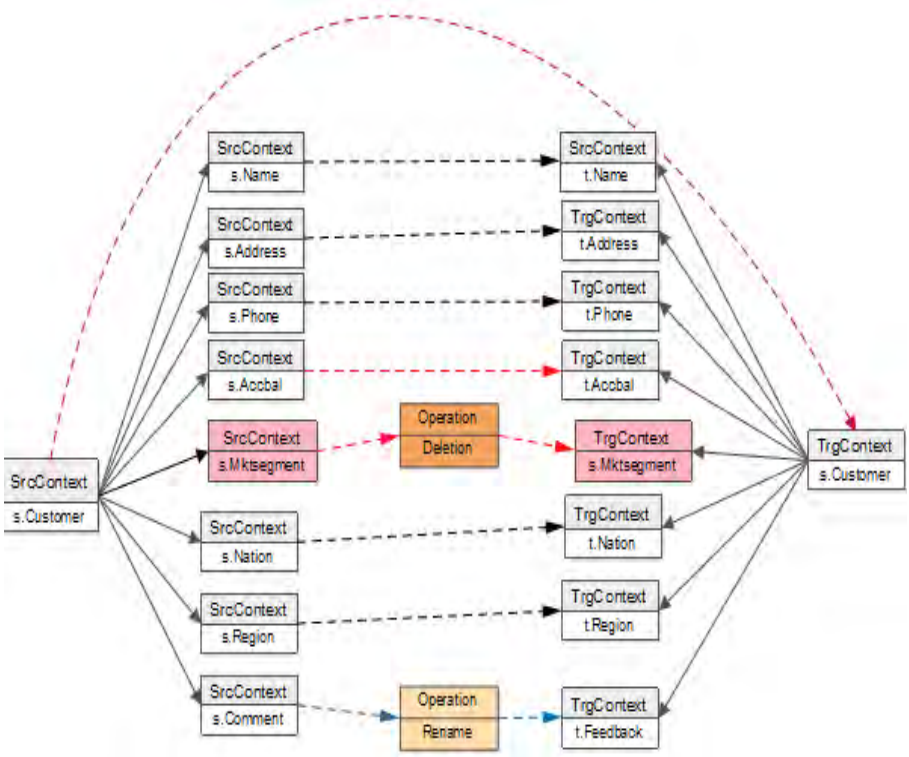

Figure 6. Representation of Mapping Updation

2) Query Rewriting

As the DW schema has evolved the queries imposed previously need to be rewritten to work over the new DW schema. First the type of change performed over DWO is extracted. If change type is addition and if MD data property is added then, the corresponding MD class in DWO is found. Next the existence of MD class name in the FROM clause of each query is checked. With user suggestion the queries with the new data property are rewritten. If change type is addition and if MD class is added then with user suggestion the queries are rewritten with the new MD class. Finally the total no. of queries rewritten is counted (steps 1-11).

If change type is deletion and if MD data property is deleted then, its existence in the SELECT, WHERE, GROUPBY clause of each query is checked. With user suggestion the queries are rewritten with the deleted data property. If change type is deletion and if MD class is deleted then, the existence of MD class name in the FROM clause of each query are checked. With user suggestion the queries are rewritten with the deleted MD class. Finally the total no. of queries affected and rewritten is counted (steps 12-33).

If change type is rename then and if MD data property is renamed then, existence of MD class name in the SELECT, WHERE, GROUPBY clause of each query is checked. With user suggestion the queries are rewritten with the renamed data property. If MD class is renamed then the existence of old MD class name is checked in the FROM clause of each query. With user suggestion the queries are rewritten with the renamed MD class. Finally the total no. of queries affected and rewritten is counted (steps 23-11). In order to compute the no. of queries affected and to rewrite the queries we apply the following steps:

1 *Addition(DWOntology,Concept _Type,QueryWorkload)*
2 **if** *Concept _Type ==Class* **then**
3 **RewriteQuery** in *FROM* clause with c_i
4 Count++
5 **else if** *Concept _Type ==DataProperty* **then**
6 $d_=$ *Domain*(dp_i)
7 **SearchQuery** for d
8 **RewriteQuery** in *SELECT, WHERE, GROUPBY* clause with dp_i
9 Count++
10 **end if**
11 **end if**
12 *Deletion(DWOntology,Concept _Type,QueryWorkload)*
13 **if** *Concept _Type ==Class* **then**
14 **RewriteQuery** in FROM clause with c_i
15 Count++
16 **else if** *Concept _Type ==DataProperty* **then**
17 $d_=$ *Domain*(dp_i)
18 **SearchQuery** for d
19 **RewriteQuery** in SELECT, WHERE, GROUPBY clause with dp_i
20 Count++
21 **end if**
22 **end if**
23 *Rename(DWOntology,Concept _Type,QueryWorkload)*
24 **if** *Concept _Type ==Class* **then**
25 **RewriteQuery** in FROM clause with c_i
26 Count++
27 **else if** *Concept _Type ==DataProperty* **then**
28 $d_=$ *Domain*(dp_i)
29 **SearchQuery** for d
30 **RewriteQuery** in SELECT, WHERE, GROUPBY clause with dp_i
31 Count++
32 **end if**
33 **end if**

3) View Rewriting

Materialized views are used in data warehouse to pre-compute and store aggregated data such as the sum of sales. They are used to vastly improve the query performance. When the underlying source and data warehouse changes its schema structure the materialized views may become out-of-dated. Hence, one important issue is to maintain the materialized views' consistency upon any structural changes. In order to find the number views affected and to rewrite the views we can use the steps given for query rewriting.

IV. IMPACT ANALYSIS

In order to evaluate the efficiency of our approach we examine the cost of manually handling evolution at the physical level with respect to our ontological approach for handling evolution. The manual effort comprises of detection, inspection and where necessary the rewriting of affected activities by an event.

Human effort for manual handling of schema evolution for a change c, over an event e, is expressed as:

$$MC_c^e = AX_c^e + RX_c^e$$

Where,

AX = no. of Query/View/Mapping per change c, affected by event e, that is manually detected.

RX = no. of Query/View/Mapping per change c, which must be manually re-written/updated/mapped to event e.

For a set of evolution operators O, in an activity A, the overall cost of manual adaption to the change c, for an event e is given as:

$$CMA = \sum_{c \in O} \sum_{e \in A} MC_c^e$$

Automatic handling of schema evolution using the proposed ontological approach is quantified as a sum of no. of changes imposed on the DW schema CS and cost of manually discovering and adjusting activities AMC that escape the automation Ad, The latter cost AMC is expressed as:

$$AMC = \sum_{c \in O} \sum_{e \in A_d} MC_c^e$$

The overall cost of automated adoption is given by,

$$CAA = CS + AMC$$

The Table 3 shows the impact of query, ETL mapping and view that is calculated values for manual and automated adoption with the no. of entities affected or corrected in each event for a change.

TABLE 3. CALCULATED VALUES OF CMA AND CAA FOR DIFFERENT CHANGE SETS

Change Set	Impact Analysis	No. of entities affected, corrected in each event for a change						
		AX	RX	MC	CS	AMC	**CMA**	**CAA**
CS1	Query	20	13	33	19	220	**251**	**239**
	ETL Mapping	74	76	150	24	76	**170**	**100**
	View	20	13	33	19	224	**255**	**243**
CS2	Query	20	13	33	18	188	**219**	**206**
	ETL Mapping	75	76	151	23	76	**171**	**99**
	View	20	13	33	18	192	**223**	**210**
CS3	Query	20	13	33	17	194	**225**	**211**
	ETL Mapping	74	81	155	27	81	**175**	**108**
	View	20	13	33	17	202	**233**	**219**
CS4	Query	20	13	33	18	203	**234**	**221**
	ETL Mapping	64	78	142	25	78	**162**	**103**
	View	20	13	33	18	211	**242**	**229**
CS5	Query	20	13	33	15	206	**237**	**221**
	ETL Mapping	76	79	155	26	79	**175**	**105**
	View	20	13	33	15	218	**249**	**223**

AX – no. of Query/View/ETL Mapping per change c, affected by event e, that is manually detected.

RX – no. of Query/View/ETL Mapping per change c, which must be manually re-written/updated/mapped to event e.

MC – human effort for manual handling of schema evolution for a change c, over an event e.

CS – no. of changes imposed on the DW schema.

AMC – cost for automated handling of schema evolution for a change c, over an event e.

CMA – the overall cost of manual adaption to the change c, for an event e.

CAA – the overall cost of manual adaption to the change c, for an event e.

The following Figure 7 shows the impact of evolution on mapping with a comparison cost of manual adaptation and automated adaptation using ontological approach for various change sets.

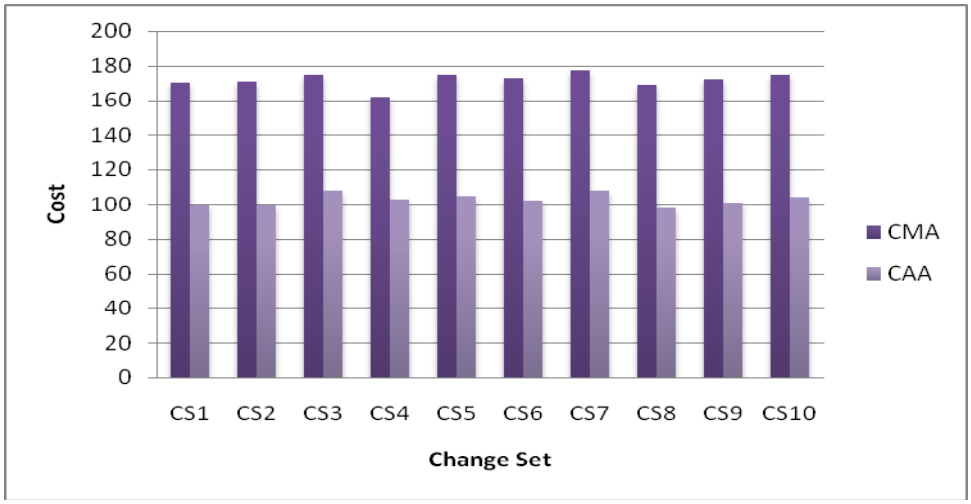

Figure 7. Impact of Evolution on Mapping

The Figure 8 shows the impact of evolution on queries with a comparison cost of manual adaptation and automated adaptation using ontological approach for various change sets.

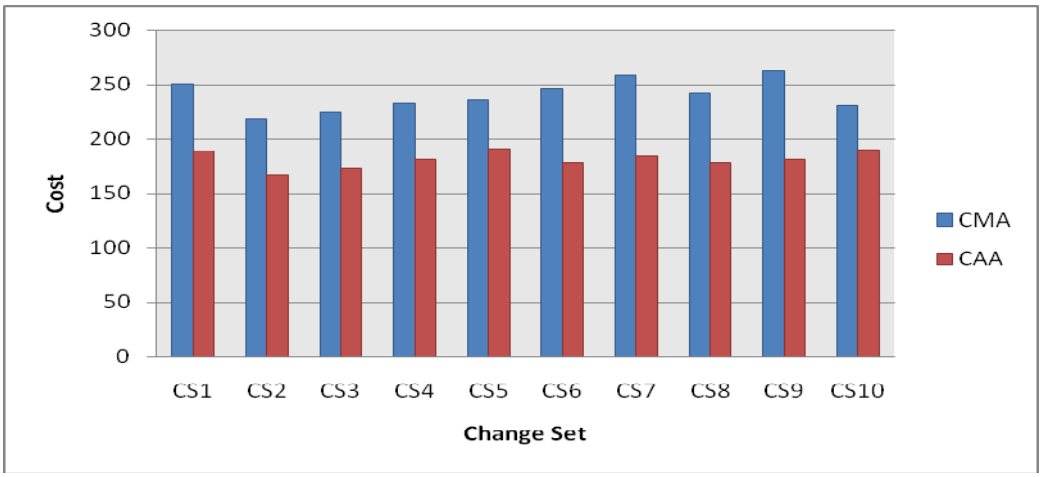

Figure 8. Impact of Evolution on Queries

The following Figure 9 shows the impact of evolution on mapping with a comparison cost of manual adaptation and automated adaptation using ontological approach for various change sets. The total adaptation cost for manual and automated approach is given in Figure 10.

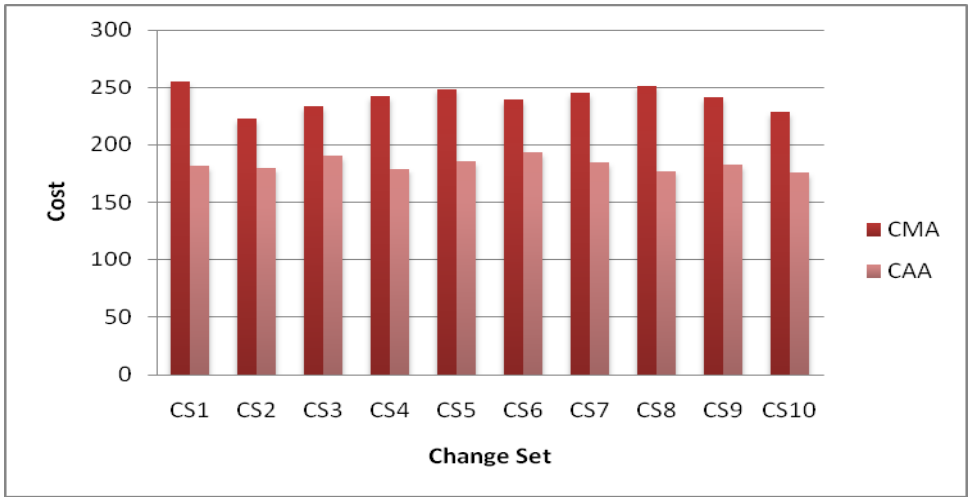

Figure 9 Impact of Evolution on Views

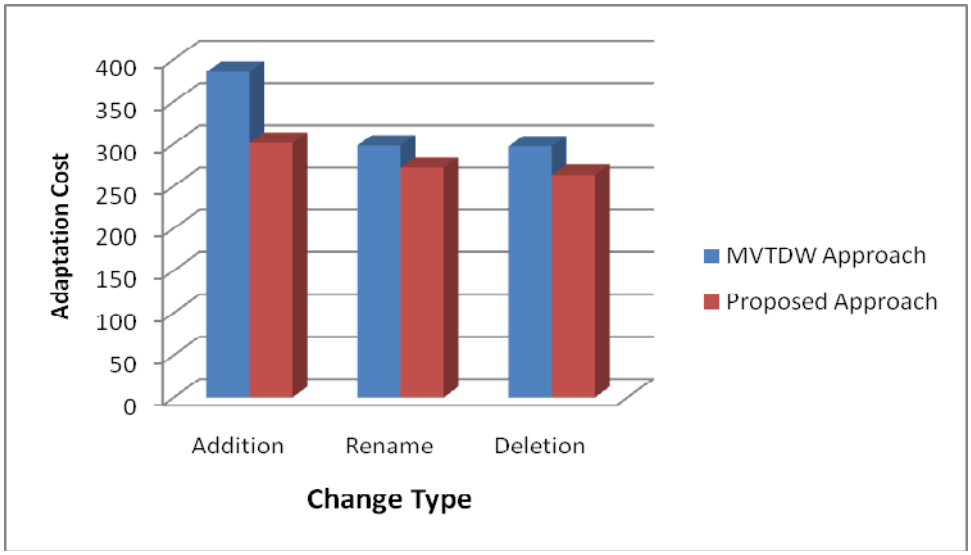

Figure 10 Total Adaptation Cost for Existing and Proposed Approach

From the above figures, it is found that the automated cost (CAA) of adaptation to Mapping, Query and view is comparatively less than that of manual cost of adoption (CMA) to ETL Mapping, Query and view.

The Figure 11 and Figure 12 shows the comparison of no. of entities affected and that are corrected by using the proposed approach by taking the evolution operators along the x-axis and of no. of entities along the y-axis.

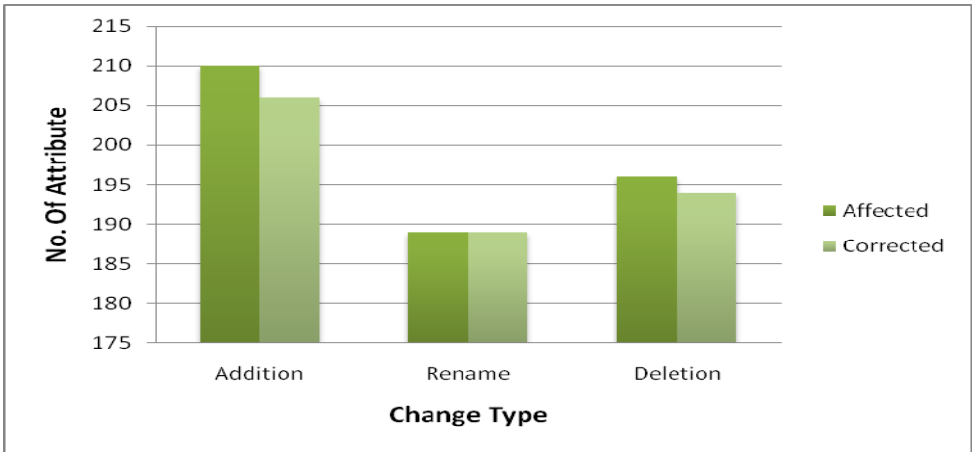

Figure 11 Total No. of Attributes Affected & Corrected for each Evolution Operators

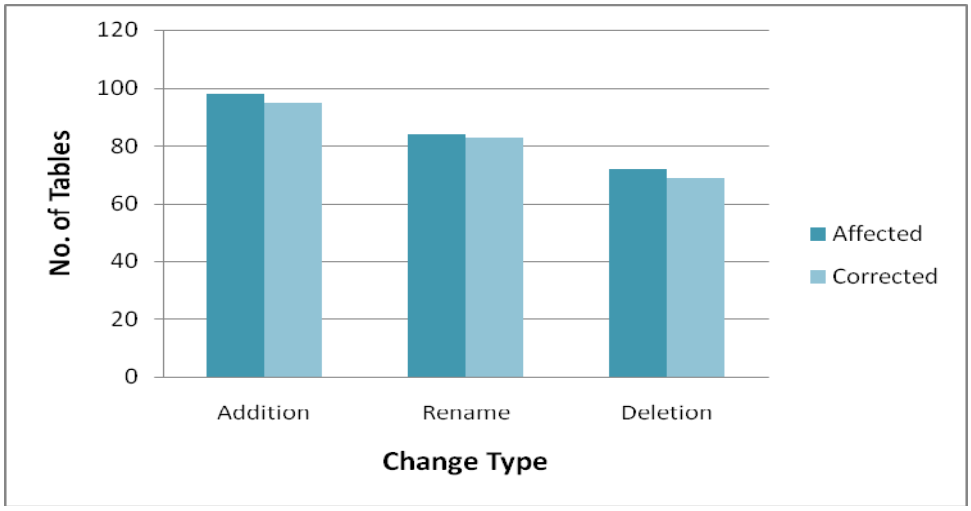

Figure 12 Total No. of Tables Affected & Corrected for each Evolution Operators

CONCLUSION

A data warehouse needs to provide up-to-date information for enabling critical business decisions. When the data source or requirements for a data warehouse evolves the schema of the data warehouse needs to evolve. In this paper we provide a framework for handling changes from data source or requirements to the data warehouse schema at ontological level. The proposed approach provides an insight of the evolution task and analyses its impact on the dependent modules. It also facilitates the automatic adaptation of the dependent modules. The automatic adaptation cost using the proposed ontological approach is lesser than the manual adaptation cost. By analyzing the impact of dependent modules the data warehouse designer can make a decision of carrying the changes over the physical schema of data warehouse.

REFERENCES

[1] Bebel, B., Z. Krolikowski, and R. Wrembel. "Formal approach to modelling a multiversion data warehouse." Palac Kultury i Nauki (2006).
[2] Council, T. P. P. (2008), TPC-H benchmark specification.[Online] www.tpc.org/tpch/ (Accessed 12 November 2013).
[3] Giorgini, Paolo, Stefano Rizzi, and Maddalena Garzetti. "Goal-oriented requirement analysis for data warehouse design." In Proceedings of the 8th ACM international workshop on Data warehousing and OLAP, pp. 47-56. ACM, 2005.
[4] Gruber, Thomas R. "A translation approach to portable ontology specifications."Knowledge acquisition 5, no. 2 (1993): 199-220.
[5] Kimball, Raiph. The data warehouse toolkit. John Wiley & Sons, 2006.
[6] Liang, Mr, Dr Alani, and Prof Shadbolt. "Ontology change management in protégé." (2005).
[7] Lin, Feiyu, and Kurt Sandkuhl. "A survey of exploiting wordnet in ontology matching." In Artificial Intelligence in Theory and Practice II, pp. 341-350. Springer US, 2008.
[8] McGuinness, Deborah L., and Frank Van Harmelen. "OWL web ontology language overview." W3C recommendation 10, no. 2004-03 (2004): 10.

[9] Noy, Natalya F., Monica Crubézy, Ray W. Fergerson, Holger Knublauch, Samson W. Tu, Jennifer Vendetti, and Mark A. Musen. "Protege-2000: an open-source ontology-development and knowledge-acquisition environment." In AMIA Annu Symp Proc, vol. 953, p. 953. 2003.
[10] Oueslati, Wided, and Jalel Akaichi. "A Multiversion Trajectory Data Warehouse to Handle Structure Changes." International Journal of Database Theory & Application 4, no. 2 (2011).
[11] Phipps, Cassandra, and Karen C. Davis. "Automating data warehouse conceptual schema design and evaluation." In DMDW, vol. 2, pp. 2-2. 2002.
[12] Solodovnikova, Darja. "Data Warehouse Evolution Framework." In SYRCoDIS. 2007.
[13] Talwar, Kanika, and Anjana Gosain. "Implementing Schema Evolution in Data Warehouse through Complex Hierarchy Semantics." International Journal of Scientific & Engineering Research 3, no. 7 (2012).
[14] Thakur, Garima, and Anjana Gosain. "DWEVOLVE: a requirement based framework for data warehouse evolution," ACM SIGSOFT Software Engineering Notes 36.6, pp.1-8, 2011.
[15] Wided, and Jalel Akaichi. "A SURVEY ON DATA WAREHOUSE EVOLUTION." International Journal of Database Management Systems 2, no. 4 (2010).
[16] Zekri, M., and A. Abdellatif. "A new approach to update the dimension hierarchies for data warehouse design." In Networked Computing and Advanced Information Management (NCM), 2011 7th International Conference on, pp. 62-66. IEEE, 2011.

AUTHORS PROFILE

M.Thenmozhi is a Research Scholar in the Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry. She is currently working as Assistant Professor in the Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, India. She completed her B.Tech in Computer Science and Engineering from Pondicherry University and M.E in Computer Science and Engineering from Anna University. Her research interest includes Data Warehousing, Data Modeling, Data mining and Ontology.

Dr.K.Vivekanandan is currently a Professor at the Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry. He completed his B.E from Bharathiyar University, M.Tech from Indian Institute of Technology, Bombay and Ph.D from Pondicherry University. He has over 150 papers in national and international journals and conferences. He has coordinated two AICTE sponsored RPS projects on “Developing Product Line Architecture and Components for e-Governance Applications of Indian Context” and “Development of a framework for designing WDM Optical Network. His special research interests include Software Engineering, Object Oriented Systems, Information Security, Web Services and Data Warehousing.