DESIGN AND REPRESENTATION OF MULTIDIMENSIONAL MODELS WITH UML AND XML TECHNOLOGIES

Name: Juan Trujillo

Affiliation: Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante

Address: Campus de San Vicente del Raspeig, Ap. Correos 99, E-03080 Alicante, Spain

Phone number: +34 965 90 34 00 ext. 2967

Fax number: +34 965 90 93 26

Email: jtrujillo@dlsi.ua.es

Name: Sergio Luján-Mora

Affiliation: Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante

Address: Campus de San Vicente del Raspeig, Ap. Correos 99, E-03080 Alicante, Spain

Phone number: +34 965 90 34 00 ext. 2962

Fax number: +34 965 90 93 26

Email: slujan@dlsi.ua.es

Name: Il-Yeol Song

Affiliation: College of Information Science and Technology, Drexel University

Address: PA 19104, USA

Phone number: +1 (215) 895-2489

Fax number: +1 (215) 895-2494

Email: songiy@drexel.edu

DESIGN AND REPRESENTATION OF MULTIDIMENSIONAL MODELS WITH UML AND XML TECHNOLOGIES

Data warehouses (DW), multidimensional databases (MDB), and On-Line Analytical Processing (OLAP) applications are based on the Multidimensional (MD) modeling. Most of these applications provide their own MD models to represent main MD properties, thereby making the design totally dependent of the target commercial application. In this chapter, we present how the Unified Modeling Language (UML) can be successfully used to abstract the representation of MD properties at the conceptual level. Then, from this conceptual model, we generate its corresponding implementation into any market OLAP tool. In our approach, the structure of the system is specified by means of a UML class diagram that considers main properties of MD modeling. If the system to be modeled is too complex, we describe how to use the *package* grouping mechanism provided by the UML to simplify the final model. To facilitate the interchange of conceptual MD models, we provide an eXtensible Markup Language (XML) Schema which allows us to represent the same MD modeling properties that can be considered by using our approach. From this XML Schema, we can directly generate valid XML documents that represent MD models at the conceptual level. Finally, we provide different presentations of the MD models by means of eXtensible Stylesheet Language Transformations (XSLT).

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

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 previously 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, 1998; OMG, 2001) needed for an adequate representation of MD modeling properties (Trujillo, 2001a; Trujillo,

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, 2002). Furthermore, we have provided a UML-compliant class notation to represent OLAP users' initial requirements (called *cube class*). 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, 1998; OMG, 2001) and the Object Query Language (OQL) (Cattell, 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. XML documents can be associated to a Document Type Definition (DTD) (W3C, 2000) or an XML Schema (W3C, 2001), both of which allow us to describe and constraint the structure of

XML documents. Moreover, thanks to the use of eXtensible Stylesheet Language Transformations (XSLT) (W3C, 1999), users can express their intentions about how XML documents should be presented, so they could be automatically transformed into other formats, e.g. HTML documents. An immediate consequence is that we can define different XSLT stylesheets to provide different presentations of the same XML document. In a previous work (Trujillo, 2004), we have presented a DTD for the representation of MD models and this DTD is then used to automatically validate XML documents.

From these considerations, in this chapter 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, 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 an XML Schema that defines the correct structure and content of a XML document representing main MD properties. Moreover, we also address the presentation of MD models on the web by means of eXtensible Stylesheet Language Transformations (XSLT): we provide XSLT stylesheets that allow us to automatically generate HTML pages from XML documents that represent MD models, thereby supporting different presentations of the same MD model easily. 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 chapter 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 XML Schema from UML. In Section 5, we present a set of case studies taken from Kimball [Kimball02] to show the benefit of our approach. Finally, Section 6 draws conclusions and sketches out new research that is currently being investigated.

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, 1998; Golfarelli, 1998; Kimball, 2002; Trujillo, 2001b; Tryfona, 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, 2002).

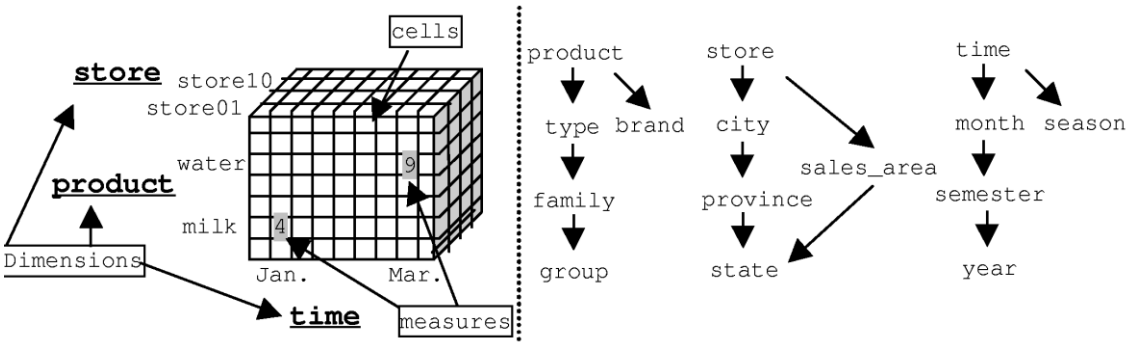

Figure 1: A data cube and classification hierarchies defined on dimensions

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, 1999). "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, 1999). 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, 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).

Star Schema

In this sub-section, we will summarize the star schema popularized by Kimball (Kimball, 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.

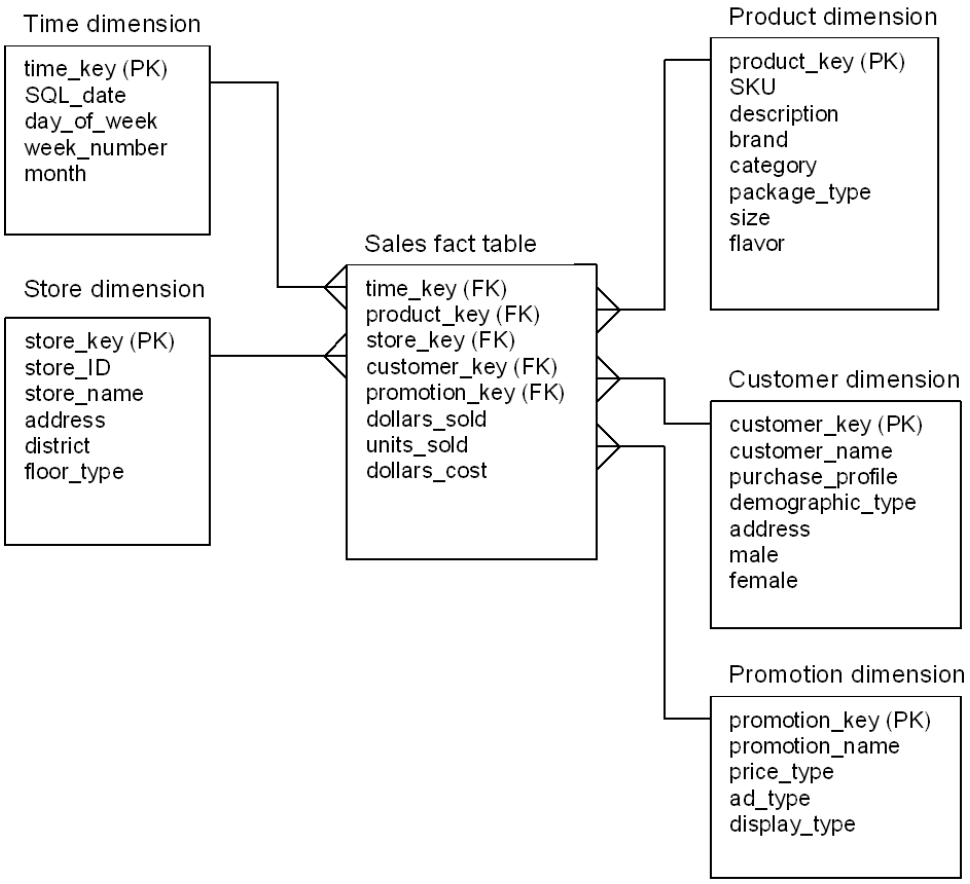

Figure 2: Sales Dimension Model

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 R. Kimball (Kimball, 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, 1998; Abello, 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 (1998), The Multidimensional/ER (M/ER) model by Sapia (1998; 1999) and The starER model by Tryfona (1999).

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, 2001) propose 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.

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 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, 2000). Then, Section 4.4 presents the corresponding representation of our approach into the XML (W3C, 2000) to allow us an easy interchange of MD information. Finally, Section 4.5 describes how to use XSLT stylesheets to automatically generate HTML pages from XML documents, thereby allowing us to manage different presentations of MD models in the Web.

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-Mora, 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, 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; Kimball, 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).

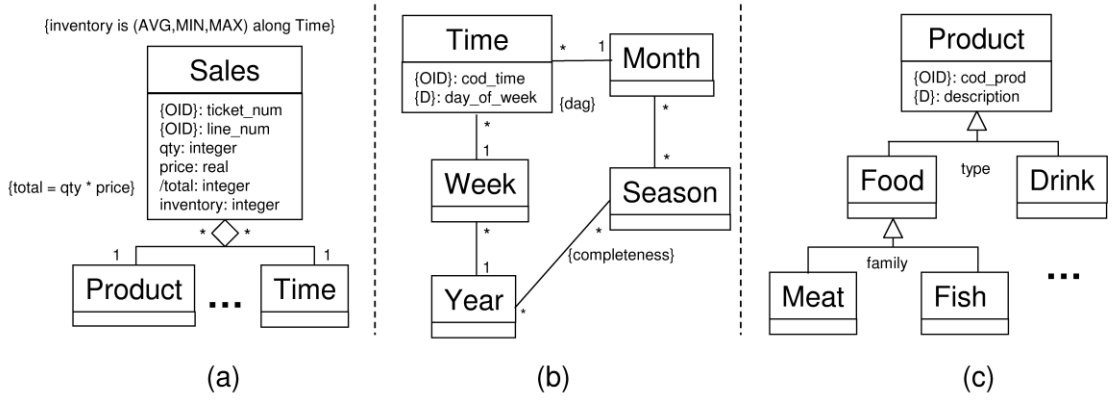

Figure 3: Multidimensional modeling using UML

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).

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

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 users' initial requirements.

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.

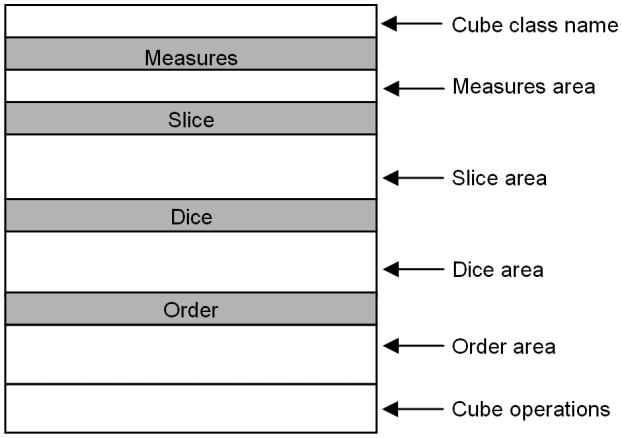

Figure 4: Cube class structure

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 4.3 for more detail on the representation of cube classes by using OQL).

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, 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[2] proposed by the Object Database Management Group (ODMG) (Cattell, 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 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 chapter, 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

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

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, 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 5 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 6. Finally, Figure 7 represents the content of the *Product* dimension package, and Figure 8 the content of the *Time* dimension package.

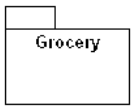

Figure 5: First level

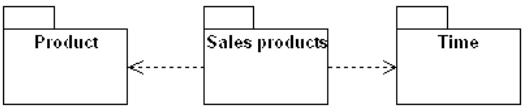

Figure 6: Second level

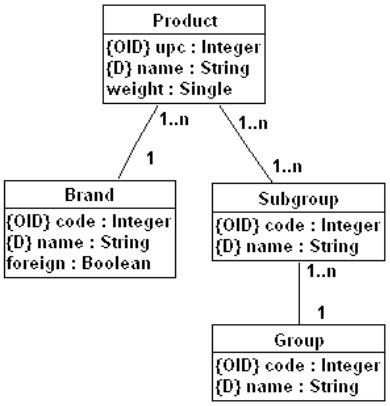

Figure 7: Product dimension (third level)

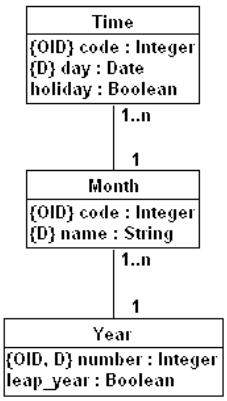

Figure 8: Time dimension (third level)

In Figure 9, 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.

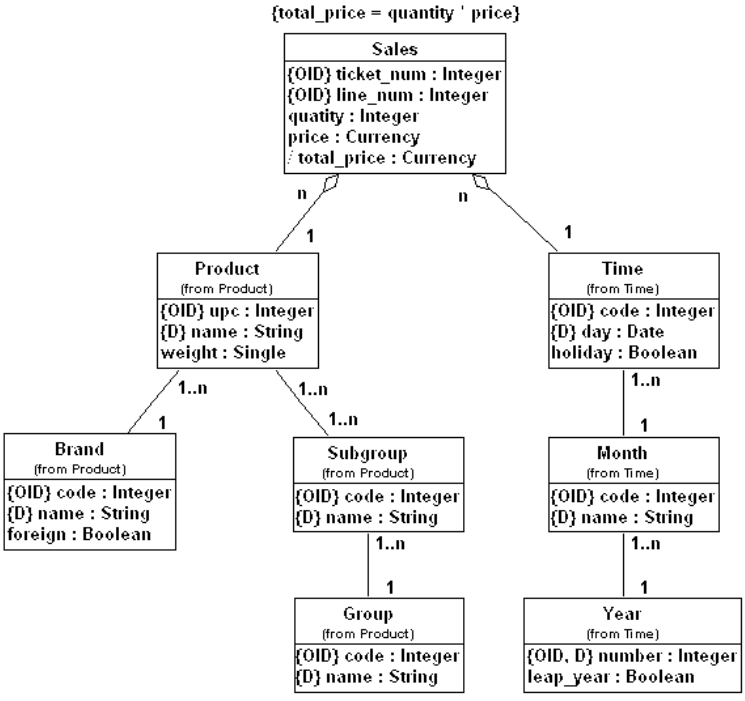

Figure 9: Third level of the MD model

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 10, 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.

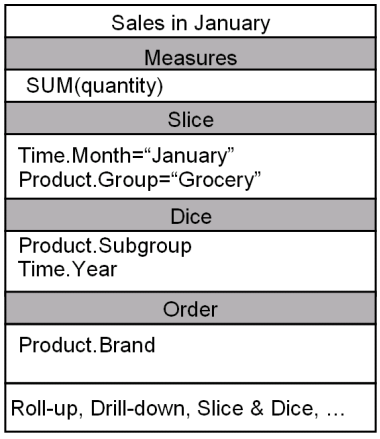

Figure 10: An example of a user's initial requirement

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 10 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 10 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
```

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 an XML Schema (W3C, 2001) that determines the correct structure and content of XML documents that represent MD models. Moreover, this XML Schema can be used to automatically validate the XML documents. In Appendix 1 we include the whole XML Schema that we have defined to represent MD models in XML. This XML Schema allows us to represent both structural and dynamic properties of MD models.

In Figure 11, Figure 12, and Figure 13[3], we have graphically represented the main rules of our XML Schema, which contains the definition of 25 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 XML Schema follows the three-level structure of our MD approach:

- An `MDMODEL` contains `PKSCHEMAS` (star schema packages) at level 1 (Figure 11).
- A `PKSCHEMA` contains at most one `PKFACT` (fact package) and many `PKDIM` (dimension packages) grouped by a `PKDIMS` element at level 2 (Figure 12).
- A `PKFACT` contains at most one `FACTCLASS` (Figure 12) and a `PKDIM` contains at most one `DIMCLASS` and many `BASECLASSES` (Figure 13) at level 3.

[3] In these figures we use the following notation: the box with three linked dots represents a sequence of elements, the range in which an element can occur is showed with numbers (the default minimum and

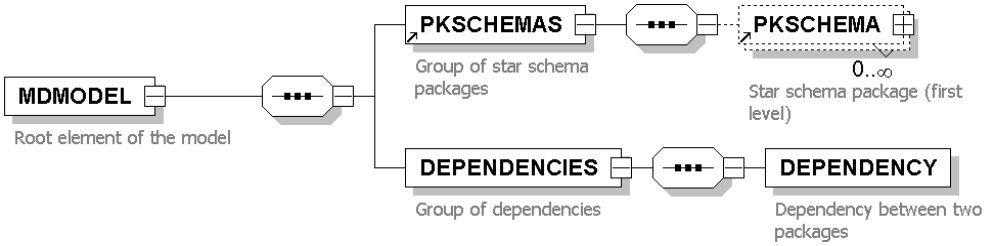

Figure 11: MDMODEL element

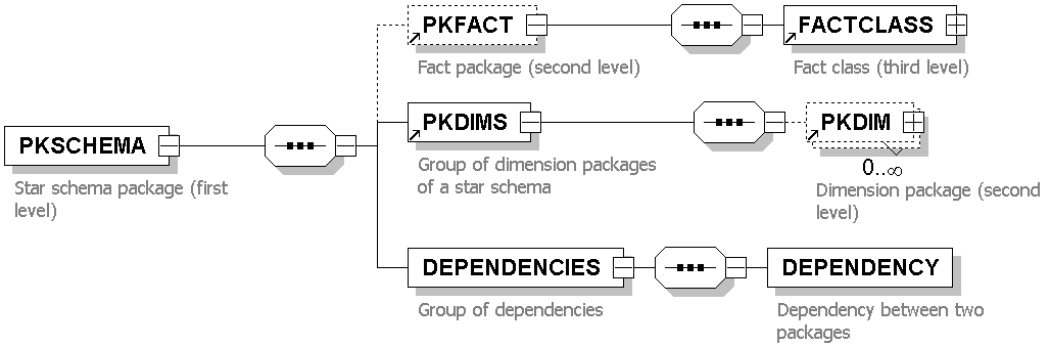

Figure 12: PKSCHEMA element

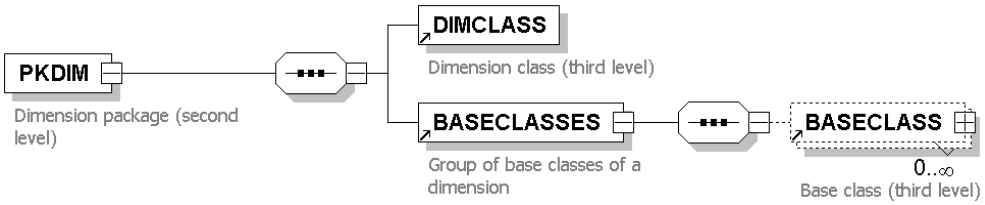

Figure 13: PKDIM element

Within our XML Schema, 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` (0 or more `FACTATT`):

---

maximum number of occurrences is 1) and graphically (a box with a dashed line indicates that the

```
<xs:element name="FACTATTS">
<xs:annotation>
<xs:documentation>Group of attributes of a fact class</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="FACTATT" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

From now on, we are going to explain the structure of our XML Schema 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. Then, the second line describes the root element of the document (`MDMODEL`) and declares the XML Schema that defines the structure of the document. An `MDMODEL` element contains two attributes: `id` and `name`. Finally, an MD model (Figure 11) contains star schema packages (`PKSCHEMAS`) with dependencies between them (`DEPENDECIES`).

In this example, the MD model only contains one star schema package (Figure 5); as there is not any dependency between star schema packages the `DEPENDENCIES` element is empty.

```
<?xml version="1.0" encoding="UTF-8"?>
<MDMODEL id="ID5" name="Grocery example" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="mdmodel.xsd">
  <PKSCHEMAS>
    <PKSCHEMA id="ID6" name="Grocery" caption="Grocery">
    ...
```

---

minimum number of occurrences is 0).

```
    </PKSCHEMA>
  </PKSCHEMAS>
  <DEPENDENCIES/>
</MDMODEL>
```

In our XML Schema, 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` (Figure 12) can contain:

- At most one `PKFACT`.
- 0 or more dimension packages (`PKDIM`) defined in the very star schema.
- 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 XML Schema, 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>
  <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 (Figure 13) contains:

- At most one dimension class (`DIMCLASS`).
- 0 or more base classes (`BASECLASS`) that represent hierarchy levels and grouped by a `BASECLASSES` element.

In this fragment we can see how the relationships between a dimension class and base classes are expressed in our XML Schema; 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 XML Schema, the *{OID}* and *{D}* constraints of our MD model are represented as boolean 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>
```

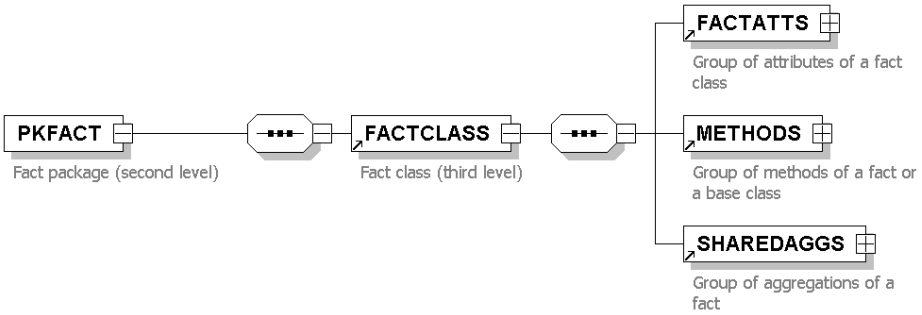

Figure 14: PKFACT element

Finally, a fact package (Figure 12) contains at most one fact class, and each fact class (Figure 14) 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 XML Schema 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>
```

Different Presentations Of MD Models

Another relevant issue of our approach was to provide different presentations of the MD models in the Web. To solve this problem, XSL Transformations (XSLT) (W3C, 1999) is a technology that allows us to define the presentation for XML documents. XSLT stylesheets describe a set of patterns (templates) to match both elements and attributes defined in an XML Schema, in order to apply specific transformations for each considered match. Thanks to XSLT, the source document can be filtered and reordered in constructing the resulting output.

Figure 15 illustrates the overall transformation process for a MD model. The MD model is stored in an XML document and an XSLT stylesheet is provided to generate different presentations of the MD model: e.g., as a Portable Document Format (PDF) file or as a HyperText Markup Language (HTML) document.

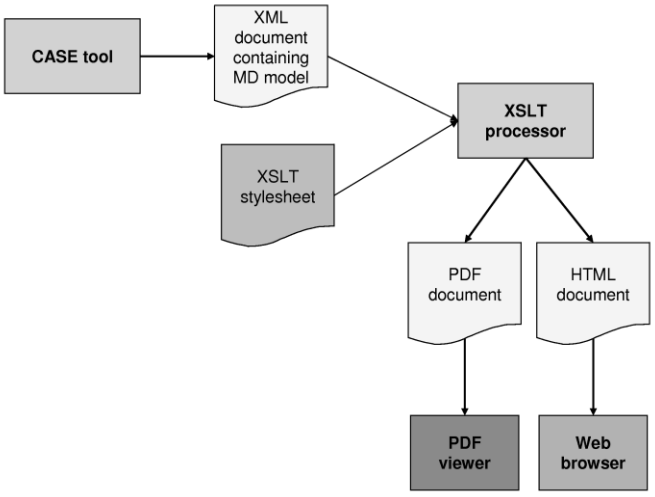

Figure 15: Generating different presentations from the same MD model

Due to space constraints, it is not possible to include the complete definition of the XSLT stylesheet here. Therefore, we only exhibit some fragments of the XSLT. The first example shows the instructions that generate the HTML code to display information about fact attributes (FACTATT):

```
<xsl:if test="FACTATTS/FACTATT">
 <tr>
  <td class="data">
  <table class="data" cellpadding="2" cellspacing="2" border="0" width="100%">
  <tr>
   <th class="name">Name</th>
   <th class="name">Type</th>
   <th class="name">Initial</th>
   <th class="name">Derivation Rule</th>
   <th class="name">DD</th>
  </tr>
<xsl:for-each select="FACTATTS/FACTATT">
  <tr>
   <td class="value"><xsl:value-of select="@name"/></td>
   <td class="value"><xsl:value-of select="@type"/></td>
   <td class="value"><xsl:value-of select="@initial"/></td>
```

```
   <td class="value"><xsl:value-of select="@derivationRule"/></td>
   <td class="value">
<xsl:choose>
 <xsl:when test="@DD"><xsl:value-of select="@DD"/></xsl:when>
 <xsl:otherwise>false</xsl:otherwise>
</xsl:choose>
   </td>
  </tr>
</xsl:for-each>
  </table>
  </td>
 </tr>
</xsl:if>
```

Notice that XSLT instructions (highlighted in bold typeface) and HTML tags are intermingled. The XSLT processor copies the HTML tags to the transformed document and interprets any XSLT instruction encountered. Applied to this example, the value of the attributes of the element `FACTATT` are inserted in the resulting document in an HTML table.

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 (Kimball, 2002), 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.

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 16 shows the first level of the model formed by three packages that represent the different star schemas in the case study.

Figure 16: 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 17 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 17, 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.

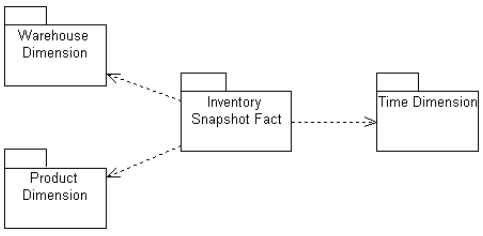

Figure 17: Level 2 of Inventory Snapshot Star

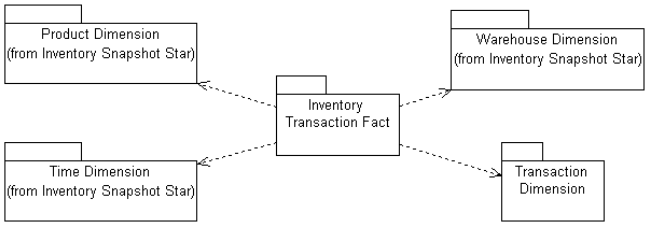

Figure 18: Level 2 of Inventory Transaction Star

Figure 18 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 17), and they are imported in this package. Therefore, the name of the package where they have 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 19 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 19, Warehouse presents alternative path classification hierarchies: (i) *ZIP*, *City*, *County*, *State*, and (ii) *SubRegion*, *Region*, *State*.

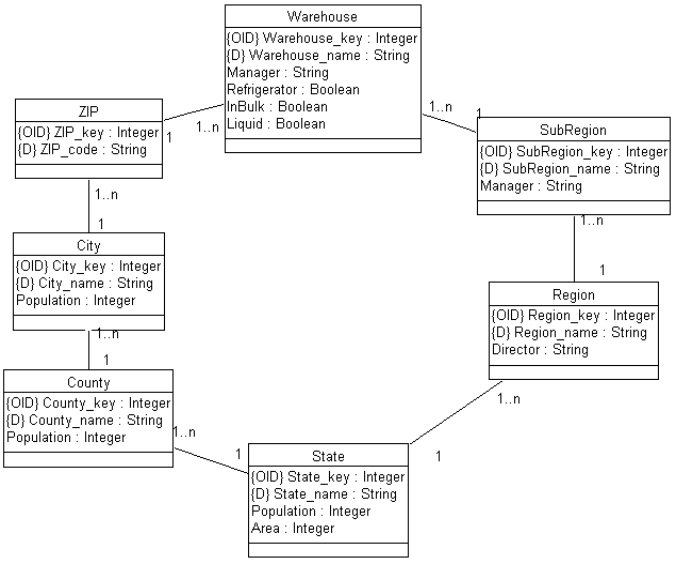

Figure 19: Level 3 of Warehouse Dimension

Finally, Figure 20 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 (nu*m*ber*_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 20.

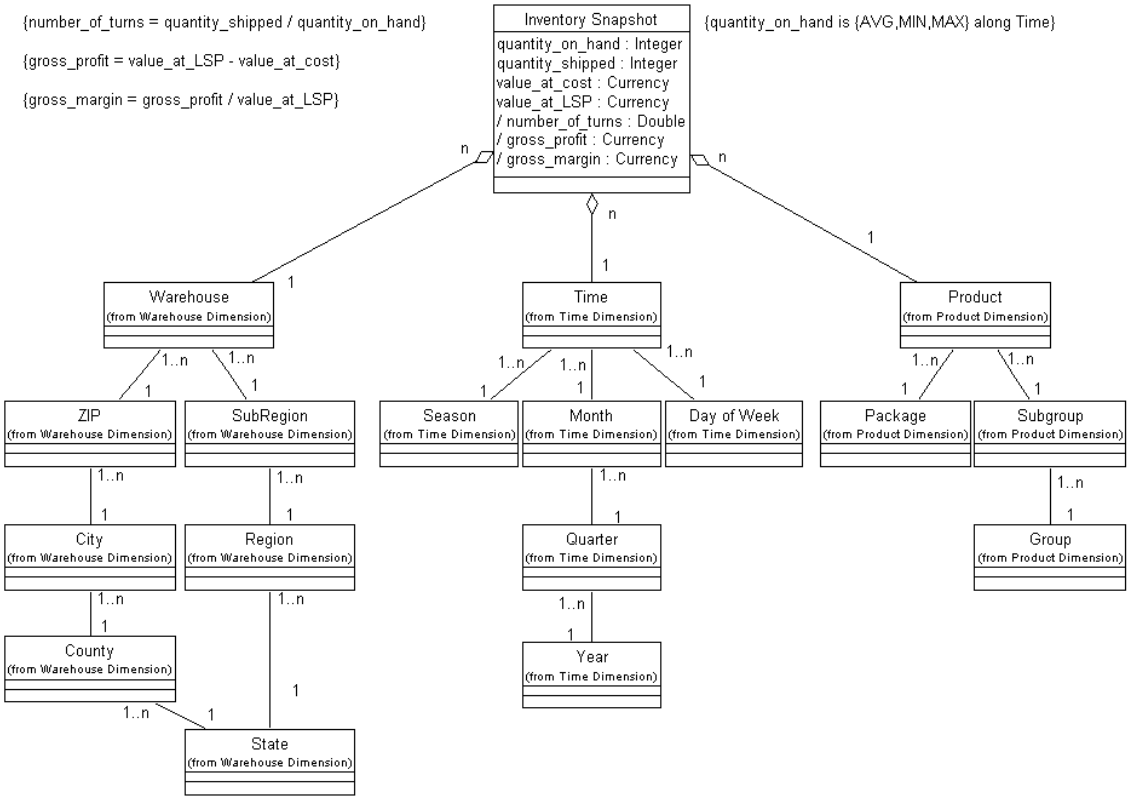

Figure 20: Level 3 of Inventory Snapshot Fact

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 20: '*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 21, 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 (sub*g*roup, re*g*ion, and sub*r*egion) 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 21 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 19).

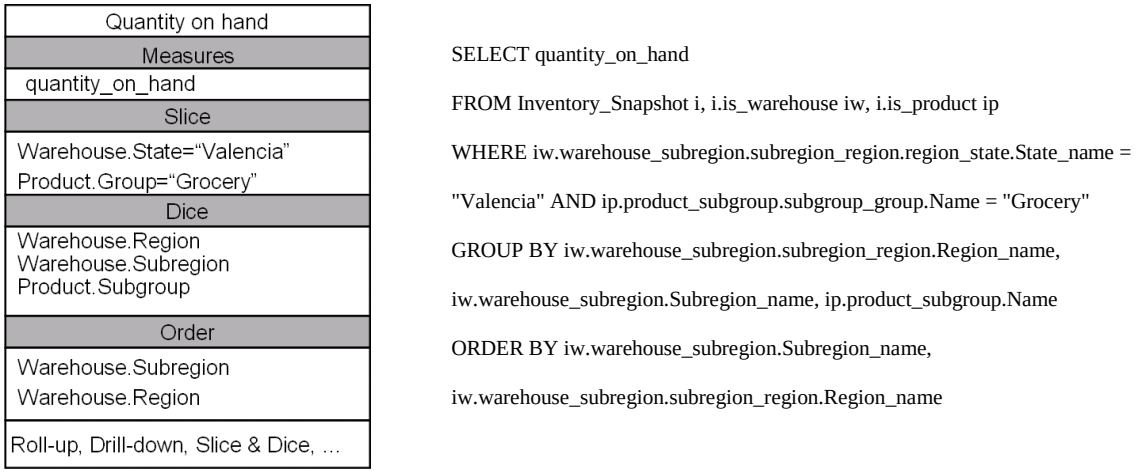

Figure 21: An example of a user's initial requirement

From the MD model stored in an XML document, we can provide different presentations thanks to the use of XSLT stylesheets. For example, we use XSLT stylesheets and XML documents in a transformation process to automatically generate HTML pages that can represent different presentations of the same MD model. As an example of the applicability of our proposal, these HTML pages can be used to document the MD models in the Web, with the advantages that it implies (standardization, access from any computer with a browser, ease of use, etc.). Moreover, the automatic generation of documentation from conceptual models avoids the problem

of documentation out of date (incoherencies, features not reflected in the documentation, etc.).

For example, in Figure 22, we show the definition of Inventory Snapshot Star on a web page. This page contains the general description of a star: name, description, and the names of the fact classes and dimension classes, which are active links that allow us to navigate through the different presentations of the model on a web browser. All the information about the MD properties of the model is represented in the HTML pages.

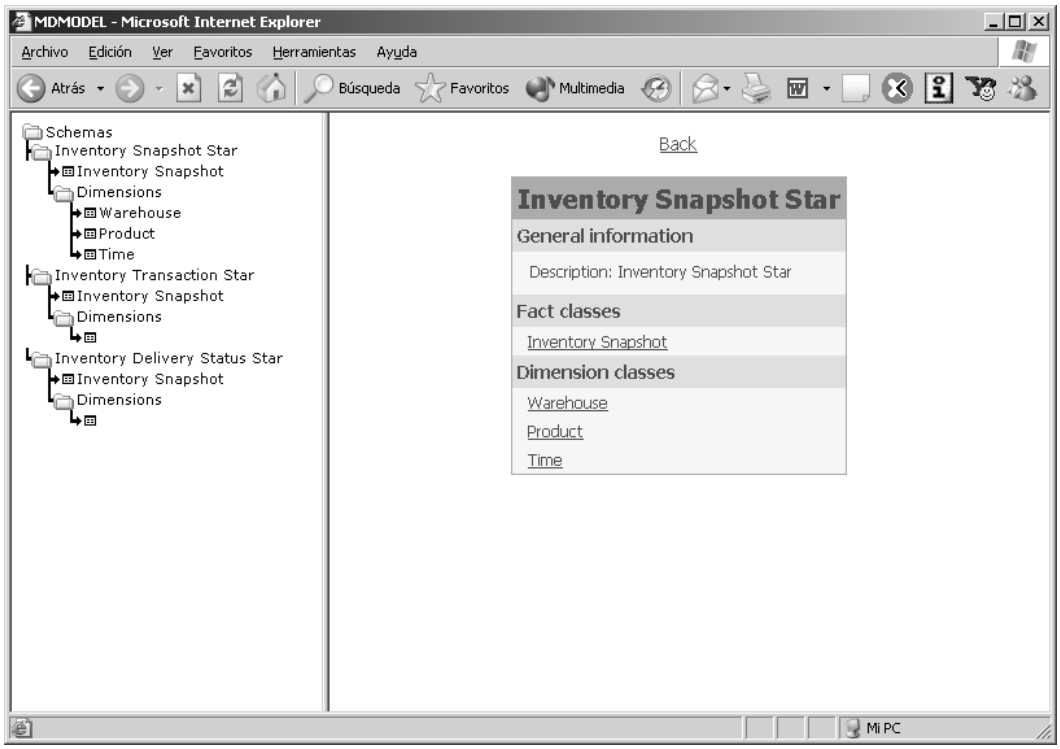

Figure 22: Multidimensional model on the Web

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 23 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 24.

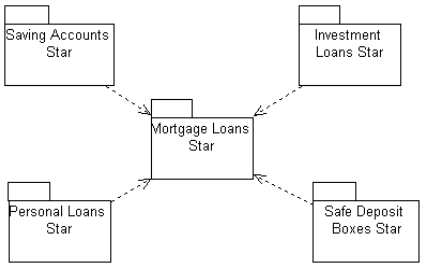

Figure 23: Level 1

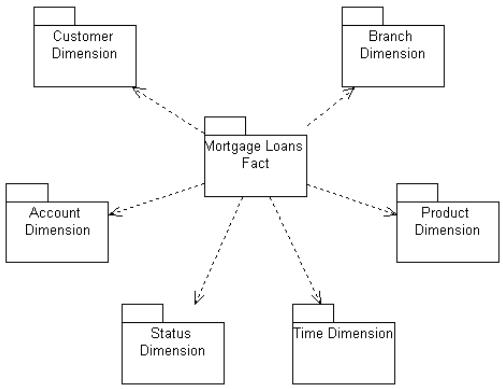

Figure 24: Level 2 of Mortgage Loans Star

Level 3 of *Mortgage Loans Fact* is shown in Figure 25. 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.

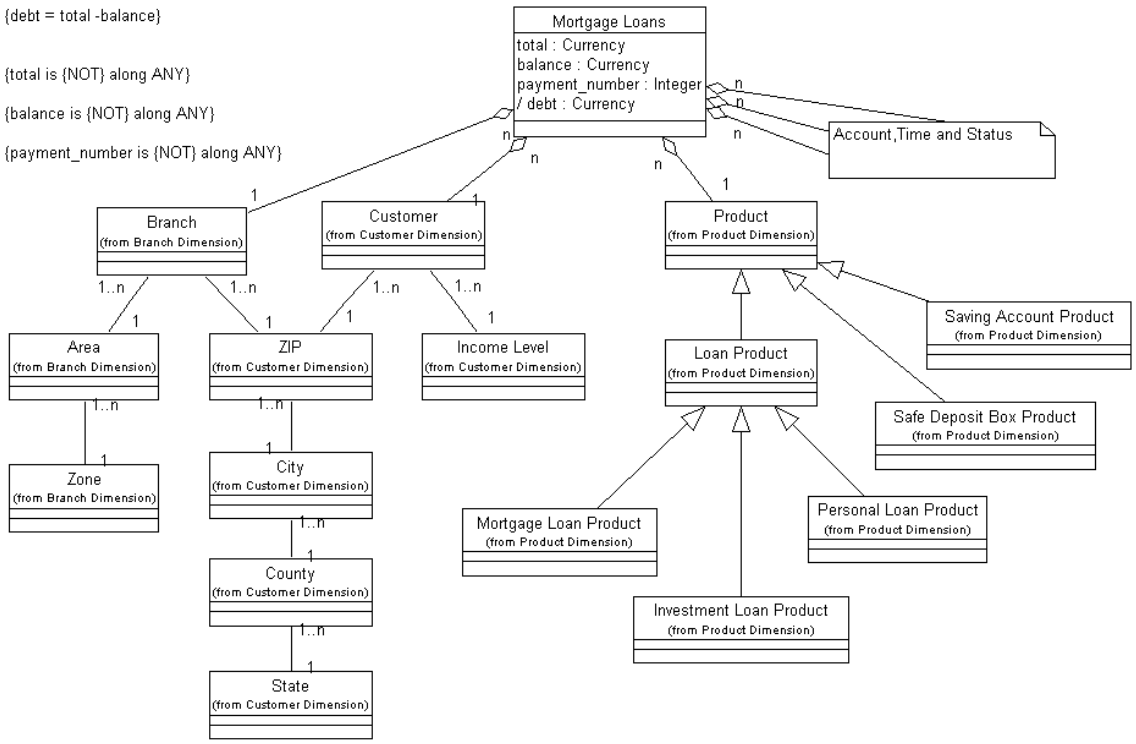

Figure 25: Level 3 of Mortgage Loans Fact

Finally, this MD model can be accessible through the Web thanks to the use of XSLT stylesheets. In Figure 26, we show the definition of the *Mortgage Loans* Star schema. On the left hand side of this figure we can notice a hierarchy index that shows the five star schemas that the MD model comprises. From this web page, if the *Mortgage Loans* link is selected, the page showed in Figure 27 is loaded.

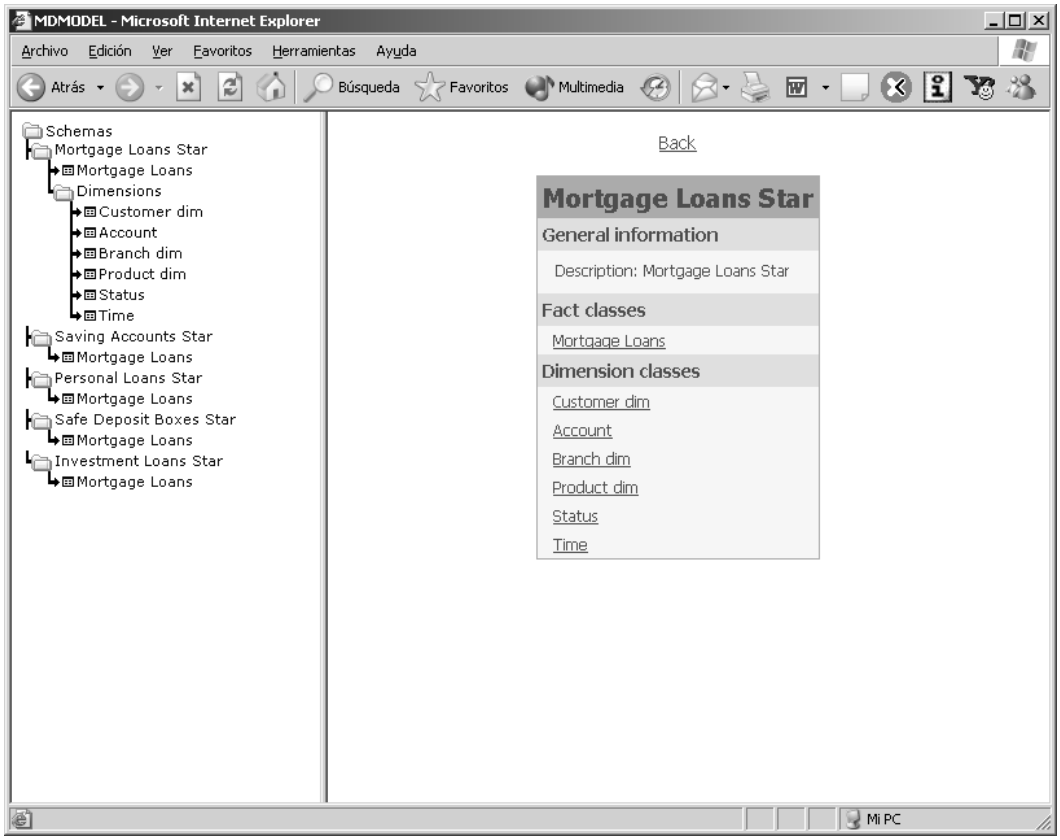

Figure 26: Multidimensional model on the Web

In Figure 27, the definition of Mortgage Loans fact class is shown: the name of the fact class, the measures, methods, and shared aggregations. In this example, *Mortgage Loans* contains three measures: total, balance, and *payment_number*. Moreover, this fact class holds six aggregation relationships with the dimensions *Account*, *Status*, *Time*, *Branch* dim, *Customer* dim, and *Product* dim, which are active links and allow us to continue exploring the model on the Web.

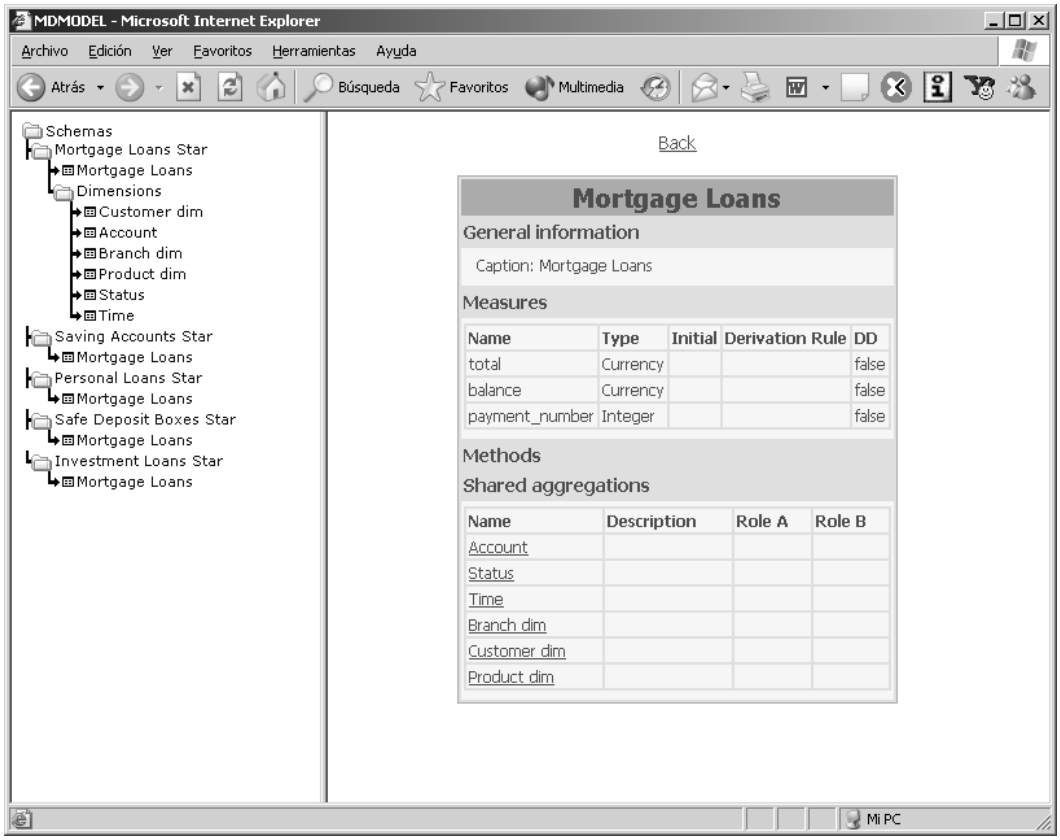

Figure 27: Multidimensional model on the Web

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 28 and Figure 29, level 1 and

level 2 of this model are depicted respectively. In this case, level 1 only contains one star package.

Figure 28: Level 1

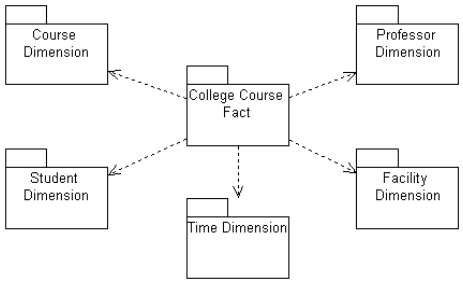

Figure 29: Level 2 of College Course Star

Figure 30 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.

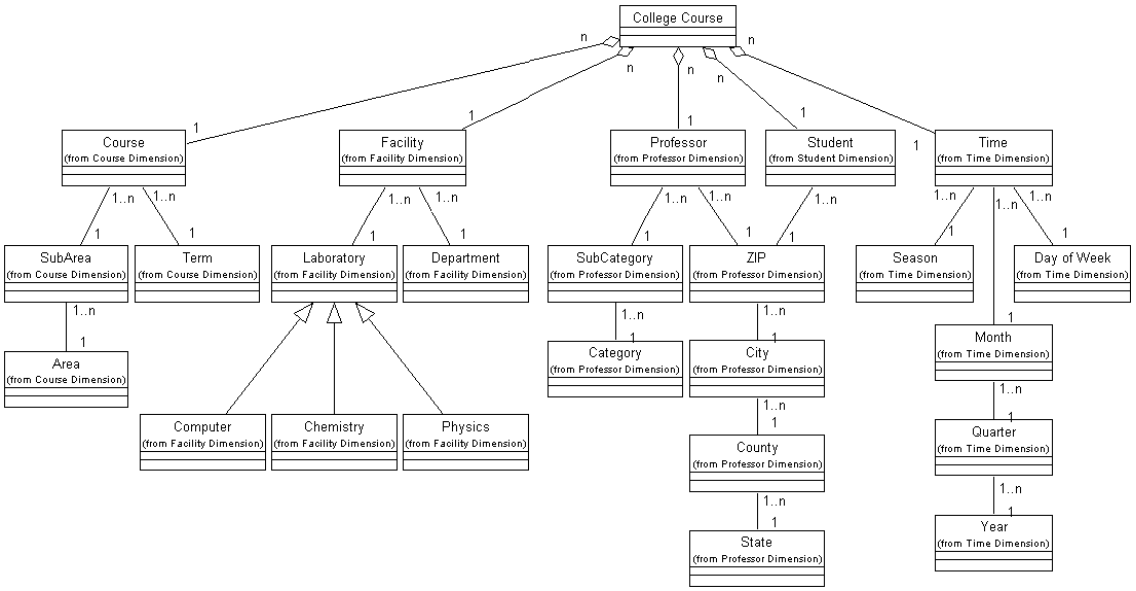

Figure 30: Level 3 of College Course Star

CONCLUSIONS

In this chapter, 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. 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 an XML Schema from which we can obtain valid XML documents. Moreover, we apply XSLT stylesheets in order to provide different presentations of the MD models. 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.

APPENDIX 1

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xs:attributeGroup name="id_name">
<xs:annotation>
<xs:documentation>Common attributes to different elements (id y name)</xs:documentation>
</xs:annotation>
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:attributeGroup>

<xs:attributeGroup name="dim_fact_atts">
<xs:annotation>
<xs:documentation>Common attributes to dimension and fact classes</xs:documentation>
</xs:annotation>
<xs:attribute name="id" type="xs:ID" use="required"/>
```

```
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="derived" type="xs:boolean" default="false"/>
<xs:attribute name="derivationRule" type="xs:string" use="optional"/>
<xs:attribute name="type" type="xs:string" use="required"/>
<xs:attribute name="initial" type="xs:string" use="optional"/>
</xs:attributeGroup>

<xs:element name="MDMODEL">
<xs:annotation>
<xs:documentation>Root element of the model</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="PKSCHEMAS"/>
<xs:element name="DEPENDENCIES">
<xs:annotation>
<xs:documentation>Group of dependencies</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element name="DEPENDENCY">
<xs:annotation>
<xs:documentation>Dependency between two packages</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="start" type="xs:IDREF" use="required"/>
<xs:attribute name="end" type="xs:IDREF" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
<xs:key name="PKSCHEMAKey">
<xs:selector xpath="PKSCHEMAS/PKSCHEMA"/>
<xs:field xpath="@id"/>
</xs:key>
<xs:keyref name="startPKSCHEMAKey" refer="PKSCHEMAKey">
<xs:selector xpath="DEPENDENCIES/DEPENDENCY"/>
<xs:field xpath="@start"/>
</xs:keyref>
<xs:keyref name="endPKSCHEMAKey" refer="PKSCHEMAKey">
<xs:selector xpath="DEPENDENCIES/DEPENDENCY"/>
<xs:field xpath="@end"/>
</xs:keyref>
</xs:element>


<xs:element name="PKSCHEMAS">
<xs:annotation>
<xs:documentation>Group of star schema packages</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="PKSCHEMA" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>


<xs:element name="PKSCHEMA">
<xs:annotation>
<xs:documentation>Star schema package (first level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="PKFACT" minOccurs="0"/>
<xs:element ref="PKDIMS"/>
<xs:element name="DEPENDENCIES">
<xs:annotation>
<xs:documentation>Group of dependencies</xs:documentation>
</xs:annotation>
<xs:complexType>
```

```
<xs:sequence>
<xs:element name="DEPENDENCY">
<xs:annotation>
<xs:documentation>Dependency between two packages</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="start" type="xs:IDREF" use="required"/>
<xs:attribute name="end" type="xs:IDREF" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
<xs:key name="DIMCLASSKey">
<xs:selector xpath="PKDIMS/PKDIM/DIMCLASS"/>
<xs:field xpath="@id"/>
</xs:key>
<xs:keyref name="sharedaggDIMCLASSKey" refer="DIMCLASSKey">
<xs:selector xpath="PKFACT/FACTCLASS/SHAREDAGGS/SHAREDAGG"/>
<xs:field xpath="@dimclass"/>
</xs:keyref>
<xs:key name="PKDIMKey">
<xs:selector xpath="PKDIMS/PKDIM"/>
<xs:field xpath="@id"/>
</xs:key>
<xs:key name="PKFACTKey">
<xs:selector xpath="PKFACT"/>
<xs:field xpath="@id"/>
</xs:key>
<xs:keyref name="startPKDIMKey" refer="PKDIMKey">
<xs:selector xpath="DEPENDENCIES/DEPENDENCY"/>
<xs:field xpath="@start"/>
</xs:keyref>
<xs:keyref name="startPKFACTKey" refer="PKFACTKey">
<xs:selector xpath="DEPENDENCIES/DEPENDENCY"/>
<xs:field xpath="@start"/>
</xs:keyref>
<xs:keyref name="endPKDIMKey" refer="PKDIMKey">
<xs:selector xpath="DEPENDENCIES/DEPENDENCY"/>
<xs:field xpath="@end"/>
</xs:keyref>
</xs:element>

<xs:element name="PKFACT">
<xs:annotation>
<xs:documentation>Fact package (second level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="FACTCLASS"/>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
</xs:element>

<xs:element name="FACTCLASS">
<xs:annotation>
<xs:documentation>Fact class (third level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="FACTATTS"/>
<xs:element ref="METHODS"/>
<xs:element ref="SHAREDAGGS"/>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
</xs:element>

<xs:element name="FACTATTS">
<xs:annotation>
<xs:documentation>Group of attributes of a fact class</xs:documentation>
```

```
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="FACTATT" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="FACTATT">
<xs:annotation>
<xs:documentation>Fact attribute</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attributeGroup ref="dim_fact_atts"/>
<xs:attribute name="DD" type="xs:boolean" default="false"/>
</xs:complexType>
</xs:element>

<xs:element name="DEGFACT">
<xs:annotation>
<xs:documentation>Degenerate fact</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="FACTATTS"/>
<xs:element ref="METHODS"/>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
</xs:element>

<xs:element name="METHODS">
<xs:annotation>
<xs:documentation>Group of methods of a fact or a base class</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="METHOD" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="METHOD">
<xs:annotation>
<xs:documentation>Method of a fact or a base class</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
</xs:element>

<xs:element name="SHAREDAGGS">
<xs:annotation>
<xs:documentation>Group of aggregations of a fact</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="SHAREDAGG" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="SHAREDAGG">
<xs:annotation>
<xs:documentation>Aggregation between a fact and a dimension</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="DEGFACT" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="dimclass" type="xs:IDREF" use="required"/>
<xs:attribute name="name" type="xs:string" use="optional"/>
<xs:attribute name="description" type="xs:string" use="optional"/>
<xs:attribute name="roleA" type="xs:string" use="optional"/>
<xs:attribute name="roleB" type="xs:string" use="optional"/>
```

```
</xs:complexType>
</xs:element>


<xs:element name="PKDIMS">
<xs:annotation>
<xs:documentation>Group of dimension packages of a star schema</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="PKDIM" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="PKDIM">
<xs:annotation>
<xs:documentation>Dimension package (second level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="DIMCLASS"/>
<xs:element ref="BASECLASSES"/>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
<xs:key name="BASECLASSKey">
<xs:selector xpath="BASECLASSES/BASECLASS"/>
<xs:field xpath="@id"/>
</xs:key>
<xs:keyref name="dimclassBASECLASSKey" refer="BASECLASSKey">
<xs:selector xpath="DIMCLASS"/>
<xs:field xpath="@baseclass"/>
</xs:keyref>
<xs:keyref name="relationasocBASECLASSKey" refer="BASECLASSKey">
<xs:selector xpath="BASECLASSES/BASECLASS/RELATIONASOCS/RELATIONASOC"/>
<xs:field xpath="@child"/>
</xs:keyref>
<xs:keyref name="relationcatBASECLASSKey" refer="BASECLASSKey">
<xs:selector xpath="BASECLASSES/BASECLASS/RELATIONCATS/RELATIONCAT"/>
<xs:field xpath="@child"/>
</xs:keyref>
</xs:element>

<xs:element name="DIMCLASS">
<xs:annotation>
<xs:documentation>Dimension class (third level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attributeGroup ref="id_name"/>
<xs:attribute name="baseclass" type="xs:IDREF" use="optional"/>
<xs:attribute name="isTime" type="xs:boolean" default="false"/>
</xs:complexType>
</xs:element>

<xs:element name="BASECLASSES">
<xs:annotation>
<xs:documentation>Group of base classes of a dimension</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="BASECLASS" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="BASECLASS">
<xs:annotation>
<xs:documentation>Base class (third level)</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="DIMATTS"/>
<xs:choice minOccurs="0">
<xs:element ref="RELATIONASOCS"/>
<xs:element ref="RELATIONCATS"/>
```

```
</xs:choice>
<xs:element ref="METHODS"/>
</xs:sequence>
<xs:attributeGroup ref="id_name"/>
</xs:complexType>
</xs:element>

<xs:element name="DIMATTS">
<xs:annotation>
<xs:documentation>Group of attributes of a base class</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="DIMATT" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="DIMATT">
<xs:annotation>
<xs:documentation>Attribute of a base class</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attributeGroup ref="id_name"/>
<xs:attribute name="derived" type="xs:boolean" default="false"/>
<xs:attribute name="derivationRule" type="xs:string" use="optional"/>
<xs:attribute name="type" type="xs:string" use="required"/>
<xs:attribute name="initial" type="xs:string" use="optional"/>
<xs:attribute name="OID" type="xs:boolean" default="false"/>
<xs:attribute name="D" type="xs:boolean" default="false"/>
</xs:complexType>
</xs:element>

<xs:element name="RELATIONASOCS">
<xs:annotation>
<xs:documentation>Group of relationships between classes</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="RELATIONASOC" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="RELATIONASOC">
<xs:annotation>
<xs:documentation>Relationship between two classes</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attribute name="ID" type="xs:ID" use="required"/>
<xs:attribute name="child" type="xs:IDREF" use="required"/>
<xs:attribute name="name" type="xs:string" use="optional"/>
<xs:attribute name="roleA" type="xs:string" use="optional"/>
<xs:attribute name="roleB" type="xs:string" use="optional"/>
<xs:attribute name="completeness" type="xs:boolean" default="false"/>
</xs:complexType>
</xs:element>

<xs:element name="RELATIONCATS">
<xs:annotation>
<xs:documentation>Group of categorization relationships between classes</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="RELATIONCAT" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="RELATIONCAT">
<xs:annotation>
<xs:documentation>Categorization relationship between two classes</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:attribute name="id" type="xs:ID" use="required"/>
<xs:attribute name="child" type="xs:IDREF" use="required"/>
```

<xs:attribute name="name" type="xs:string" use="optional"/>
</xs:complexType>
</xs:element>
</xs:schema>