Journal of Database Management, Vol. 15, No.1, 2004, pp. 41-72.

Applying UML and XML for designing and interchanging information for data warehouses and OLAP applications

Juan Trujillo*, Sergio Luján-Mora* and Il-Yeol Song+

* Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante, Alicante. SPAIN
{jtrujillo,slujan}@dlsi.ua.es

\+ College of Information Science and Technology
Drexel University, Philadelphia. USA
songiy@drexel.edu

Abstract: Multidimensional (MD) modeling is the basis for Data warehouses (DW), multidimensional databases (MDB) and On-Line Analytical Processing (OLAP) applications. In this paper, we present how the Unified Modeling Language (UML) can be successfully used to represent both structural and dynamic properties of these systems at the conceptual level. The structure of the system is specified by means of a UML class diagram that considers the main properties of MD modeling with minimal use of constraints and extensions of the UML. If the system to be modeled is too complex, thereby leading us to a considerable number of classes and relationships, we describe how to use the *package* grouping mechanism provided by the UML to simplify the final model. Furthermore, we provide a UML-compliant class notation (called cube class) to represent OLAP users' initial requirements. We also describe how we can use the UML state and interaction diagrams to model the behavior of a data warehouse system. To facilitate the interchange of conceptual MD models, we provide a Document Type Definition (DTD) which allows us to represent the same MD modeling properties that can be considered by using our approach. From this DTD, we can directly generate valid eXtensible Markup Language (XML) documents that represent MD models at the conceptual level. We believe that our innovative approach provides a theoretical foundation for simplifying the conceptual design of MD systems and the examples included in this paper clearly illustrate the use of our approach.

Keywords: Data warehouses, multidimensional databases, OLAP, conceptual modeling, UML, object orientation, ODBMS, XML

1 Introduction

Multidimensional (MD) modeling is the foundation for Data warehouses (DW), multidimensional databases (MDB) and On-Line Analytical Processing (OLAP) applications. The benefit of using MD modeling is two-fold. On one hand, the MD model is close to data analyzers' way of thinking; therefore, it helps users understand data. On the other hand, the MD model supports performance improvement as its simple structure allows us to predict final users' intentions.

Some approaches have been proposed lately (presented in Section 3) to accomplish the conceptual design of these systems. Unfortunately, none of them have been accepted as a

standard for DW conceptual modeling. These proposals try to represent main MD properties at the conceptual level with special emphasis on MD data structures. A conceptual modeling approach for DW, however, should also concern other relevant aspects such as users' initial requirements, the behavior of the system (e.g. main operations to be accomplished on MD data structures), available data sources, specific issues for automatic generation of the database schema and so on. We claim that object orientation with the UML provides an adequate notation for modeling every aspect of a DW system (MD data structures, the behavior of the system, etc.) from user requirements to implementation.

We have proposed an object-oriented (OO) approach to accomplish the conceptual modeling of DW, MDB and OLAP applications that introduces a set of minimal constraints and extensions of the UML (Booch, Rumbaugh, and Jacobson, 1998) (OMG, 2001), needed for an adequate representation of MD modeling properties (Trujillo, 2001) (Trujillo, Palomar, Gómez, and Song, 2001b). These extensions are based on the standard mechanisms provided by the UML to adapt it to a specific method or model (e.g. constraints, tagged values). We have also presented how to group classes into *packages* to simplify the final model in case that the model becomes too complex due to the high number of classes (Luján-Mora, Trujillo, and Song, 2002). Furthermore, we have provided a UML-compliant class notation to represent OLAP users' initial requirements (called *cube class*). From these cube classes, we then describe the use of state and interaction diagrams to model the behavior of the system based on the applied OLAP operations (Trujillo, Palomar, and Gómez, 2000). We have also discussed issues such as identifying attributes and descriptor attributes that set the basis for an adequate semi-automatic generation of a database schema and user requirements in a target commercial OLAP tool.

The UML can also be used with powerful mechanisms such as the Object Constraint Language (OCL) (Warmer and Kleppe, 1998) (OMG, 2001) and the Object Query Language (OQL) (Cattell *et al.* 2000) to embed DW constraints (e.g. additivity and derived attributes) and users' initial requirements in the conceptual model. In this way, when we model a DW system, we can obtain simple yet powerful extended UML class diagrams that represent main MD properties at a conceptual level.

On the other hand, a salient issue these days in the scientific community and in the business world is the interchange of information. The eXtensible Markup Language (XML) (W3C, 2000) is rapidly being adopted as the standard syntax for the interchange of un-structured, semi-structured and structured data. XML is an open neutral platform and vendor independent meta-language, which allows us to reduce the cost, complexity, and effort required in integrating data within and between enterprises.

From these considerations, in this paper we present the following contributions. We believe that our innovative approach provides a theoretical foundation for the possible use of Object-Oriented Databases (OODB) and Object-Relational Databases (ORDB) for DW and OLAP applications. For this reason, we provide the representation of our approach into the standard for OODB proposed by the Object Database Management Group (ODMG) (Catell *et al.* 2000). We also believe that a relevant feature of a conceptual model should be its capability to share information in an easy and standard form. Therefore, we also present

how to represent MD models, accomplished by using our approach based on the UML, by means of the XML. In order to do this, we provide a Document Type Definition (DTD) that defines the correct structure and content of a XML document representing main MD properties. Finally, to show the benefit of our approach, we include a set of case studies to show the elegant way in which our proposal represents both structural and dynamic properties of MD modeling.

The remainder of this paper is organized as follows: Section 2 details the major features of MD modeling that should be taken into account for a proper MD conceptual design. Section 3 summarizes the most relevant conceptual approaches proposed so far by the research community. In Section 4, we present how we use the UML to consider main structural and dynamic MD properties at the conceptual level. We also present how to facilitate the interchange of MD models by generating the corresponding standard provided by the ODMG and the DTD from UML. In Section 5, we present a set of case studies taken from Kimball (Kimball and Ross, 2002) to show the benefit of our approach. Finally, Section 6 draws conclusions and sketches out new research that is currently being investigated.

2 Multidimensional modeling

In MD modeling, information is structured into **facts** and **dimensions**. A fact is an item of interest for an enterprise, and is described through a set of attributes called **measures** or **fact attributes** (**atomic** or **derived**), which are contained in cells or points in the data cube. This set of measures is based on a set of dimensions that determine the granularity adopted for representing facts (i.e. the context in which facts are to be analyzed). Moreover, dimensions are also characterized by attributes, which are usually called **dimension attributes**. They are used for grouping, browsing, and constraining measures.

Let us consider an example in which the fact is the product sales in a large store chain and the dimensions are as follows: product, store, customer and time. On the left hand side of Figure 1, we can observe a data cube typically used for representing an MD model. In this particular case, we have defined a cube for analyzing measures along the product, store and time dimensions.

We note that a fact usually represents a *many-to-many* relationship between any of two dimensions. For example, a product is sold in many stores and a store sells many products. We also assume that there is a *many-to-one* relationship between a fact and each particular dimension. For example, for each store there are many sale tickets, but each sale ticket belongs to only one store.

Nevertheless, there are some cases in which a fact may be associated with a particular dimension as a ***many-to-many*** relationship. For example, the fact product_sales is considered as a particular *many-to-many* relationship to the product dimension, as one ticket may consist of more than one product even though every ticket is still purchased in only one store by one customer and at one time.

With reference to measures, the concept of **additivity** or summaribility ((Blaschka, Sapia, Höfling, and Dinter, 1998), (Golfarelli, Maio, and Rizzi, 1998), (Kimball and Ross, 2002), (Trujillo *et al.*, 2001b), (Tryfona, Busborg, and Christiansen, 1999)) on measures along dimensions is crucial for MD data modeling. A measure is *additive* along a dimension if the SUM operator can be used to aggregate attribute values along all hierarchies defined on that dimension. The aggregation of some fact attributes (roll-up, in OLAP terminology), however, might not be semantically meaningful for all measures along all dimensions. A measure is *semi-additive* if the SUM operator can be applied to some dimensions, but not all the dimensions. A measure is *non-additive* if the SUM operator cannot be applied to any dimension. In our example, number of clients (estimated by counting the number of purchased receipts for a given product, day and store) is not additive along the product dimension. Since the same ticket may include other products, adding up the number of clients along two or more products would lead to inconsistent results. However, other aggregation operators (e.g. SUM, AVG and MIN) could still be used along other dimensions such as time. Thus, number of clients is semi-additive. Finally, examples of non-additive measures would be those measures that record a static level such as inventory financial account balances or measures of intensity such as room temperatures (Kimball and Ross, 2002).

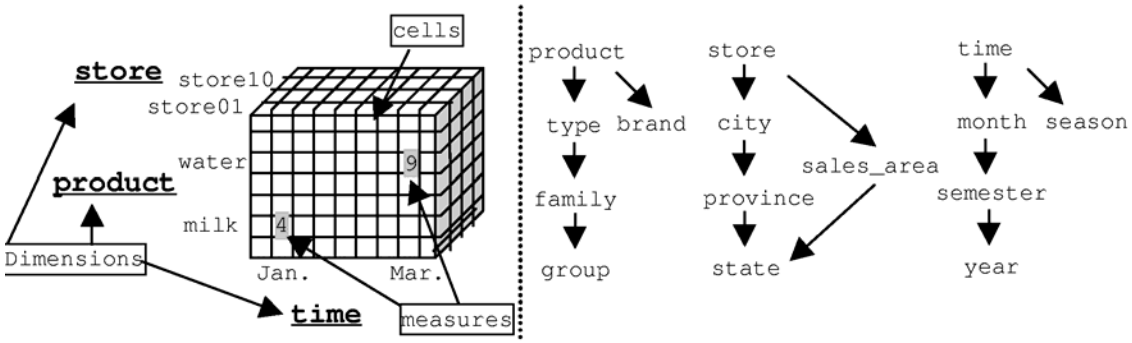

Figure 1: A data cube and classification hierarchies defined on dimensions

Regarding dimensions, the **classification hierarchies** defined on certain dimension attributes are crucial because the subsequent data analysis will be addressed by these classification hierarchies. A dimension attribute may also be aggregated (related) to more than one hierarchy. Therefore, **multiple classification hierarchies** and **alternative path hierarchies** are also relevant. For this reason, a common way of representing and considering dimensions with their classification hierarchies is by means of Directed Acyclic Graphs (DAG).

On the right hand side of Figure 1, we can observe different classification hierarchies defined on the product, store and time dimensions. On the product dimension, we have considered a multiple classification hierarchy to be able to aggregate data values along two different hierarchy paths: (i) product, type, family, group and (ii) product, brand. On the other hand, we can also find attributes that are not used for aggregating purposes, instead they provide features for other dimension attributes (e.g. product name). On the store dimension, we have defined an alternative classification hierarchy with two different paths that converge into the same hierarchy level: (i) store, city, province, state and (ii) store,

sales_area, state. Finally, we have defined another multiple classification hierarchy with the following paths on the time dimension: (i) time, month, semester, year and (ii) time, season.

Nevertheless, classification hierarchies are not so simple in most cases. The concepts of **strictness** and **completeness** are quite important, not only for conceptual purposes, but also for further design steps of MD modeling (Tryfona *et al.*). *Strictness* means that an object of a lower level in a hierarchy belongs to *only* one in a higher level, e.g. a province is only related to one state. *Completeness* means that all members belong to one higher-class object which consists of those members only. For example, suppose that the classification hierarchy between the state and province levels is "complete". In this case, a state is formed by *all* the provinces recorded and all the provinces that form the state are recorded.

OLAP scenarios sometimes become very large as the number of dimensions increases significantly, which may then lead to extremely sparse dimensions and data cubes. In this way, there are some attributes that are normally valid for all elements within a dimension while others are only valid for a subset of elements (also known as the **categorization of dimensions** ((Lehner, 1998), (Tryfona *et al.*)). For example, attributes alcohol percentage and volume would only be valid for drink products and will be "null" for food products. Thus, a proper MD data model should be able to consider attributes only when necessary, depending on the categorization of dimensions.

Furthermore, let us suppose that apart from a high number of dimensions (e.g. 20) with their corresponding hierarchies, we have a considerable number of facts (e.g. 8) sharing dimensions and classification hierarchies. This system will lead us to a very complex design, thereby increasing the difficulty in reading the modeled system. To avert a convoluted design, an MD conceptual model should also provide techniques to **avoid flat diagrams**, allowing us to group dimensions and facts to simplify the final model.

Once the structure of the MD model has been defined, OLAP users usually 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 operations (usually called OLAP operations) (Chaudhuri and Dayal, 1997) to the MD view of data for further data analysis. These OLAP operations are usually as follows: *roll-up* (increasing the level of aggregation) and *drill-down* (decreasing the level of aggregation) along one or more classification hierarchies, *slice-dice* (selection and projection) and *pivoting* (re-orienting the MD view of data which also allows us to exchange dimensions for facts; i.e. **symmetric** treatment of facts and dimensions).

2.1 Star Schema

In this sub-section, we will summarize the star schema popularized by Kimball (Kimball and Ross, 2002), as it is the most well-known schema representing MD properties in relational databases.

Kimball claims that the star schema and its variants fact constellations schema and the snowflake schema are logical choices for MD modeling to be implemented in relational

systems. We will briefly introduce this well-known approach using Sales Dimensional Model.

Figure 2 shows an example of Kimball's Sales Dimensional Model. In this model, the fact is the name of the middle box (Sales fact table). Measures are the non-foreign keys in the fact table (dollars_sold, units_sold, and dollars_cost). Dimensions are the boxes connected to the fact table in a one-to-many relationship (Time, Store, Product, Customer, and Promotion). Each dimension contains relevant attributes: day_of_week, week_number, and month in Time; store_name, address, district, and floor_type in Store, and so on.

From Figure 2, we can easily see that there are many MD features that are not reflected in the Dimensional Model: Which are the classification hierarchies defined on dimensions? Can we use all aggregation operators on all measures along all dimensions? What are these classification hierarchies like (non-strict, strict, and complete)? And many more properties. Therefore, we argue that for a proper DW and OLAP design, a conceptual MD model should be provided to better reflect user requirements. This conceptual model could then be translated into a logical model for a later implementation. In this way, we can be sure that we are analyzing the real world as users perceive it.

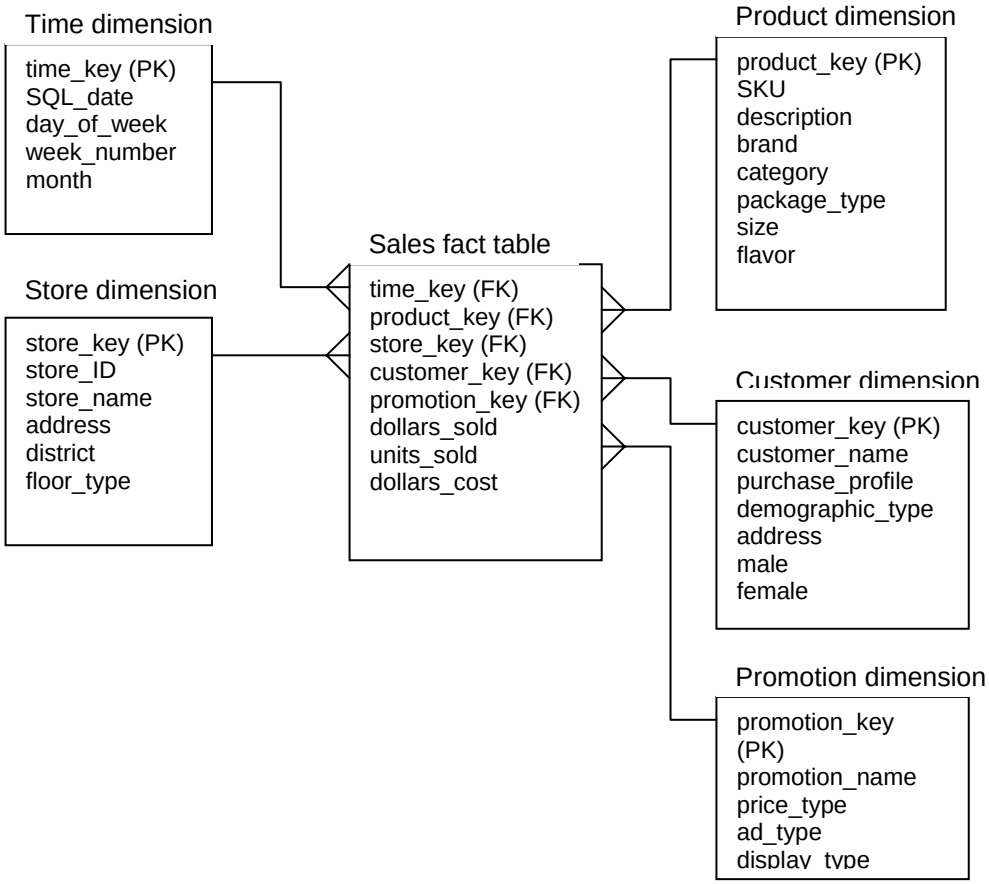

Figure 2: Sales Dimensional Model

3 Related work

Lately, several MD data models have been published. Some of them fall into the logical level (such as the well-known star-schema by Kimball (Kimball and Ross, 2002). Others may be considered as formal models, as they provide a formalism to consider main MD properties. A review of the most relevant logical and formal models can be found in Blaschka *et al.* and Abello, Samos and Saltor (2001).

In this section, we will only briefly make reference to the most relevant models that we consider "pure" conceptual MD models. These models provide a high level of abstraction for the main MD modeling properties presented in Section 2 and are totally independent from implementation issues. These are as follows: The Dimensional-Fact (DF) model by Golfarelli *et al.* (1998), The Multidimensional/ER (M/ER) model by Sapia, Blaschka, Höfling, and Dinter (1998) and Sapia (1999) and The starER model by Tryfona *et al.*.

In Table 1, we provide the coverage degree of each above-mentioned conceptual model regarding the main MD properties described in the previous section. To start with, to the best of our knowledge, no proposal provides a grouping mechanism to avoid flat diagrams and to simplify the conceptual design when a system becomes complex due to a high number of dimensions and facts sharing dimensions and their corresponding hierarchies. Regarding facts, only the starER model considers *many-to-many* relationships between facts and particular dimensions by indicating the exact cardinality (multiplicity) between them. None of them consider derived measures or their derivation rules as part of the conceptual schema. The DF and the starER models consider the additivity of measures by explicitly representing the set of aggregation operators that can be applied on non-additive measures. With reference to dimensions, all of the models consider multiple and alternative path classification hierarchies by means of Directed Acyclic Graphs (DAG) defined on certain dimension attributes. However, only the starER model considers non-strict and complete classification hierarchies by specifying the exact cardinality between classification hierarchy levels. As both the M/ER and the starER models are extensions of the Entity Relationship (ER) model, they can easily consider the categorization of dimensions by means of *Is-a* relationships.

Multidimensional modeling properties	Model		
	DF	M/ER	StarEr
Structural level			
Grouping mechanism	No	No	No
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 users' initial 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

Table 1: Comparison of conceptual multidimensional models

With reference to the dynamic level of MD modeling, the starER model is the only one that does not provide an explicit mechanism to represent users' initial requirements. On the other hand, only the M/ER model provides a set of basic OLAP operations to be applied from these users' initial requirements, and it models the behavior of the system by means of state diagrams.

We note that all the models provide a graphical notation that facilitates the conceptual modeling task to the designer. On the other hand, only the M/ER model provides a framework for an automatic generation of the database schema into a target commercial OLAP tool (particularly into Informix Metacube and Cognos Powerplay).

Finally, none of the proposals from Table 1 provide a mechanism to facilitate the interchange of the models following standard representations. Regarding MD modeling and the eXtensible Markup Language (XML) (W3C, 2000), some proposals have been presented. All of these proposals make use of XML as the base language for describing data. In (Pokorný, 2001), an innovative data structure called an XML-star schema is presented with explicit dimension hierarchies using DTDs that describe the structure of the objects permitted in XML data. The approach presented in (Golfarelli, Rizzi and Vrdoljak, 2001) proposes a semi-automatic approach for building the conceptual schema for a data mart starting from the XML sources. However, these approaches focus on the presentation of the multidimensional XML data rather than on the presentation of the structure of the multidimensional conceptual model itself.

From Table 1, one may conclude that none of the current conceptual modeling approaches consider all MD properties at both the structural and dynamic levels. Therefore, we claim that a standard conceptual model is needed to consider all MD modeling properties at both the structural and dynamic levels. We argue that an OO approach with the UML is the right way of linking structural and dynamic level properties in an elegant way at the conceptual level.

4 Multidimensional modeling with UML

In this section, we summarize how our OO MD model, based on a subset of the UML, can represent main structural and dynamic properties of MD modeling. In Section 4.1, we will present how to represent main structural properties by means of a UML class diagram. Section 4.2 summarizes how users' initial requirements are easily considered by what we call *cube classes*. Section 4.3 describes how to model the behavior of MD databases by using UML state and interaction diagrams from the information represented in these cube

classes. Section 4.4 sketches how we automatically transform an MD model accomplished by following our approach into the Object Database Standard defined by the Object Database Management Group (ODMG) (Cattell *et al.*, 2000). Finally, Section 4.5 presents the corresponding representation of our approach into the XML (W3C, 2000) to allow us an easy interchange of MD information.

4.1 Structural properties by using UML class diagrams

The main structural features considered by UML class diagrams are the many-to-many relationships between facts and dimensions, degenerate dimensions, multiple and alternative path classification hierarchies, and non-strict and complete hierarchies.

It is important to remark that if we are modeling complex and large DW systems, we are not restricted to using flat UML class diagrams. Instead, we can make use of the grouping mechanism provided by the UML called *package* to group classes together into higher level units to create different levels of abstraction, therefore, simplifying the final model (Luján *et al.*, 2002). In this way, a UML class diagram improves and simplifies the system specification accomplished by classic semantic data models such as the ER model. Furthermore, necessary operations and constraints (e.g. additivity rules) can be embedded in the class diagram by means of OCL expressions ((Warmer and Kleppe, 1998), (OMG, 2001).

In this approach, the main structural properties of MD models are specified by means of a UML class diagram in which the information is clearly separated into facts and dimensions. Dimensions and facts are represented by *dimension classes* and *fact classes*, respectively. Then, fact classes are specified as composite classes in shared aggregation relationships of *n* dimension classes. The flexibility of shared aggregation in the UML allows us to represent *many-to-many* relationships between facts and particular dimensions by indicating the 1..* cardinality on the dimension class role. In our example in Figure 3 (a), we can see how the fact class Sales has a many-to-one relationship with both dimension classes.

By default, all measures in the fact class are considered additive. For non-additive measures, *additivity rules* are defined as constraints and are included in the fact class. Furthermore, *derived measures* can also be explicitly considered (indicated by */*) and their *derivation rules* are placed between braces near the fact class, as shown in Figure 3 (a).

This OO approach also allows us to define *identifying attributes* in the fact class, by placing the constraint *{OID}* next to an attribute name. In this way we can represent *degenerate dimensions* ((Giovinazzo, 2000) and (Kimball and Ross, 2002)), thereby representing other fact features in addition to the measures for analysis. For example, we could store the ticket number (ticket_num) and the line number (line_num) as degenerate dimensions, as reflected in Figure 3 (a).

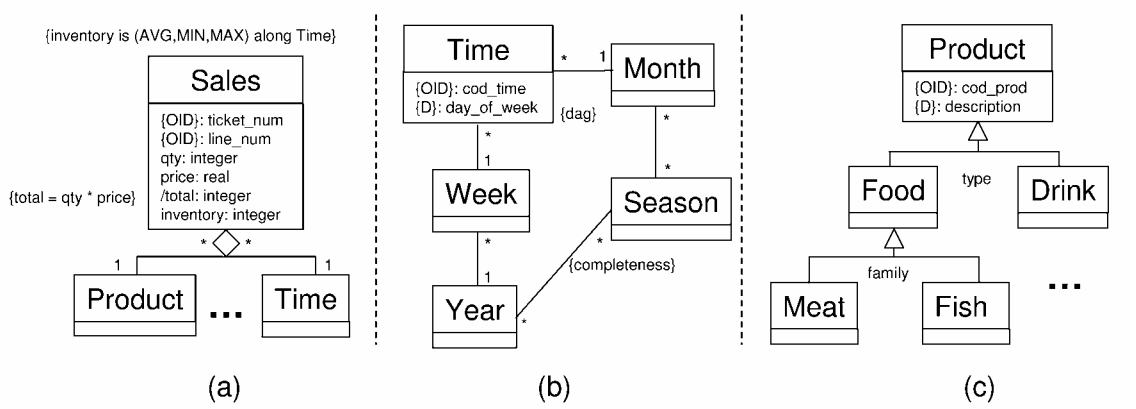

Figure 3: Multidimensional modeling using UML

With respect to dimensions, every *classification hierarchy* level is specified by a class (called a *base class*). An association of classes specifies the relationships between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the dimension class (constraint *{dag}* placed next to every dimension class). The DAG structure can represent both alternative path and multiple classification hierarchies. Every classification hierarchy level must have an *identifying* attribute (constraint *{OID}*) and a *descriptor* attribute[1] (constraint *{D}*). These attributes are necessary for an automatic generation process into commercial OLAP tools, as these tools store the information in their metadata. The multiplicity *1* and *1..* ,* defined in the target associated class role, addresses the concepts of *strictness* and *non-strictness*, respectively. Strictness means that an object at a hierarchy's lower level belongs to only one higher-level object (e.g., as one month can be related to more than one season, the relationship between them is non-strict). Moreover, defining the *{completeness}* constraint in the target associated class role addresses the completeness of a classification hierarchy (see an example in Figure 3 (b)). By completeness we mean that all members belong to one higher-class object and that object consists of those members only. For example, all the recorded seasons form a year, and all the seasons that form the year have been recorded. Our approach assumes all classification hierarchies are non-complete by default.

Finally, the *categorization of dimensions*, used to model additional features for a class's subtypes, is represented by means of *generalization-specialization* relationships. However, only the dimension class can belong to both a classification and a specialization hierarchy at the same time. An example of categorization for the Product dimension is shown in Figure 3 (c).

4.2 Dynamic properties

Regarding dynamic properties, this approach allows us to specify users' initial requirements by means of a UML-compliant class notation called *cube class*. After requirements are specified, behavioral properties are usually then related to these cube classes that represent

[1] A descriptor attribute will be used as the default label in the data analysis.

users' initial requirements. We particularly use **state** and **interaction diagrams** (see Section 4.3) to model the behavior (evolution) of these cube classes based on the applied OLAP operation.

Cube classes follow the query by example (QBE) method: the requirements are defined by means of a template with blank fields. Once requirements are define, the user can then enter conditions for each field that are included in the query. We provide a graphical representation to specify users' initial requirements because QBE systems are considered easier to learn than formal query languages. The structure of a cube class is shown in Figure 4:

- Cube class name.
- Measures area, which contains the measures from the fact to be analyzed.
- Slice area, which contains the constraints to be satisfied in the dimensions.
- Dice area, which contains the dimensions and their grouping conditions to address the analysis.
- Order area, which specifies the order of the result set.
- Cube operations, which cover the OLAP operations for a further data-analysis phase.

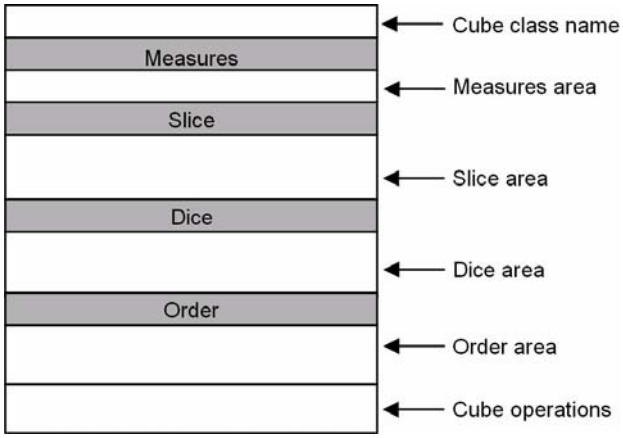

Figure 4: Cube class structure

We should point out that this graphical notation of the cube class aims at facilitating the definition of users' initial requirements to non-expert UML or databases users. In a more formal way, every one of these cube classes has its underlying OQL specification. Moreover, an expert user can directly define cube classes by specifying the OQL sentences (see Section 4.5 for more detail on the representation of cube classes by using OQL).

4.3 Behavioral properties by using state and interaction diagrams

From these cube classes seen in the previous section, final users may start a navigational process by applying certain OLAP operations (*roll-up*, *drill-down*, etc.) in the further data analysis phase. These operations are closed as they generate another cube class as an

output. Thus, we use **state** and **interaction diagrams**[2] to model the behavior (evolution) of these cube classes based on the applied OLAP operation. These diagrams contain information about the most probable evolution of final users' requirements from the specified initial requirement. The information contained in these diagrams can be used by OLAP designers to predict user behaviors, and therefore, help them design a proper view maintenance policy.

Regarding **state diagrams**, one state diagram is defined for each initial cube class. The diagram specifies that certain OLAP operations lead users to cube classes that allow them to analyze the same data (the same measures along the same dimensions) in different ways (navigating through the classification hierarchies defined along the dimensions considered). In these diagrams, each classification hierarchy level defined on a dimension included in the Dice area is considered as a valid state. Every one of these valid states will be a new cube class. Then, the provided OLAP operations allow us to navigate along the states to define new cube classes.

In Figure 5, we can see an example of state diagrams. One state is defined for every level considered in the classification hierarchy of the dimensions included in the corresponding Dice area of the *cube class*. The data analysis will start in the initial state that corresponds to the finest condition specified in the Slice area. Let us suppose that we are interested in navigating along both the product and store dimensions. Classical Roll-up and Drill-down OLAP operations will allow us to aggregate and de-aggregate data (measures) respectively along the hierarchy levels defined in the classification hierarchies. Finally, from every state, we can finish the data analysis with the destroy operation that will lead us to the final state.

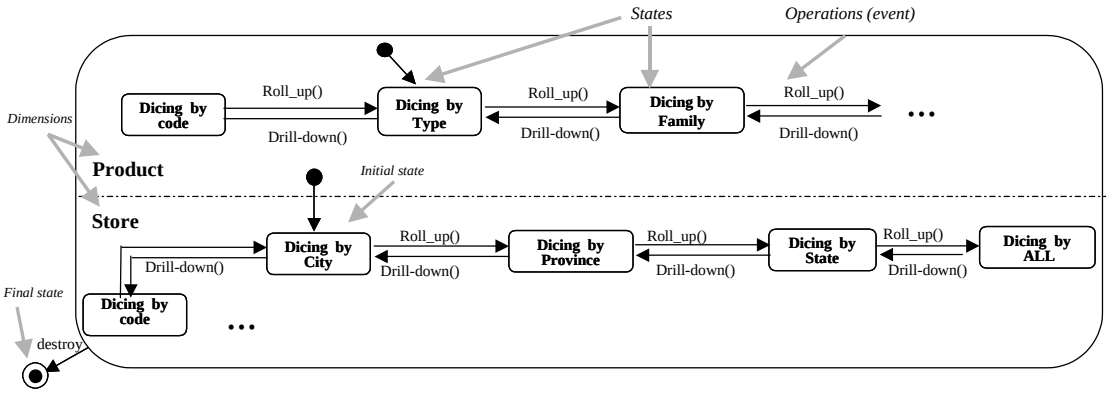

Figure 5: An example of state interaction diagram

On the other hand, an **interaction diagram** can also be defined for each UML class diagram. In our approach, we have adopted sequence diagrams ((Booch *et al.*), (OMG, 2001)) for their clarity and low complexity. This interaction diagram shows interactions among cube classes, changed by OLAP operations such as rotate, pivot, slice, or dice. Thus, we can specify that certain OLAP operations (e.g. dice) lead users to cube classes which

[2] See Chapter 4 [Trujillo01a] and [Trujillo00] for more information about the provided OLAP operations and how to build state and interaction diagrams.

will show completely different data. Thus, these new cube classes represent the most probable new requirements a final user wishes to execute.

In Figure 6, we can see an example of interaction diagrams. Let us suppose that we have only defined two cube classes to specify two initial requirements. Then, we specify the operation needed to switch from one cube class into the other. In this particular case, the Rotate operation indicates the transition that lead us to the CC_2 from the CC_1. In concrete, we are also interested in analyzing data along the Customer dimension. As we are still interested in the dimensions defined in CC_1, we do not eliminate any dimension in this operation. It is easy to see that we always define the reverse operation to give analyzers the opportunity of returning to the initial point of analysis.

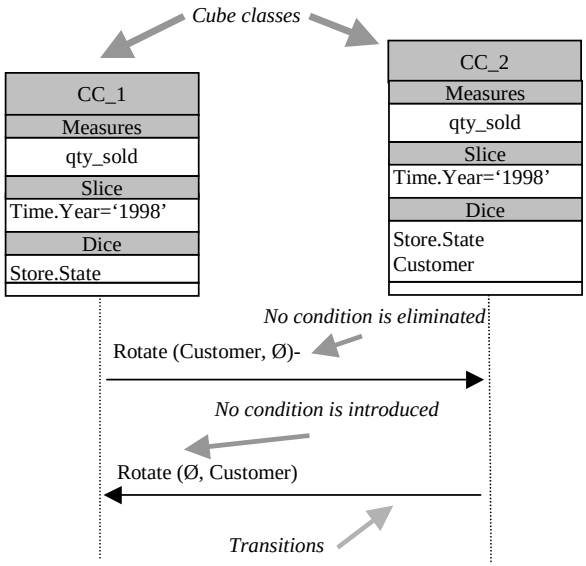

Figure 6: An example of interaction diagrams

4.4 Standard representation by using the ODMG proposal

Our approach generates the corresponding representation of an MD model in most of the relational database management systems such as Oracle, Informix, Microsoft SQL Server, IBM DB2 and so on (Trujillo *et al.*, 2001b). Furthermore, we also provide the corresponding representation into object-oriented databases. However, this representation is totally dependent on the object database management system (ODBMS) used for the corresponding implementation. For this reason, in this Section, we present the corresponding representation of an MD model accomplished by our approach following the standard for ODBMS[3] proposed by the Object Database Management Group (ODMG) (Cattell *et al.*, 2000). The adoption of this standard ensures the portability of our MD model across platforms and products, thereby facilitating the use of our approach. However, we

[3] The ODMG defines an ODBMS as "[…] a DBMS that integrates database capabilities with object-oriented programming language capabilities".

also point out some properties that cannot be directly represented by using this standard and that should be taken into account when transforming this ODBM into a particular object-oriented model of the target ODBMS.

The major components of the ODMG standard are the Object Model, the Object Definition Language (ODL), the Object Query Language (OQL), and the bindings of the ODMG implementations to different programming languages (C++, Smalltalk, and Java). In this paper, we will start by providing the corresponding representation for structural properties into the ODL, a specification language used to define the specifications of object types. Then, we will sketch how to represent *cube classes* into the OQL, a query language that supports the ODMG data model. The great benefit of this OQL is that is very close to SQL, and is therefore, a very simple-to-use query language.

ODL definition of an MD model

Our three-level MD model cannot be represented in an ODBMS, because the ODL uses a flat representation for the class diagram without providing any package mechanism in the ODL. Therefore, we start the transformation of the MD models from the third level in the fact package, because it contains the complete MD model definition: fact classes, dimension classes, base classes, classification hierarchy properties, etc.

In the following, we are going to use an actual example to clarify our approach. We have selected a simplification of the grocery example taken from Kimball's book (Kimball and Ross, 2002). In this example, the corresponding MD model contains the following elements:

- One fact (Sales) with three measures (quantity, price and total_price) and two degenerate dimensions (ticket_num and line_num).
- Two dimensions: Product, with three hierarchy levels (Brand, Subgroup, and Group) and Time, with two hierarchy levels (Month and Year).

The first level of the MD model is represented in Figure 7 and only contains one star schema package, as the example only contains one fact. The second level contains one fact package (Sales product) and two dimension packages (Product and Time), as it can be seen in Figure 8. Finally, Figure 9 represents the content of the Product dimension package, and Figure 10 the content of the Time dimension package.

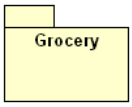

Figure 7

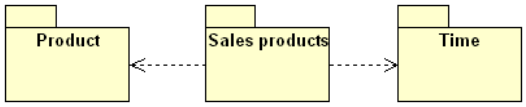

Figure 8

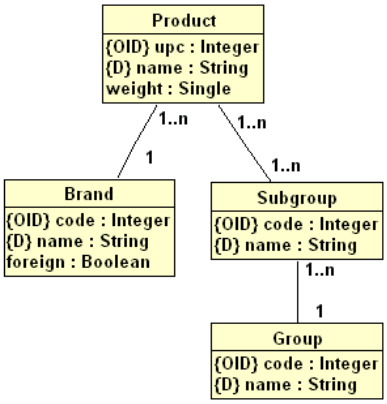

Figure 9

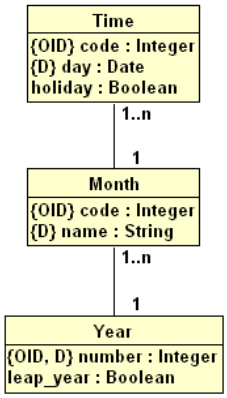

Figure 10

In Figure 11, we can see the content (level 3) of the Sales products fact package, where the complete definition of the MD model is available. The transformation process starts from this view of the MD model.

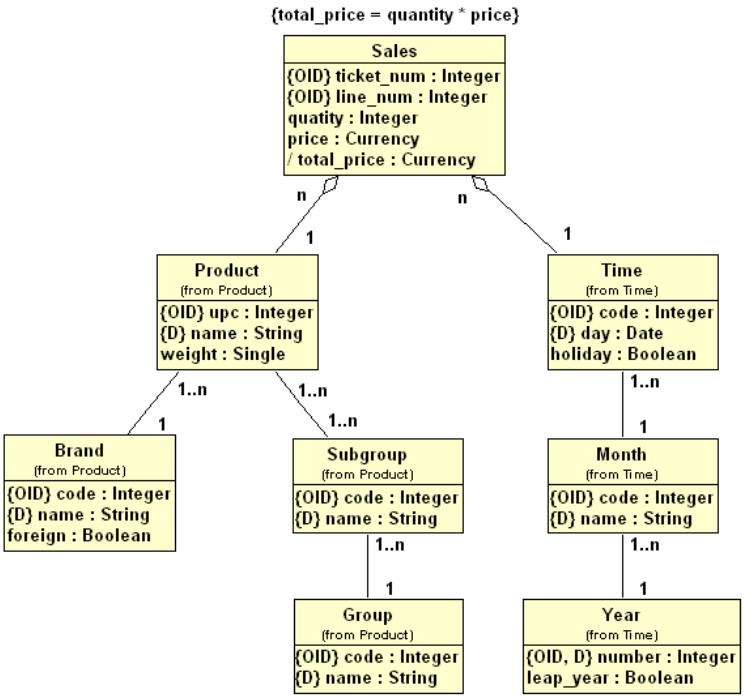

Figure 11: Third level of the MD model

For the sake of simplicity, we show the ODL representation of only three classes: Sales, Product, and Time (the representation of the other classes is very similar). The transformation process starts from the fact class (Sales). Since OID attributes cannot be

represented in ODL, we have decided to use the `unsigned long` type to represent them. Aggregation relationships cannot be directly represented, but we transform them to association relationships. Moreover, maximum cardinality of relationships can be expressed, but the minimum cardinality is lost in the transformation process. In ODL, the definition of a relationship includes designation of the target type, the cardinality on the target side, and information about the inverse relationship found in the target side. The ODL definition for the Sales fact class is as follows:

```
class Sales
{
  attribute unsigned long ticket_num;
  attribute unsigned long line_num;
  attribute long quantity;
  attribute double price;
  attribute double total_price;
  relationship Product sales_product inverse Product::product_sales;
  relationship Time sales_time inverse Time::time_sale;
};
```

For expressing the cardinality ?-to-many, we use the ODL constructor `set`. For example, the Product class has three relationships: with Sales class (?-to-many), with Brand class (?-to-one) and with Subgroup class (?-to-many). In order to know the cardinality of the relationships in this side, we have to consult the inverse relationship in the target side. For example, the relationship between Product and Sales is one-to-many, since the type of the relationship is set<Sales> (many) in this side, but in the inverse relationship (Sales::sales_product) it is Product (one). Product and Time dimension classes are specified in ODL as:

```
class Product
{
  attribute unsigned long upc;
  attribute string name;
  attribute float weight;
  relationship set<Sales> product_sales inverse Sales::sales_product;
  relationship Brand product_brand inverse Brand::brand_product;
  relationship set<Subgroup> product_subgroup inverse Subgroup::subgroup_product;
};

class Time
{
  attribute unsigned long code;
  attribute date day;
  attribute boolean holiday;
  relationship set<Sales> time_sales inverse Sales::sales_time;
  relationship Month time_month inverse Month::month_time;
};
```

Loss of expressiveness

As previously commented, some MD properties that are captured in our approach cannot be directly considered by using ODL. This is an obvious problem, because the ODL is a general definition language that is not oriented to represent MD properties used in a conceptual design. Specifically, we ignore or transform the following properties:

- Identifying attribute (OID) and descriptor attribute (D) are ignored because they are considered to be an implementation issue that will be automatically generated by the ODBMS.
- Initial values are ignored. This is not a key issue in conceptual MD modeling.
- Derived attributes and their corresponding derivation rules are ignored. These derivation rules will have to be specified when defining user requirements by using the OQL.
- Additivity rules are ignored because the ODL specification cannot represent any information related to the aggregation operators that can be applied on measures.
- Minimum cardinality cannot be specified either.
- Completeness of a classification hierarchy is also ignored.

Up to now, these ignored properties have to be considered as footnotes in the ODMG specification. For an unambiguous specification of MD models using the ODMG specification, a formal constraint language should be used. Unfortunately, a constraint language is completely missing from the ODMG standard specification.

Cube classes represented by using OQL

The OQL is not easy to use for defining users' initial requirements, because the user needs to know the underlying ODL representation corresponding to the MD model. Due to this fact, we also provide cube classes, which allow the user to define initial requirements in a graphical way. These cube classes can automatically be transformed into OQL sentences, and can therefore be used to query an ODBMS that stores an MD model. For example, let us suppose the following initial requirement:

```
The quantity sold of the products belonging to the "Grocery" Group during
  "January", grouped according to the product Subgroup and the Year and
                  ordered by the Brand of the product
```

In Figure 12, we can see the corresponding cube class to the previous requirement. It is easy to see how the cube class is formed:

- Measures contains the goal of the analysis: SUM(quantity).
- Slice the restrictions defined on the Time and Product dimensions.
- Dice the grouping conditions required along the Product and Time dimensions.
- And Order defines the order of the result set.

Figure 12: An example of a user's initial requirement

The cube class can be automatically translated into OQL. The algorithm uses the corresponding ODL definition of the MD model to obtain the paths from the fact class (the core of the analysis) to the rest of classes (dimension and base classes). For example, the path from the Sales fact class to the Year base class along the Time dimension traverses the relationships sales_time in Sales fact class, time_month in Time dimension class, and month_year in Month base class. Moreover, when attributes' names are omitted in the cube class, the algorithm automatically selects the descriptor attribute defined in the MD model. For example, the expression Time.Month="January" of the cube class in Figure 12 involves the use of the descriptor attribute from the Month base class, because no further attribute is specified. In the same way, the order expression Product.Brand involves the use of the descriptor attribute from Brand. The OQL for the corresponding cube class in Figure 12 is as follows:

```
SELECT SUM(s.quantity)
FROM Sales s, s.sales_time st, s.sales_product sp
WHERE st.time_month.name = "January" AND
sp.product_subgroup.subgroup_group.name = "Grocery"
GROUP BY sp.product_subgroup.name AND st.time_month.month_year.number
ORDER BY sp.product_brand.name
```

4.5 XML to interchange multidimensional properties

One key aspect in the success of an MD model should be its capability to interchange information in an easy and standard format. The eXtensible Markup Language (XML) (W3C, 2000) is rapidly being adopted as the standard for the exchange of un-structured, semi-structured and structured data. Furthermore, XML is an open neutral platform and vendor independent meta-language, which allows users to reduce the cost, complexity, and effort required in integrating data within and between enterprises. In the future, all applications may exchange their data in XML and then conversion utilities will not be necessary any more.

We have adopted the XML to represent our MD models due to its advantages, such as standardization, usability, versatility and so on. We have defined a Document Type Definition (DTD) that determines the correct structure and content of XML documents that represent MD models. Moreover, this DTD can be used to automatically validate the XML documents. In Appendix 1 we include the whole DTD that we have defined to represent MD models in XML. This DTD allows us to represent both structural and dynamic properties of MD models.

In Table 2, we have summarized the main rules of our DTD, which contains 38 elements (tags). We have defined additional elements (in plural form) in order to group common elements together, so that they can be exploited to provide optimum and correct comprehension of the model, e.g. elements in plural like PKSCHEMAS or DEPENDENCIES.

The DTD follows the three-level structure of our MD approach:

- An MD model contains PKSCHEMAS (star schema packages) at level 1 (Table 2, line 1).
- A PKSCHEMA contains at most one PKFACT (fact package) and many PKDIMS (dimension packages) and IMPPKDIMS (imported dimensions) at level 2 (Table 2, line 2).
- A PKFACT contains at most one FACTCLASS (Table 2, line 4) and a PKDIM contains at most one DIMCLASS and many BASECLASSES (Table 2, line 3) at level 3.

Within our DTD, fact classes labeled FACTCLASS may have no fact attributes to consider fact-less fact tables, as can be observed in the content of the element FACTATTS (Table 2, line 6): 0 or more FACTATT.

1.	`<!ELEMENT MDMODEL (PKSCHEMAS, DEPENDENCIES, CUBECLASSES)>`
2.	`<!ELEMENT PKSCHEMA (PKFACT?, PKDIMS, IMPPKDIMS, DEPENDENCIES)>`
3.	`<!ELEMENT PKDIM (DIMCLASS?, BASECLASSES, IMPBASECLASSES)>`
4.	`<!ELEMENT PKFACT (FACTCLASS?)>`
5.	`<!ELEMENT FACTCLASS (FACTATTS, METHODS, SHAREDAGGS)>`
6.	`<!ELEMENT FACTATTS (FACTATT*)>`

Table 2: DTD main rules

From now on, we are going to explain the structure of our DTD by means of the grocery example presented in the previous section. In the next fragment of the XML document that represents the grocery example, the first line defines the XML version and the character encoding used in the document. The next line declares the DTD that defines the structure of the document. Finally, the third line describes the root element of the document (MDMODEL). An MD model (Table 2, line 1) contains star schema packages (PKSCHEMAS)

with dependencies between them (DEPENDECIES) and users' initial requirements (CUBECLASSES).

In this example, the MD model only contains one star schema package (Figure 7); as there is not any dependency between star schema packages the DEPENDENCIES element is empty. Finally, the CUBECLASSES element is also empty as no initial requirement has been specified yet.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MDMODEL SYSTEM "MDModel.dtd">
<MDMODEL id="ID5" name="Grocery example">
  <PKSCHEMAS>
    <PKSCHEMA id="ID6" name="Grocery" caption="Grocery">
    ...
    </PKSCHEMA>
  </PKSCHEMAS>
  <DEPENDENCIES/>
  <CUBECLASSES/>
</MDMODEL>
```

In our DTD, every MD element has an ID attribute that must be unique to the whole XML document. The value of this attribute is automatically generated by our exportation process and is used in the definition of relationships between elements in our MD model, e.g., in the definition of the dependencies between packages.

The next fragment represents the definition of the star schema Grocery. A PKSCHEMA (Table 2, line 2) can contain:

- At most one PKFACT.
- 0 or more dimension packages (PKDIMS) defined in the very star schema.
- 0 or more dimension packages imported form other star schemas (IMPPKDIMS).
- Dependencies between the dimensions packages (DEPENDENCIES).

Every package, regardless being a fact or a dimension package, has a name used in the exportation process and a caption used in the graphical representation. As seen in this fragment of the XML document, two dependencies have been defined from the Sales products package (ID7) to the Product package (ID8) and the Time package (ID9). Thanks to the use of the IDREF attribute type in the DTD, we can define that start and end attributes of DEPENDENCY element that must take a value from an ID attribute of an element in the XML document.

```
<PKSCHEMA id="ID6" name="Grocery" caption="Grocery">
  <PKFACT id="ID7" name="Sales_products" caption="Sales products">
  ...
  </PKFACT>
  <PKDIMS>
    <PKDIM id="ID8" name="Product">
    ...
    </PKDIM>
    <PKDIM id="ID9" name="Time">
    ...
    </PKDIM>
  </PKDIMS>
  <IMPPKDIMS/>
```

```
  <DEPENDENCIES>
    <DEPENDENCY id="ID10" start="ID7" end="ID8"/>
    <DEPENDENCY id="ID11" start="ID7" end="ID9"/>
  </DEPENDENCIES>
</PKSCHEMA>
```

The following fragment defines the dimension package Product. A dimension package (Table 2, line 3) contains:

- At most one dimension class (DIMCLASS).
- 0 or more base classes that represent hierarchy levels (BASECLASSES).
- 0 or more imported bases classes from other dimension packages (IMPBASECLASSES).

In this fragment we can see how the relationships between a dimension class and base classes are expressed in our DTD; the cardinality of the relationship is expressed by means of the attributes roleA and roleB. We can also see the definition of the three attributes of the Product dimension class: upc, name, and weight. In the DTD, the *{OID}* and *{D}* constraints of our MD model are represented as attributes OID and D of the DIMATT element.

```
<PKDIM id="ID8" name="Product">
  <DIMCLASS id="ID12" name="Product">
    <DIMATTS>
      <DIMATT id="ID15" name="upc" atomic="true" type="Integer"
       OID="true" D="false"/>
      <DIMATT id="ID16" name="name" atomic="true" type="String"
       OID="false" D="true"/>
      <DIMATT id="ID17" name="weight" atomic="true" type="Single"
       OID="false" D="false"/>
    </DIMATTS>
    <RELATIONASOCS>
      <RELATIONASOC id="ID35" child="ID18" roleA="1..M" roleB="1"
       completeness="false"/>
      <RELATIONASOC id="ID36" child="ID25" roleA="1..M" roleB="1..M"
       completeness="false"/>
    </RELATIONASOCS>
    <RELATIONCATS/>
    <METHODS/>
  </DIMCLASS>
  <BASECLASSES>
    <BASECLASS id="ID18" name="Brand">
    ...
    </BASECLASS>
    <BASECLASS id="ID25" name="Subgroup">
    ...
    </BASECLASS>
    <BASECLASS id="ID30" name="Group">
    ...
    </BASECLASS>
  </BASECLASSES>
  <IMPBASECLASSES/>
</PKDIM>
```

Finally, a fact package (Table 2, line 4) contains at most one fact class, and each fact class (Table 2, line 5) can contain fact attributes (FACATTS), methods (METHODS) and shared aggregations with the dimension classes (SHAREDAGGS). Notice that many-to-many relationships between facts and dimensions can also be expressed by assigning the same value "M" to both attributes roleA and roleB in the DTD element SHAREDAGG.

```
<PKFACT id="ID7" name="Sales_products" caption="Sales products">
  <FACTCLASS id="ID70" name="Sales">
    <FACTATTS>
      <FACTATT id="ID71" name="ticket_num" atomic="true"
       type="Integer" OID="true"/>
      <FACTATT id="ID72" name="line_num" atomic="true" type="Integer"
       OID="true"/>
      <FACTATT id="ID73" name="quantity" atomic="true" type="Integer"
       OID="false"/>
      <FACTATT id="ID74" name="price" atomic="true" type="Currency"
       OID="false"/>
      <FACTATT id="ID75" name="total_price" atomic="true"
       type="Currency" derivationRule="quantity * price" OID="false"/>
    </FACTATTS>
    <METHODS/>
    <SHAREDAGGS>
      <SHAREDAGG id="ID80" dimclass="ID12" roleA="1" roleB="1..M"/>
      <SHAREDAGG id="ID81" dimclass="ID49" roleA="1" roleB="1..M"/>
    </SHAREDAGGS>
  </FACTCLASS>
</PKFACT>
```

5 Case studies

The aim of this section is to exemplify the usage of our conceptual modeling approach on modeling MD databases. We have selected three different examples taken from Kimball's book [Kimball02], each of which introduces a new particular modeling feature: a warehouse, a large bank, and a college course. Due to the lack of space, we will only apply our complete modeling approach for the first example: we will apply all of the diagrams we use for modeling a DW (package diagrams, class diagrams, interaction diagrams, etc.). For the rest of the examples, due to space constraints, we will only focus on representing the structural properties of MD modeling by specifying the corresponding UML class diagram. This class diagram is the key one in our approach since the rest of diagrams can be easily obtained from it.

5.1 The warehouse

This example explores three inventory models of a warehouse. The first one is the inventory snapshot, where the inventory levels are measured every day and are placed in separate records in the database. The second model is the delivery status model, which contains one record for each delivery to the warehouse and the disposition of all the items is registered until they have left the warehouse. Finally, the third inventory model is the transaction model, which records every change of the status of delivery products as they arrive at the warehouse, are processed into the warehouse, etc.

This example introduces two important concepts: the *semi-additivity* and the *multistar model* (also known as fact constellations). The former has already been introduced in Section 2 and refers to the fact that a measure cannot be summarized by using the *sum* function along a dimension. In this example, the inventory level (stock) of the warehouse is semi-additive, because it cannot be summed along time dimension, but it can be averaged

along the same dimension. The multistar (fact constellations) concept refers to the fact that the same MD model has multiple facts.

To start with, in our approach we model multistar models by means of package diagrams. In this way, at the first level, we create a package diagram for each one of the facts considered in the model. At this level, connecting package diagrams means that a model will use elements (e.g. dimensions, hierarchies) defined in the other package. Figure 13 shows the first level of the model formed by three packages that represent the different star schemas in the case study.

Figure 13: Level 1

Then, we explore each package diagram at the second level to define packages for each one of the facts and dimensions defined in the corresponding package diagram. Figure 14 shows the content of the package Inventory Snapshot Star at level 2. The fact package Inventory Snapshot Fact is represented in the middle of Figure 14, and the dimension packages (Product Dimension, Time Dimension, and Warehouse Dimension) are placed around the fact package. As can be seen, a dependency is drawn from the fact package to each one of the dimension packages, because the fact package comprises the whole definition of the star schema. At level 2, it is possible to create a dependency from a fact package to a dimension package or between dimension packages (when they share some hierarchy levels), but not from a dimension package to a fact package.

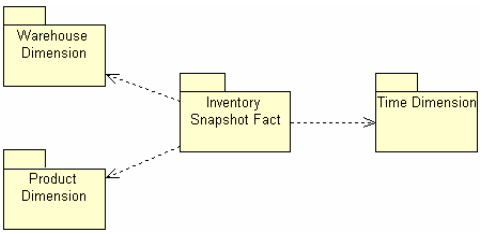

Figure 14: Level 2 of Inventory Snapshot Star

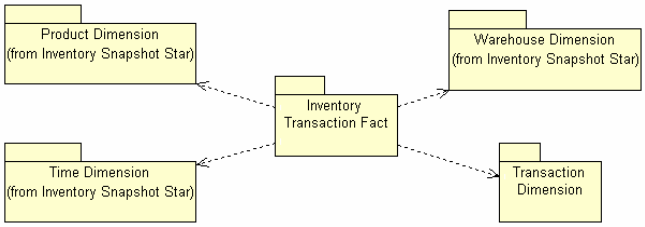

Figure 15: Level 2 of Inventory Transaction Star

Figure 15 shows the content of the package Inventory Transaction Star at level 2. As in the Inventory Snapshot Star, the fact package is placed in the middle of the figure and the dimension packages are placed around the fact package in a star fashion. Three dimension packages (Product Dimension, Time Dimension, and Warehouse Dimension) have been previously defined in the Inventory Snapshot Star (Figure 14), and they are imported in this package. Therefore, the name of the package where it has been previously defined appears below the package name (from Inventory Snapshot Star).

The content of the dimension and fact packages is represented at level 3. The diagrams at this level are only comprised of classes and their associations. For example, Figure 16 shows the content of the package Warehouse Dimension at level 3. In a dimension package, a

class is drawn for the dimension class (Warehouse) and a class for each classification hierarchy level (ZIP, City, County, State, SubRegion, and Region). For the sake of simplicity, the methods of each class have not been depicted in the figure. As can be seen in Figure 16, Warehouse presents alternative path classification hierarchies: (i) ZIP, City, County, State, and (ii) SubRegion, Region, State.

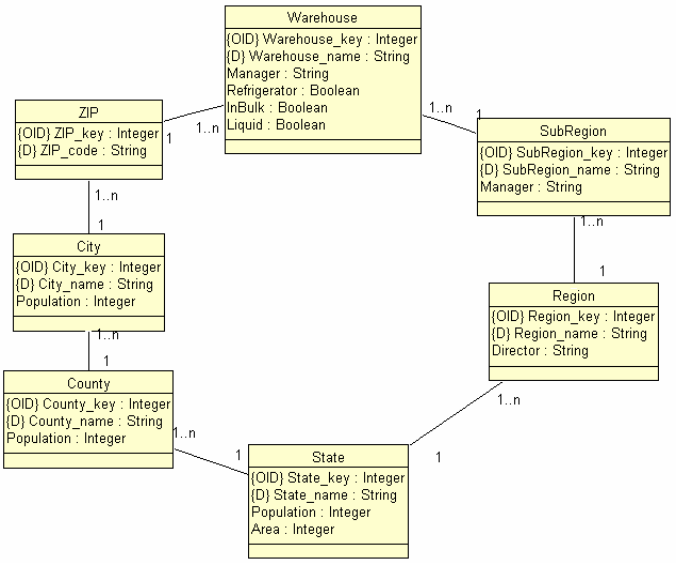

Figure 16: Level 3 of Warehouse Dimension

Finally, Figure 17 shows the content of the package Inventory Snapshot Fact. In this package, the whole star schema is displayed: the fact class (Inventory Snapshot) is defined and the dimensions with their corresponding hierarchy levels are imported from the dimension packages. To avoid unnecessary details, we have hidden the attributes and methods of dimensions and hierarchy levels, but the measures of the fact are shown as attributes of the fact class: four atomic measures (quantity_on_hand, quantity_shipped, value_at_cost, and value_at_LSP), and three derived measures (number_of_turns, gross_profit, and gross_margin). The definition of the derived measures is included in the model by means of derivation rules. Regarding the additivity of the measures, only quantity_on_hand is semi-additive; because of this, an additivity rule has been added to the model. Finally, Warehouse presents alternative path classification hierarchies and Time and Product present multiple classification hierarchies, as can be seen in Figure 17.

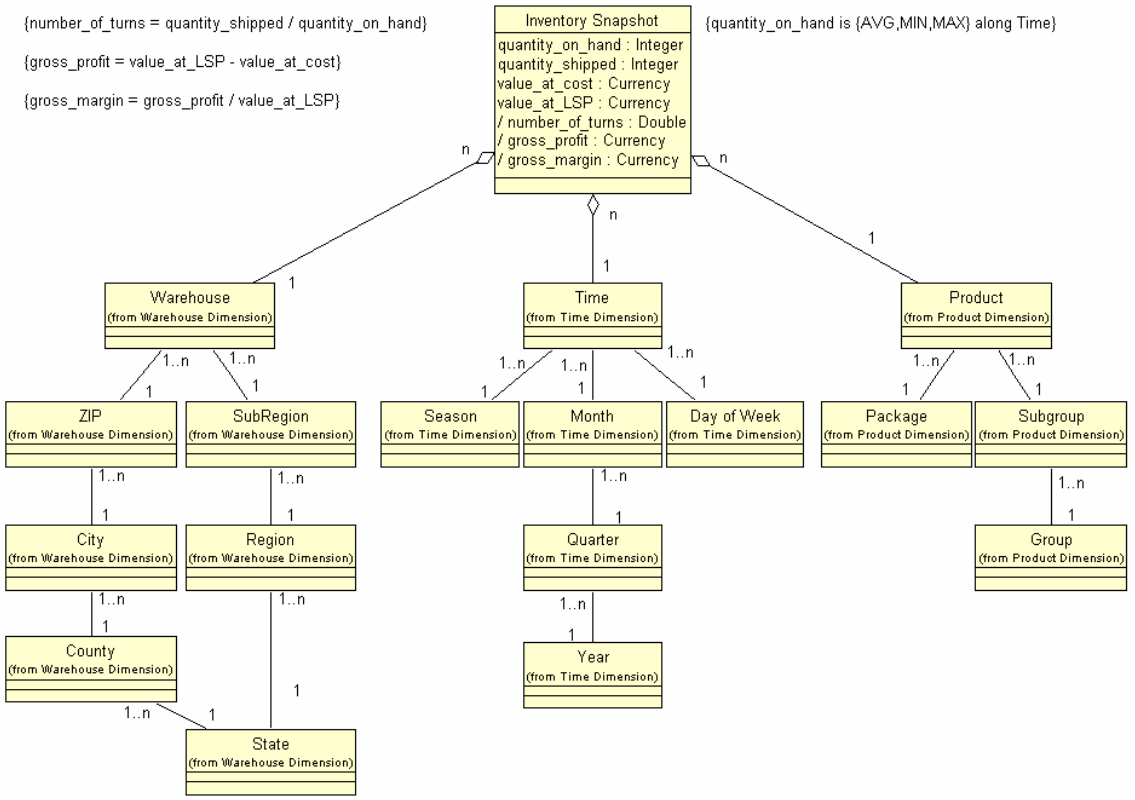

Figure 17: Level 3 of Inventory Snapshot Fact

Regarding the dynamic part of the model, let us suppose the following user's initial requirement on the MD model specified by the UML class diagram of Figure 17: '*We wish to analyze the quantity_on_hand of products* ***where*** *the group of products is "Grocery" and the warehouse state is "Valencia",* ***grouped*** *according to the product subgroup* ***and*** *the warehouse region and subregion, and* ***ordered*** *by the warehouse subregion and region'*. On the left hand side of Figure 18, we can observe the graphical notation of the cube class that corresponds to this requirement. The measure to be analyzed (quantity_on_hand) is specified in the measure area. Constraints defined on dimension classification hierarchy levels (group and state) are included in the slice area, while classification hierarchy levels along which we are interested in analyzing measures (subgroup, region, and subregion) are included in the dice area. Finally, the available OLAP operations are specified in the CO (Cube Operations) section (in this example the CO are omitted to avoid unnecessary detail). On the right hand side of Figure 18 the OQL sentence corresponding to the cube class is shown. We can notice how the descriptor attributes from the MD model are used when the attributes of the hierarchy levels are omitted in the analysis. For example, the expression Warehouse.State="Valencia" of the cube class involves the use of the descriptor attribute from the State base class (Figure 16).


```
SELECT quantity_on_hand
FROM Inventory_Snapshot i, i.is_warehouse iw,
i.is_product ip
WHERE
iw.warehouse_subregion.subregion_region.region_state.S
tate_name = "Valencia" AND
ip.product_subgroup.subgroup_group.Name = "Grocery"
GROUP BY
iw.warehouse_subregion.subregion_region.Region_name,
iw.warehouse_subregion.Subregion_name,
ip.product_subgroup.Name
ORDER BY iw.warehouse_subregion.Subregion_name,
iw.warehouse_subregion.subregion_region.Region_name
```

Figure 18: An example of a user's initial requirement

Regarding **state diagrams**, one state diagram is defined for each initial cube class. The diagram specifies that certain OLAP operations lead users to cube classes that allow them to analyze the same data (the same measures along the same dimensions) in different ways. For example, in Figure 19 we can see the corresponding state diagram of the cube class definition of Figure 18. It may be observed, for example, that roll-up and drill-down operations applied on the classification hierarchies levels defined on the Warehouse and Product dimensions will allow us to navigate up and down along the classification hierarchies defined in both dimensions.

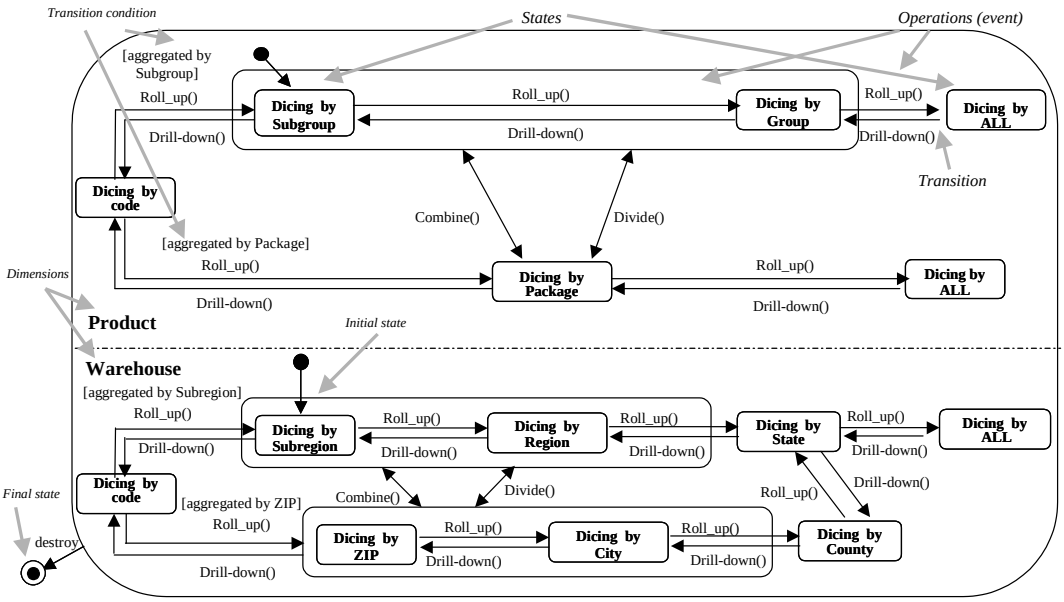

Figure 19: An example of state diagram

On the other hand, an **interaction diagram** can also be defined for each UML class diagram. This interaction diagram shows interactions among cube classes, changed by OLAP operations such as rotate, pivot, slice, or dice. In Figure 20, we can see an example

of an interaction diagram, in which we have considered three cube classes that specify the user's initial requirements. We have then defined the OLAP operations needed to switch between these cube classes.

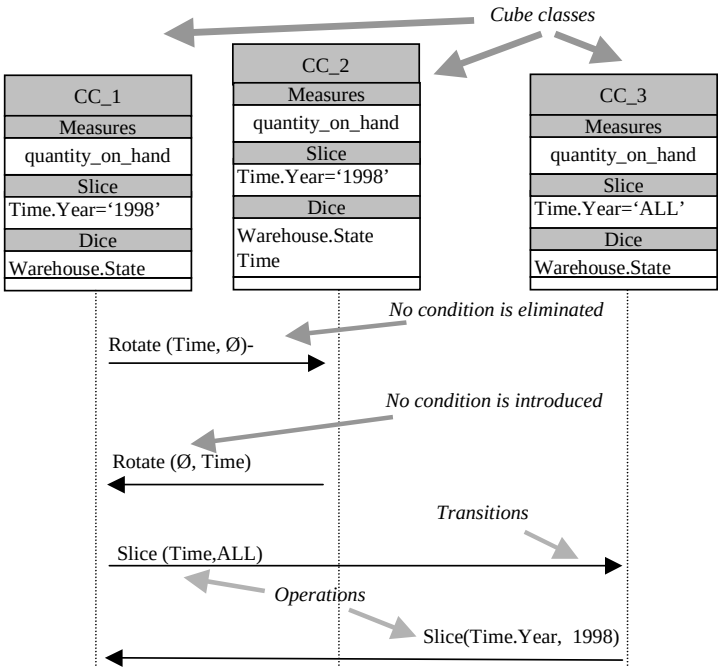

Figure 20: An example of interaction diagram

5.2 A large bank

In this example, a DW for a large bank is presented. The bank offers a significant portfolio of financial services: checking accounts, savings accounts, mortgage loans, safe deposit boxes, and so on.

This example introduces the following concepts:

- Heterogeneous dimension: a dimension that describes a large number of heterogeneous items with different attributes. Kimball's recommended technique is "to create a core fact table and a core dimension table in order to allow queries to cross the disparate types and to create a custom fact table and a custom dimension table for querying each individual type in depth". However, our conceptual MD approach can provide an elegant and simple solution to this problem, thanks to the categorization of dimensions.
- Categorization of dimensions: it allows us to model additional features for a dimension's subtypes.
- Shared classification hierarchies between dimensions: our approach allows two or more dimensions to share some levels of their classification hierarchies.

Figure 21 represents level 1, which comprises five star packages: Saving Accounts Star, Personal Loans Star, Investment Loans Star, Safe Deposit Boxes Star, and Mortgage Loans Star. For now, we will only center on the Mortgage Loans Star. The corresponding level 2 of this star package is depicted in Figure 22.

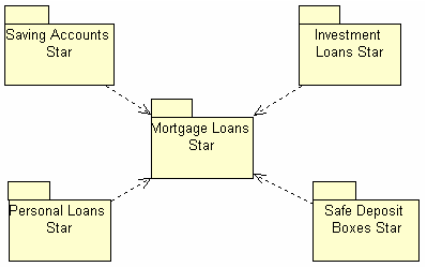

Figure 21: Level 1

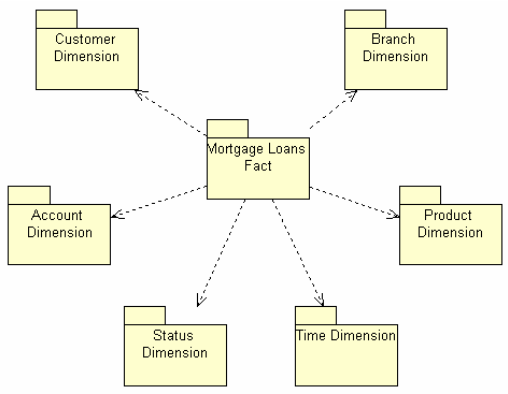

Figure 22: Level 2 of Mortgage Loans Star

Level 3 of Mortgage Loans Fact is shown in Figure 23. To avoid unnecessarily complicating the figure, three of the dimensions (Account, Time, and Status) with their corresponding hierarchies are not represented. Moreover, the attributes of the represented hierarchy levels have been omitted. The fact class (Mortgage Loans) contains four attributes that represent the measures: total, balance, and payment_number are atomic; whereas debt is derived (the corresponding derivation rule is placed next to the fact class). None of the measures is additive. Consequently, the additivity rules are also placed next to the fact class.

In this example, the dimensions present two special characteristics. On one hand, Branch and Customer share some hierarchy levels: ZIP, City, County, and State. On the other hand, the Product dimension has a generalization-specialization hierarchy. This kind of hierarchy allows us to easily deal with heterogeneous dimensions: the different items can be grouped together in different categorization levels depending on their properties.

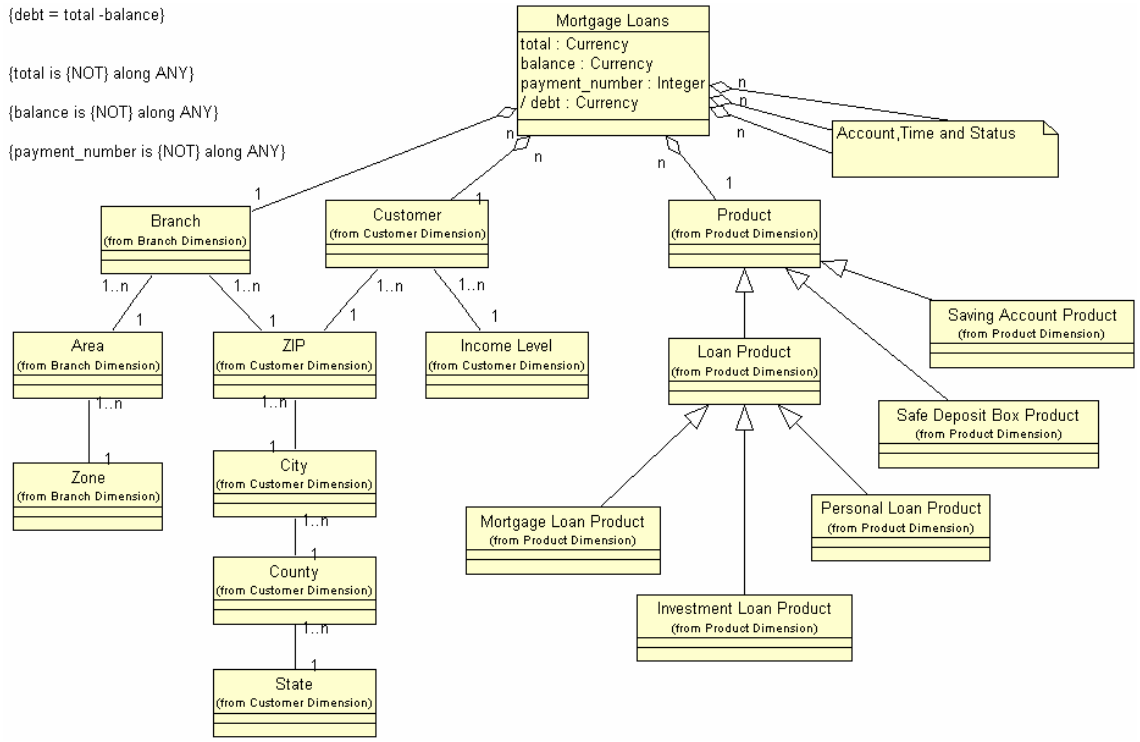

Figure 23: Level 3 of Mortgage Loans Fact

5.3 The college course

This example introduces the concept of the *factless fact table* (FFT): fact tables for which there are no measured facts. Kimball distinguishes two major variations of FFT: *event tracking tables* and *coverage tables*. In this example we will focus on the first type.

Event tracking tables are used when a large number of events need to be recorded as a number of dimensional entities coming together simultaneously. In this example, we will model daily class attendance at a college. In Figure 24 and Figure 25, level 1 and level 2 of this model are depicted respectively. In this case, level 1 only contains one star package.

Figure 24: Level 1

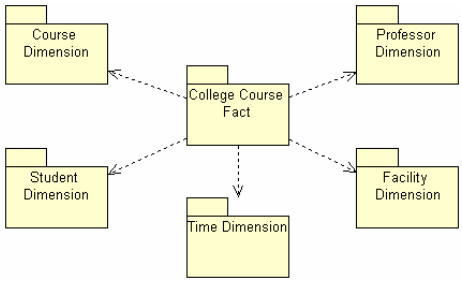

Figure 25: Level 2 of College Course Star

Figure 26 shows level 3 of College Course Fact. For the sake of simplicity, the attributes and methods of every class have not been depicted in the figure. As shown, the fact class

College Course contains no measures because it is a FFT. In FFT, the majority of the questions that users create imply counting the number of records that satisfy a constraint, such as: which facilities were used most heavily? Or, which courses were the least attended?

Regarding the dimensions, Course and Time present multiple classification hierarchies, Professor and Student share some hierarchy levels, and Facility presents a categorization hierarchy.

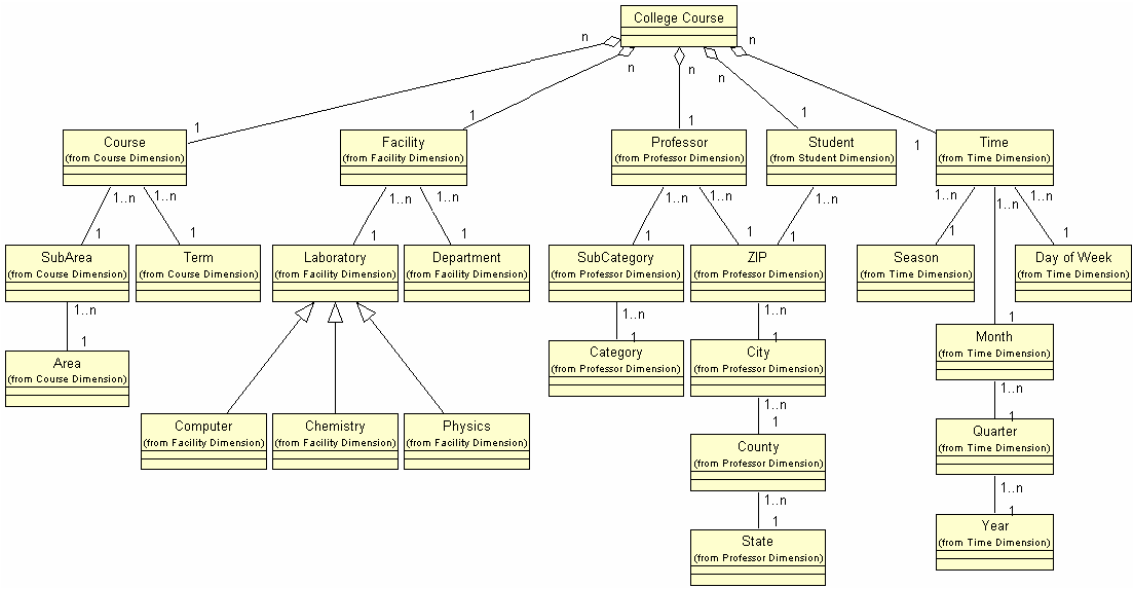

Figure 26: Level 3 of College Course Star

6 Conclusions

In this paper, we have presented an OO conceptual modeling approach, based on the UML, to design DWs, MD databases and OLAP applications. Structural aspects of MD modeling are easily specified by means of a UML class diagram in which classes are related through association and shared aggregation relationships. In this context, thanks to the flexibility and the power of the UML, all the semantics required for proper MD conceptual modeling are considered, such as *many-to-many* relationships between facts and particular dimensions, multiple path hierarchies of dimensions, the strictness and completeness of classification hierarchies, and categorization of dimension attributes. Regarding dynamic aspects, we provide a UML-compliant class graphical notation (called *cube classes*) to specify users' initial requirements at the conceptual level. We have also described how we use state and interaction diagrams to model the behavioral aspects of the system regarding these cube classes based on the set of the applied OLAP operations. Moreover, we have sketched out how to represent a conceptual MD model accomplished by our approach in the ODMG standard as a previous step for a further implementation of MD models into OODB and ORDB. Furthermore, to facilitate the interchange of MD models, we provide a DTD from which we can obtain valid XML documents. Finally, we have selected three case

studies from Kimball's book and modeled them following our approach. This shows that our approach is a very easy-to-use yet powerful conceptual model that represents main structural and dynamic properties of MD modeling in an easy and elegant way.

Currently, we are working on several issues. On one hand, we are extending our approach to key issues in MD modeling, including temporal and slowly changing dimensions. On the other hand, we are also working on the definition of a formal constraint language for ODMG that allows us to represent the MD modeling necessarily ignored in the generation process from our approach based on the UML.

References

Abello, A., Samos, J. & Saltor, F. (2001). A framework for the Classification and Description of Multidimensional Data Models. *Proceedings of the Twelfth Database and Expert Systems Applications (DEXA'01)*, Munich, Germany, Lecture Notes in Computer Science (LNCS), 2113, 668-677.

Blaschka, M., Sapia, C., Höfling, G. & Dinter, B. (1998). Finding Your Way Through Multidimensional Models. *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria, IEEE Computer Society, 198-203.

Booch, G., Rumbaugh, J. & Jacobson, I. (1998). *The Unified Modeling Language*. New York: Addison-Wesley.

Cattell, R. G. G. *et al*. (2000). *The Object Data Standard: ODMG 3.0*. New York: Morgan Kaufmann Publishers, 2000.

Chaudhuri, S. & Dayal, U. (1997). An Overview of Data Warehousing and OLAP Technology. *ACM Sigmod Record*, 26(1), 65-74.

Giovinazzo, W. (2000). *Object-Oriented Data Warehouse Design. Building a star schema*. New York: Prentice-Hall.

Golfarelli, M., Maio, D. & Rizzi, S. (1998). The Dimensional Fact Model: a Conceptual Model for Data Warehouse. *International Journal of Cooperative Information Systems*, 7(2&3), 215-247.

Golfarelli, M., Rizzi, S. & Vrdoljak, B. (2001). Data warehouse design from XML sources. *Proceedings of the ACM Fourth International Workshop on Data Warehousing and OLAP (DOLAP'01)*, Atlanta, USA, 40-47.

Kimball, R. & Ross, M. (2002). *The Data Warehouse Toolkit* (2nd ed.). New York: John-Wiley & Sons.

Lehner, W. (1998). Modeling Large Scale OLAP Scenarios. *Proceedings of the Sixth International Conference On Extending Database Technology (EDBT'98)*, Valencia, Spain, Lecture Notes in computer Science (LNCS), 1377, 153-167.

Luján-Mora, S., Trujillo, J. & Song, I.-Y. (2002). Multidimensional Modeling with UML Package Diagrams". *Proceedings of the Twenty-first International Conference on*

Conceptual Modeling (ER'02), Tampere, Finland, Lecture Notes in Computer Science, 2503, 199-213.

Object Management Group (OMG) (2001). *Unified Modeling Language Specification 1.4*. [Online]. Available: http://www.omg.org.

Pokorný, J. (2001). Modelling Stars Using XML. *Proceedings of the ACM Fourth International Workshop on Data Warehousing and OLAP (DOLAP'01)*, Atlanta, USA, 24-31.

Sapia, C., Blaschka, M., Höfling, G. & Dinter, B. (1998). Extending the E/R Model for the Multidimensional Paradigm. *ER Workshops 1998*, Singapore, Lecture Notes in Computer Science (LNCS), 1552, 105-116.

Sapia, C. (1999). On Modeling and Predicting Query Behavior in OLAP Systems. *Proceedings of the First International Workshop on Design and Management of Data Warehouse (DWDM'99)*, Heidelberg, Germany.

Trujillo, J., Palomar, M. & Gomez, J. (2000). Modeling the Behavior of OLAP Applications Using an UML Compliant Approach. *Proceeedings of the First Conference on Advances in Information Systema (ADVIS'00).* Izmir, Turkey, Lecture Notes in Computer Science (LNCS), 1909, 14-23.

Trujillo, J (2001a). *The GOLD model: an object oriented conceptual model for the design of OLAP applications*. Unpublished doctoral dissertation, Universidad de Alicante, Alicante, Spain.

Trujillo, J., Palomar, M., Gomez, J. & Song, I.-Y. (2001b). Designing Data Warehouses with OO conceptual models. *IEEE Computer, special issue on Data Warehouses*, 34 (12), 66-75.

Tryfona, N., Busborg, F. & Christiansen, J.G.B. (1999). starER: A Conceptual Model for Data Warehouse Design. *Proceedings of the ACM Second International Workshop on Data Warehousing and OLAP (DOLAP'99)*, Kansas City, USA, 3-8.

World Wide Web Consortium (W3C) (2000, October). *Extensible Markup Language (XML) 1.0* (2nd ed.). [Online]. Available: http://www.w3.org/TR/2000/ REC-xml-20001006.

Warmer, J. & Kleppe, A. (1998). *The Object Constraint Language: Precise Modeling with UML*. New York: Addison-Wesley.

Appendix 1

In this section we include the whole DTD that we have defined to represent MD models in XML. This DTD allows us to represent both structural and dynamic properties of MD models and initial requirements (cube classes).

```
<!ENTITY % Boolean '(true|false)'>
<!ENTITY % Multiplicity '(0|1|M|1..M)'>
<!ENTITY % Operator '(eq|lt|gt|let|get|noteq|like|notlike|in|notin)'>
<!ELEMENT MDMODEL (PKSCHEMAS, DEPENDENCIES, CUBECLASSES)>
<!ATTLIST MDMODEL
        id ID #REQUIRED
        name CDATA #REQUIRED
        creationDate CDATA #IMPLIED
        lastModified CDATA #IMPLIED
        description CDATA #IMPLIED
        responsible CDATA #IMPLIED>
<!ELEMENT PKSCHEMAS (PKSCHEMA*)>
<!ELEMENT PKSCHEMA (PKFACT?, PKDIMS, IMPPKDIMS, DEPENDENCIES)>
<!ATTLIST PKSCHEMA
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT PKFACT (FACTCLASS?)>
<!ATTLIST PKFACT
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT PKDIMS (PKDIM*)>
<!ELEMENT PKDIM (DIMCLASS?, BASECLASSES, IMPBASECLASSES)>
<!ATTLIST PKDIM
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT IMPPKDIMS (IMPPKDIM*)>
<!ELEMENT IMPPKDIM (IMPDIMCLASS?, BASECLASSES, IMPBASECLASSES)>
<!ATTLIST IMPPKDIM
        id ID #REQUIRED
        pkdim IDREF #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT IMPDIMCLASS (RELATIONASOCS, RELATIONCATS)>
<!ATTLIST IMPDIMCLASS
        id ID #REQUIRED
        dimclass IDREF #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED
        isTime %Boolean; "false">
<!ELEMENT DEPENDENCIES (DEPENDENCY*)>
<!ELEMENT DEPENDENCY EMPTY>
<!ATTLIST DEPENDENCY
        id ID #REQUIRED
        start IDREF #REQUIRED
        end IDREF #REQUIRED>
<!ELEMENT FACTCLASS (FACTATTS, METHODS, SHAREDAGGS)>
<!ATTLIST FACTCLASS
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT FACTATTS (FACTATT*)>
```

```
<!ELEMENT FACTATT (ADDITIVITY*)>
<!ELEMENT ADDITIVITY EMPTY>
<!ATTLIST ADDITIVITY
        id ID #REQUIRED
        dimclass IDREF #REQUIRED
        isNOT %Boolean; "false"
        isSUM %Boolean; "false"
        isMAX %Boolean; "false"
        isMIN %Boolean; "false"
        isAVG %Boolean; "false"
        isCOUNT %Boolean; "false">
<!ATTLIST FACTATT
        id ID #REQUIRED
        name CDATA #REQUIRED
        atomic %Boolean; "true"
        type CDATA #REQUIRED
        description CDATA #IMPLIED
        initial CDATA #IMPLIED
        derivationRule CDATA #IMPLIED
        OID %Boolean; "false">
<!ELEMENT METHODS (METHOD*)>
<!ELEMENT METHOD EMPTY>
<!ATTLIST METHOD
        id ID #REQUIRED
        name CDATA #REQUIRED>
<!ELEMENT SHAREDAGGS (SHAREDAGG*)>
<!ELEMENT SHAREDAGG EMPTY>
<!ATTLIST SHAREDAGG
        id ID #REQUIRED
        dimclass IDREF #REQUIRED
        name CDATA #IMPLIED
        description CDATA #IMPLIED
        roleA %Multiplicity; "M"
        roleB %Multiplicity; "1">
<!ELEMENT DIMCLASS (DIMATTS, RELATIONASOCS, RELATIONCATS, METHODS)>
<!ELEMENT DIMATTS (DIMATT*)>
<!ELEMENT DIMATT EMPTY>
<!ATTLIST DIMATT
        id ID #REQUIRED
        name CDATA #REQUIRED
        atomic %Boolean; "true"
        type CDATA #REQUIRED
        description CDATA #IMPLIED
        initial CDATA #IMPLIED
        derivationRule CDATA #IMPLIED
        OID %Boolean; "false"
        D %Boolean; "false">
<!ELEMENT BASECLASSES (BASECLASS*)>
<!ELEMENT BASECLASS (DIMATTS, (RELATIONASOCS | RELATIONCATS)?, METHODS)>
<!ELEMENT RELATIONASOCS (RELATIONASOC*)>
<!ELEMENT RELATIONASOC EMPTY>
<!ATTLIST RELATIONASOC
        id ID #REQUIRED
        child IDREF #REQUIRED
        name CDATA #IMPLIED
        description CDATA #IMPLIED
        roleA %Multiplicity; "1"
        roleB %Multiplicity; "M"
        completeness %Boolean; "false">
<!ELEMENT RELATIONCATS (RELATIONCAT*)>
<!ELEMENT RELATIONCAT EMPTY>
<!ATTLIST RELATIONCAT
        id ID #REQUIRED
        child IDREF #REQUIRED
        name CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ATTLIST BASECLASS
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
```

```
<!ATTLIST DIMCLASS
        id ID #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED
        isTime %Boolean; "false">
<!ELEMENT IMPBASECLASSES (IMPBASECLASS*)>
<!ELEMENT IMPBASECLASS (RELATIONASOCS | RELATIONCATS)?>
<!ATTLIST IMPBASECLASS
        id ID #REQUIRED
        baseclass IDREF #REQUIRED
        name CDATA #REQUIRED
        caption CDATA #IMPLIED
        description CDATA #IMPLIED>
<!ELEMENT CUBECLASSES (CUBECLASS*)>
<!ELEMENT CUBECLASS (MEASURES, SLICES, DICES)>
<!ELEMENT MEASURES (MEASURE*)>
<!ELEMENT MEASURE EMPTY>
<!ATTLIST MEASURE
        factatt IDREF #REQUIRED>
<!ELEMENT SLICES (SLICE*)>
<!ELEMENT SLICE EMPTY>
<!ATTLIST SLICE
        dimattORlevel IDREF #REQUIRED
        operator %Operator; #REQUIRED
        value CDATA #REQUIRED>
<!ELEMENT DICES (DICE*)>
<!ELEMENT DICE EMPTY>
<!ATTLIST DICE
        level IDREF #REQUIRED>
<!ATTLIST CUBECLASS
        id ID #REQUIRED
        name CDATA #REQUIRED
        creationDate CDATA #IMPLIED
        lastModified CDATA #IMPLIED
        description CDATA #IMPLIED>
```