Automatically Generating Structural and Dynamic Information of OLAP Applications from Object-Oriented Conceptual Models

Juan Trujillo, Sergio Luján-Mora
Departamento de Lenguajes y Sistemas Informáticos, Universidad de Alicante
Campus de San Vicente del Raspeig
Ap. Correos 99 - 03005 Alicante, Spain
{jtrujillo, slujan}@dlsi.ua.es

Abstract

Graphical conceptual models for On-Line Analytical Processing (OLAP) applications should semi-automatically generate the database schema and the corresponding multidimensional (MD) model for a specific target commercial OLAP tool. However, this generation process is not immediate as the semantics represented by these conceptual models are different from those considered by the underlying MD models of OLAP tools. Therefore, some transformations for these differences are needed in this process.

In the context of graphical conceptual models, we provide an object-oriented conceptual model that provides a Unified Modeling Language (UML) graphical notation to represent both structural and dynamics properties of MD models and initial user requirements at the conceptual level. In this paper, on one hand, we present how to semi-automatically generate the database schema and the underlying MD model for one of the most leading commercial OLAP tools from our model. In this process, some semantics represented in the model are transformed into those considered by the underlying MD model of the target OLAP tool. On the other hand, initial user requirements are translated into their corresponding definitions in the target OLAP tool. In this way, the final user is able to start the analysis process from the initial requirements specified at the conceptual level. Finally, we present the prototype of the Computer Aided Software Engineering (CASE) tool that gives support to both the model definition and this generation process.

Keywords: Conceptual modeling, OLAP, UML, Multidimensionality, Data warehouse

1 Introduction

On-Line Analytical Processing (OLAP) tools, based on the multidimensional (MD) model, are the most popular front-end tools to analyze data in data warehouses. These tools consider the implementation of the MD model from two different perspectives:

- The structural, part which refers to:
  - The structures that form the database schema to house MD data.
  - The underlying MD model that provides the OLAP tool to consider the MD semantics (e.g. facts, fact attributes, dimensions, hierarchy paths, etc.).
- The dynamic part: it refers to the definition of initial user requirements and OLAP operations to further analyze data.

These OLAP tools are mainly MOLAP (Multidimensional OLAP) or ROLAP (Relational OLAP) depending on the kind of structures used to implement the database schema of the MD model. MOLAP tools directly implement the MD model into multidimensional vector structures. ROLAP tools are based on the relational model and the tables of the database schema are normally organized in form of the star schema (and its variants snowflake and constellations) [1].

Each commercial OLAP tool provides its own MD model to consider the main semantics and concepts of MD modeling. As a consequence, different OLAP tools consider different semantics and properties of the MD model. These tools provide a graphical user interface to define the MD model from the structures (multidimensional vectors or relational tables) that form the MD database schema. Therefore, they first require the database schema to be defined. Once both the database schema and the MD model have been defined, an easy "point-and-click" graphical user interface allows the user to define initial requirements.

On the other hand, several proposals have lately been made to accomplish the graphical conceptual design of OLAP applications [2, 3, 4, 5, 6, 7, 8]. Ideally, within the context of OLAP applications, these graphical proposals should semi-automatically generate the implementation of the MD model to be directly queried in a commercial OLAP tool. In doing this, the generation process should generate the implementation of the:

- Structural part:
  - Structures that form the database schema.
  - The MD concepts of the underlying MD model of the target OLAP tool.
- Dynamic part: initial user requirements considered in the conceptual modeling phase.

To the best of our knowledge, the work presented by Hahn *et al.* in [9] is the only one in considering this semi-automatic generation process with outstanding results. In this process, only the implementation of the underlying MD model into the target OLAP tool is taken into

consideration. Thus, the MD model accomplished with the Multidimensional/Entity Relationship model (M/ER) [5] is translated into the corresponding MD model of the target OLAP tool. In the M/ER model, some functional information such as derived measures or additivity are not considered and, therefore, cannot be generated. As pointed out in [9], the semantics and concepts considered by the different MD models of commercial OLAP tools are different from those considered by the graphical conceptual approaches above-presented. Therefore, it is necessary to transform some semantics and properties in the generation process trying to preserve their initial semantics as much as possible.

In this context, our model is an object-oriented (OO) conceptual model to accomplish the conceptual design of both the structural and dynamic parts of OLAP applications. In order to facilitate the use of the modeling constructors, the model provides a Unified Modeling Language (UML) [10] compliant graphical notation [7, 8] in which each modeling constructor has its corresponding graphical notation. This fact allows the designer to accomplish a correct conceptual design with no need of parsing the graphical notation.

In this paper, we present how to semi-automatically generate the implementation of both the structural and dynamic part from our OO model into Informix Metacube (IM). In [13] we presented how to generate the structural part. In this paper, we extend the latter work by providing details on how to generate the dynamic part.

With respect to the structural part, the process first generates the star schema that will house the MD data and then, the corresponding MD model of IM from our modeling constructors used in the conceptual design. Nevertheless, some of the constructors do not have their corresponding representation into IM and, therefore, some are ignored while others are transformed trying to preserve their initial semantics as much as possible. With reference to the dynamic part, initial user requirements defined in our conceptual model are translated into IM requirements. Thanks to this, the final user is able to load them in the subsequent analysis phase and can immediately start the data analysis from them.

Finally, in this paper we also present a Computer Aided Software Engineering (CASE) tool that gives support to all the theoretical aspects presented in this paper.

The remainder of this paper is organized as follows. Section 2 summarizes how to accomplish the conceptual modeling of the structural and dynamic part of OLAP applications with our model. Section 3 presents how IM stores information about its underlying MD model and the requirements defined by the user. Section 4 describes the generation process of both the structural and dynamic part from our model into IM. In section 5 we present a summary of the CASE tool that gives support to both our model and this generation process. Finally, in section 6, we present the conclusions and sketch some works that are currently being carried out.

2 The Object-Oriented Conceptual Model

In this section, we will describe how the UML compliant graphical notation we provide represents both the structural and dynamic parts of OLAP applications.

2.1 The Structural Part

Let us consider from now on an example of a sales system in which facts are considered as the tickets issued in stores of a great-store chain. Figure 1 shows the class diagram of our model for this example in which the fact Sales_products is considered along four dimensions (Product, Customer, Store and Time).

Facts are considered as composite classes in a shared aggregation relation of *n* dimension classes. The minimum cardinality on dimension class roles is 1. *Many-to-many* relationships between the fact class and any dimension class are considered by the cardinality of 1..* on the dimension class role. In our example (see Figure 1), there has been defined a *many-to-many* relationship between the fact Sales_products and the Product dimension as one ticket may contain several products.

Moreover, the designer may define identifying attributes in the fact class with the constraint *{OID}* if they are needed to identify the instances of the fact class unambiguously. These *{OID}* attributes are needed, for example, when there is a *many-to-many* relationship between the fact class and any dimension class. These *{OID}* attributes also allows us to define *degenerate dimensions* [11], which are those dimensions whose identifiers exist in a fact table, but do not have the corresponding dimension explicitly represented. In our example, as a ticket may include several products, the identifying attributes num_ticket and num_line have been defined to distinguish the ticket a sold product belongs to.

As well as atomic measures, derived measures can also be represented with the constraint "/" placed next to the measure_name. Their derivation rules are placed as constraints between brackets around the class. In Figure 1, we can see that the fact class contains three derived measures (qty_sold, total_price, and num_clients) and their corresponding derivation rules.

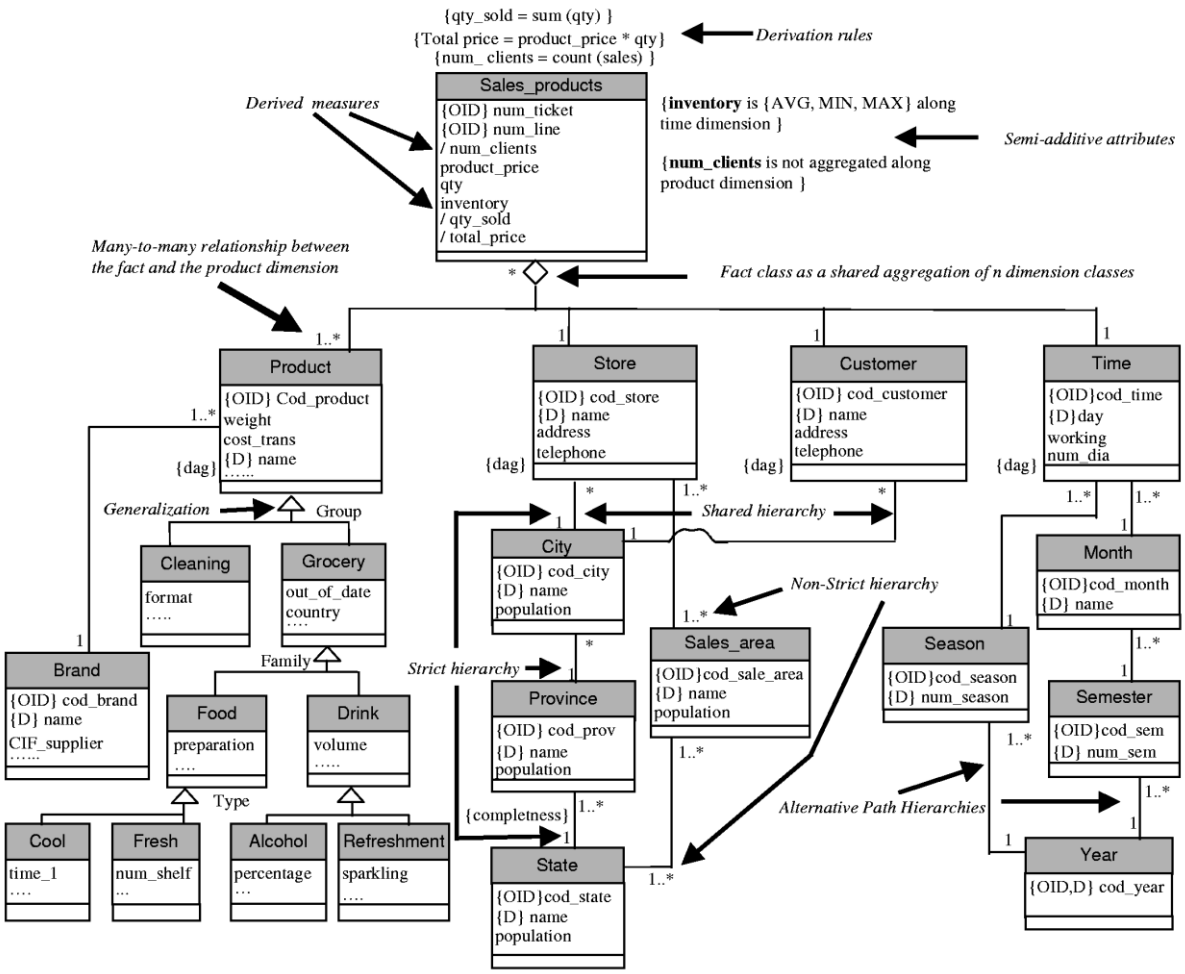

Figure 1: The UML notation to represent the structural part of the sales system

Regarding additivity, all measures are additive by default. Semi-additivity and non-additivity are considered by defining constraints on measures and also placed around the class. These constraints are shown to the designer in a non-strict natural language syntax although they have its formal underlying formulae. In Figure 1, we can see that the attribute num_clients cannot be aggregated along the Product dimension.

With respect to dimensions, each level of a classification hierarchy is considered as a class. These classes must contain an identifying attribute (*{OID}*) to identify the instances of a hierarchy level and a descriptor attribute (*{D}*) that will be used as the default label in the data analysis in the target OLAP tool. These two attributes are necessary as in the semi-automatic generation of the implementation of the model, the OLAP tool will need to know the existence of these two attributes. The classes that represent classification hierarchies must form a Directed Acyclic Graph (DAG) (constraint *{dag}*) starting from each dimension class. The DAG structure can represent both alternative path and multiple classification hierarchies.

The peculiarities of classification hierarchies such as the strictness (an object of a lower level of a hierarchy belongs to only one of a higher level) and completeness (all members belong to one higher-class object and that object consists of those members only) are also considered. In concrete, these features are specified by means of the cardinality of the roles of the associations and the constraint *{completeness}* respectively, as seen in the Store dimension (see Figure 1). Finally, dimensions can be categorized by means of generalization and specialization hierarchies, as observed in the Product dimension. In this way, we can model additional features for an entity's subtypes.

2.2 The Dynamic Part

Our model also allows the user to represent initial user requirements at the conceptual level by means of cube classes. The basic components of these classes are as follows:

- Head area (H): name of the cube class.

- Measures area (M): representation of the measures to be analyzed.
- Slice area (S): restrictions to be satisfied.
- Dice area (D_C): dimensions and their grouping conditions to address the analysis.
- Cube operations (CO): OLAP operations that are provided by the model (roll-up, drill-down, etc.).

Let us suppose the following user initial requirement:

The qty_sold of products where the State is "Valencia" and the Group of products is "Grocery" must be grouped according to the Store Province and City and the Product Family and Brand

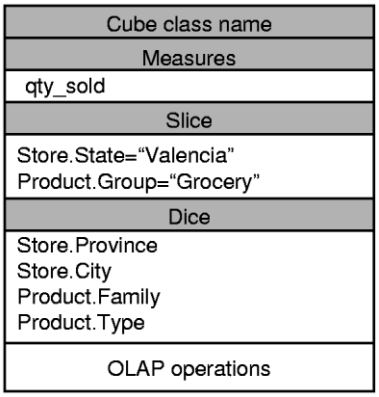

Figure 2: Graphical notation for an initial user requirement

In Figure 2, we can see the graphical notation of the cube class that is used by the designer to represent the previous user initial requirement. It is easy to see the different sections of cube classes above-presented:

- Section Facts contains the aim of the analysis: qty_sold.
- Slice the restrictions defined on the dimensions Store and Product.
- Section Dice, the grouping conditions required along the Store and Product dimensions can easily be identified.

For nonexpert UML or database users, the cube class's graphical notation facilitates the definition of initial user requirements. Every cube class has a more formal underlying OQL specification.

3 Informix Metacube

In this section, we will present how Informix Metacube (IM) [12] represents the structural and dynamic parts of multidimensional modeling and the tools it provides for these tasks.

3.1 The Structural Part

IM works with both the star and snowflake schema. However, the snowflake schema is partial in the sense that the tables that represent different levels of hierarchies are not related between them. Thus, these tables are related to the one that represents the minimum level of hierarchy. Therefore, in our generation process, we will only generate the database schema that corresponds to the star schema.

On the other hand, the underlying MD model of IM is called Decision Support System (*DSS*). The content of the DSS can be defined through an easy graphical user interface by the tool Data Warehouse Manager (*DWM*). In order to accomplish this, it is necessary to have the database schema (star or snowflake) previously defined. The information about the MD models defined in IM is stored in relational tables which contain information about:

- The MD elements defined in the DSS (e.g. facts, dimensions, hierarchy levels, etc.).
- The logical information on these MD elements (e.g. fact tables, primary key of the fact tables, attributes in the relational tables to identify instances of hierarchy levels, etc.).

To clarify the main MD properties considered by the DSS, we have modeled the DSS model with UML (see Figure 3). Thus, the relational tables of the DSS have been modeled with classes and relationships between them. The name of the classes and relationships are the same as their corresponding relational tables in the DSS. It can be observed that not only information on the defined MD concepts is considered (facts, dimensions, etc.) but also logical information (primary keys, table column that corresponds to one measure, etc.).

The classes Fact_table and Dim store information about the facts and dimensions defined in the DSS respectively. The relationship Fact_dim_mapping represents the information about which dimensions are related to which facts through the corresponding foreign keys defined in the fact table (information considered in the associated class Foreign_key), as several facts and dimensions can be defined in the DSS. The class Fact stores information on the measures defined in the DSS, so that every measure must always be contained in a fact (see cardinality of the relationship). The derivation rules of derived measures are represented in the class Dss_string.

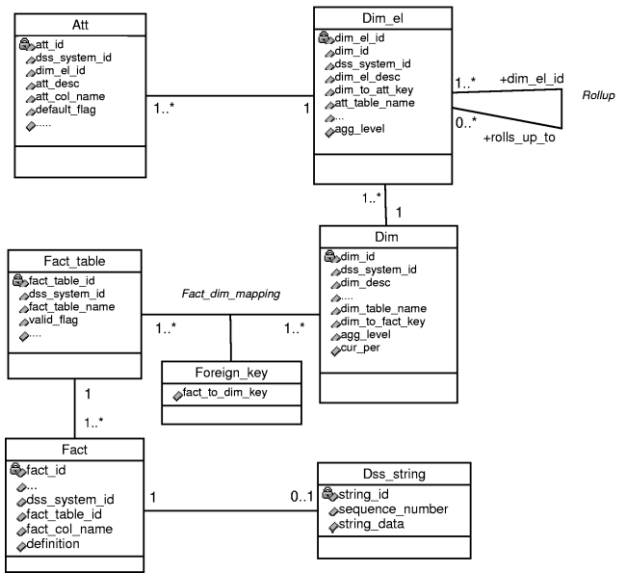

Figure 3: Modeling the DSS model of Informix Metacube (structural part) with UML

In the DSS, hierarchy levels are called dimensional groups and are considered in the class Dim_el. Every dimensional group only belongs to one dimension (see cardinality of the relationship). This means that even though two or more dimensions share the same dimensional group, the same dimensional group has to be defined per each dimension. The relationship Rollup stores information about what dimensional group is connected to (rolls_up_to) which dimensional group, thus representing classification hierarchies. These dimensional groups can be connected to form multiple and alternative path hierarchies (see cardinality of the relationship). Finally, the class Att stores information on the attributes defined in each dimensional group. An attribute must only belong to one dimensional group. For example, if the attribute name has been defined in both the City and Community classes, the attribute name must be defined twice as in the DSS they will be considered as different attributes (name in the City class and name in the Community class).

After a brief review of the main MD features considered by the DSS, we will summary some important MD properties at the conceptual level that cannot be considered by the DSS model:

- *Many-to-many* relationships between a fact and one dimension cannot be considered as the primary key of the fact table is only composed by the foreign keys of the dimension tables to which the fact table is related. This would require more attributes to be part of the primary key of the fact table or additional relational tables to represent these *many-to-many* relationships.
- Additivity cannot be considered, i.e. there is not way of indicating that a certain fact attribute cannot be aggregated along a dimension. Not either it is possible to restrict the set of aggregation operators that can be applied on a fact attribute (e.g. it is not possible to specify that only MAX and MIN can only be applied on a specific fact attribute).
- The relationships between dimensional groups (relationship *Rollup*) are considered strict by default and, therefore, aspects about the cardinality of these relationships are not considered. In the star schema managed by IM, an instance of a dimensional group is only related to one instance of a higher hierarchy level. That is, non-strict and completeness classification hierarchies are not considered.
- The standard star schema does not allow the consideration of the categorization of dimensions as all attributes that correspond to all possible categories of a dimension are defined as attributes within the same relational table in the star schema.

3.2 The Dynamic Part

IM allows the definition of initial user requirements from a DSS with the Informix Metacube Explorer (IME) tool. These requirements are defined from a model defined in the DSS and from them users can apply the set of OLAP operations provided by this tool. User requirements can be saved and then, users can load them whenever it is necessary, as a start point of the information analysis phase. The information about these user requirements is also stored in relational tables.

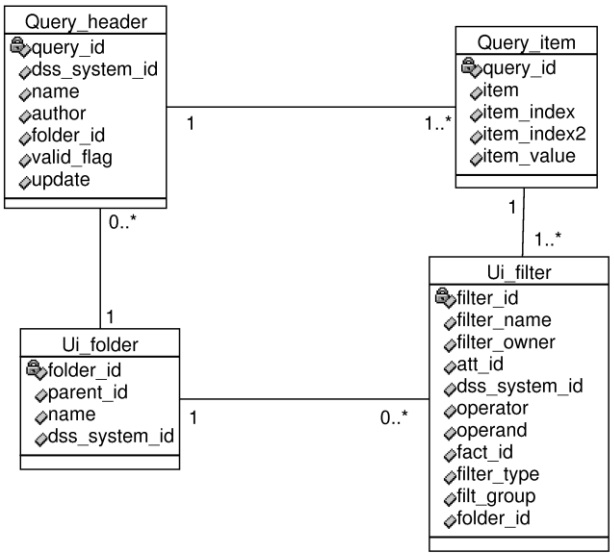

Figure 4: Modeling the DSS model of Informix Metacube (dynamic part) with UML

To facilitate the comprehension about the information on user requirements stored in the relational tables, in Figure 4, we have modeled these tables with UML. To start with, in the class Query_header, every instance represents the basic information of a defined requirement, i.e. its number in the DSS, number of the DSS to which it belongs to, the author and the folder where it has been saved. In the class Query_item all elements of a user requirement are specified, i.e. dimensions and their dimensional groups considered, fact attributes, etc. The information about the filters defined in a user requirement is considered by the class Ui_filter (filters allow us to define constraints on data, e.g. `year = 2002`). Both filters and user requirements are stored in folders and, therefore, the class Ui_folder contains information about the folder where they have been saved.

4 From theObject-Oriented Conceptual Model into the DSS Model

In this section, we will present the main steps of the semi-automatic generation process that obtains from a OO model its corresponding DSS model to implement it in IM. This generation process consists of two algorithms, one for the structural part and another one for the dynamic one.

4.1 The Structural Part

The algorithm of the structural part reads a conceptual MD model and generates two SQL script files:

- The first file contains the SQL sentences needed to create the relational tables that correspond to the star schema that will form the database schema.
- The second one contains the SQL sentences to register the MD concepts in the DSS that correspond to the modeling constructors used in our model.

In this generation process, we have to handle that certain modeling constructors of our OO model do not have their corresponding representation into the DSS model. In some cases, it has been possible to carry out a minimal semi-automatic transformation (the designer has to decide if this transformation is carried out in some cases) of the modeling constructors to be able to represent them in the DSS trying to preserve their initial semantics. In other cases, such a transformation is not possible and, therefore, those modeling constructors have been ignored with the corresponding lack of expressiveness in the final representation of a model. Due to the lack of space, in this paper, we will only describe the transformations accomplished for some modeling constructors; the whole algorithm has been implemented in the CASE tool (see next section) we have developed.

The Table 1 shows the correspondence between the modeling constructors of our model and the MD concepts considered by the DSS model. From now on, we will only mention the modeling constructors ignored as well as we will only remark the main transformations accomplished.

To start with, every class that represents a hierarchy level is defined as a dimensional group. Then, the process reads all associations for every one of these classes and defines a Rollup relationship between the two associated classes. This means that even though two or more dimensions share the same hierarchy level, this level is defined for each dimension as a dimensional group. This is required by the DSS model where every dimensional group must only belong to one dimension.

On the other hand, non-strict and complete classification hierarchies are ignored as in the database schema managed by the DSS model an instance of a hierarchy level can only refer to a one instance of a higher level of the classification hierarchy. Thus, we have to ignore those properties in the generation algorithm.

Finally, the additivity of measures is not considered by the DSS model and, therefore, this property is ignored in the generation process. In the follow, we will describe how to accomplish the transformations described in Table 1.

4.1.1 Specialization Hierarchies

Specialization hierarchies are transformed into strict classification hierarchies. Every concept of the specialization hierarchy is transformed into one dimensional group (level) of the classification hierarchy. Every attribute within a class defined under this specialization concept is considered as an attribute of the new dimensional group. These new dimensional groups are related by means of strict classification hierarchies. Finally, every new dimensional group will have defined as identifying and default attribute the attribute specialization_name_ID.

The GOLD Model	The DSS Model
Dimensions	
Dimension	Dimension
Time dimension	Attribute current period
Dimension class	Base dimensional group
Base class	Dimensional group (D.G.)
Class identifying attribute {OID}	D.G. identifying attribute
Class attribute	D.G. attribute
Class descriptor attribute {D}	D.G. default attribute
Classification hierarchies	Roll-up relationships between D.G.
Alternative path and multiple hierarchies	Alternative path and multiple hierarchies
Non-strict and complete hierarchies	**NOT CONSIDERED**
Specialization hierarchies	**TRANSFORM** into roll-up relationships between D.G.
Specialization concept	D.G. "specialization concept"
Specialized class attribute	Attribute of the D.G. just created
Specialization relationships	Roll-up relationships
Facts	
Fact class	Fact
Many-to-many relationships between a fact and one dimension	**NOT CONSIDERED**. Ask for identifying attribute *{OID}*
Shared aggregation with a class	Fact-dimension relationship
Fact class identifying attribute {OID}	**TRANSFORM** Create dimension, D.G. and the only attribute of that D.G.
Fact attribute	Measure
Derivation rule	Derivation rule
Additivity of measures	**NOT CONSIDERED**

Table 1: Correspondence between the modeling constructors of our model and the MD concepts of the DSS model

In Figure 5, we can see an example of this transformation accomplished for the Product dimension from a conceptual point of view. The specialization levels Group, Family and Type will be transformed into their corresponding classification levels. The identifying and default attribute will be called specialization_name_ID of type String that will have the possible values of the name of the classes defined under the specialization concept that represents. For example, the different values the attribute Type_ID can have are Cool, Fresh, Alcohol and Refreshments. Finally, the arrows in the Figure 5 show how all attributes defined under a specialization concept are included in the corresponding new classification level.

4.1.2 Many-to-many Relationships between a Fact and one Dimension

The DSS model does not consider the *many-to-many* relationship between a fact and one dimension. If the algorithm reads a cardinality higher than 1 in a shared aggregation on the role of a dimension class, this property will be ignored if no identifying attribute has been defined in the fact class. Let us remind (see section 2) that our model allows the designer to define identifying attributes *{OID}* in the fact class that may be needed to represent *many-to-many* relationships between a fact and one dimension.

Therefore, in our generation process, if the designer has defined identifying attributes in the fact class, every one of theses attributes will be transformed into a new dimension with only the base dimensional group with only one attribute (identifying and descriptor at the same time).

In Figure 6, we can see the transformation accomplished from a conceptual point of view. The identifying attributes *{OID}* num_ticket and *{OID}* num_line will be transformed into two new dimensions with only one base dimensional group for each new dimension. The only attribute defined within these new dimensional groups is the same as the one defined in the fact class with the properties of *{OID, D}* (i.e. identifying and descriptor attribute).

4.2 The Dynamic Part

On the other hand, as commented in section 2, the model allows the definition of initial user requirements at the conceptual level by means of cube classes. The dynamic part generation process generates a SQL script file needed to register the initial requirements in IM. The whole process is summarized in Figure 7.

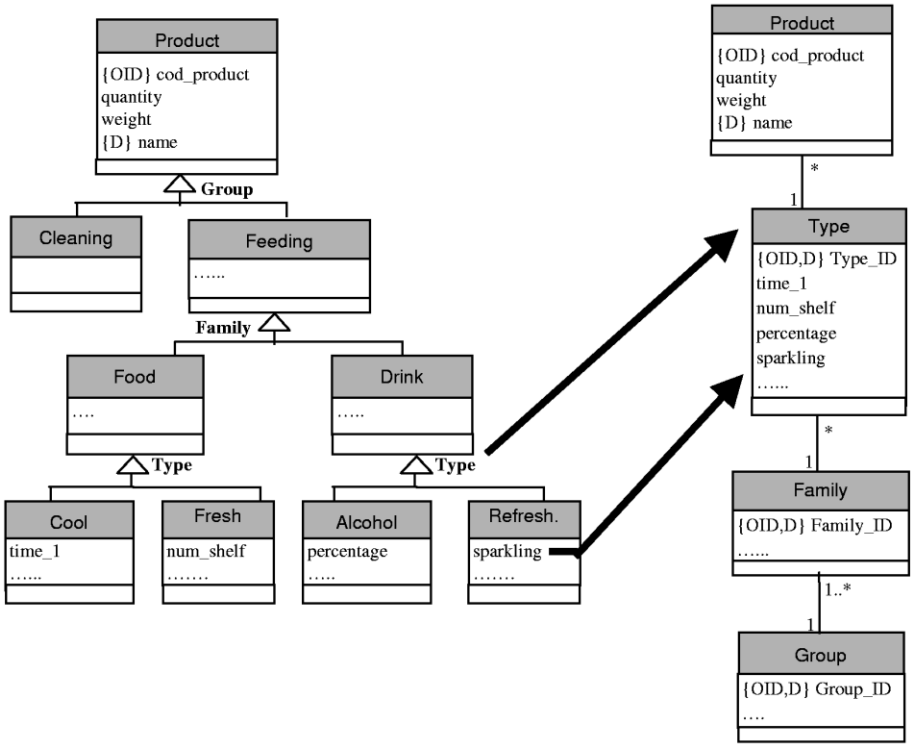

Figure 5: Transformation of specialization hierarchies into strict classification hierarchies

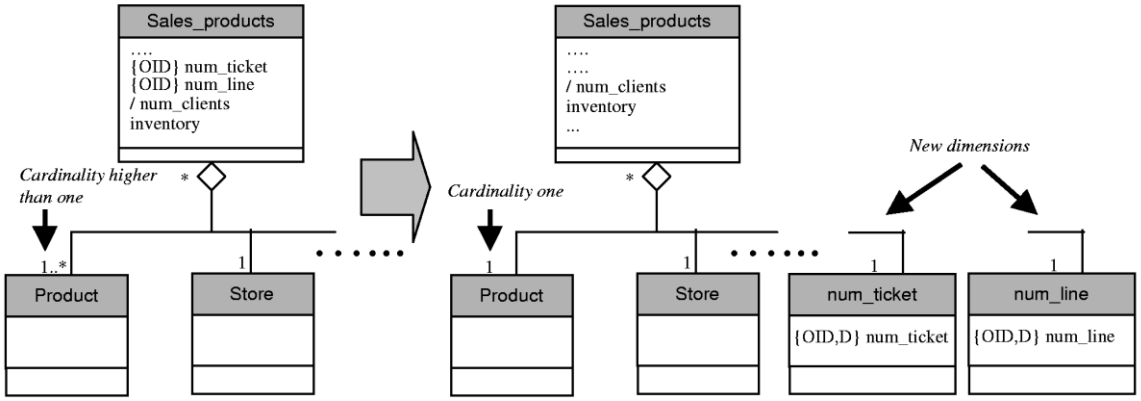

Figure 6: Transformation of identifying attributes (OID) of Fact classes into new dimensions

The algorithm reads every element defined in a cube class and its corresponding definition in the star schema generated with the algorithm of the structural part previously-commented. This is necessary as every element defined in a cube class refers to an element defined in the conceptual model and its logical information is also needed. For example, with reference to the requirement considered in Figure 2, one of the grouping conditions considered is Store.City. The generation process needs to know that this element corresponds to the logical element Store.City_name, i.e. the attribute City_name defined in the relational table Store that represents the Store dimension.

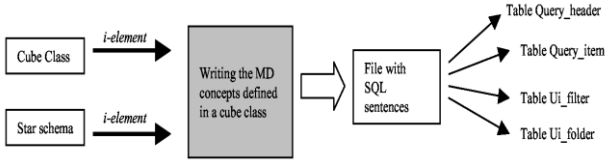

Figure 7: Generation process of initial user requirements

In this way, the final user can load all initial user requirements specified at the conceptual level and, from them, to start the further data analysis phase by applying OLAP operations. Nevertheless, the administrator of the database has to fill in some attributes of theses relational

tables such as privileges, user name, etc., for a correct execution of the requirements.

5 The CASE Tool

In this section, we briefly present the CASE tool that gives support to both the model definition and the generation process described through the paper. The tool provides a comfortable interface for elaborating MD conceptual designs independently of implementation issues. In [14], the architecture of the CASE tool and a complete explanation of its use were presented.

First of all, the designer has to define both the structural and the dynamic part of a conceptual model. The CASE tool allows the designer to hide the attributes and methods defined in every class to have a complete view of the class diagram (Figure 8).

Then, an option of the File menu allows us to start with the generation process of the structural part of a model into Informix Metacube (IM). To carry out the generation of the dynamic part, it is absolutely necessary to have the structural part previously generated. As commented through our paper, the process is semi-automatic as the interaction with the designer is sometimes necessary. For example, the process warns the designer that every *{OID}* attribute defined in the Fact class will be transformed into a new dimension and waits confirmation. In this moment, the designer may decide to abort the process and define new *{OID}* attributes in the Fact class.

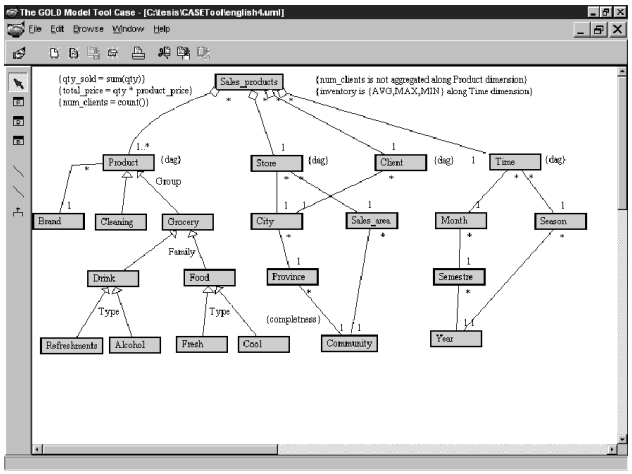

Figure 8: A multidimensional model in the CASE tool

If the generation process ends successfully a window will inform the designer that the generation process has concluded successfully. The CASE tool also allows the designer to view the SQL script files generated. For example, we can see the SQL sentences that will be needed to define the corresponding star schema (Figure 9) and those needed to register every MD concept of the DSS model of IM.

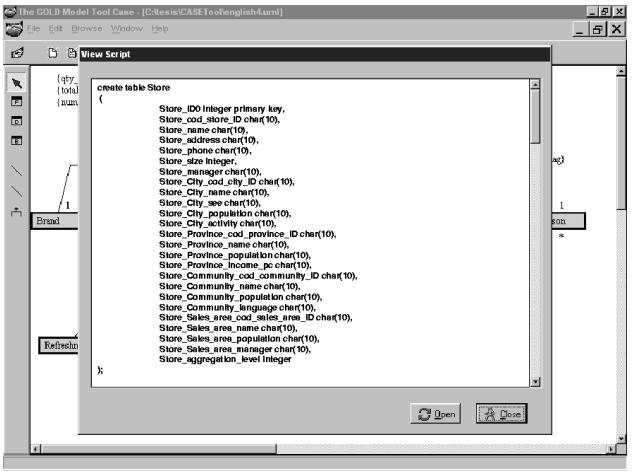

Figure 9: SQL sentences generated from a multidimensional model

6 Conclusions

We have previously proposed an object-oriented approach to accomplish the conceptual modeling of data warehouses, MDB, and OLAP applications [7, 8]. To facilitate the conceptual design, the model provides an easy UML graphical notation that will be used by the designer in the CASE tool that gives support to the model.

In this paper, we have presented how to semi-automatically generate all the needed information to implement our model into Informix Metacube (IM). On one hand, the generation of the structural part consists of generating both the star schema to house the MD data and the MD concepts of the DSS model that correspond to the modeling constructors used in the design. In this process, it has been necessary to transform some modeling constructors that do not have their corresponding representation in IM. On the other hand, with reference to the dynamic part, we have generated the user initial requirement information in IM format. This means that the final user will be able to start the data analysis from these initial requirements.

Finally, we have presented a CASE tool that gives support to our approach. The generation process described throughout the paper has also been implemented in the CASE tool. We are currently working on using some dynamic information used in our model such as state and interaction diagrams to generate more user requirements than only the initial ones.

References

[1] R. Kimball, "*The data warehousing toolkit*", New York: John Wiley & Sons, 2002.

[2] M. Golfarelli, and S. Rizzi, "A methodological Framework for Data Warehouse Design", in *Proceedings of the ACM 1st International Workshop on Data*

warehousing and OLAP (DOLAP'98), Washington D.C., USA, 1998, pp. 3-9.

[3] M. Golfarelli, D. Maio, and S. Rizzi, "Conceptual Design of DataWarehouses from E/R Schemes", in *Proceedings of the 31st Hawaii Conference on System Sciences (HCSS'31)*, Kona (Hawaii), USA, 1998, pp. 334-343.

[4] L. Cabibbo and R. Torlone, "From a Procedural to a Visual Query Language for OLAP", in *Proceedings of the 10th Intl. Conference on Scientific and Statistical Database Management (SSDM'98)*, Capri, Italy, 1998, pp. 74-83.

[5] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter, "Extending the E/R Model for the Multidimensional Paradigm", in *Proceedings of the First International Workshop on Data Warehouse and Data Mining (DWDM'98)*, 1998, vol. 1552 of Lecture Notes in Computer Science, pp. 105-116.

[6] C. Sapia, "On Modeling and Predicting Query Behavior in OLAP Systems", in *Proceedings of the Internacional Workshop on Design and Management of Data Warehouses (DMDW'99)*, Heidelberg, Germany, 1999, pp. 1-10.

[7] J. Trujillo, J. Gómez, and M. Palomar, "Modeling the Behavior of OLAP Applications Using an UML Compilant Approach", in *Proceedings of the First International Conference On Advances in Information Systems (ADVIS'00)*, Izmir, Turkey, 2000, vol. 1909 of Lecture Notes in Computer Science, pp. 14-23.

[8] J. Trujillo, M. Palomar, J. Gómez, and I. Song, "Designing Data Warehouses with OO Conceptual Models", *IEEE Computer, special issue on Data Warehouses*, vol. 34, no. 12, pp. 66-75, December 2001.

[9] K. Hahn, C. Sapia, and M. Blaschka, "Automatically Generating OLAP Schemata from Conceptual Graphical Models", in *Proceedings of the ACM 3rd International Workshop on Data warehousing and OLAP (DOLAP'00)*, Washington D.C., USA, 2000.

[10] Object Management Group (OMG), "Unified Modeling Language (UML)", Internet: http://www.omg.org/cgi-bin/doc?formal/01-09-67, January 2001.

[11] W. Giovinazzo, "*Object-Oriented Data Warehouse Design. Building a star schema*", New Jersey: Prentice-Hall, 2000.

[12] Informix, "Informix Metacube", Internet: http://www.informix.com, December 1999.

[13] J. Trujillo, and S. Luján-Mora, "Automatically Generating Database Schemas into OLAP Tools from Object-Oriented Conceptual Models", in *International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA'02)*, Foz do Iguazu, Brazil, June 2002, pp. 102-107.

[14] J. Trujillo, S. Luján-Mora, and E. Medina, "The GOLD Model CASE Tool: an environment for designing OLAP applications", in *Proc. of the 4th International Conference on Enterprise Information Systems (ICEIS 2002)*, Ciudad Real, Spain, April 2002, pp. 699-707.