Designing Data Warehouses with OO Conceptual Models

Based on a subset of the Unified Modeling Language, the authors' object-oriented approach to building data warehouses frees conceptual design from implementation issues.

Juan Trujillo
Manuel Palomar
Jaime Gomez
Universidad de Alicante

Il-Yeol Song
Drexel University

Most developers agree that data warehouse, multidimensional database (MDB), and online analytical processing (OLAP) applications emphasize multidimensional modeling, which offers two benefits. First, the multidimensional model closely parallels how data analyzers think and, therefore, helps users understand data. Second, this approach helps predict what final users want to do, thereby facilitating performance improvements.

Developers have proposed various approaches for the conceptual design of multidimensional systems. These proposals try to represent the main multidimensional properties at the conceptual level with special emphasis on data structures.

A conceptual modeling approach for data warehouses, however, should also address other relevant aspects such as initial user requirements, system behavior, available data sources, and specific issues related to automatic generation of the database schemes. We believe that object orientation with the Unified Modeling Language can provide an adequate notation for modeling every aspect of a data warehouse system from user requirements to implementation.

We propose an OO approach to accomplish the conceptual modeling of data warehouses, MDB, and OLAP applications. This approach introduces a set of minimal constraints and extensions to UML[1] for representing multidimensional modeling properties for these applications. We base these extensions on the standard mechanisms that UML provides for adapting itself to a specific method or model, such as constraints and tagged values. Our work builds on previous research,[2-4] which provided a foundation for the results we report here and for earlier versions of our work. We believe that our innovative approach provides a theoretical foundation for the use of OO databases and object-relational databases in data warehouses, MDB, and OLAP applications.

We use UML to design data warehouses because it considers an information system's structural and dynamic properties at the conceptual level more naturally than do classic approaches such as the Entity-Relationship model. Further, UML provides powerful mechanisms—such as the Object Constraint Language[1] and the Object Query Language[1]—for embedding data warehouse constraints and initial user requirements in the conceptual model. This approach to modeling a data warehouse system yields simple yet powerful extended UML class diagrams that represent main data warehouse properties at the conceptual level.

MULTIDIMENSIONAL MODELING PROPERTIES

Multidimensional modeling structures information into *facts* and *dimensions*. We define a fact as an item of interest for an enterprise, and describe it through a set of attributes called *measures* or *fact attributes*—atomic or derived—which are contained in cells or points within the data cube. We base this set of measures on a set of dimensions that derive from the granularity chosen for representing the facts. These dimensions thus present the context for analyzing the facts. Further, attributes—usually called *dimension attributes*—provide the specifics that characterize dimensions.

For example, consider the *product sales* category for a large chain of stores, which has the following

0018-9162/01/$17.00 © 2001 IEEE

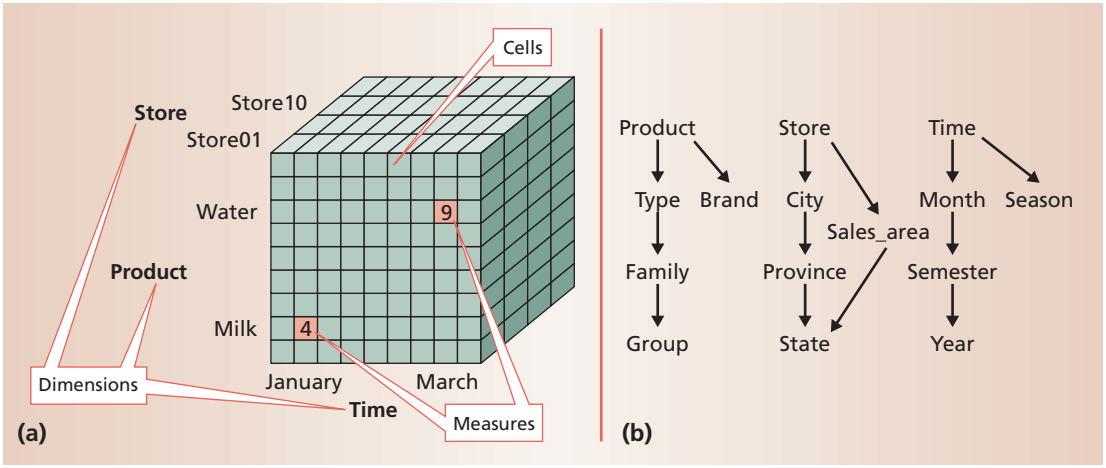


Figure 1. A multidimensional model data cube: (a) The cube itself is composed of cells that define fact attributes, while (b) the classification hierarchies display the dimensions that define the cube—*product*, *store*, *and* time.

dimensions: *product*, *store*, *customer*, and *time*. Figure 1 shows both a data cube and classification hierarchies. In Figure 1a, we see a data cube typically used for multidimensional modeling. In this particular case, we have defined a cube for analyzing measures along the *product*, *store*, and *time* dimensions, as shown in Figure 1b's classification hierarchies.

We usually consider facts as *many-to-many* relationships between all dimensions and as *many-to-one* relationships between the fact and every particular dimension. In our example, *product sales* is related to only one *product* that is sold in one *store* to one *customer* at one *time*.

In some cases, facts can represent *many-to-many* relationships between particular dimensions. Suppose that we want to associate *product sales* with the purchase tickets each chain store generates. The sales and tickets form a *many-to-many* relationship to the *product* dimension because one ticket can consist of more than one *product*, although every ticket is still purchased in only one *store* by one *customer* at one *time*.

The concept of applying additivity or summarizability[4-7] to measures along dimensions is crucial to multidimensional data modeling. A measure is additive along a dimension if we can use the SUM operator to aggregate attribute values along all hierarchies defined on that dimension. The aggregation of some fact attributes—called roll-up in OLAP terminology—might not, however, be semantically meaningful for all measures along all dimensions.

In our example, *number of clients*—estimated by counting the number of purchase receipts for a given *product*, *customer*, *day*, and *store*—is not additive along the *product* dimension. Because the same ticket can include other *products*, adding up the *number of clients* for two or more *products* would lead to inconsistent results. However, other aggregation operators—such as SUM, AVG, and MIN—could be applied to other dimensions, such as *time*.

Defining the classification hierarchies of certain dimension attributes is crucial because these classification hierarchies provide the basis for the subsequent data analysis. Because a dimension attribute can also be aggregated to more than one other attribute, multiple classification hierarchies and alternative path hierarchies are also relevant. For this reason, directed acyclic graphs provide a common way of representing and analyzing dimensions with their classification hierarchies.

Figure 1b shows the different classification hierarchies defined for the *product*, *store*, and *time* dimensions. On the *product* dimension, we have defined a multiple classification hierarchy so that we can aggregate data values along two different hierarchy paths:

- *product–type–family–group* and
- *product–brand*.

Other attributes, not used for aggregating purposes, can provide features for other dimension attributes, such as *product name*. For the *store* dimension, we have defined an alternative path classification hierarchy with two different paths that converge into the same hierarchy level:

- *store–city–province–state* and
- *store–sales_area–state*.

Finally, we have also defined another alternative path classification hierarchy with the following paths for the *time* dimension:

- *time–month–semester–year* and
- *time–season*.

In most cases, however, classification hierarchies are not so simple. The concepts of *strictness* and *completeness* are important for both conceptual purposes and for further multidimensional modeling design

Table 1. Comparison of conceptual multidimensional models.

Multidimensional modeling properties	Model		
	DF	M/ER	StarEr
Structural level			
Facts			
Many-to-many relationships with particular dimensions	No	No	Yes
Atomic measures	Yes	Yes	Yes
Derived measures	No	No	No
Additivity	Yes	No	Yes
Dimensions			
Multiple and alternative path classification hierarchies	Yes	Yes	Yes
Nonstrict classification hierarchies	No	No	Yes
Complete classification hierarchies	No	No	Yes
Categorization of dimensions	No	Yes	Yes
Dynamic level			
Specifying initial user requirements	Yes	Yes	No
OLAP operations	No	Yes	No
Modeling system behavior	No	Yes	No
Graphical notation	Yes	Yes	Yes
Automatic generation into a target OLAP commercial tool	No	Yes	No

steps.[7] We use *strictness* here to mean that an object at a hierarchy's lower level belongs to only one higher-level object. Thus, for example, a *province* can only relate to one *state*. By *completeness* we mean that all members belong to one higher-class object and that object consists of those members only. Thus, only the recorded *provinces* can form a *state*. In a "complete" classification hierarchy between the *state* and *province* levels, all the recorded *provinces* form the *state*, and all the *provinces* that form the *state* have been recorded.

OLAP scenarios sometimes become extensive as the number of dimensions increases significantly, a trend that can lead to extremely sparse dimensions and data cubes. In such a scenario, some attributes are normally valid for all elements within a dimension while others are only valid for a subset of elements, known as the *categorization of dimensions*.[7,8] For example, the attributes *alcohol percentage* and *volume* would only be valid for *drink products* and would be null for *food products*. A proper multidimensional data model should consider attributes only when necessary, depending on the categorization of dimensions.

Once developers define the multidimensional model structure, users can define a set of initial requirements as a starting point for the subsequent data-analysis phase. From these initial requirements, users can apply a set of OLAP operations[6,9] to the multidimensional view of data for further data analysis. These OLAP operations usually include the following:

- *roll-up*, which increases the level of aggregation along one or more classification hierarchies;
- *drill-down*, which decreases the level of aggregation along one or more classification hierarchies;
- *slice-dice*, which selects and projects the data; and
- *pivoting*, which reorients the multidimensional data view to allow exchanging dimensions for facts symmetrically.

RELATED WORK

The well-known star schema by Ralph Kimball is characterized as a logical multidimensional data model.[6] Other data models provide a formalism for assessing multidimensional properties. However, we will only mention the most relevant conceptual multidimensional models. These models provide a high level of abstraction for assessing multidimensional properties. These models include the Dimensional-Fact (DF) model by Matteo Golfarelli and colleagues,[5] the Multidimensional/Entity-Relationship (M/ER) model by Carsten Sapia and colleagues,[10,11] and the StarEr model by Nectaria Tryfona and colleagues.[7]

Table 1 shows the multidimensional properties of these three types of conceptual models. Only the StarEr model considers the *many-to-many* relationships between facts and particular dimensions by indicating the exact cardinality between them. None of the models includes derived measures or their derivation rules as part of their conceptual schema.

The DF and StarEr models explicitly represent the set of aggregation operators that can be applied to nonadditive measures. With reference to dimensions, all three models use directed acyclic graphs (DAGs) to define certain dimension attributes for multiple and alternative path classification hierarchies. However, only the StarEr model specifies the exact cardinality levels for nonstrict and complete classification hierarchies. As both the M/ER and the StarEr models derive from the Entity-Relationship model, they use *is-a* relationships to categorize dimensions.

Only the StarEr model lacks an explicit mechanism for representing initial user requirements for dynamic

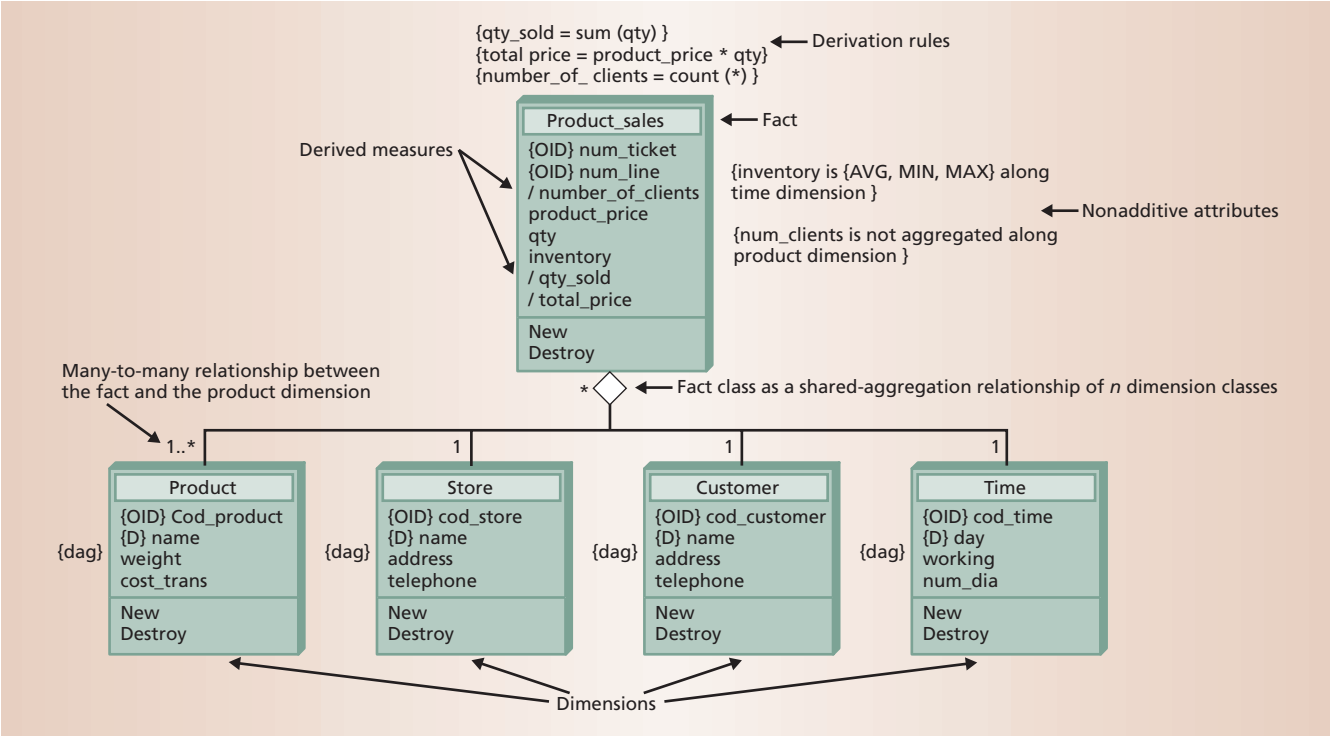


Figure 2. A fact class as a shared-aggregation relationship of n dimension classes. The* Product_sales *class—which has derivation rules, derived measures, and nonadditive attributes—has a shared-aggregation relationship with the* product, store, customer, *and* time *dimensions.

multidimensional modeling. Only the M/ER model uses state diagrams to model the system's behavior and provide a set of basic OLAP operations to be applied from these initial user requirements.

All the models provide a graphical notation that helps designers perform conceptual modeling. Only the M/ER model, however, provides a framework for automatically generating the database schema into a target OLAP commercial tool, either Informix MetaCube or Cognos PowerPlay.

None of the conceptual modeling approaches considers all multidimensional properties at both the structural and dynamic levels. However, an OO approach can elegantly represent multidimensional properties at both levels.

OO CONCEPTUAL MODELING APPROACH

Our approach uses a UML class diagram to specify the structure of a multidimensional model.

Structural level

This OO approach is not restricted to using flat UML class diagrams to model large, complex data warehouse systems. UML's *package* grouping mechanism groups classes into higher-level units, creating different levels of abstraction and simplifying the final model. In this way, a UML class diagram improves and simplifies the system specifications created with classic semantic data models such as the Entity-Relationship model. Further, OCL expressions can embed operations and constraints in the class diagram. Our approach clearly separates the structure of a multidimensional model specified with a UML class diagram into facts and dimensions.

Facts and dimensions

Fact classes represent facts and the measures we are interested in, defined as attributes within these classes. *Dimension classes* represent dimensions.

We then consider fact classes as composite classes in a shared-aggregation relationship of *n* dimension classes. We define the minimum cardinality of dimension-class roles as 1 to indicate that a fact object instance is always related to object instances from all dimensions. We define the fact class role's cardinality as * to indicate that a dimension object can be part of one, zero, or more fact object instances.

Figure 2 shows the *Product_sales* fact class—which consists of tickets—and the dimension classes *Product*, *Store*, *Customer*, and *Time*. The fact class is thus specified as a shared-aggregation relationship between all dimension classes.

Derived measures

We can also explicitly consider *derived measures* by placing the constraint /next to a measure in the fact class. Derivation rules appear between braces. In Figure 2, three derived measures—*number_of_clients, qty_sold* and *total_price*—have been defined and their derivation rules placed between braces atop the fact class.

Many-to-many relationships

Thanks to the flexibility of shared-aggregation relationships that UML provides, we can consider *many-to-many* relationships between facts and particular dimensions. We do so by indicating the 1..* cardinality on the dimension class role to show that a fact object instance can be related to one or more dimen-

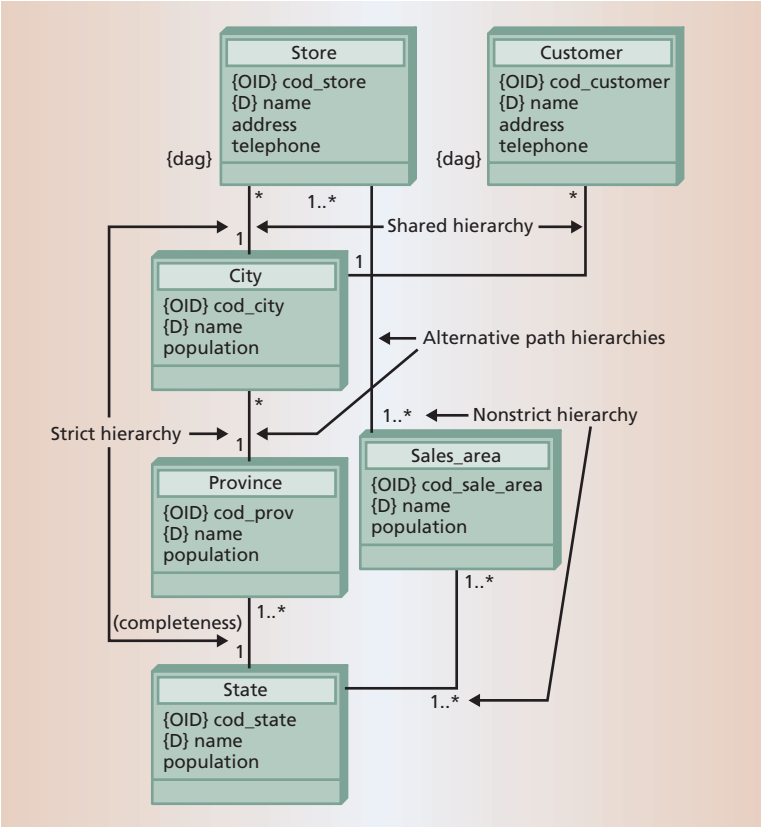


Figure 3. Multiple classification hierarchy that includes shared, strict, complete, alternative-path, and nonstrict hierarchies, with class methods omitted for clarity.

sion object instances. In Figure 2, the 1..* cardinality on the *Product* dimension class role in the shared-aggregation relationship indicates an instance of a *Product_sales* fact object—one ticket—that can relate to more than one *product*.

Our approach also lets us define identifying attributes that can be defined in the fact class, if convenient, by placing the constraint {*OID*} next to a measure name, also known as degenerate dimensions[12]—dimensions whose identifiers exist in a fact table only, and not as actual dimensions. This approach provides other fact features in addition to the measures for analysis. In our example, we could store the ticket and line numbers as other ticket features.

These {*OID*} attributes can also be used for automatic generation of the database schema into a target relational commercial OLAP tool, mainly if a *many-to-many* relationship exists between the fact class and a particular dimension class. In this case, the designer may want to use these {*OID*} attributes to identify fact instances in a relational database instead of creating bridge tables.

The identifying attributes *num_ticket* and *num_line* correspond to the ticket and line numbers of a physical ticket. In an automatic generation process, these attributes could be used to unambiguously distinguish the ticket line that a sold product belongs to.

Additivity

By default, we consider all measures additive: The SUM operator can be applied to aggregate measure values along all dimensions. Nonadditivity defines constraints on measures between braces and places them somewhere around the fact class. They have formal underlying formulas[2] and contain the allowed operators, if any, along the dimension for which the measure is not additive. For clarity, the UML notation represents these constraints in a property tag.

For simplicity and clarity, the class diagram contains the additivity rules and derivation rules for derived attributes. OCL expressions can embed both kinds of rules in a large data warehouse system, thereby avoiding cluttered class diagrams.

Classification hierarchies

For dimensions, a base class represents every classification hierarchy level. An association of classes specifies the relationships between two levels of a classification hierarchy. The only required constraint is that the classes used for defining a classification hierarchy along a dimension must define a DAG rooted in the dimension class. The DAG structure can represent both *alternative path* and *multiple classification hierarchies*. Thus, a class B of a hierarchy is considered an association of a class A. Placing the constraint *{dag}* next to every dimension class in the UML class diagram specifies that any class within a classification hierarchy must define a DAG.

Figure 3 shows a classification hierarchy specified along the *store* and *customer* dimensions that omits class methods for clarity. The constraint {*dag*} has been placed next to the *Store* and *Customer* dimension classes. For simplicity, the *store* and *customer* dimensions share a classification hierarchy path.

The base classes, including the dimension class, that belong to the classification hierarchy must contain an explicitly defined identifying attribute. We do this by placing the constraint {*OID*} next to one attribute in every class. This attribute is necessary to automatically generate the database schema from the UML class diagram into a target relational OLAP tool because these tools store the attribute in their metadata to unambiguously identify every instance of a classification hierarchy level.

Relational commercial OLAP tools, however, use a default attribute within every classification hierarchy level that will be used in the subsequent data analysis phase. A *default* is a dimensional attribute that users want to analyze with a target commercial OLAP tool. This default attribute displays every time the user

applies an OLAP operation, rather than having the tool prompt the user to specify which attribute to display before executing each operation.

For example, a commercial relational OLAP tool can define the *name* attribute of the *city* hierarchy level as the default attribute in the *store* dimension. Applying a roll-up operation to aggregate measure values from the *store* hierarchy level into the *city* level would analyze the city names where the *products* were sold because *city* hierarchy level defines the *city name* attribute as the default.

Therefore, to plan for subsequent automatic generation into a target relational OLAP tool, we must qualify the default attribute for every hierarchy level in our UML class diagram. We call a default attribute a *descriptor* because we consider the term "default" too general. Thus, we define a descriptor in every class that represents a classification hierarchy level, thereby indicating that the commercial OLAP tool will use this as the default attribute. We do this by defining the constraint {*D*} next to an attribute. Finally, we can define the default and identifying attributes simultaneously. In Figure 3, both the identifying {*OID*} and descriptor {*D*} attributes have been defined for every class.

Strictness and completeness

The multiplicity 1 and 1..* defined in the target associated class role address the concepts of *strictness* and *nonstrictness*. Defining the {*completeness*} constraint in the target associated class role addresses the completeness of a classification hierarchy. By default, our approach considers all classification hierarchies noncomplete.

In Figure 3, adding the constraint {*completeness*} to the *state* associated class role in the association to the *province* class indicates that a *state* object instance consists of only those *province* object instances and no others. Further, the *store* and *sales_area* classes form a nonstrict association because a *sales_area* object instance can refer to more than one *state* object instance. Moreover, if the multiplicity from *sales_area* to *state* were * instead of 1..*, *sales_areas* could be related to no *states*, forming a noncovering relationship.[13]

Categorizing dimensions

In some cases, however, a multidimensional conceptual model should consider the categorization of dimensions to model additional features for an entity's subtypes. Our approach uses a generalization-specialization relationship to categorize entities that contain subtypes. This approach imposes the important constraint that no class other than the dimension class can belong to both a classification and specialization hierarchy at the same time.

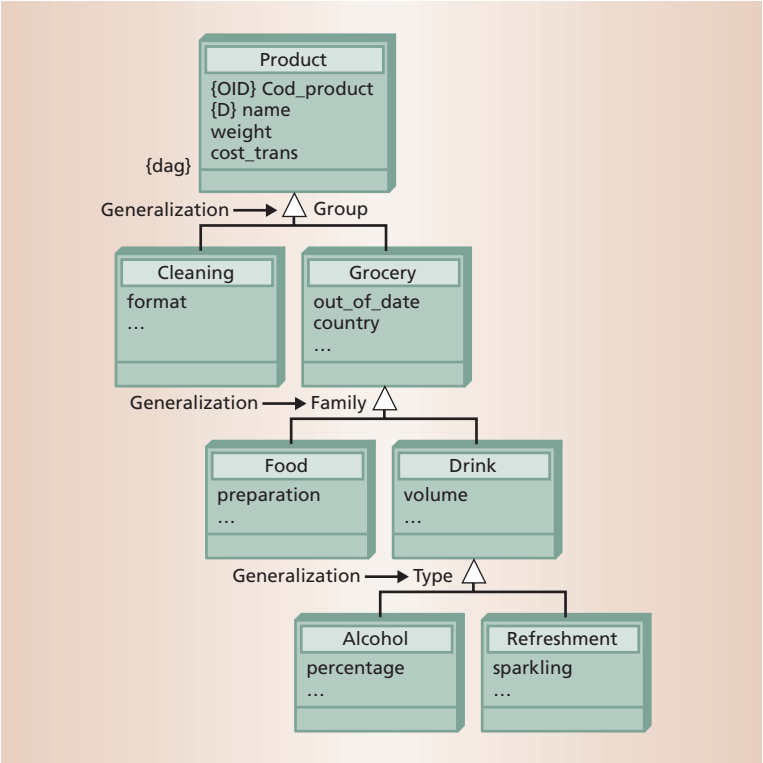


***Figure 4. The* Product *dimension class, modeled by defining its different* subtypes—Group, Family, *and* Type.**

In Figure 4, the *Product* dimension class has been modeled depending on the different subtypes—*Group*, *Family*, and *Type*—considered in the system.

Dynamic level

We use *cube classes* to represent initial user requirements as the starting point for the subsequent data-analysis phase. A UML-compliant class notation properly and easily defines these classes. The basic components of the cube classes include the

- head area, which contains the cube class's name;
- measures area, which contains the measures to be analyzed;
- slice area, which contains the constraints to be satisfied;
- dice area, which contains the dimensions and their grouping conditions to address the analysis; and
- cube operations, which cover the OLAP operations for a further data-analysis phase.

Consider the following initial user requirement for the multidimensional model, as specified by a UML class diagram: We want to analyze the quantity of products sold *where* the group of products is "Gro-

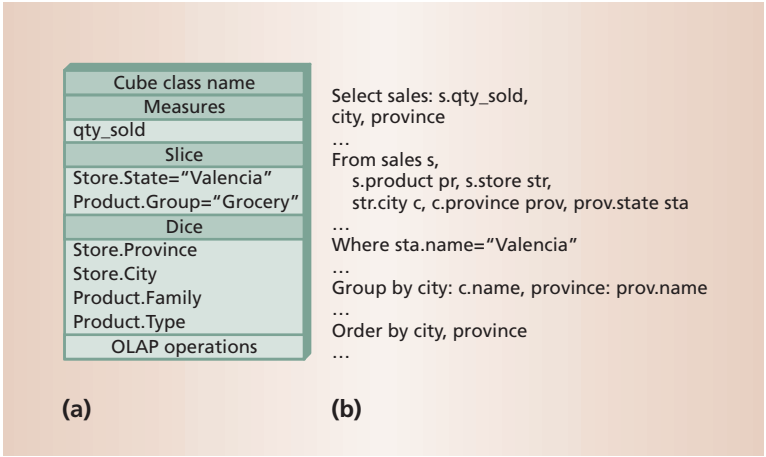


Figure 5. (a) Cube class example with parameters specified in the* measures, slice, dice, *and* operations *areas; and (b) the class's corresponding Object-Query Language specification.

cery" and the *store_state* is "Valencia," *grouped* according to the *product family* and *type* and the *store province* and *city*. Figure 5 shows both the graphical notation of the cube class that corresponds to this requirement and its accompanying Object Query Language (OQL) specification. Figure 5a shows that the measure area specifies the measure to be analyzed, *qty_sold*. Constraints defined on dimension classification hierarchy levels—*group* and *state*—appear in the slice area, and the classification hierarchy levels for which we want to analyze measures—*family*, *type*, *province* and *city*—appear in the dice area. Finally, the cube operations section specifies the available OLAP operations.

For nonexpert UML or database users, the cube class's graphical notation facilitates the definition of initial user requirements. Every cube class has a more formal underlying OQL specification. Experts can use OQL to define cube classes by specifying the appropriate OQL sentences. Figure 5b shows how an OQL syntax could specify the cube class example in Figure 5a.

Behavioral properties

OO analysis and design techniques can elegantly link a system's structural and behavioral properties at the conceptual level. In our OO approach, behavioral properties mainly relate to cube classes that represent initial user requirements. Users can apply certain OLAP operations in the further-data-analysis phase to start a navigational process. These operations are close as they generate another cube class as an output.

UML's state and interaction diagrams can model the behavioral evolution of these cube classes based on the applied OLAP operation.[2,3] These diagrams contain information about the most probable evolution of the final user requirements from the specified user requirement. OLAP designers can use the information these diagrams contain to predict user behavior, helping them design a proper view-maintenance policy.

To specify that certain OLAP operations lead users to cube classes that analyze the same data in different ways, we define one *state diagram* for each initial cube class. In these diagrams, we consider as a valid state every classification hierarchy level defined on a dimension included in the dice area. Each of these valid states will form a new cube class. For example, consider the cube class definition in Figure 5a. Users can apply roll-up and drill-down operations on the classification hierarchy levels defined on the *store* and *product* dimensions to identify a valid state of that cube class. Therefore, an OLAP designer can observe the most visited state and provide the corresponding views that answer it.

We can also define an *interaction diagram* for each UML class diagram. In our approach, we have adopted sequence diagrams[1] for their clarity and simplicity. The interaction diagram shows interactions among cube classes changed by OLAP operations such as rotate, pivot, slice, or dice. Certain OLAP operations can lead users to cube classes that will show completely different data. These cube classes represent the most probable new requirements a final user may want to execute. Thus, just as with the state diagram, an OLAP designer can define a corresponding view for each of these new cube classes in the interaction diagram.

CASE TOOL

The computer-assisted software engineering (CASE) environment's most significant contribution to our work is its ability to implement both the structural and dynamic levels of the conceptual model into a target commercial OLAP tool. Different implementation issues imposed by different commercial OLAP tools—as well as the particular issues to be tackled in data warehouse conceptual modeling—have prompted us to design our own CASE tool instead of extending another one.

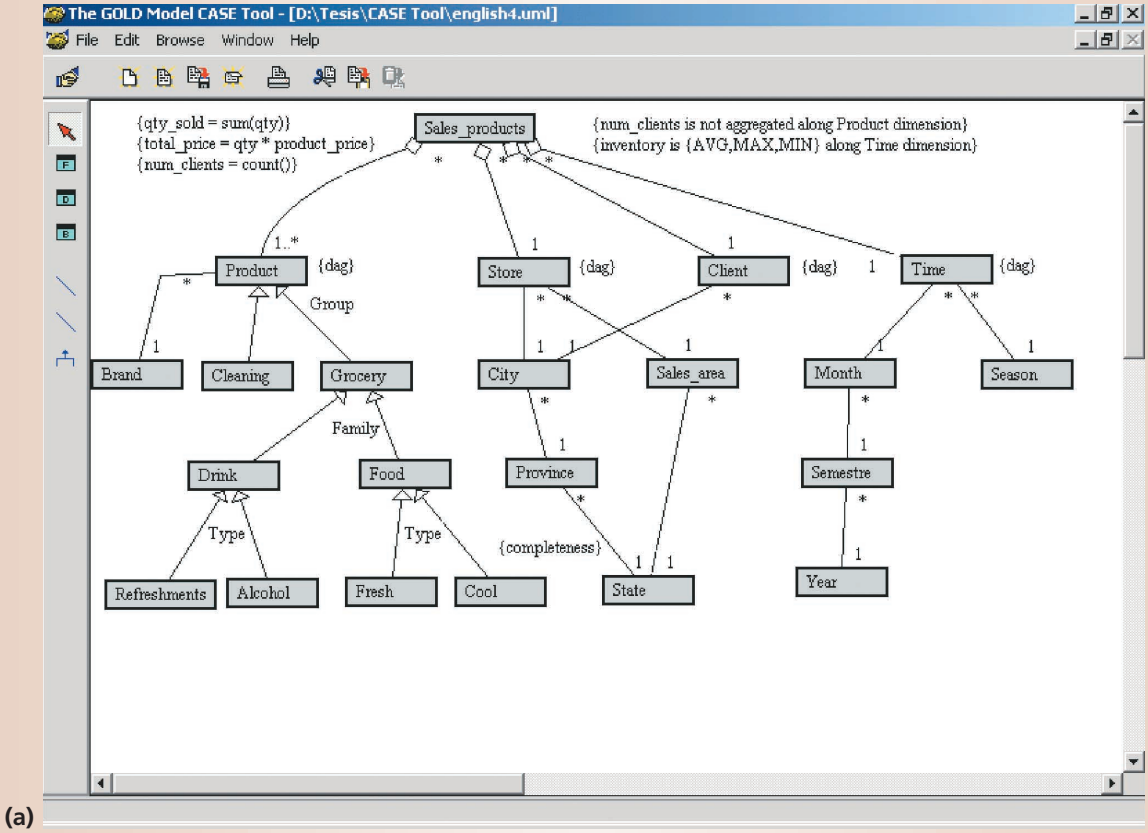
We designed the GOLD Model CASE Tool to provide an operational environment that supports our conceptual modeling approach. The tool provides a comfortable interface for elaborating data warehouse conceptual designs independently of implementation issues. Figure 6 consists of two screens from the tool:

- Figure 6a shows a design screen from the CASE tool that displays the structural aspects of a multidimensional model. Class attributes and methods have been hidden for the sake of simplicity.
- Figure 6b shows the window that lets the tool's operator see initial user requirements and their corresponding state diagrams.

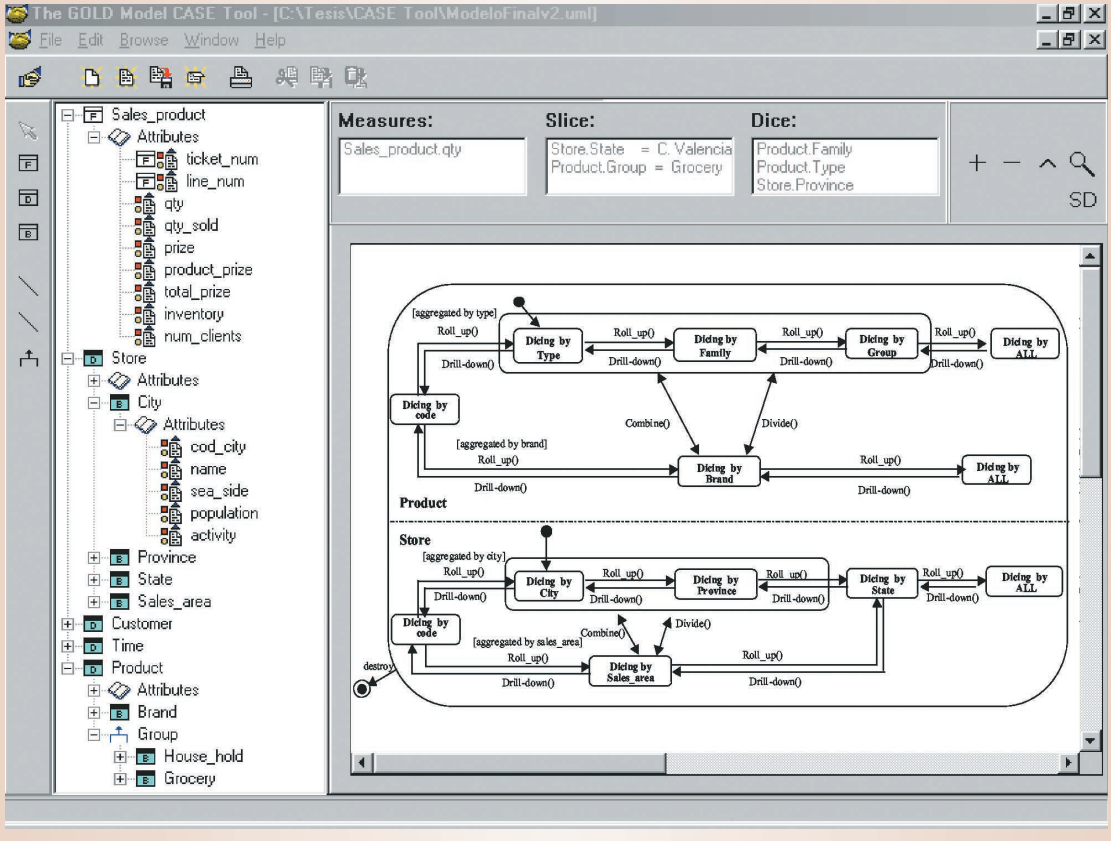
In the second screen, 6b, the structural aspects of the underlying multidimensional model appear as a tree structure on the window's left side.

AUTOMATIC GENERATION INTO AN OLAP TOOL

OLAP tools implement a multidimensional model from two different levels:



(a)



(b)

Figure 6. The GOLD Model CASE Tool. The operational environment supports a conceptual modeling approach: (a) A design screen displays the structural aspects of a multidimensional model; (b) the tool's operator can view initial user requirements and the corresponding state diagrams.

```
create table Store (
  ID0 integer primary key,
  cod_store_ID char(10),
  name char(10),
  address char(10),
  City_cod_city_ID integer,
  City_name char(10),
  City_population integer,
  Province_cod_province_ID integer,
  Province_name char(10),
  Province_population integer,
  State_cod_state_ID integer,
  State_name char(10),
  State_population integer,
  Sales_area_cod_sales_ID integer,
  Sales_area _name char(10),
  Sales_area _population integer,
  level_attribute integer );
```

```
insert into dim
(dim_id,dss_system_id,dim_desc,dim_type,dim_squema_name,dim_table_name,
dim_to_fact_key,agg_level_col)
values (1, 1, 'Store', 0, 'user name','Store','cod_store_ID', 'level_attribute');

insert into dim_el
(dim_el_id,dss_system_id,dim_id,dim_el_desc,dim_to_att_key,att_squema_name,
att_table_name, att_to_dim_key,base_element,agg_level)
values (1, 1, 1, 'Store', 'cod_store_ID','user name','Store','ID0','N', 1);

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_col_name, default_flag, att_col_type,
sort_column) values (1, 1, 1, 'cod_store_ID', 'cod_store_ID', 'N', 1, 'cod_Store_ID');

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_col_name, default_flag, att_col_type,
sort_column) values (2, 1, 1, 'name', 'name', 'Y', 1, 'name');
```

Figure 7. SQL sentences for defining star schema tables and multidimensional issues in the Informix MetaCube Decision Support System.

- *Structural*—the structures that form the database schema for housing multidimensional data and the underlying multidimensional model—also known as the metadata—that provides the model's key semantics, such as facts, measures, and dimensions.
- *Dynamic*—refers to the definition of final user requirements and OLAP operations for further analyzing data.

Each commercial OLAP tool provides its own model for assessing multidimensional modeling's main semantics and concepts. Consequently, different OLAP tools focus on different semantics and properties. Ideally, a proper multidimensional design uses a conceptual approach totally independent of implementation concerns, and developers generate the model's implementation directly into a commercial OLAP tool.[14] We chose the Informix MetaCube's (http://www.informix.com) relational OLAP tool as being representative of our process, but it can be applied to any commercial OLAP tool. The multidimensional model provided by the Informix MetaCube to consider multidimensional properties—a Decision Support System (DSS)—is implemented in relational tables.

At the structural level, our process first generates the star schema that will house the multidimensional data, then it generates the corresponding multidimensional information in DSS format from the modeling constructors used in the conceptual design. However, some constructors lack a corresponding Informix MetaCube representation, thus we ignore some and transform others while trying to preserve their initial semantics. Specifically, we ignore or transform the following semantic properties:

- Nonstrict and complete classification hierarchies convert to strict classification hierarchies, the only type Informix MetaCube considers.
- We ignore the defined additivity rules because Informix MetaCube does not store any information related to the additivity of measures.
- Specialization hierarchies transform into strict classification hierarchies.
- To transform *many-to-many* relationships between the fact class and any particular dimension class, we convert every {*OID*} attribute defined in the fact class into a new dimension with only one hierarchy level and one attribute, the proper {*OID*} attribute. In our example, *num_ticket* will be transformed into a new dimension.

At the dynamic level, every cube class our conceptual design specifies—initial user requirements, state, and interaction diagrams—translates into Informix MetaCube requirements. Thanks to this conversion, the final user can use the relational OLAP tool to load the initial user requirements specification in the subsequent data-analysis phase. Figure 7 shows an example of the SQL sentences contained in the scripts our CASE tool generates.

In Figure 7's left column, the SQL sentences that define the relational table correspond to the *store* dimension. All attributes defined in the different classification hierarchy classes are defined as attributes within the same table. Figure 7's right column shows an example of the SQL sentences that define multidimensional issues in the DSS model. The CASE tool registers the *store* dimension in the *dim* table, then defines the *store* base hierarchy level in the *dim_el* table and, finally, registers attributes within this hierarchy level in the *att* table.

Currently, we're working on the automatic implementation of a multidimensional model derived from our approach to using object-oriented and object-relational databases for data ware-

house and OLAP applications. We are also investigating identification of materialized views by developing state and interaction diagrams from cube classes. Further, we plan to integrate commercial OLAP tool facilities within our GOLD Model Case Tool, a task that involves data warehouse prototyping and sample data generation issues. ✸

Acknowledgment

We thank our colleague Cristina Cachero for her helpful comments on previous versions of this material.

References

1. G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison Longman Wesley, Reading, Mass., 1998.
2. J. Trujillo, *The GOLD Model: An Object-Oriented Conceptual Model for the Design of OLAP Applications*, doctoral dissertation, Languages and Information Systems Dept., Alicante University, Spain, June 2001.
3. J. Trujillo, J. and M. Palomar, "Modeling the Behavior of OLAP Applications Using an UML Compliant Approach," *Proc. 1st Int'l Conf. Advances in Information Systems* (ADVIS 00), vol. 1909, Lecture Notes in Computer Science, Springer-Verlag, New York, 2000, pp. 14-23.
4. J. Trujillo, M. Palomar, and J. Gómez, "Applying Object-Oriented Conceptual Modeling Techniques to the Design of Multidimensional Databases and OLAP Applications," *Proc. 1st Int'l Conf. Web-Age Information Management* (WAIM 00), vol. 1846, Lecture Notes in Computer Science, Springer-Verlag, New York, 2000, pp. 83-94.
5. M. Golfarelli, D. Maio, and S. Rizzi, "The Dimensional Fact Model: A Conceptual Model for Data Warehouses," *Int'l J. Cooperative Information Systems* (IJCIS), vol. 7, no. 2-3, 1998, pp. 215-247.
6. R. Kimball, *The Data Warehousing Toolkit*, John Wiley & Sons, New York, 1996.
7. N. Tryfona, F. Busborg, and J.G. Christiansen, "StarER: A Conceptual Model for Data Warehouse Design," *Proc. ACM 2nd Int'l Workshop Data Warehousing and OLAP* (DOLAP 99), ACM Press, New York, 1999, pp. 3-8.
8. W. Lehner, "Modelling Large-Scale OLAP Scenarios," *Proc. 6th Int'l Conference On Extending Database Technology* (EDBT 98), vol. 1377, Lecture Notes in Computer Science, Springer-Verlag, New York, 1998, pp. 153-167.
9. S. Chaudhuri and U. Dayal, "An Overview of Data Warehousing and OLAP Technology," *ACM Sigmod Record*, vol. 26, no. 1, 1997, pp. 65-74.
10. C. Sapia, "On Modeling and Predicting Query Behavior in OLAP Systems," *Proc. Int'l Workshop on Design and Management of Data Warehouses* (DMDW 99), Swiss Life, Switzerland, 1999, pp. 1-10.
11. C. Sapia et al., "Extending the E/R Model for the Multidimensional Paradigm," *Proc. 1st Int'l Workshop on Data Warehousing and Data Mining* (DWDM 98), vol. 1552, Lecture Notes in Computer Science, Springer-Verlag, New York, 1998, pp. 105-116.
12. W. Giovinazzo, *Object-Oriented Data Warehouse Design: Building a Star Schema*, Prentice-Hall, Upper Saddle River, N.J., 2000.
13. T.B. Pedersen and C.S. Jensen, "Multidimensional Data Modeling of Complex Data," *Proc. 15th IEEE Int'l Conf. Data Eng.* (ICDE 99), IEEE CS Press, Los Alamitos, Calif., 1999, pp. 336-345.
14. K. Hahn, C. Sapia, and M. Blaschka, "Automatically Generating OLAP Schemata from Conceptual Graphical Models," *Proc. ACM 3rd Int'l Workshop Data Warehousing and OLAP* (DOLAP 00), ACM Press, New York, 2000, pp. 9-16.

Juan Trujillo *is a professor at the Computer Science School at the University of Alicante, Spain. His research interests include database modeling, conceptual design of data warehouses, multidimensional databases, OLAP, and object-oriented analysis and design with UML. Trujillo received a PhD in computer science from the University of Alicante, Spain. Contact him at jtrujillo@dlsi.ua.es.*

Manuel Palomar *is the head of the Language and Information Systems Department at the Computer Science School at the University of Alicante, Spain. His research interests include databases, object-oriented conceptual modeling of data warehouses, OLAP, and natural-language processing. Palomar received a PhD in computer science from the Technical University of Valencia, Spain. Contact him at mpalomar@dlsi.ua.es.*

Jaime Gomez *is a professor of software engineering at the Computer Science School at the University of Alicante, Spain. His research interests include object-oriented conceptual modeling of data warehouses, OLAP, Web engineering, model-based code generation, and component-based development. Gomez received a PhD in computer science from the University of Alicante, Spain. Contact him at jgomez@dlsi.ua.es.*

Il-Yeol Song *is a professor at Drexel University. His research interests include database modeling and design and performance optimization of data warehouses, OLAP, database support for e-commerce systems, and object-oriented analysis and design with UML. Song received a PhD in computer science from Louisiana State University, Baton Rouge. Contact him at songiy@drexel.edu.*