Jurnal Antarabangsa (Teknologi Maklumat) 3(2002): 11-32

Translation of ER Model to Multidimensional Model for Data Warehouse – An Automated Approach

OPIM SALIM SITOMPUL & SHAHRUL AZMAN MOHD NOAH

ABSTRACT

Multidimensional data model has been considered by the database community as a model to support the application of data warehouse. While this model has been the de-facto of successful data warehouse implementation, little has been said on how to carry out the data warehouse conceptual design. This paper, however, presents an approach and a tool for the automatic conceptual data warehouse design. The approach uses an existing ER model described by using an ER specification language, whereby the model is progressively translated and extended to include the dimensional functionality necessary in data warehousing. The output of the translation model is a basic multidimensional construct such as facts, measures, dimensions, and dimension hierarchies. Although there is no universally agreed method on the conceptual design of data warehouse, the approach and tool presented in this paper has at least demonstrated the practical exploitation of ER model as a basis for such a design. The paper concludes with a discussion of areas of future research and development.

Keywords: data warehouse design, multidimensional model, entity relationship model, automated design tool

ABSTRAK

Komuniti pangkalan data telah mengiktiraf model data multidimensi sebagai sebagai model utama dalam pembangunan aplikasi gudang data. Walaubagaimanapun perbincangan dan usulan berhubung dengan pendekatan dalam proses reka bentuk konseptual gudang data masih lagi kurang mendapat perhatian. Melalui kertas kerja ini, satu pendekatan dan perisian dalam mereka bentuk model konseptual gudang data telah diketengahkan. Pendekatan yang dicadangkan ini melibatkan penterjemahan dan pengembangan model perhubungan entiti sedia ada (ditakrifkan melalui bahasa spesifikasi perhubungan entiti) dengan mempertimbangkan fungsian dimensi yang diperlukan dalam penggudangan data. Hasil daripada proses penterjemahan ini ialah satu model asas data multidimensi yang melibatkan fakta, had, *dimensi dan hirarki dimensi. Kertas kerja ini telah menunjukkan kemungkinan untuk menggunakan model perhubungan entiti sedia ada sebagai asas dalam mereka bentuk gudang data, walaupun masih lagi tiada satu metod yang dipersetujui bersama di kalangan penyelidik dan pembangun dalam mereka bentuk gudang data. Kesimpulan dan perluasan penyelidikan dan pembangunan dalam bidang ini diutarakan di akhir kertas kerja ini.*

INTRODUCTION

The concept of data warehouse often viewed as encircling the aspects of application tools, architectures, information service, and communication infrastructures to synthesize information useful for decision-making from distributed heterogeneous operational data sources (Golfarelli *et al.* 1998). The model intended to support the application and implementation of a data warehouse is called the multidimensional data model.

In multidimensional model, data are represented in terms of facts and dimensions where each fact is associated to multiple dimensions. In this manner, facts are the focus of interest by which they are analyzed through the quantifying context stored in measures and the qualifying context determined through dimension levels (Hüsemann *et al.* 2000). Categorizing data along dimensions is a mean to organize them into hierarchical levels so that data can be viewed from their finer to coarser granularities (Agrawal *et al.* 1997).

While it has universally agreed that the implementation of data warehouse rest on the multidimensional model, little agreement has been said on how to carry out its conceptual design. The most popular opinion would be of using an existing ER model whereby the model is progressively translated and extended to include the dimensional functionality necessary in data warehousing (Golfarelli *et al.* 1998, Tryfona *et al.* 1998, Hüsemann *et al.* 2000, Moody & Kortink 2000, Phipps & Davis 2002).

In this paper, we will discuss an approach and a tool for the automatic translation of the ER model into a multidimensional model. First, we proposed an ER specification language meant for describing the properties and semantic structure of an ER model. Output of the translation process is a basic multidimensional construct such as facts, measures, dimensions, and dimension hierarchies. The basic constructs generated could be represented in different graphical multidimensional models such as the Dimensional Fact (DF) model (Golfarelli *et al.* 1998) and the Multidimensional ER (ME/R) model (Hahn *et al.* 2000).

The rest of the paper is organized as follows. In Section 2 we will overview some related works in the area of conceptual data warehouse design. Section 3 will describe the specification of the ER language including the entity class structure and syntax diagrams of the entity class members. In

Section 4 we will describe the methodology for creating the multidimensional model, and presents the model in form of the DFM and the ME/R model. Conclusion and current research will be discussed in Section 5.

RELATED RESEARCH

Research works in the area of conceptual data warehouse design has been initiated since 1998 with the works by Golfarelli *et al.* (1998) and Tryfona *et al.* (1998). Subsequent works that directly relate to conceptual data warehouse design are presented in Hüsemann *et al.* (2000), Moody & Kortink (2000), and Phipps & Davis (2002). However, from a framework of automated conceptual data warehouse design, we can only notice two works that provide concise methodology and algorithms, namely the work by Phipps & Davis (2002) and Golfarelli *et al.* (1998).

Phipps and Davis (2002) proposed a five-step algorithm for translating an ER schema (represented as table data structures) into a set of ME/R schema in tabular form, consisting of: finding entities with numeric fields and creating fact nodes from each entity found; creating numeric attributes of each fact node; creating date and time levels (dimensions) from the date/time type field of each fact node; creating dimension for the remaining attributes; and examining relationship among entities to add dimension hierarchies. The algorithm uses numeric fields and relationships between entities as the basis to create the ME/R schemas. Such a concept can be used with most semantic models including the ER model.

Golfarelli *et al.* (1998) derived their graphical conceptual model for a data warehouse from the ER schema based on the Dimensional Fact (DF) model. In their work, they proposed the following steps to build the DF model: defining facts; building attribute tree; pruning and grafting the tree; defining dimensions; defining fact attributes; and defining hierarchies. Even though the algorithm for the process of translating the ER scheme into the conceptual model was not as complete and automatic as those of Phipps & Davis (2002), they do provide some algorithms as the basis of the translation, such as the algorithm to build attribute trees and algorithm to grafting unwanted vertices.

The rest of the research works on conceptual data warehouse design mentioned previously do not provide any algorithm for the translation of ER model into conceptual model. Hüsemann *et al.* (2000) provided a systematic approach to derive a conceptual schema from the global operational ER schema to design data warehouse conceptual model. However, they emphasize their work on obtaining data warehouse schema that is in generalized multidimensional normal form (GMNF) and suggested three sequential phases of conceptual data warehouse design process, namely context definition of measures, dimensional hierarchy design, and definition of summarisability constraints. Moody & Kortink (2000) derived their dimensional model from typical ER data model used by operational (OLTP) systems. They divided their method into four main steps: classify entities, identifies hierarchies, produce dimensional models, and evaluation and refinement. As the output of the method, they proposed various alternatives to the dimensional model such as flat schema, terraced schema, star schema (constellation schema and galaxy schemas), snowflake schema, and star cluster schema. Tryfona *et al.* (1998) presented a set of user modelling requirements to build a conceptual model named starER that combines the star structure and the ER model. The input for the modelling requirements is an existing ER schema for a certain enterprise. The modelling concepts have five stages: representing facts and their properties; connecting temporal dimension to facts; representing objects, capturing their properties and the associations among objects; recording the associations between objects and facts; and distinguish dimensions and categorize them into hierarchies.

Table 1 provides a summary of the proposed methodologies for the conceptual design of data warehouse. As can be seen, all the methodologies reviewed use the ER model as the basis for the conceptual design of a data warehouse but differ in terms of the multidimensional models generated as the output. Therefore, our decision of using an ER model in this research is parallel with the current approaches and methodologies.

The ER Specification Language

In order for a tool or system to 'understand' the properties and semantic content of an ER model prior to processing, we have proposed an ER specification language in the form of object-oriented class constructs. Three main constructs of our ER specification language are entity, attribute and relationship.

The entity class construct contains the specification lists of an entity participated in the ER diagram. To make the model simple, each entity description only keeps basic properties of an entity such as entity name, attributes, identifier, and relationship where the entity participated. In order to support the enhance properties of the ER model, we also include other properties such as subclass and aggregation abstraction concepts. In general, the structure of the entity class construct can be described as follows.

The above class construct is depicted in the form of syntax diagram as shown in Figure 1.

As can be seen from Figure 1, the class construct starts with a keyword **CLASS** followed by the class name, five class components, and ends with the keyword **END-CLASS**; a new line (*CrLf*) character separates each component.

TABLE 1. Summary of proposed methodologies

Source	Methodology	Translation Algorithm	Output
Phipps & Davis (2002)	• Create fact nodes • Create numeric attributes • Create date/time level • Create other level • Add hierarchical level from relationship	yes	ME/R
Hüsemann *et al.* (2000)	• Define measures context • Design dimensional hierarchy • Define summarizability constraints	no	MNF
Moody & Kortink (2000)	• Classify entities • Identifies hierarchies • Produce dimensional models • Evaluation and refinement	no	Multidimensional Schemas
Tryfona *et al.* (1998)	• Represents facts and properties • Connect temporal dimensions to facts • Associates objects and properties • Record association • Categorize dimensions into hierarchy	no	StarER
Golfarelli *el al.* (1998)	For each defined fact: • Build attribute tree • Pruning and grafting the tree • Define dimension • Define fact attributes • Define hierarchy	yes	DFM

```
CLASS class-name
ATTRIBUTE List of attributes
IDENTIFIER List of identifiers
SUBCLASS List of subclasses
AGGREGATION List of object aggregation
RELATIONSHIP List of relationship, which consists of:
        Rel-Name: Name of the relationship
    Part-Obj: Participating objects in the relationship
    Rel-List: List of relationship attributes
    F-Constr: First participation constraint on the object side.
    S-Constr: Second participation constraint on the Part-Object side.
END-CLASS
```

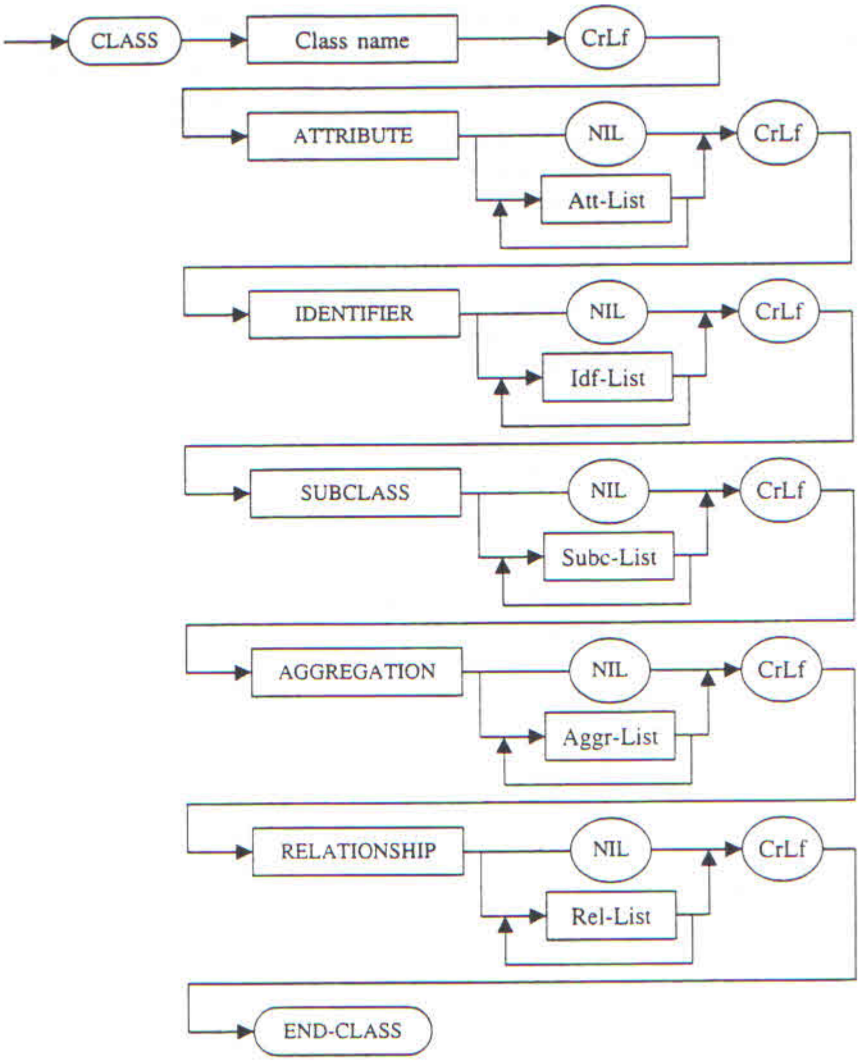

FIGURE 1. Syntax diagram of the Class construct

The class component is composed of the keyword of the component, followed by either *NIL* or list of the sub-components.

The syntax diagram describing the **Att-List** for the **ATTRIBUTE** class component is depicted in Figure 2.

The attribute list (**Att-List**) must be enclosed by parenthesis; either contains at least one composite attribute, or one simple attribute. Composite attribute is preceded by a composite name (*Comp-Name*) and followed by one or more attribute specification (*Att-Spec*) enclosed in parenthesis.

Att-List :

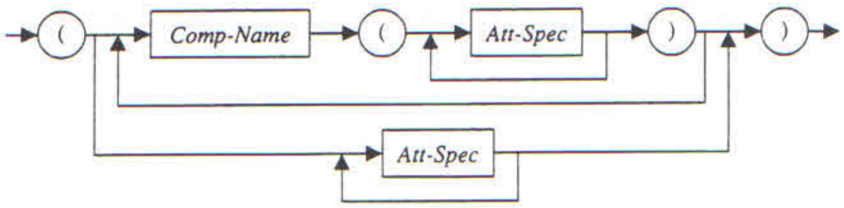

FIGURE 2. Syntax diagram for the attribute list

Likewise, the simple attribute is just a list of one or more attribute specification. The *Att-Spec* sub-component is a list of attribute name, a colon, and a type, enclosed in parenthesis (Figure 3).

The *Class-Name*, *Comp-Name*, and *Att-Name* illustrated in Figure 1, 2, and 3 respectively are of similar syntax, requiring that every name element in the language model should be double-quoted alphanumeric characters preceded by an alphabet (Figure 4).

The syntax diagram of the alphabet (*Alpha*) and digit (*Digit*) in Figure 4 is shown in Figure 5(a) and (b).

The *Type-Name* sub-component of the *Att-Spec* from Figure 3 is a list of allowed type for an attribute. In this case, allowable types are String, Date, Time, Integer, and Float. However, the data types are not only limited to these types, as we can add more data types to the language by modifying the syntax diagram and corresponding program codes. The syntax diagram for the *Type-Name* construct can be seen in Figure 6.

Att-Spec :

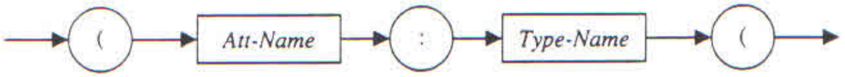

FIGURE 3: Syntax diagram for the attribute specification

Class-Name, Comp-Name, or Att-Name :

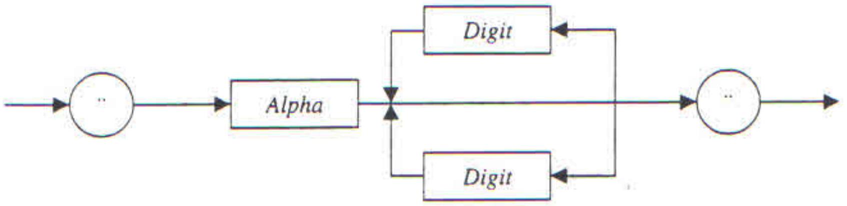

FIGURE 4. Syntax diagram for class name, component name, and attribute name

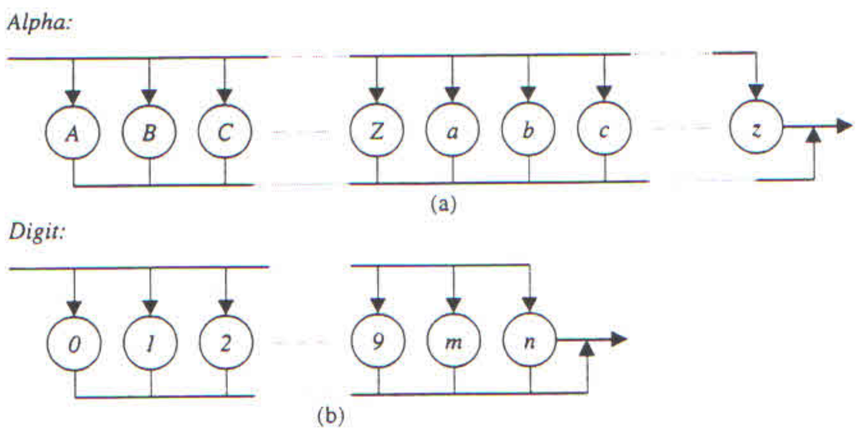

FIGURE 5. Syntax diagram for alphanumeric characters

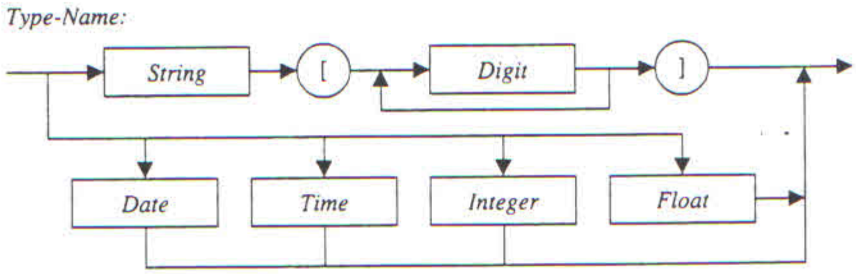

FIGURE 6: Syntax diagram showing the available data types

Using this syntax diagram, we can specify the attributes of an entity either they have simple attributes, composite attributes, or both. The following example illustrates how the attributes of a "PERSON" entity from a university database (Elmasri & Navathe 2000) is constructed using syntax diagram for the attribute list:

```
ATTRIBUTE ("Name" (("Fname": String[15]) ("MInit": String[3]) ("Lname": String[20]))
          ("Ssn": String[12]) ("Bdate": Date) ("Sex": String[1])
          "Address" (("No": String[4]) ("Street": String[20]) ("AptNo": String[4])
                     ("City": String[15]) ("State": String[2]) ("Zip": String[5])))
```

The **Idf-List**, **Subc-List**, and **Agg-List** for the **IDENTIFIER**, **SUBCLASS**, and **AGGREGATION** class components have the same syntax as shown in Figure 7, in which the construct of the *Att-Name* sub-component is the same as one depicted in Figure 4.

Idf-List, Subc-List or Aggr-List:

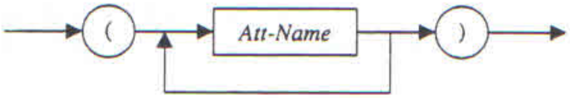

FIGURE 7: Syntax diagram for identifier list, subclass list, or aggregation list

For example, the syntax of IDENTIFIER, SUBCLASS, and AGGREGATION components of a "PERSON" entity can be specified as the following:

```
IDENTIFIER ("Ssn")
SUBCLASS ("FACULTY" "STUDENT")
AGGREGATION NIL
```

The last component that is the RELATIONSHIP class component has five sub-components as depicted in Figure 8. In this case, if an entity has more than one relationship, each relationship should be listed in one line, separated with a continuation character "\".

Rel-List:

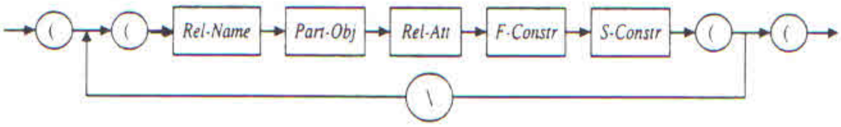

FIGURE 8. Syntax diagram for the relationship list

The *Rel-Name* and *Part-Obj* sub-components of the relationship list has the same syntax as one in Figure 4, while the *Rel-Att* sub-component is described in Figure 9 with the same *Att-Spec* sub-component as one depicted in Figure 3.

Rel-Att:

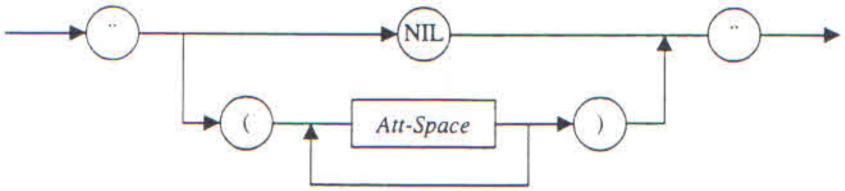

FIGURE 9. Syntax diagram for the relationship attribute

As indicated in Figure 9, whether the RELATIONSHIP component does not have any attribute (specified by *NIL*) or has a list of attributes, it should be delimited by double quotes.

The last two sub-components of the relationship list are the *F-Constr* and *S-Constr* stand for first constraint and second constraint, respectively. These two components have the same syntax and are described in Figure 10.

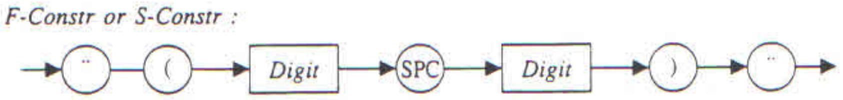

FIGURE 10. Syntax diagram for first and second constraint of the relationship list

The *F-Constr* and *S-Constr* have a straight syntax showing no optional paths, in which the *Digit* sub-component has the same syntax as one in Figure 5(b), while the SPC sub-component represents a white space character.

For example, an entity "FACULTY" with a list of five relationships can be specified as follows:

```
RELATIONSHIP (("chairs" "DEPARTMENT" "NIL" "(1 1)" "(1 1)")\
              ("belongs" "DEPARTMENT" "(("Count": Integer))" "(1 n)" "(1 m)")\
              ("pi" "GRANT" "NIL" "(1 n)" "(1 1)")\
              ("committee" "GRAD_STUDENT" "(("NumberOaf": Integer))" "(1 n)" "(1 m)")\
              ("advisor" "GRAD_STUDENT" "NIL" "(1 n)" "(1 1)"))
```

Deriving Multidimensional Model

In general, our proposed translation of the ER model to multidimensional tool consists of three main modules, namely the ER language module, the object class module, and the multidimensional module as shown in the Figure 11.

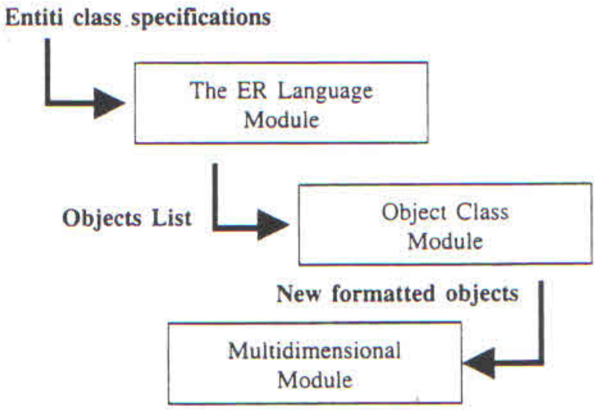

FIGURE 11. Main modules for EER-Multidimensional translation

The translation approach consists of three-step process conforming to the three aforementioned modules as described in Figure 12 below.

- Specifications of input
    - o Specify input in the ER language construct
    - o Read the input file containing the entity classes and generate objects list.
- Processing Objects For each object in the objects list:
    - o Specify direct superclass
    - o Specify indirect objects
    - o Reformat relationship
    - o Get new objects from numerical relationship attributes
    - o Get inherited attributes and identifiers
    - o Reformat and classify attributes
- Creating multidimensional data model
    - o Select fact from the objects list
    - o Specify measures
    - o Get dimensions
    - o Add dimension hierarchies

FIGURE 12. Three-step translation process

THE ER LANGUAGE MODULE

The initial input for the translation process is an ER schema of an operational database specified using the ER language syntax (discussed in Section 3) and saved into an ASCII text file. The ER language module reads this file and parses each line to obtain the name of each entity and all its properties. The algorithm used is shown in Figure 13, where _, *eof*, and *eoln* respectively denote an empty string, end of file, and end of line.

```
Algorithm ReadInputFile(E )
Input: Data set E of entity class file
    whole-line _ _
    while not eof (E )
        a-line _ read-line(E )
        if a-line ∉ _
            if eoln(a-line) is in {"\"}
                concatenate(whole-line, substring(a-line, 0, (length(a-line) - 1)))
            else
                concatenate(whole-line, a-line)
                a-name _ left-string (whole-line)
                r-str _ right-string (whole-line)
                if a-name ∉ _
                    if a-name = "CLASS"
                        instantiate-object (r-str)
                    else
                        insert-obj-property (r-str, O)
        whole-line _ _
Output: List of objects O
```

FIGURE 13. Algorithm for reading input file of the entity class

Each entity obtained is instantiated whereby all the properties attached to the entity are added to the object instance, and finally all object instances are inserted into an objects list structure. The structure of the entity object is represented in the following Common Lisp Object System (CLOS) constructs (Keene 1989).

```
(defclass EntityObject ()
    ((name :accessor obj-name :initform nil :initarg :name)
    (attribute :accessor obj-attribute :initform nil :initarg :attribute)
    (identifier :accessor obj-identifier :initform nil :initarg :identifier)
    (subclass :accessor obj-subclass :initform nil :initarg :subclass)
    (superclass :accessor obj-superclass :initform nil :initarg :superclass)
    (aggregation :accessor obj-aggregation :initform nil :initarg :aggregation)
    (relationship :accessor obj-relationship :initform nil :initarg :relationship))
    (:documentation
    "An object represents an entity. It holds the name, attributes, and its association
    with other entities."))
```

THE OBJECT CLASS MODULE

The main purpose of the object class module is to process the properties of each object in the objects list to facilitate an intermediate input for the translation process. Each object property such as attribute and relationship is converted into an association list (in form of list of sub lists), by which the first element of each sub list is used as a key for recovering the entire sub lists (Winston & Horn 1989). In addition, the process also includes derivation of other object properties such as superclass, indirect subclass, indirect superclass, inherited attribute, and inherited identifier. The algorithm used in this stage is shown in Figure 14.

A. SPECIFYING DIRECT SUPERCLASS

Direct superclass is obtained based on the existence of the SUBCLASS component of an object instance. If an object has subclasses, then this object becomes direct superclass of those subclasses. This activity is done recursively by re-examining the existence of SUBCLASS component of each subclass obtained earlier.

B. SPECIFYING INDIRECT OBJECTS

There are two kinds of indirect objects, i.e. indirect subclass and indirect superclass. To specify the indirect subclass of an object, first we have to check if the object has a subclass. If the subclass exists, then we recursively check the subclass of this subclass. Likewise, to specify indirect superclass, the superclass of the object is initially checked. If the superclass exists, then the superclass of this superclass is checked recursively. The indirect objects are a collection of subclass and superclass obtained in this manner.

```
Algorithm ProcessObject(O)

Input: Data Set $O = \{o_i \mid i = 1, ..., n\}$

    for each $o_i \in O$
        SpecifyObjectSuperclass(o)
        SpecifyIndirectObject($o_i$)
        ReformatRelationship(o)

let N $\leftarrow \emptyset$, where N $= \{n_j \mid j = 0, ..., p;\ p \leq n\}$ of new objects
let m $= \{m_k \mid k = 0, ..., q;\ q \leq n\}$ list of new object name from relationship

for each $o_i \in O$
    m $\leftarrow$ GetObjectFromRelationship(o)
    if m $\notin \emptyset$
        for each $m_k \in$ m
            new-att $\leftarrow$ GetNewAttribute($m_k$, $o_i$)
            obj-id $\leftarrow$ GetNewIdentifier(part-obj)
            part-obj $\leftarrow$ GetParticipatingObject($m_k$, $o_i$)
            new-constr $\leftarrow$ SetRelConstraint($m_k$, $o_i$)
            new-rel $\leftarrow$ SetNewRelationship(new-constr, part-obj)
            InstantiateNewObject ($m_k$); Add $m_k$ to N.
            ModifyOldObject(o)

    O $\leftarrow$ O $\cup$ N

    for each $o_i \in O$
        AddInheritedAttribute($o_i$)
    AddInheritedIdentifier($o_i$)
        ReformatAttribute($o_i$)
    ClassifyAttribute($o_i$)

Output: Modified set of objects list O
```

FIGURE 14. Algorithm for processing objects in the objects list

C. REFORMATTING RELATIONSHIP

The relationship properties are obtained from the relationship of each object in the objects list. The purpose of reformatting the relationship is to convert all its subcomponents into association lists. The first part of the association list is the name of the subcomponent and the second part is its value.

D. EXTRACTING NEW OBJECTS FROM NUMERICAL RELATIONSHIP ATTRIBUTES

A new object could be added to the objects list originating from a relationship, namely a relationship that has numeric attributes. Such relationship is converted into an object by giving the name of the relationship as the name of the new object. As a result of this conversion, the new object and the corresponding objects participated in the relationship need further processing. For the new object, its attributes are derived from the relationship attributes and its identifiers are the combination of the identifiers of the participating objects. If the relationship is a many-to-many relationship, the relationship is split into two many-to-one relationships with the new object in the many side. Each participating objects get a new relationship with the new object and the new relationships are given a default name "of ".

E. EXTRACTING INHERITED ATTRIBUTES AND IDENTIFIERS

If an object has one or more superclass, then its attributes are the union of its attributes and the attributes of its superclass. The attribute of the object can be obtained using the *obj-attribute* accessor function of the *EntityObject* class, whereas the inherited attributes are acquired from the attributes of its superclass. The identifiers of an object are obtained in the same manner.

F. REFORMATTING AND CLASSIFYING ATTRIBUTES

There are two activities performed in this step: reformatting attributes and classifying the attributes according to their types. First, attributes of an object are reformatted in the form of association lists. The first part of the association list is the name of the attribute and the second part is the domain attribute type. If the attribute is a composite attribute, then the composite attributes are reformatted by associating a word "Composite" in front of the composition name to indicate the beginning of the composite attributes and an association list ("Composite" . "End") is inserted to indicate the end of the composition. Next, the new formatted attributes are classified into three categories: *numeric* attributes, *temporal* attributes, and *other* attributes. *Numeric* attributes are from a domain of numbers such as integer or float; *temporal* attributes are from the date and time domain; and *other* attribute is commonly attributes from the string domain.

The object class module delivers an intermediate output containing all objects obtained from the entity class file with new formatted properties. Given below is a portion of the intermediate output resulted from this module.

```
Object name: PERSON
  Attribute(s):
    Numeric Attribute(s): NIL
    Date Atribute(s):
      (Bdate . Date)
    Other Attribute(s):
      (Composite . Name)
        (Fname . String[15])
        (MInit . String[3])
        (Lname . String[20])
      (Ssn . String[12])
      (Sex . String[1])
      (Composite . Address)
        (No . String[4])
        (Street . String[20])
        (AptNo . String[4])
        (City . String[15])
        (State . String[2])
        (Zip . String[5])
Identifier(s): (("Ssn"))
Direct Subclass(es): ("FACULTY" "STUDENT")
Indirect Subclass(es):
  ("GRAD_STUDENT")
  ("MASTERS_STUDENT")
  ("PHD_STUDENT")
Direct Superclass(es): NIL
Indirect Superclass(es): NIL
Aggregation(s): NIL
Relationship(s): (NIL)
```

The Multidimensional Module

The multidimensional module is a module that produces the basic constructs of the multidimensional model such as facts, measures, dimensions, and dimension hierarchies. Input to this module is a list of objects with formatted properties resulted from the object module. The methodology for creating multidimensional model consists of four steps: deriving facts from the objects list, specifying measures, getting dimensions, and adding dimension hierarchies (Figure 15).

```
Algorithm CreateDimensionalModel(O)

Input: A list of objects O = {$o_i$ | $i$ = 1, ..., $n$}

    let $L_f$ _ Ø, where $L_f$ = { $l_j$ | $j$ = 1, ..., $m$;  $m \le n$} of facts assoc. list
    if O ∉ NIL
        for each $o_i \in$ O
            ct-num _ CountNumericalAttributes ($o_i$)
            if (ct-num > 0)
        c-node _  obj-name ($o_i$)
                cons (c-node, ct-num)
                append c-node into $L_f$
        sort $L_f$  descending on ct-num

    let F _ Ø, where F = { $f_k$ | $k$ = 1, ..., $m$;  $m \le n$} of fact objects

    for each $l_j \in L_f$
        $l_j$' _ first part of association list $l_j$
        $f_k$ _ instantiate-fact-object ($l_j$')
        num-att _ SpecifyMeasures ($f_k$)
        obj-num-att ($f_k$) _ num-att
        obj-fact-dim ($f_k$) _ GetDimensions (obj-name $f_k$)
        obj-hierarchy ($f_k$) _ AddDimHierarchy (obj-name $f_k$)
        Append $f_k$ into F  Output: F
```

FIGURE 15. Algorithm for creating multidimensional data model

A. DERIVING FACTS

Facts are focus of analysis in multidimensional model and characterized by properties, which are usually numerical data, and can be summarized (or aggregated) to extract further information (Tryfona *et al.* 1998). Referring to this, facts for the multidimensional model are selected based on whether an object has numeric attributes or not. Since each object has a numerical attribute classification for its attribute, the selection of fact becomes trivial. Each fact obtained in this step is inserted into a list of fact-objects class called *FactObject*, which is a subclass of the *EntityObject* class with the following structure.

```
(defclass FactObject (EntityObject)
   ((measure :accessor obj-fact-measure :initform NIL :initarg :measure)
   (dimension :accessor obj-fact-dim :initform NIL :initarg :dimension)
   (dim-hierarchy :accessor obj-dim-hierarchy :initform NIL :initarg :dim-hierarchy))
   (:documentation
   "A fact object is a candidate for a fact node. It keeps the numeric attributes
(measures),
   dimensions, and dimension hierarchies."))
```

In addition, as suggested by Phipps and Davis (2002) the count of numeric attributes an object has could be regarded also as a feature in determining which object would most likely be chosen as a fact; so each chosen fact inserted into the list of facts is sorted in descending order based on the count of its numerical attributes. The rationale for this sorting is that the greater the count of the numerical attributes, the most likely that the object is chosen as a fact.

B. SPECIFYING MEASURES

Measures are the summary properties of a fact and could be acquired from the *numeric* attributes of the *EntityObject* class. Since the fact object is a subclass of the entity object class, the attributes of the object could be accessed using the *obj-attribute* accessor function of the *EntityObject* class by which a list of attributes (including inherited attributes) in forms of association list is obtained. Next, the type of each attribute is checked from the second part of the association list to see whether it is a numeric attribute. Finally, each numeric attribute obtained is added to the measure of the fact object using the *obj-fact-measure* accessor function of the *FactObject* class.

C. EXTRACTING DIMENSIONS

Dimensions are chosen among the attributes of an object, which has been classified into three categories as previously described. Among the three categories, only *temporal* and *other* categories that will be used for dimensions, because *numeric* category has been used as measures of the fact. *Temporal* dimension is considered a very important dimension in data warehouse as this dimension could provide a view of the evolution of data over a specified time range. For this reason, it is necessary that each fact should have the time/date as one of its dimensions. *Other* dimension is also crucial for the design of data warehouse because they represent different point of view of the fact being considered. However, it is not all of the *other* attributes could be added as dimensions, some of these attributes need to be pruned and grafted because aggregating them into different granularities may not give any significant contribution to the view of multidimensional data (Golfarelli *et al.* 1998). Another attribute that may not give significant contribution is what so called non-dimensional attributes and most likely those attributes are having a domain attribute of type string.

D. ADDING DIMENSION HIERARCHIES

In addition to the dimensions obtained in previous step, additional dimensions could be added from relationships, superclass, and aggregations. These dimensions will contribute to the creation of dimension hierarchies of the multidimensional model. Objects are added to the dimension hierarchies based on the existence of many-to-one relationships amongst objects. The first level of the hierarchy is created whenever a many-to-one relationship exists between a fact object and another object (where the many side is on the fact object). If the object has many-to-one relationship with another object, then the other object is also added to the existing hierarchy. This step is done recursively until there is no other many-to-one relationship exists for the newly added object. Dimension hierarchy could also be added from superclasses of a fact object in which the direct superclass of the fact object is added to the first level of the hierarchy and inserting each subsequent indirect superclass in different hierarchy level, correspondingly. In case where a fact object has some aggregation objects, a new dimensional hierarchy could also be created. In this case, each component object of the aggregation is placed at the same hierarchy level.

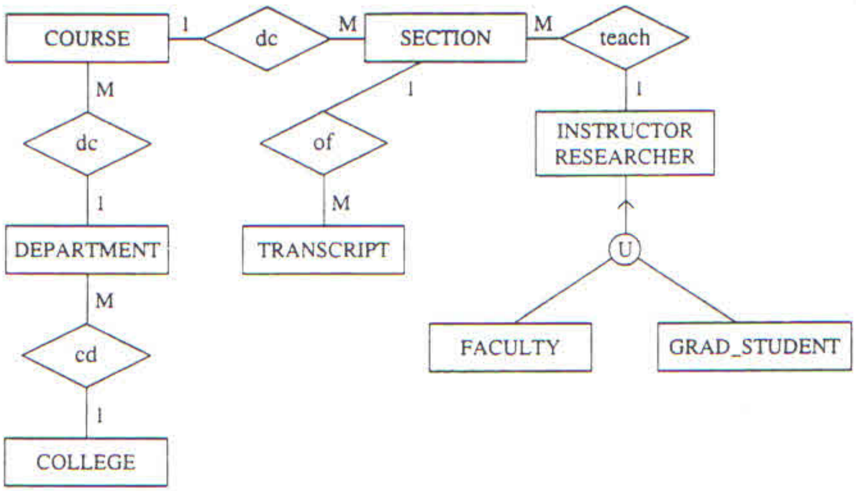

FIGURE 16. Portion of ER diagram for a "SECTION" object

To assist understanding of the previously described process of the multidimensional module, consider an object called "SECTION" emanated from a university data base of Elmasri and Navathe (2000), which participates in two many-to-one relationships and one one-to-many relationship as shown below.

```
(Name  .  cs)
    (Participating-obj  .  COURSE)
    (Rel-Attribute  .  NIL)
    (First-constraint  .  (1  1))
    (Second-constraint  .  (1  n))
(Name  .  teach)
    (Participating-obj  .  INSTRUCTOR_RESEARCHER)
    (Rel-Attribute  .  NIL)
    (First-constraint  .  (1  1))
    (Second-constraint  .  (1  n))
(Name  .  of)
    (Participating-obj  .  TRANSCRIPT)
    (Rel-Attribute  .  NIL)
    (First-constraint  .  (1  m))
    (Second-constraint  .  (1  1))
```

A portion of the ER diagram showing this relationship is shown in Figure 16.

From the three relationships, only those with many-to-one relationships will be added to the dimension hierarchy, namely the "COURSE" and the "INSTRUCTOR_RESEARCHER" objects, whereas the "TRANSCRIPT" object is not included. The "COURSE" object also has many-to-one relationship with the "DEPARTMENT" object and this object has a many-to-one relationship with the "COLLEGE" object, so the algorithm recursively includes the two objects into the dimension level. In case of the "INSTRUCTOR_RESEARCHER" object, it does not have further many-to-one relationship but it has two aggregation objects instead, namely "FACULTY" and "GRAD_STUDENT". These two objects are added to the dimension at the same level. Result of this process is two dimension levels for the "SECTION" object.

After completing all four steps in multidimensional module, we have a set of fact scheme for the "SECTION" fact as shown below.

```
Fact  Node:  "SECTION"
    Measure(s):
        (Year  .  Integer)
    Dimensions:
        Temporal  Dimension:
            NIL
        Other  Dimension:
            Sect#
            Qtr
    Hierarchical  Level:
        "INSTRUCTOR_RESEARCHER"
            "FACULTY"
            "GRAD_STUDENT"
        "COURSE"
            "DEPARTMENT"
                "COLLEGE"
```

The fact scheme for the "SECTION" fact can be represented in graphical model such as the ME/R model from Hahn, Sapia & Blaschka (2000) and the DF model from Golfarelli, Maio & Rizzi (1998). The resulting graphical multidimensional model in form of DF model and ME/R diagram can be seen in Figure 17 and Figure 18, respectively.

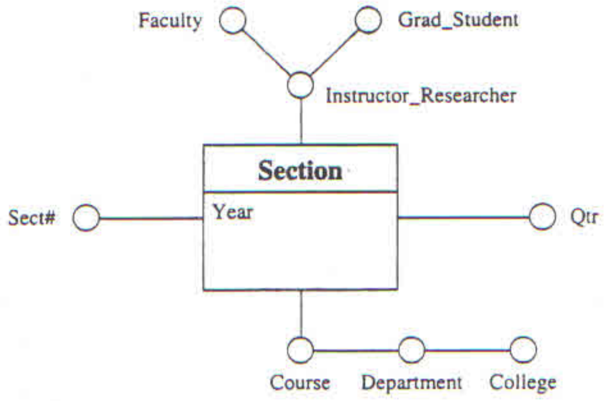

FIGURE 17: Dimensional Fact Model for "SECTION" fact

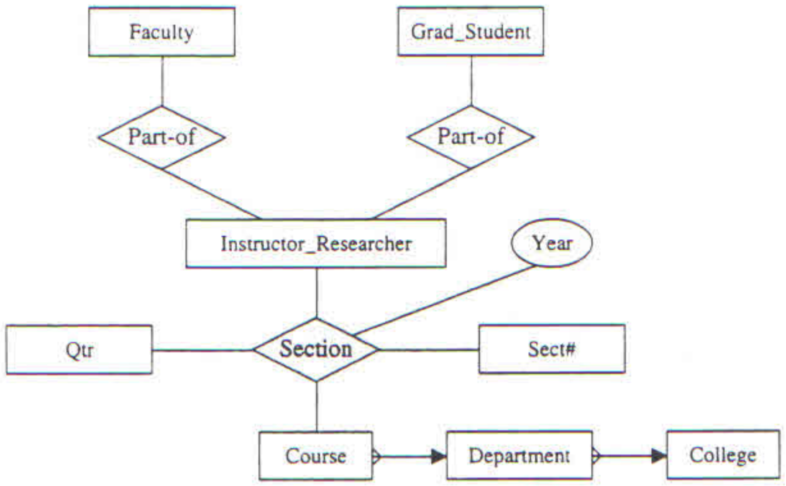

FIGURE 18: ME/R model for the "SECTION" fact

CONCLUSION AND CURRENT RESEARCH WORK

In this paper we have proposed an approach for automating the translation process of an ER model into a multidimensional model. The proposed automated approach is implemented according to three modules: the ER language module, the object class module, and the multidimensional module. As to date, the approach only produced an initial multidimensional data model in the form of basic multidimensional constructs such as facts, measures, dimensions, and dimension hierarchies. Our current research work has been focused towards constructing a module for enhancing and refining the basic multidimensional data model by incorporating refinement elements such as grafting and pruning attributes; inserting additional dimensions such as time dimension; revising measures or facts; and choosing an appropriate fact. Techniques from the field of artificial intelligence (AI) may be a fruitful direction towards implementing the aforementioned elements as has been achieved within the context of database analysis and design (Noah and Williams 1998 & 2000).

REFERENCES

Agrawal, R., Gupta, A. and Sarawagi, S. 1997. Modeling multidimensional databases. *Proceedings of the 13th International Conference On Data Engineering (ICDE'97).* Birmingham, U.K: 232-243.

Chaudhuri, S. and Dayal, U. 1997. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record* **26**(1): 65-74.

Elmasri, R. and Navathe, S. B. 2000. *Fundamentals of Database Systems.* 3rd Edition. Addison-Wesley: Reading, MA.

Golfarelli, M., Maio, D. and Rizzi, S. 1998. Conceptual design of data warehouses from E/R schemes. *Proceedings of the Hawaii International Conference on System Sciences* **VII**. Kona, Hawaii: 334-343.

Hahn, K., Sapia, C. and Blaschka, M. 2000. Automatically generating OLAP schemata from conceptual graphical models. *Proceedings of the 3rd International Workshop on Data Warehousing and OLAP (DOLAP, in connection with CIKM).* Washington D.C.: 9-16.

Hüsemann, B., Lechtenbörger, J. and Vossen G. 2000. Conceptual data warehouse design. *Proceedings of the International Workshop on Design and Management of Data Warehouse (DMDW '2000).* Stockholm, Sweden: 6-1 – 6-11.

Keene, S. E. 1989. *Object-Oriented Programming in Common LISP: A Programmer's Guide to CLOS.* Addison-Wesley: Reading, MA.

Kimbal, R. 1997. A dimensional modeling manifesto: Drawing the line between dimensional modeling and ER modeling techniques. *DBMS Online* **10**(9): 59.

Moody, D. and Kortink, M.A.R. 2000. From enterprise models to dimensional models: A methodology for data warehouse and data mart design. *Proceedings of the International Workshop on Design and Management of Data Warehouses 2000 (DMDW'2000).* Stockholm, Sweden: 5-1 – 5-12.

Noah, S. A. and Williams, M. 1998. An evaluation of two approaches to exploiting real-world knowledge by intelligent database design tools. *Proceedings of the 17th International Conference on Conceptual Modelling, Singapore*. Springer-Verlag, Berlin: 197-210.

Noah, S. A. and Williams, M. 2000. Exploring and validating the contributions of real-world knowledge to the diagnostic performance of automated database design tools. *Proceedings of the Fifteenth IEEE International Conference on Automated Software Engineering (ASE 2000)*. Grenoble, France:177-185.

Phipps, C. and Davis, K. C. 2002. Automating data warehouse conceptual schema design and evaluation. *Proceedings of the 4th International Workshop on Design and Management of Data Warehouses 2002 (DMDW'2002)*. Toronto, Canada: 23-32.

Samtani S., Mohania, M., Kumar, V. and Kambayashi, Y. 1998. Recent advances and research problems in data warehousing. *Proceedings of the International Workshop on Data Warehousing & Data Mining, Mobile Data Access, and New Database Technologies for Collaborative Work Support & Spatio -Temporal Data Management*. Singapore: 81-92.

Sapia, C., Blaschka, M., Höfling, G. and Dinter, B. 1998. Extending the E/R model for the multidimensional paradigm. *Proceedings of the 1st International Workshop on Data Warehousing and Data Mining (DWDM 98)*. New York, N.Y.: 105-116.

Tryfona, N., Busborg, F. and Christiansen, J. G. B. 1999. starER: A conceptual model for data warehouse design. *Proceedings of the ACM 2nd International Workshop on Data Warehousing and OLAP*. New York, N.Y.: 3-8.

Widom, J., 1995. Research Problems in Data Warehousing. *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*. Baltimore, Maryland: 25 – 30.

Winston, P. H. and Horn, B. K. P. 1989. *LISP*. 3rd Edition. Addison-Wesley: Reading, MA.

MAKLUMAT PENGARANG
Opim Salim Sitompul & Shahrul Azman Mohd Noah
Department of Information Science
Faculty of Information Science & Technology
Universiti Kebangsaan Malaysia
43600 Bangi, Selangor, Malaysia
oss19877@ftsm.ukm.my, samn@ftsm.ukm.my