International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 3 (2011) pp. 127-136

Implementation of Graph Semantic Based Multidimensional Data Model: An Object Relational Approach

Anirban Sarkar[1], Sankhayan Choudhury[2], Nabendu Chaki[2] and Swapan Bhattacharya[3]

[1]Department of Computer Applications,
National Institute of Technology, Durgapur,India
sarkar.anirban@gmail.com

[2]Department of Computer Science & Engineering,
University of Calcutta, Kolkata, India
{sankhayan@gmail.com, nabendu@ieee.org}

[3] Department of Computer Science & Engineering,
Jadavpur University, Kolkata, India
bswapan2000@yahoo.co.in

***Abstract*: Data Warehouse (DW) design demands a methodical support for the designer of DW to specify the execution system efficiently from its conceptual level design. In [19], we have proposed a generic model for Data warehouse at conceptual level named Graph Object Oriented Multidimensional Data Model (GOOMD). This paper proposes a systematic approach for object relational implementation of the GOOMD model. The approach is governed by some set of conversion rules to specify the GOOMD model construct in SQL 2003 compatible Object Relational (OR) Schemas. Moreover, the set of OLAP operators defined in GOOMD model have been mapped using OR SQL which will operate on the OR schemas. The concept of GOOMD model also has been implemented in Generic Modeling Environment (GME) and an interpreter has been developed to automate the proposed approach.**

Keywords: Data Warehouse, Multidimensional Databases, OLAP, Conceptual Modeling, Object Orientation, Graph Data Model.

I. Introduction

Complex, online and multidimensional analysis of data is done by fetching just-in-time information from subjective, integrated, consolidated, non–volatile, historical collection of data. Data Warehouse (DW) and On Line Analytical Processing (OLAP) in conjunction with multidimensional database are typically used for such analysis. DW facilitates data navigation, analysis, and business oriented visualization of data using multidimensional cube and OLAP query processing. DW design framework spans in three levels namely, Conceptual, Logical and Physical. Conceptual models with graphical notations are closer to the perception of users about an application domain whereas the logical models concentrate more to the way as a designer perceives an application domain. Data Warehouse design is a highly complex engineering task. It requires a proper methodological support to specify a DW system at conceptual level and moreover the ability to map the system from conceptual to the equivalent logical level. But in context of multidimensional data modeling, there is a semantic gap between advanced conceptual data models and multidimensional implementations of data cubes at logical level [16]. More research scope is there to identify the methodology to preserve all information captured by advanced conceptual multidimensional models in its implementation. Thus a systematic and efficient approach for mapping of conceptual model to its execution system is a necessary requirement for providing an integrated solution of a DW implementation.

Several proposed formal multidimensional data models at conceptual level [2, 3, 4, 5, 6, 7, 8] have the necessary mapping scheme to the relational model at logical design phase. But the relational model, however, have serious deficiencies in many aspects [9, 10, 11]. In some other approaches [12, 13, 14, 15] the object oriented paradigm has been considered for conceptual level design of DW. But majority of these approaches do not facilitate OLAP operational model. Also very few of these approaches have been implemented at execution level. An approach by extending the OCL has been described in [16] for implementation of conceptual model described in [12]. The OCL has been mapped in relational calculus. In [17], a methodology has been described to enable OLAP users to exploit simultaneously the features of OLAP and object systems. A prototypical OLAP language called SumQL++ also has been defined to demonstrate the capabilities of the proposed method. Further, the object oriented specification and related CASE tool for multidimensional databases also has been addressed in [18] based on GOLD model [14]. However, the GOLD model itself lacks from semantic enriched graphical notations and OLAP operational model.

The Graph Object Oriented Multidimensional Data Model (GOOMD) [19] provides a novel graph based semantic with simple but powerful algebra for multidimensional data model to conceptualize the multidimensional data visualization and operational model for OLAP based on object oriented paradigm. The model is a multidimensional extension of Graph Data Model proposed in [11] to support the conceptualization of DW system. The GOOMD model has been revealed a set of concepts to the conceptual level design phase of DW along with rich set of graphical notations. It makes GOOMD more understandable to the users, independent of implementation issues. Moreover the model provides a set of constructs along with a set of operations to facilitate the need of designers.

This paper proposes a systematic approach to specify the modeling construct in SQL 2003 (compatible to Object Relational (OR) constructs) for GOOMD model to facilitate the designer of DW. Using the proposed approach, conceptual multidimensional schemas can be specified in terms of object types and its hierarchies in object relational database system. Moreover, OLAP operations at the execution level through the set of OLAP operators of GOOMD model can be mapped in terms of OR SQL. Further, the concepts of GOOMD model have been implemented using Generic Modeling Environment (GME) [20] which is a meta-configurable modeling environment. The GME implementation can be used as prototype CASE tools for modeling multidimensional databases using GOOMD model. An interpreter has been developed also to implement the mapping scheme from GOOMD model schema into its equivalent Object Relational schema definition. Hence the proposed solution is able to offer an integrated executable automatic solution for modeling of Data warehouse. The priliminary version of this work has been published in [22].

II. GOOMD Model with Example

In this section, we will summarize the basic concepts of GOOMD model [19]. The GOOMD model is the core of the comprehensive object oriented model of a DW containing all the details that are necessary to specify a data cube, a description of the dimensions, the classification hierarchies, a description fact and measures.

A. The GOOMD Model

The GOOMD model allows the entire multidimensional database to be viewed as a Graph (V, E) in layered organization. At the lowest layer, each vertex represents an occurrence of an attribute or measure, e.g. product name, day, customer city etc. A set of vertices semantically related is grouped together to construct an Elementary Semantic Group (ESG). So an ESG is a set of all possible instances for a particular attribute or measure. On next, several related ESGs are group together to form a Contextual Semantic Group (CSG) – the constructs to represent any context of business analysis. A set of vertices of any CSG those determine the other vertices of the CSG, is called *Determinant Vertices* of said CSG. The most inner layer of CSG is the construct of highest level of granularity of fact in Multidimensional

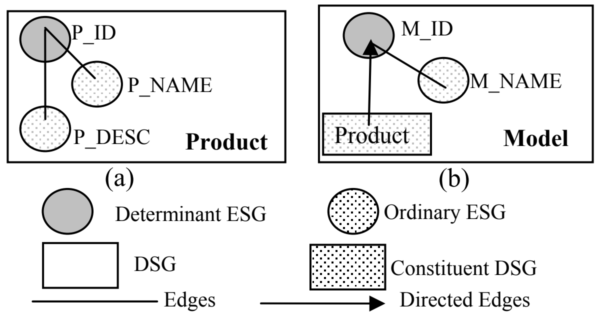


Figure 1: (a) Lowest Level DSG, (b) Higher Level DSG

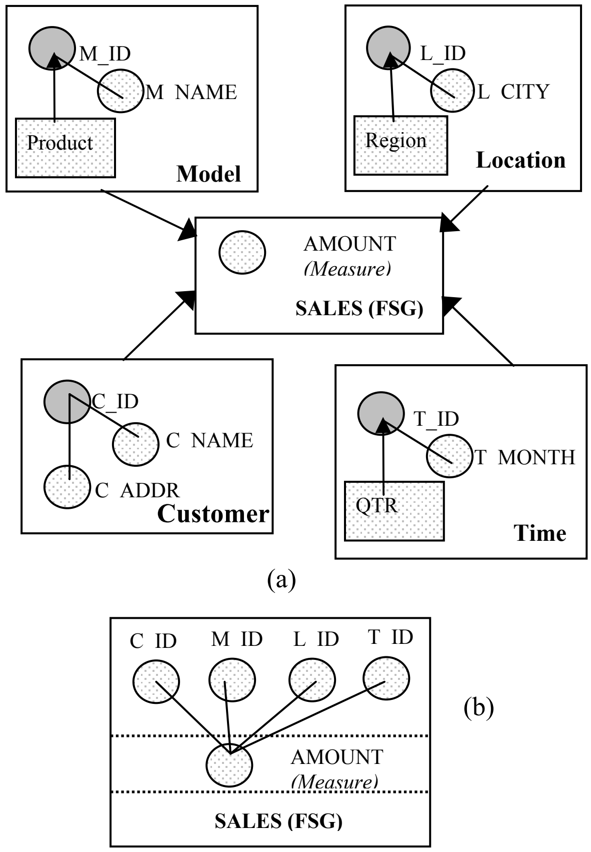


Figure 2: (a) Schema for Sales Application (b) SALES FSG construct after inheritance

database formation. This layered structure may be further organized by combination of two or more CSGs as well as ESGs to represent next upper level layers and to achieve further lower level granularity of contextual data. From the topmost layer the entire database appears to be a graph with CSGs as vertices and edges between CSGs as the association amongst them. *Dimensional Semantic Group (DSG)* is a type of CSG to represent a dimension member, which is an encapsulation of one or more ESGs along with extension and / or composition of one or more constituent DSGs. *Fact Semantic Group (FSG)* is a type of CSG to represent a fact, which is an inheritance of all related DSGs and a set of ESG defined on measures. Two types of edges has been used in GOOMD model, (i) directed edges from DSGs to FSG or constituent DSG to determinant vertex of parent DSG to represent the one – to – many associations and (ii) undirected edges between constituent ESGs and determinant ESGs to represent the association within the members of any CSG.

Since, In order to materialize the cube, one must ascribe values to various measures along all dimensions and can be created from FSG. The cube will also obey a functional constraint $f{:}D_1 \times D_2 \times \ldots \times D_p \rightarrow MI$. Where any D_i is a member of all related top level DSGs and *MI* is instances of set of measures *M*. For schema containing multiple FSGs with shared DSGs, the DSG set $\{D_1, D_2, \ldots D_p\}$ are the common set of DSGs for all FSGs of the schema.

Let consider an example, based on Sales Application with *Sales Amount* as measure and with four dimensions – *Customer*, *Model*, *Time* and *Location* with the set of attributes *{C_ID, C_NAME, C_ADDR}*, *{M_ID, M_NAME, P_ID, P_NAME, P_DESC}*, *{T_ID, T_MONTH, Q_ID, Q_NAME, YEAR}* and *{L_ID, L_CITY, R_ID, R_NAME, R_DESC}* respectively. Model, Time and Location dimensions have upper level hierarchies as Product *{P_ID, P_NAME, P_DESC}*, QTR *{Q_ID, Q_NAME, YEAR}* and Region *{R_ID, R_NAME, R_DESC}* respectively. Then in the notation of GOOMD model there will be four DSGs $D_{Sales} = \{D_{Customer}, D_{Model}, D_{Location}, D_{Time}\}$ with hierarchy. Each DSG will be comprised of either a set of ESGs $E_X \subseteq E_{Sales}$ or a combined set of ESGs and DSGs. As described above the lower layer DSG will be comprised of ESGs only. The *Product* DSG $D_{Product}$ is comprised of only ESGs like E_{P_ID}, E_{P_NAME} *and* E_{P_DESC} and will be represented as the inner layer of the graph. Whereas, DSG for *Model*, D_{Model} is an extension of $D_{Product}$ as well as encapsulation of E_{M_ID} and E_{M_NAME}. The $D_{Product}$ and D_{Model} DSG graphically can be represented as Figure 1. The FSG for the database can be described as $F_{Sales} = \{DET(D_{Customer}), DET(D_{Model}), DET(D_{Location}), DET(D_{Time}), E_{AMOUNT}\}$. Where E_{AMOUNT} is the ESGs defined on the measure *AMOUNT*. The schema from the topmost layer has shown in Figure 2.

B. OLAP Algebra

GOOMD model also provides a concept of OLAP algebra that will operate on different semantic groups and consists of a set of operators. The set of operators can efficiently manipulate the set of instances of Cube, DSGs or even ESGs.

(i) dSelect (π): The operator will extract vertices from some ESG or CSG, depending on some Predicate P. The algebraic notation of the operator is $\pi_P(S) = S_O$, where S is the original ESG or CSG and the S_O is the output ESG or CSG.

(ii) Retrieve (σ): The *Retrieve* operator extracts vertices from the cube C using some constraint *CON* over one or more dimensions or ESGs defined on measures. The algebraic notation of the operator is $\sigma_{CON}(C) = C_O$, where constraint *CON* will be in the form, $CON = (\pi_{P1}(D_1)\text{<op>}\pi_{P2}(D_2)\text{<op>}\ldots\text{<op>}\pi_{Pj}(D_j))$ AND $(\pi_{Pj+1}(E_{m1})\text{<op>}\pi_{Pj+2}(E_{m2})\text{<op>}\ldots\text{<op>}\pi_{Pj+k}(E_{mk}))$. The *Retrieve (σ)* operator is helpful to realize *Slice* and *Dice* operation in connection to OLAP.

(iii) Aggregation (α and +α): The *Aggregation* operators perform aggregation on one or more DSG vertices and will operate on base cube C. The output of the operators will be another cube C_O. The algebraic notations of the operators are $\alpha_{F,\,m,\,DS}(C) = C_O$ and $+\alpha_{F,\,m,\,DS}(C) = C_O$, where F is the relational aggregation function and will perform for measure m. DS is the set of DSGs on which F will operate. The ***α*** *operator* will perform the aggregation function *F* for measure *m* on the specified set of DSGs *DS*. Whereas +***α*** *operator* will perform the aggregation function *F* for measure *m* on each DSG of outer layers including the specified set of DSGs and also persist the output of the corresponding ***α*** operation. Since these realize another important OLAP operation *Roll - Up*. Also the output of ***α*** operation related to the +***α*** operation realize the *Drill-Down* operation.

(iv) Union, Intersection and Difference Operator (∪, ∩ and –): The operators will find Union, Intersection and Difference of two cubes. The algebraic notation for the operators is $C_1 \oplus C_2 = C_O$, where C_O is the output cube and the symbol ⊕ should be replaced by ∪ for Union operation, ∩ for Intersection operation and – should be replaced for Difference operation. The operations can be performed iff both the cube C_1 and C_2 are related with identical set of DSGs and Measures.

(v) Cartesian Product(x): It is a binary operator to relate any two cubes. The algebraic notation of the operator is $C_1 \times C_2 = C_O$.

(vi) Join (| x |): The *Join* operator is a special case of *Cartesian Product* operator. The algebraic notation of the operator is $C_1 \mid\times\mid C_2 = C_O$, where C_O is the output cube. The Join operation between C_1 and C_2 is possible iff $D1 \cap D2 \neq \varnothing$, where D1 and D2 is the set of DSGs associated with C_1 and C_2 respectively. the *Join* operator can be expressed as, $C_1 \mid\times\mid C_2 = \sigma_{CON}(C_1 \times C_2)$, where *CON* will equate the similar DSGs of C_1 and C_2.

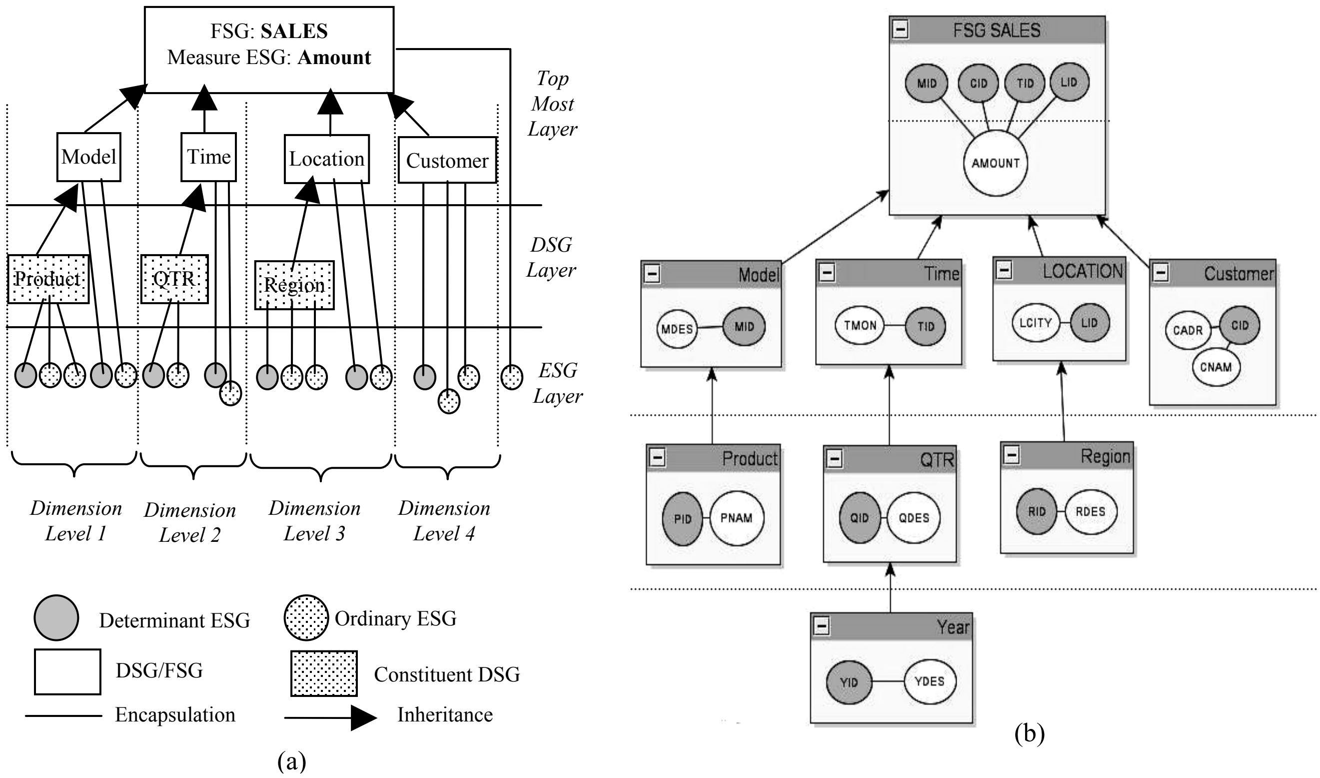
Since the operators like *Union, Intersection, Difference* and *Cartesian Product* can operate on any ESG (e.g. ESG defined on measures) also. In that case the operators will extract the set of vertices from the ESGs on which it is operating and will form another related ESG, without changing the meaning of the operator as defined above.

C. The Hierarchical View of GOOMD Model

The hierarchical views of the above described Sales Application in GOOMD notation has shown in Figure 3.(a). The Sales application in hierarchical view consists of three layers. Lowest layer is *ESG Layer*, which is collection of all ESGs Defined on attributes or measures. Constituent DSGs are placed in intermediate layer i.e. *DSG Layer*. GOOMD schema may have multiple DSG Layers.

Since, constituent DSGs may be extended or may be encapsulated like other ESGs to define upper layer DSGs. DSGs with different granularity level for any Dimension Level may be placed in different layer and so there may exists multiple DSG layers. The *Top Most Layer* consists of DSGs with lowest level granularity and FSGs. For the Sales Application, there are four DSGs and one FSG.

Different *Dimension Levels* in the Figure 4.(a) shows the hierarchy of different dimension members, each consists of DGSs and their corresponding constituent DSGs. For the Sales Application, there are four *Dimension Levels*. The ESGs not associated with any Dimension Levels are the ESGs defined on different measures. In Figure 3.(a) only one such ESG exist, that is defined on measure *Amount*. As discussed earlier, the Cube with lowest level granularity or highest detail of data can be materialized from the top most layer of



(a) (b)

Figure 3: (a) Hierarchical View of GOOMD Model, (b) Example Sales Schema in Hierarchical View with GOOMD Notation (with encapsulated ESGs)

the GOOMD model. The detail schema of Sales Application has been shown in Figure 3.(b).

III. Object relational implementation of GOOMD Model

The concept of any multidimensional data model consists of three basic construct namely, (1) Dimensions, where each can consist of a multi-level classification hierarchy, (2) Facts and (3) Measures. In object oriented concept different object types need to specify for Dimension members and Fact constructs type. Object identification or OID must address the key attributes specification. In the context of GOOMD model, the construct like DSG and FSG will be realized by the object type definitions. The determinant ESG will realize the OID for the specific semantic construct. Further, it is important to note that, the dimension hierarchy level can be represented by corresponding inheritance tree of object types defined on the hierarchy of DSGs.

In this section, a set of rules to specify equivalent Object Relational schemas for GOOMD Model has been proposed. Based on those rules, different object types have been specified corresponding to the GOOMD model construct and its graphical notations, with the purpose of system level implementation of DW from its conceptual design model.

We have used the SQL 2003 standard for mapping GOOMD model schema into the Object Relational model.

Since SQL 2003 supports *structured user-defined types or object data types*, which are analogous to class declarations in object languages. Object data types group semantically-related attributes, which can be of any SQL type and of public visibility. Further object data types can include other encapsulated object data types as complex attributes. The type hierarchy also can be defined using object data types but with the restriction to single inheritance only i.e. a subtype can directly derive only from a single super type. Further, using definition of some object data types, either objects can be stored in columns of relational tables or object data can be stored in object tables, where each row is an object. As example of a commercial object-relational DBMS we have used Oracle 10g Release 2 which is compliance to SQL2003 standard [21].

A. Implementation of GOOMD Model Constructs

For the implementation of GOOMD model constructs into equivalent Object Relational Schema we are setting the following implementation rules.

Rule 1: All ESGs will be mapped directly into simple attributes.

Rule 2: Lowest Layer's DSGs will be mapped directly into the object type of OR features. For the storage of instances of such object type, the table structure will be defined on the object type with the OID as specified by Determinant ESGs.

An example has been shown in Figure 4. Since, *tProduct* in the example is an Object Table where every row object is correspond to the *tyProduct* object type and *pid* is the object identifier (OID) which uniquely identifies each object in the object table.

Rule 3: Higher Layer's DSGs with single inheritance will be mapped into the object type with inheritance. For the storage of instances of such object types the object table structures will be defined for both supertype and subtype object types with the OIDs as specified by Determinant ESGs.

An example has been shown in Figure 5. Since, tyModel is a specialized object type of tyProduct. To maintain the referential integrity within the instances of generalized and specialized object type, referential integrity constraint has been imposed in the object table of specialized object type. In the context, this is important to note that, in the concept GOOMD model the referential integrities are maintained inherently.

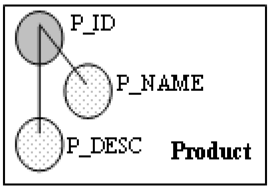
GOOMD Construct	Equivalent Object Type & Object Table Definition
	CREATE TYPE tyProduct AS OBJECT (pid NUMBER, pname varchar2(10), pdes varchar2(10)) NOT FINAL / CREATE TABLE tProduct OF tyProduct (pid PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY; /

Figure 4: Implementation of Rule 2

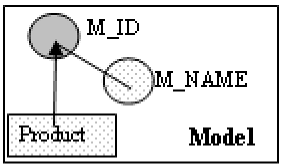
GOOMD Constructs	Equivalent Object Type & Object Table Definition
	CREATE TYPE tyModel UNDER tyProduct (mid NUMBER, mdes varchar2(10)) NOT FINAL; / CREATE TABLE tModel OF tyModel (mid PRIMARY KEY, mid REFERENCES(tProduct)) OBJECT IDENTIFIER IS PRIMARY KEY; /

Figure 5: Implementation of Rule 3

Rule 4: Higher Layer's DSGs with encapsulation of constituent DSG will be mapped into the object type with nesting. For the storage of instances, the object table structures will be defined only on the parent object type with the OIDs as specified by Determinant ESGs.

An example has been shown in Figure 6. In the example, tyCADDR is an object type and addr is an encapsulated object type in the parent object type tyCustomer. As the rule described the object table tCustomer has been created only for the object type tyCustomer and addr has been treated as column object of tCustomer table.

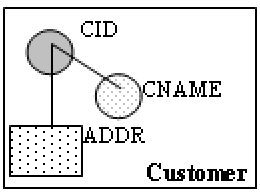
GOOMD Constructs	Equivalent Object Type & Object Table Definition
	CREATE TYPE tyCADDR AS OBJECT (City varchar2(10), ……) / CREATE TYPE tyCustomer (cid NUMBER, cname varchar2(10), addr tyCADDR) NOT FINAL; / CREATE TABLE tCustomer OF tyCustomer (cid PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY; /

Figure 6: Implementation of Rule 4

Rule 5: Higher Layer's DSGs or FSGs with multiple inheritances will be mapped into the object table with scoped references of the parent objects. Since OR feature does not support multiple inheritance. An example has been shown in Figure 7.

Rule 6: Cube will be treated as object view of Fact Object Type. Multiple views can be created with different level of details from the same fact object type. Also a Cube may be created from multiple fact object tables with common set of dimension hierarchies.

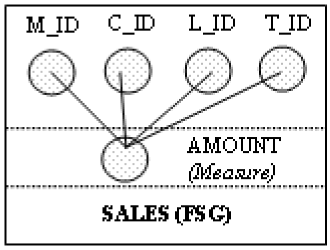
GOOMD Constructs	Equivalent Object Type & Object Table Definition
	CREATE TYPE tyFSales as object (mr REF tyModel, lr REF tyLocation, cr REF tyCustomer, tr REF tyTime, AMOUNT number(7,2)); / CREATE TABLE tFSales (mr REF tyModel SCOPE IS tModel, lr REF tyLocation SCOPE IS tLocation, cr REF tyCustomer SCOPE IS tCustomer, tr REF tyTime SCOPE IS tTime, AMOUNT number(7,2)); /

Figure 7: Implementation of Rule 5

Since the view for the base cube can be formed from the SALES FSG definition of Rule 4 example using following view definition,

CREATE VIEW vCubeSales OF tyFSales WITH OBJECT IDENTIFIER (mr.mid, lr.lid, cr.cid, tr.tid) AS SELECT e.mr, e.lr, e.cr, e.tr, e.Amount FROM tFSales e;

/

B. *Cube Representation from Multiple Facts*

In GOOMD model concept a Cube can be materialized from multiple Fact Object Tables with shared dimension hierarchy where multiple measures attributes can be taken from different Fact Object Tables. Further using two *Retrieve* operations on the same Cube one can change the cell value in multidimensional space without changing the associated top level DSGs. This will realize the *Drill-Across* operation of OLAP, which is implicit in the Cube concept of GOOMD Model. Also there is no restriction to define a derived FSG with new set of measure ESGs by inheriting the existing base FSG. By defining a Cube on new FSG, one can perform the Drill-Across operation between new set and existing set of measure ESGs.

For example let the schema of Figure 1 containing another FSG *SalesQty* with *Quantity* as attributes along with the existing FSG Sales. Also let both FSGs are sharing the same dimension hierarchy. Then two fact object tables can be created as follows,

CREATE TABLE tFSales (mr REF tyModel SCOPE IS tModel, lr REF tyLocation SCOPE IS tLocation, cr REF tyCustomer SCOPE IS tCustomer, tr REF tyTime SCOPE IS tTime, Amount number(7,2));

/

CREATE TABLE tFSalesQty (mr REF tyModel SCOPE IS tModel, lr REF tyLocation SCOPE IS tLocation, cr REF tyCustomer SCOPE IS tCustomer,

tr REF tyTime SCOPE IS tTime, Quantity number(5));

/

The cube from the multiple fact object types can be created as follows,

CREATE TYPE tyFS AS OBJECT (mr REF tyModel, lr REF tyLocation, cr REF tyCustomer, tr REF tyTime, Amount number(7,2), Quantity number(5));

/

CREATE VIEW vCubeSalesAmtQty OF tyFS WITH OBJECT IDENTIFIER (mr.mid,lr.lid,cr.cid,tr.tid) AS SELECT a.mr, a.lr, a.cr, a.tr, a.Amount, b.Quantity FROM tFSales a, tFSalesQty b where a.mr = b.mr, a.lr = b.lr, a.cr = b.cr, a.tr = b.tr;

/

Now, two queries associated to *Retrieve* operator, one with measure attribute *Amount* and another with measure attribute *Quantity* can be formed along the common dimension hierarchy to realize the *Drill-Across* operation.

C. Implementation of OLAP Operators

In this section we will focus on implementation of OLAP operators as defined in GOOMD model, into object relational SQL which will operate on the Object View created for Cube or on Object Tables created for FSG constructs. The dimension hierarchy levels for a cube can be represented by corresponding inheritance trees of Object Tabled defined on corresponding DSGs. For the purpose of query on the cube, the general form of Object Relational SQL will be as follows,

SELECT DimObjTab$_1$.OID, ..., DimObjTab$_n$.OID, AggrFun(Cube.Measure$_1$), ..., (Cube.Measure$_2$)
FROM Cube c, DimObjTab$_1$ d$_1$, ..., DimObjTab$_n$ d$_n$
WHERE c.RefDim$_1$.OID=d$_1$.OID AND ... AND c. RefDim$_n$.OID=d$_n$
AND *{Other Predicates on Dimension Object Tables}*
GROUP BY DimObjTab$_1$.OID, ..., DimObjTab$_n$.OID
ORDER BY DimObjTab$_1$.OID, ..., DimObjTab$_n$.OID;

/

The general form of the query on OLAP imposes certain interesting properties,

Property 1: The SELECT clause contains the aggregate functions described on measure attributes of Cube and the object identifiers of object tables defined on corresponding DSGs. Each dimension object table corresponds to the top most layer DSGs of each Dimension hierarchy level [Figure 3.(a)]. The OIDs are corresponding to the identifiers of dimension object tables of specific layers at which we want to aggregate the measures.

Property 2: Further as discussed earlier, a Cube may be materialized from multiple fact object tables with common dimension hierarchy, so different measure attributes of the cube may belong to different fact object tables.

Property 3: The FROM clause contains the Object View on object tables defined on FSG and topmost layer DSGs.

Property 4: The WHERE clause use to link the Cube and the top most layer dimension object tables using referenced OID and it also contain other predicated defined on dimension object tables.

Property 5: The GROUP BY clause contains the object identifiers of dimension object table of some specific layer at which we want to aggregate the measures.

Property 6: The ORDER BY clause use to sort the output of the queries based on the object identifiers.

Using the above said general form of OLAP query on cube and its properties, the GOOMD model operators (discussed in section 2.2) can be implemented as follows,

a) Retrieve Operator: $\sigma_{(\pi RDES\ =\ \text{“North”}\ (Location)\ \wedge\ \pi YDES\ =\ 2007\ (Time))}(SALES) = C_{Result}$

Query:

SELECT d1.mid, d2.lid, d3.cid, d4.tid, f.Amount FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid AND d4.ydes = ‘2007’ AND d2.rdes=’North’
GROUP BY d2.lid, d1.mid, d3.cid, d4.tid
ORDER BY d2.lid, d1.mid, d3.cid, d4.tid;

/

Since Retrieve Operator realize the SLICE and DICE operations of OLAP. In the above query the *vCubeSales* will be sliced by the predicate *ydes =’2007’ and rdes =’North’.* The dice operation can be performed only by using a subset of Dimension Object Tables of specific layers in SELECT, WHERE, GROUP BY and ORDER BY clause respectively.

b) Aggregation Operator – I: $\alpha_{SUM,\ AMOUNT,\ \{tYear\}}$ (SALES) = C_{Result}

Query:

SELECT d1.mid, d2.lid, d3.cid, d4.yid, SUM(f.Amount) FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid
GROUP BY d2.lid, d1.mid, d3.cid, d4.yid
ORDER BY d2.lid, d1.mid, d3.cid, d4.yid;

/

The above query results the Roll-Up operation upto the Year DSG [see Figure 4.(b)] from the base cube vCubeSales.

c) Aggregation Operator – II: $+\alpha_{SUM,\ AMOUNT,\ \{tYear\}}$ (SALES) = C_{Result}

Query:

SELECT d1.mid, d2.lid, d3.cid, SUM(f.Amount) FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid
GROUP BY d2.lid, d1.mid, d3.cid
ORDER BY d2.lid, d1.mid, d3.cid;

Intermediate Result 1:IC_1

SELECT d1.mid, d2.lid, d3.cid, d4.yid, SUM(f.Amount) FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid
GROUP BY d2.lid, d1.mid, d3.cid, d4.yid
ORDER BY d2.lid, d1.mid, d3.cid, d4.yid;

/

Intermediate Result 2:IC_2

SELECT d1.mid, d2.lid, d3.cid, d4.qid, SUM(f.Amount) FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid
GROUP BY d2.lid, d1.mid, d3.cid, d4.qid
ORDER BY d2.lid, d1.mid, d3.cid, d4.qid;
/

On execution of above +α Aggregation Operator, the query will result C_0 the total Amount of sales on other given dimension. Also it will generate the Intermediate Cube IC_1 and IC_2. This operator will facilitate the Drill-Down Operation also. Since IC_2 is Drill-Down output of IC_1 and IC_1 is Drill-Down Output of C_0.

d) UNION Operator: $\sigma_{(\pi YDES = 2007 (Time))}(SALES) U \sigma_{(\pi YDES = 2008 (Time))}(SALES) = C_{Result}$

Query:

SELECT d1.mid, d2.lid, d3.cid, d4.tid FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid AND d4.ydes = '2007'
GROUP BY d2.lid, d1.mid, d3.cid, d4.tid
ORDER BY d2.lid, d1.mid, d3.cid, d4.tid;
UNION
SELECT d1.mid, d2.lid, d3.cid, d4.tid FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, vCubeSales f
WHERE f.mr.mid = d1.mid AND f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid AND d4.ydes = '2008'
GROUP BY d2.lid, d1.mid, d3.cid, d4.tid
ORDER BY d2.lid, d1.mid, d3.cid, d4.tid;
/

e) Join Operator: Let the query is "Find those models having total sales during the first QTR greater than half of their sales for the entire year for 2007".

Query:

The join operator for the query as described in section 2.2 may be expressed in the following way,

$\sigma_{CON}(C_1 \mid x \mid C_2) = C_{Result}$

Where,

$CON = \pi_{C2.Amount \geq 1/2*C1.Amount}(C1.Amount \ X \ C2.Amount)$,

$\sigma_{CON1}(SALES) = C1$, $\sigma_{CON2}(SALES) = C2$,

$CON1 = \pi_{YDES=2007}(TIME)$ and $CON2 = \pi_{YDES=2007 \wedge QDES="1h7"}(TIME)$

For the purpose we are creating two Cubes based on the Fact tables as follows,

CREATE VIEW C1 AS SELECT d2.lid, d3.cid, d4.yid, sum(f.Amount) "Amount" FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, tFSales f WHERE f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid AND f.tr.ydes='2007' GROUP BY d2.lid, d3.cid, d4.yid ORDER BY d2.lid, d3.cid, d4.yid;
/
CREATE VIEW C2 AS SELECT d2.lid, d3.cid, d4.yid, sum(f.Amount) "Amount" FROM tModel d1, tLocation d2, tCustomer d3, tTime d4, tFSales f WHERE f.lr.lid = d2.lid AND f.cr.cid = d3.cid AND f.tr.tid = d4.tid AND f.tr.ydes='2007' AND f.tr.qdes ='1h7' GROUP BY d2.lid, d3.cid, d4.yid ORDER BY d2.lid, d3.cid, d4.yid;
/

Now the query for the join of two Cubes C1 and C2 is as follows,

SELECT C2.lid, C2.cid, C2.yid, C2."Amount" from C1, C2 WHERE C1.lid=C2.lid AND C1.cid = C2.cid AND C1.yid=C2.yid AND C2."Amount" >= 0.5*C1."Amount";
/

IV. Implementation of GOOMD Model Using GME

The Generic Modeling Environment (GME) provides meta-modeling capabilities and where a domain model can be configured and adapted from meta-level specifications (representing the Conceptual modeling) that describe the domain concept. It is common for a model in the GME to contain several numbers of different modeling elements with hierarchies that can be in many levels deep. The GME supports the concept of a viewpoint as a first-class modeling construct, which describes a partitioning that selects a subset of conceptual modeling components as being visible.

Moreover, GME support the programmatic access of the metadata of GME models. Most usual techniques for such programmatic access is to write GME interpreter for some metamodel. The interpreter will be able to interpret any domain model based on that predefined metamodel. GME interpreters are not standalone programs, they are components (usually Dynamic Link Libraries) that are loaded and executed by GME upon a user's request. Most GME components are built for the Builder Object Network (BON), an inbuilt framework in GME and provide a network of C++ objects. Each of these represents an object in the GME model database. C++ methods provide convenient read/write access to the objects' properties, attributes, and relations described in GME metamodel.

In the context of GOOMD model, the lower layers can be conceptualized using levels in GME. The interpreter for the model has been developed using BON in Visual C++ IDE. The interpreter will generate the equivalent Object – Relational (OR) data definitions for any given GME model configured using meta-level specifications of GOOMD model.

Recalling the example of DW system based on *Sales Application* (Figure 2 and Figure 3) with *Sales Amount* as measure and with four dimensions – *Customer, Model, Location* and *Time*. Also, one ESG has been defined on the measure *AMOUNT*. Say for the Sales application the set of ESGs are E_{Sales}. The meta-level specifications of GOOMD model using GME has been shown in Figure 8.

The GOOMD model schema specification of Sales Application using GME has been shown in Figure 9. The BON based interpreter for GOOMD model can run from the GME interface to interpret any GOOMD model schema like Sales Application schema to generate the equivalent Object Relational data definition language. The interpreter output of Sales Application schema has been shown in Figure 10.

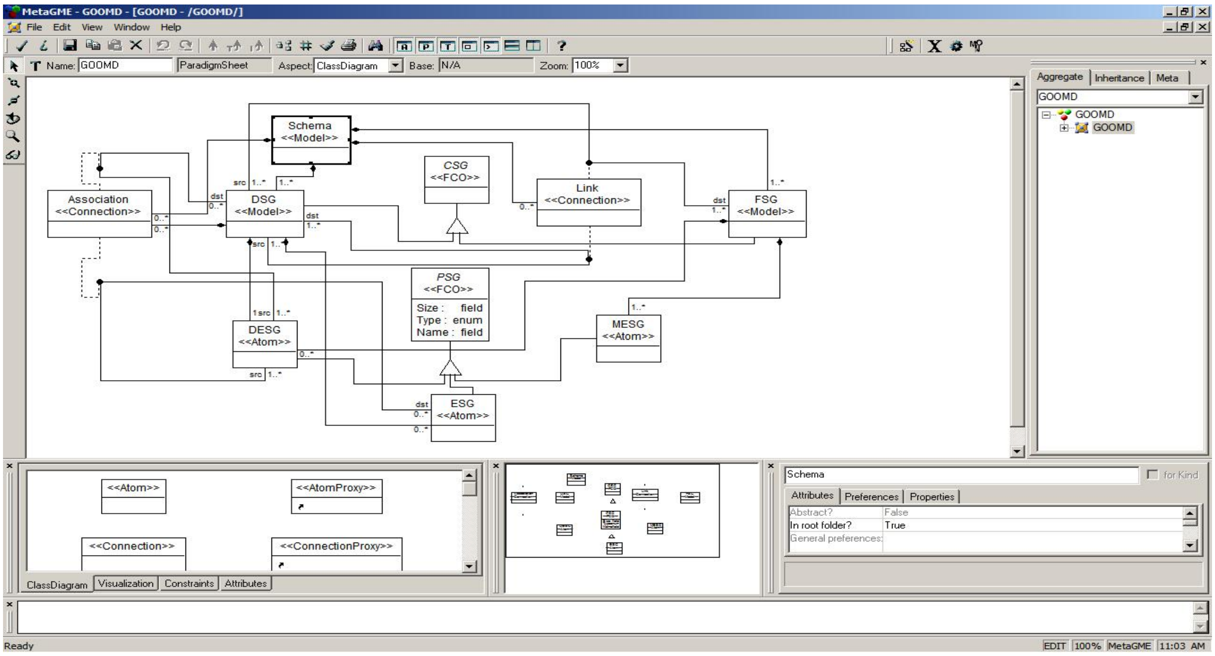


Figure 8: Meta-Level Specifications of GOOMD model using GME

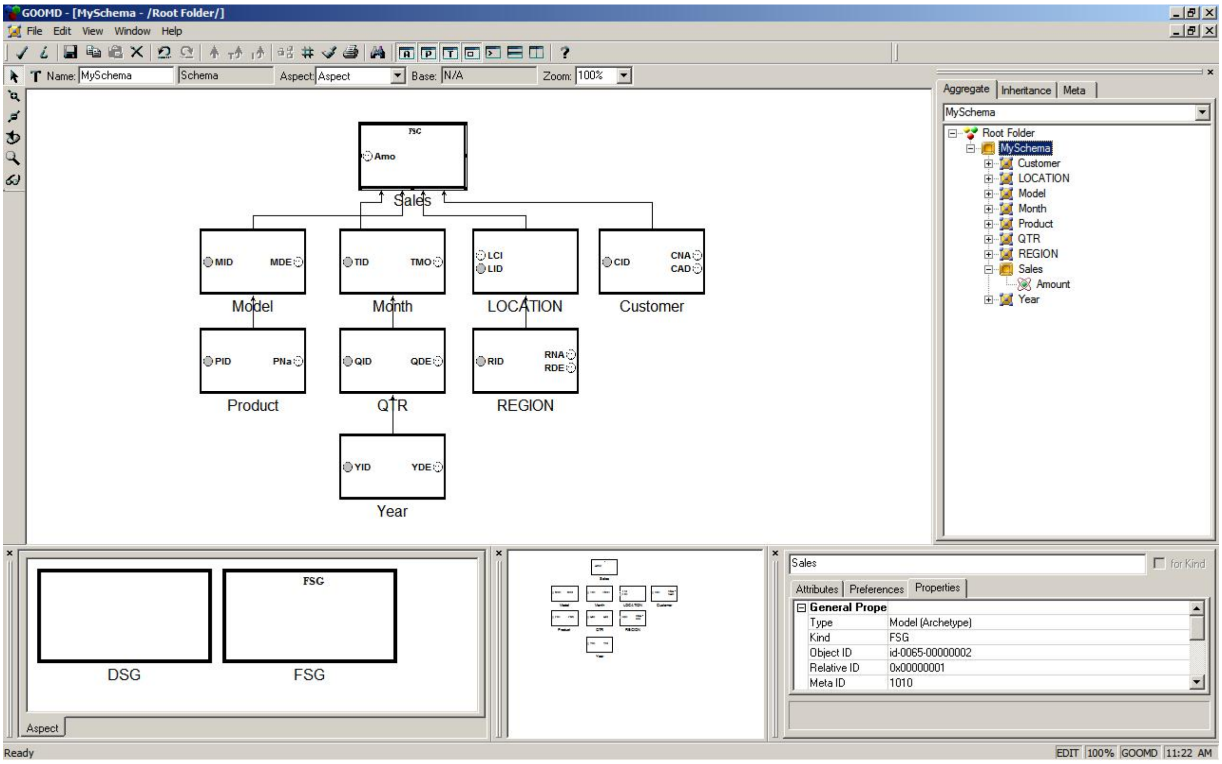


Figure 9. GOOMD Model Schema of Sales Application using GME

V. Conclusion

In this paper a systematic approach has been proposed to specify the conceptual level multidimensional data model called, GOOMD model, into an equivalent object types. It is compatible to SQL 2003 standard. The proposed rule based approach can express the concepts, graphical notations and the OLAP operators of the GOOMD model at the system level implementation of DW. The expressive power of the proposed approach also has been demonstrated using typical examples.

The main objective of the proposed methodology is to remove the semantic gap between advanced conceptual level data models and multidimensional implementations of data cubes. The advantage of this approach is multifold. *Firstly*, it provides a systematic approach to express the formal conceptual multidimensional data model at execution level and facilitate the designer of DW to specify the operational system for DW more effectively. *Secondly*, the proposed rule based approach is simple, powerful, expressive, and has been drawn from basic concept of object orientation. *Thirdly*, the implementation of proposed approach exhibit a

```
CREATE TYPE tyYear AS OBJECT (YID  NUMBER (2), YDES  VARCHAR2 (6)) NOT FINAL;
CREATE TABLE tbYear OF tyYear (YID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
CREATE Type tyQTR UNDER tyYear (QID  NUMBER (2), QDES  VARCHAR2 (5)) NOT FINAL;
CREATE TABLE tbQTR OF tyQTR (QID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
CREATE Type tyMonth UNDER tyQTR (TID  NUMBER (2), TMON  VARCHAR2 (10)) NOT FINAL;
CREATE TABLE tbMonth OF tyMonth (TID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
CREATE TYPE tyProduct AS OBJECT (PID  NUMBER (2), PNAME  VARCHAR2 (15)) NOT FINAL;
CREATE TABLE tbProduct OF tyProduct (PID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
CREATE Type tyModel UNDER tyProduct (MID  NUMBER (2), MDES  VARCHAR2 (10)) NOT FINAL;
CREATE TABLE tbModel OF tyModel (MID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY KEY;
CREATE TYPE tyCustomer AS OBJECT (CID  NUMBER (2), CNAME  VARCHAR2 (15), CADDR
VARCHAR2 (20)) NOT FINAL;
CREATE TABLE tbCustomer OF tyCustomer (CID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY
KEY;
CREATE TYPE tyREGION AS OBJECT (RID  NUMBER (3), RNAME  VARCHAR2 (15), RDESC
VARCHAR2 (30)) NOT FINAL;
CREATE TABLE tbREGION OF tyREGION (RID   PRIMARY KEY) OBJECT IDENTIFIER IS PRIMARY
KEY;
CREATE Type tyLOCATION UNDER tyREGION (LID  NUMBER (3),  LCITY  VARCHAR2 (15)) NOT
FINAL;
CREATE TABLE tbLOCATION OF tyLOCATION (LID   PRIMARY KEY) OBJECT IDENTIFIER IS
PRIMARY KEY;
CREATE TABLE FSGSales (refMonth REF tyMonth SCOPE IS tbMonth, refModel REF tyModel SCOPE
IS tbModel, refCustomer REF tyCustomer SCOPE IS tbCustomer, refLOCATION REF tyLOCATION
SCOPE IS tbLOCATION, Amount  NUMBER (10));
```

Figure 10: GOOMD Interpreter Output for Sales Applications Schema

comprehensive guideline for automatic generation of execution model like SQL2003 compatible Object relational schemas from the conceptual model and its graphical notations. And *finally*, the proposed methodology, in general, can be used with any conceptual multidimensional data model with proper mapping rules to specify the model at execution level.

The proposed approach also has been automated through the GME based interpreter for GOOMD model. The meta-level specification of GOOMD model along with the interpreter can be used as a CASE tool for the model by the DW designer.

References

[1] Stefano Rizzi, Alberto Abelló, Jens Lechtenbörger, Juan Trujillo, “Research in data warehouse modeling and design: dead or alive?”, Proceedings of the 9th ACM international workshop on Data warehousing and OLAP, PP 3 – 10, November 2006.

[2] E. Franconi and U. Sattler,"A Data Warehouse Conceptual Data Model for Multidimensional Aggregation: a preliminary report", Italian Association for Artificial Intelligence AI*IA Notizie, Vol. 1, PP 9-21, 1999.

[3] Anindya Datta and Helen Thomas, "The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses", Decision Support Systems, Vol. 27(3), PP 289-301, December 1999.

[4] Nectaria Tryfona, Frank Busborg, Jens G. Borch Christiansen, "starER: A Conceptual Model For Data Warehouse Design", Proceedings of the 2nd ACM international workshop on Data warehousing and OLAP, PP 3 – 8, November 1999.

[5] Matteo Golfarelli and Dario Maio and Stefano Rizzi,"The Dimensional Fact Model: A Conceptual Model for Data Warehouses", International Journal of Cooperative Information Systems, Vol 7, No 2-3, PP 215-247, 1998.

[6] Karl Hahn, Carsten Sapia, Markus Blaschka, "Automatically Generating OLAP Schemata From

Conceptual Graphical Models", Proceedings of the 3rd ACM international workshop on Data warehousing and OLAP DOLAP, PP 9 – 16, November 2000.
[7] Nguyen Thanh Binh, A. Min Tjoa, "Conceptual Multidimensional Data Model Based On Object-Oriented Metacube", Proceedings of the 2001 ACM symposium on Applied computing, PP 295 – 300, March 2001.
[8] Aris Tsois and Nikos Karayannidis and Timos K. Sellis, "MAC: Conceptual data modeling for OLAP", Booktitle "Design and Management of Data Warehouses", PP 5, 2001.
[9] Senko M. E., “Information Systems: Records, relations, set, entities and things”, Information Systems, Vol. 1.1, PP 3 – 13, 1975.
[10] Hideko S.Kunii, “Graph Data Model and its Data Language”, Springer – Verlag, 1990.
[11] S. Choudhury, N. Chaki, S. Bhattacharya, “GDM: A New Graph Based Data Model Using Functional Abstraction”, Journal of Computer Science and Technology, Vol. 21(3), PP 430 – 438, 2006.
[12] Sergio Luján-Mora, Juan Trujillo and Il-Yeol Song, "A UML Profile For Multidimensional Modeling in Data Warehouses", Data & Knowledge Engineering, Vol. 9(3), PP 725 – 769, December 2006.
[13] Nicolas Prat, Jacky Akoka and Isabelle Comyn-Wattiau, "A UML-based Data Warehouse Design Method", Decision Support Systems, Vol 42(3), PP 1449 – 1473, December 2006.
[14] J. Trujillo, “The GOLD Model: An Object-Oriented ConceptualModel for the Design of OLAP Applications”, Doctoral Dissertation, Languages and Information Systems Dept., Alicante University, Spain, June 2001.
[15] Zepeda, L.; Celma, M., “A model driven approach for data warehouse conceptual design”, 7th International Baltic Conference on Databases and Information Systems, PP 114 – 121, 2006.
[16] Jesús Pardillo, Jose-Norberto Mazón, Juan Trujillo, "Extending OCL for OLAP querying on conceptual multidimensional models of data warehouses", Information Sciences, Vol.180(5), PP 584-601, 2010
[17] T. B. Pedersen, J. Gu, A. Shoshani, C. S. Jensen, "Object-extended OLAP querying", Data & Knowledge Engineering, Vol.68(5), PP 453-480, 2009.
[18] Juan Trujillo, Manuel Palomar, Jaime Gómez, “The GOLD definition language (GDL): An Object Oriented Formal Specification Language For Multidimensional Databases”, Proceedings of the ACM symposium on Applied Computing, Vol. 1, PP 346 – 350, March 2000.
[19] Anirban Sarkar, Sankhayan Choudhury, Nabendu Chaki, Swapan Bhattacharya, “Conceptual Level Design of Object Oriented Data Warehouse: Graph Semantic Based Model”, INFOCOMP Journal of Computer Science, Vol 8(4), PP 60 – 70, 2009.
[20] Ákos Lédeczi, Arpad Bakay, Miklos Maroti, Peter Volgyesi, Greg Nordstrom, Jonathan Sprinkle, and Gábor Karsai, “Composing Domain-Specific Design Environments”, IEEE Computer, PP 44-51, 2001.
[21] Oracle Corp. (2005), http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/ap_standard_sql003.htm# i7719, 2005.
[22] Anirban Sarkar, Sankhayan Choudhury, Nabendu Chaki, Swapan Bhattacharya, “Object Relational Implementation of Graph Based Conceptual Level Multidimensional Data Model”, 9th International Conference on Computer Information Systems and Industrial Management Applications (CISIM 2010), PP 154 – 159, October 2010.A. Bonnaccorsi. “On the Relationship between Firm Size and Export Intensity”, *Journal of International Business Studies*, XXIII (4), pp. 605-635, 1992. *(journal style)*

Author Biographies

Anirban Sarkar is presently a faculty member in the Department of Computer Applications, National Institute of Technology, Durgapur, India. He received his PhD degree from National Institute of Technology, Durgapur, India in 2010. His areas of research interests are Database Systems and Software Engineering. His total number of publication in various international platforms is about 20.

Sankhayan Choudhury is presently a faculty member in the Department of Computer Science & Engg., University of Calcutta, Kolkata, India. He received his Ph.D. degree from Jadavpur University, India in 2006. His areas of research interests are Distributed Computing and Database Systems. He has published about 30 papers in International Conferences and journal. He is also actively involved in organizing international conferences on distributed computing.

Nabendu Chaki is Head and Associate Professor in the Department of Computer Science & Engineering, University of Calcutta, Kolkata, India. He received his Ph.D. degree from Jadavpur University, India in 2000. His areas of research interests include Distributed Computing and Software Engineering. Dr. Chaki has supervised in the Ph.D. program in Software Engineering in Naval Postgraduate School, Monterey, CA, USA during 2001–2002, as a Research Assistant Professor. He is a visiting faculty member for many Universities including the University of Ca'Foscari, Venice, Italy. Besides being in the editorial board of a few International Journals, Dr. Chaki has also served in the committees of several international conferences. His total number of publications in referred international journals and conferences is more than 70.

Swapan Bhattacharya is presently working as Professor in Department of Computer Science & Engineering, Jadavpur University, Kolkata, India. He has served as Director of National Institute of Technology, Durgapur, India during 2005 – 2010. He did his Ph.D. in Computer Science in 1991 from University of Calcutta, India. His areas of research interests are distributed computing and software engineering. He had received Young Scientist Award from UNESCO in 1989. As a Sr. Research Associate of National Research Council, USA, he had also served as the coordinator of Ph.D. program in Software Engg. in Naval Postgraduate School, Monterey, CA during 1999–2001. He has published over 100 research papers in various international platforms. He is actively involved in collaborative research with several Institutes in UK and USA and also in organizing international conferences on software engineering and distributed computing.