The GOLD Model CASE Tool: an environment for designing OLAP applications

Juan Trujillo, Sergio Luján-Mora, Enrique Medina
Departamento de Lenguajes y Sistemas Informáticos. Universidad de Alicante.
Campus de San Vicente del Raspeig. Apartado 99. 03080 Alicante. SPAIN
Email: {jtrujillo,slujan,emedina}@dlsi.ua.es

Keywords: CASE Tool, Data Warehouse, Multidimensional, OLAP, UML

Abstract: The number of On-Line Analytical Processing (OLAP) applications in the market has dramatically increased in the last years. Most of these applications provide their own multidimensional (MD) models to represent main MD properties, thereby making the design totally dependent of the target commercial OLAP application. In this paper, we present a Computer-Aided Software-Engineering (CASE) operational environment to accomplish the design of an OLAP applications totally independent of the target commercial OLAP tool. The designer uses a Unified Modeling Language (UML) compliant approach to represent MD properties at the conceptual level. Once the conceptual design is finished, the CASE tool semi-automatically generates the corresponding implementation into the target commercial OLAP tool. Therefore, our approach frees conceptual design from implementation issues.

1 Introduction

Data warehouse (DW) systems consolidate and integrate data from heterogeneous sources and provide powerful mechanisms to manage the information, enabling the knowledge workers to make better and faster decisions for the enterprise (Chaudhuri and Dayal, 1997)(Inmon, 1996). In this context, OLAP tools, based on the multidimensional (MD) modeling, are the predominant front-end client tools for querying and analyzing data in a DW. OLAP tools are mainly classified into ROLAP (Relational OLAP) and MOLAP (Multidimensional OLAP), depending on whether the underlying multidimensional structures are relational tables or propietary MD vectors, respectively.

During the last years, many OLAP tools have increasingly arisen (Dinter et al., 1998). Most of these OLAP tools provide their own MD model to consider main MD properties. As a consequence,

1. not all MD properties are considered by all commercial OLAP tools and,
2. a MD conceptual design does not usually have the same representation in the different commercial OLAP tools.

Therefore, nowadays the design process of an OLAP application is totally dependent on the target OLAP tool and its underlying MD model.

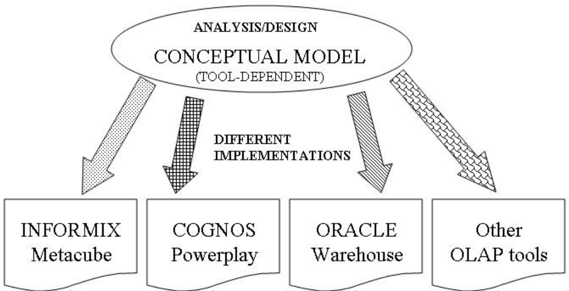


Figure 1: Development of an OLAP application.

Figure 1 illustrates the above-mentioned problem. In the phase of analysis/design, the user has to define one specific conceptual MD model for each particular OLAP tool so that the implementation can be successfully achieved. In this context, a conceptual MD model in Informix, for example, is different from one another in Oracle.

We argue that this dependence could be solved by providing a MD conceptual model that allows us to accomplish a MD design totally independent of implementation issues without taking into consideration the target commercial OLAP tool. Then, from this conceptual model, we should ideally be able to generate its corresponding implementation into any market OLAP tool by providing the specific information to build the corresponding MD model.

In this context, some conceptual approaches have been lately proposed by the research community to accomplish the conceptual design of OLAP applications, such as the Dimensional-Fact Model by M. Golfarelli *et al.* (Golfarelli and Rizzi, 1998), the starER model by N. Tryfona *et al.* (Tryfona et al., 1999), the ME/R model by C. Sapia *et al.* (Hahn et al., 2000; Sapia et al., 1998) and the GOLD model by J. Trujillo *et al.* (Trujillo, 2001; Trujillo et al., 2000; Trujillo et al., 2001). To the best of our knowledge, the only works that have explicitly provided details regarding the automatic implementation from a conceptual model into a target OLAP tool are (Hahn et al., 2000; Trujillo et al., 2001).

In this sense, it is generally avowed that both the design of the conceptual model and the generation process should be accomplished within a CASE tool framework. This environment should have two design phases clearly identified: (i) graphical notations should be provided by a conceptual MD model to facilitate the designer with the specification of the conceptual MD model and (ii) an automatic generation process should perform the implementation of the designed conceptual MD model into a target commercial OLAP tool. This generation process should generate both (i) the database structures to house the MD data and (ii) the underlying OLAP tool metadata which stores information on the created MD model such as facts, measures, dimensions and classification hierarchies. However, some MD issues of the original conceptual MD may not have their corresponding implementation in the target commercial OLAP tool. In this particular case, a well-founded transformation policy should be accomplished to transform some original MD issues while trying to preserve their original meaning as much as possible. Unfortunately, there will also be some MD issues that will not have a corresponding transformation, and therefore, they will be ignored with the corresponding lack of expressiveness.

In this paper, we propose 'The GOLD Model CASE Tool' (GMCT), a graphical environment for the design of OLAP applications. The conceptual design is accomplished by using the GOLD model (Trujillo, 2001; Trujillo et al., 2000; Trujillo et al., 2001), an object-oriented conceptual model that facilitates the design of conceptual MD models by using a subset of the Unified Modeling Language (UML) (Object Management Group (OMG), 2001). The semantic rules used by this model for a proper MD design have been incorporated to the CASE tool, therefore providing the mechanisms to ensure the correctness, consistency and integrity of the accomplished model. Once the conceptual MD design has been accomplished, the designer can pose the generation process that semi-automatically generates the corresponding implementation into the most popular front-end OLAP tools such as Informix Metacube, Oracle Express and Cognos PowerPlay. During this process, the CASE tool informs the designer about every needed transformation to obtain the corresponding implementation.

The remainder of this paper is structured as follows: Section 2 summarizes the main MD features considered by the GOLD model. Based on the GOLD model, Sections 3 and 4 discuss the layered architecture and the usage of the GMCT respectively. Finally, Section 5 presents the conclusions and future works that are currently being considered.

2 The GOLD model: an object-oriented conceptual model for OLAP tools

The GOLD model allows us to specify both the *structural* (Trujillo et al., 2001) and *dynamic* parts (Trujillo et al., 2000) of an OLAP application at the conceptual level (Trujillo, 2001). The structural part regards to the MD model structural properties such as facts, dimensions, measures, classification hierarchies, additivity and so on. On the other hand, the dynamic one refers to the initial user requirements and the evolution (behaviour) of these requirements based on the applied OLAP operations.

Let us consider an example in which the fact is the product sales category for a large chain of stores, which has the following dimensions: product, store, customer, and time.

Regarding the structural part, a MD model is specified by means of a UML class diagram in which the information is clearly separated into dimensions and facts.

Dimensions and facts are considered by *dimension classes* and *fact classes* respectively. Then, facts classes are specified as composite classes in a shared aggregation relationship of `n` dimension classes. Thanks to the flexibility of shared-aggregation relationships that UML provides, *many-to-many* relationships between facts and particular dimensions can be considered. This is done by indicating the 1..* cardinality on the dimension class role to show that a fact object instance can be related to one or more dimension object instances. Derived measures can also be explicitly considered (constraint /) and their derivation rules are placed between braces in somewhere around the fact class. By default, all measures are considered additive. For non-additive measures, additive rules are defined as constraints and also placed in somewhere around the fact class.

Our approach also allows us to define identifying attributes in the fact class, if convenient, by placing the constraint {OID} next to a measure name, also known as degenerate dimensions (Giovinazzo, 2000; Kimball, 1996). This provides other fact features in addition to the measures for analysis. In our example, we could store the ticket and line numbers as other ticket features.

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 required constraint is that these classes must define a Directed Acyclic Graph (DAG) rooted in the dimension class. The DAG structure can represent both alternative path and multiple classification hierarchies. Therefore, the constraint {*dag*} is placed next to every dimension class. Every classification hierarchy level has an identifying attribute (constraint {*OID*}) and a descriptor attribute (constraint {*D*}). These attributes are necessary for an automatic generation process into a commercial ROLAP tool as these tools store this information in their metadata. The multiplicities *1* and *1..** defined in the target associated class role address the concepts of strictness and nonstrictness. Defining the {*completeness*} constraint in the target associated class role addresses the completeness of a classification hierarchy. By default, our approach considers all classification hierarchies noncomplete.

The categorization of dimensions, used to model additional features for an entity's subtypes, is considered by means of generalization-specialization relationships. However, no class other than the dimension class can belong to both a classification and specialization hierarchy at the same time.

Regarding the dynamic part, we provide a UML-compliant class notation (called *cube class*) to specify initial user requirements. A set of basic OLAP operations (e.g. roll-up, drill-down, slice, etc.) is provided to simulate the further data analysis phase from these cube classes. Then, we model the behaviour (evolution) of these cube classes by means of state and interaction diagrams based on the applied OLAP operations (Trujillo et al., 2000). These diagrams contain information about the most probable evolution of the final user requirements from the specified cube classes. OLAP designers can use the information these diagrams contain to predict user behaviour, thereby helping them in the design of a proper view maintenance policy.

Finally, this model provides a set of modules with adequate transformation rules to export the accomplished conceptual model into a commercial OLAP tool, which allows us to check the validity of the proposed approach.

3 Architecture of The GOLD Model CASE Tool

The GMCT[1] is based on a layered architecture of different modules and was designed and implemented applying the object-oriented (OO) methodology (Booch, 1994). The OO methodology enables a modular design and implementation, aimed at helping the task of maintenance and extension of the code.

This layered architecture is structured in the following modules, as shown in Figure 2: Graphical User Interface (GUI) and Core Tool (3.1), Repository (3.2), Export Model (3.3), Export Queries (3.4) and Query Generator (3.5). In the following sections, we will summarize each one of these modules and their connections.

3.1 Graphical User Interface and Core Tool

The GUI is based on Microsoft Windows interface and uses Multiple Document Interface (MDI)[2] technology. It is similar to OO commercial tools like the Rational Rose Family from Rational Software (Rational Software, 2001).

The GUI presents three different interfaces:

- **Class Diagram**: as above commented, the class diagram allows us to represent the static part of the GOLD Model (structural properties: dimensions, facts, classification hierarchies, etc.). It uses standard graphical model-

[1]This tool is developed using *Borland C++ Builder 5* under Microsoft Windows 98.

[2]MDI allows to load multiple documents in the same program. It is possible to copy and paste elements from one document into another.

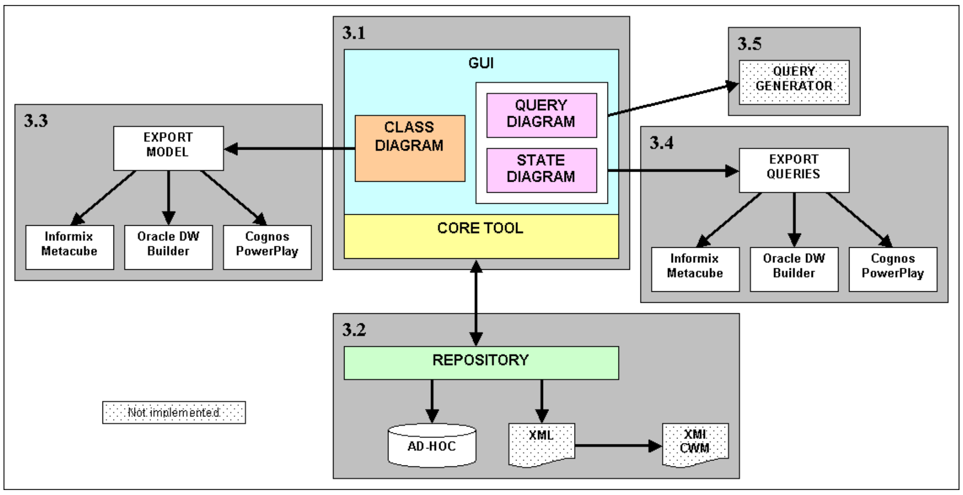


Figure 2: Architecture of The GOLD Model CASE Tool

ing elements from UML: classes, aggregations, associations, etc.

- **Query Diagram**: allows us to represent the dynamic part of the GOLD model. This module allows the user to define initial requirements from the class diagram that are represented as cube classes in this model.
- **State Diagram**: describes the behaviour of each cube class according to the different OLAP operations (roll-up, drill-down, combine, etc.).

To ensure the correctness of the model designed using the GMCT, the Core Tool module checks the semantic constraints that have to be satisfied by the model (e.g. a dimension class has to be connected to a fact class, or a dimension class needs an identifier and a descriptor attribute, etc.). This module also ensures the correctness of the cube classes. All these checkings are performed while the class diagram and the cube classes are being constructed. Once the model has been correctly checked, the model is stored internally in the repository.

3.2 Repository

This part of the tool stores the metadata that describes the internal definition of the different models created by the user. For a proper CASE tool internal use, we use an ad-hoc repository, but we are currently working on the specification of an XML DTD[3] to store the metadata in an XML format, so that it can be exported to different OLAP tools.

[3]Document Type Definition.

3.3 Export Model

The GMCT exports the implementation of the model represented in a class diagram to different OLAP tools. We have implemented the generation process for one representative of each set of OLAP tools: Informix Metacube, Cognos PowerPlay, IBM Warehouse Manager, etc. The only prerequisite for an OLAP tool to be integrated as a target in the GMCT is the existence of any form of external and accessible interface, e.g. a model definition language in Cognos or metadata relational tables in Informix that can be accessed with SQL. This is a semiautomatic process, as the user has to confirm possible changes (transformations) made to the original model, since some semantics and constraints may be misspelled due to the limitations of the target OLAP tool. In this paper, we will only show the generation process for the Informix Metacube ROLAP tool, which is divided into two modules: Warehouse Manager, to construct DSS[4], and Explorer, to query previously constructed DSS. Once the model has been exported, the cube classes (queries) can be also exported.

3.4 Export Queries

The purpose of this module is to export the cube classes (queries) to the target OLAP tool. Whether the target OLAP tool has the possibility of defining queries externally, this module translates cube classes in the query diagram to the target query language. Again in this paper, we

[4]A multidimensional model is called Decision Support System (DSS) in Informix Metacube.

will only focus on the export query process for the module Explorer of the ROLAP tool Informix Metacube.

3.5 Query Generator

In order to allow the designer to execute user queries directly to the DW from the GMCT, this module connects to the data stored in the DW through ODBC and presents the results of the execution of the queries in a spreadsheet style, as the grid-oriented, two-dimensional layout structure used to analyze multidimensional data. This module is currently under construction.

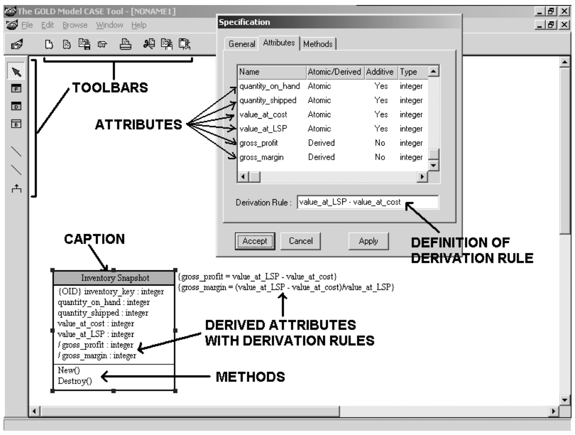
4 Working with the GOLD Model CASE Tool environment

This section presents the main steps needed to design the static and dynamic parts of the GOLD model. We have chosen two well-known examples, “the advanced inventory snapshot model” and “the inventory transaction model” from Chapter 3 (Kimball, 1996) to help the explanation. Summarizing, in the first model, the DW stores the inventory levels in separate records for each time period (days, weeks, etc.). On the second model, the DW stores every change in the status of the products (arrival, processing, departure, etc.).

4.1 Graphical User Interface and Core Tool

The GUI is the main window of the tool and is shown in Figure 3. To accomplish the design of a class diagram, the GUI provides the designer with the toolbar on the left hand side where the different icons are the following (from top to bottom): cursor (selection), fact class, dimension class, base class, shared aggregation, association and generalization. These icons switch to ‘enabled/disabled’ depending on the actual state of the design, not allowing the designer to perform ‘non-applicable’ actions.

The GMCT does not necessary force the designer to follow a creation order for the design of the elements in the class diagram. However, the first class to include in the diagram is usually the fact class; then, dimension classes are created. Figure 3 shows the fact class `InventorySnapshot`. Notice that every class has two parts: attributes (`inventory_key`, `quantity_on_hand` and `quantity_shipped`, ...) and methods (`New()` and `Destroy()`). In order to define attribute features (name, type, additivity rule, etc.) and methods, the GMCT provides



Name	Atomic/Derived	Additive	Type
quantity_on_hand	Atomic	Yes	integer
quantity_shipped	Atomic	Yes	integer
value_at_cost	Atomic	Yes	integer
value_at_LSP	Atomic	Yes	integer
gross_profit	Derived	No	integer
gross_margin	Derived	No	integer

Derivation Rule : value_at_LSP - value_at_cost

```
Inventory Snapshot
{OID} inventory_key : integer
quantity_on_hand : integer
quantity_shipped : integer
value_at_cost : integer
value_at_LSP : integer
/ gross_profit : integer
/ gross_margin : integer
New()
Destroy()
```

{gross_profit = value_at_LSP - value_at_cost}
{gross_margin = (value_at_LSP - value_at_cost)/value_at_LSP}

Figure 3: GUI representation and Specification dialog.

the **Specification** dialog, that can also be seen in Figure 3. This dialog also allows us to define derived attributes, both for the fact and the dimension classes, e.g. definition of derivation rule for `gross_profit`.

Hierarchy levels can also be defined within the dimensions by creating a new class for every level. Depending on how these classes are conected, different and multiple hierarchy paths can be represented for the same dimension. Furthermore, merging dimensions can also be used. Figure 4 shows two hierarchy paths: `Product` > `Subcategory` > `Category` and `Warehouse` > `Zip`.

Once the dimension classes have been created, the fact and dimension classes can be associated. Figure 4 illustrates how the fact class `InventoryTransaction` is connected to four dimension classes (`Product`, `Transaction`, `Time` and `Warehouse`) by means of shared aggregations. Next, it is possible to define the additivity rules of the measures within the fact classes. This task is performed by using the **Additivity** dialog shown in Figure 4. As a example, Figure 4 shows the additivity rules that have been defined for the fact class in our example: specifically, `PO_number` is nonadditive across any dimension and `amount` is nonadditive across `Time`.

Once the class diagram (static part of the GOLD model) is finished, the user can define cube classes by using the query diagram. As it can be seen in Figure 5, the query diagram presents a different interface that displays facts and dimensions from the class diagram on the left panel of the window via a tree structure. This interface allows the user to compose new queries by using the “drag’n’drop” technique. In this sense, measures and attributes can be dragged from the tree structure and dropped into any of the three

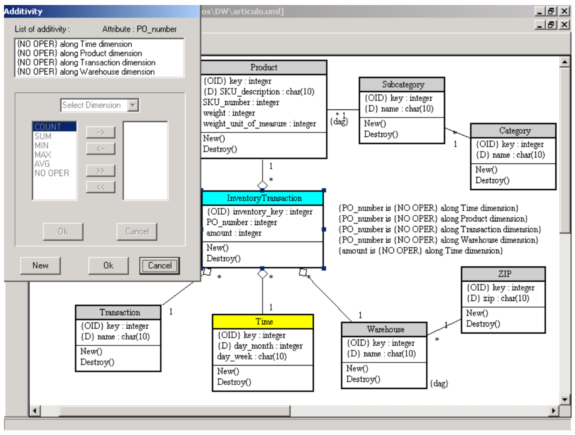


Figure 4: Definition of semi-additive measures using the Additivity dialog.

boxes named measures (fact attributes to be analyzed), slice (constraints to be satisfied) and dice (dimensions and grouping conditions to address the analysis) respectively. As outlined in 3.1, the Core Tool module ensures the correctness of the constructed queries (cube classes).

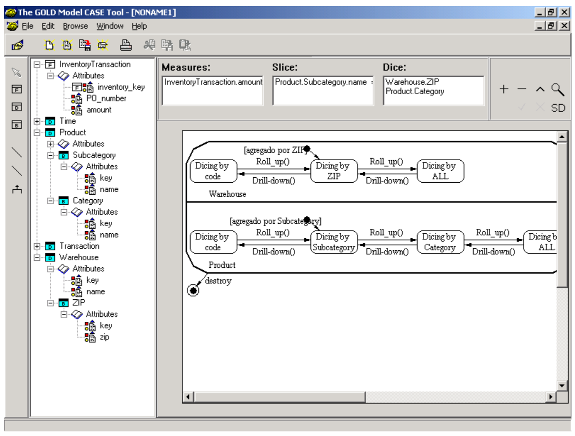
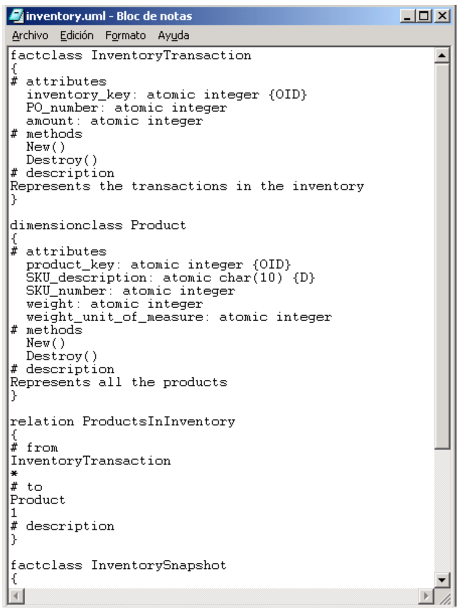


Figure 5: Interface for both the query and the state diagram.

For every cube class, the GMCT can automatically represent its corresponding state diagram, as reflected in Figure 5 for an example cube class. This diagram shows the different states and OLAP operations that can be performed along with the corresponding transitions between states. To follow the example shown in Figure 5, if the user performs a roll-up operation in a cube starting from the state `Dicing by Subcategory`, then the cube changes to the new state `Dicing by Category`.

4.2 Repository

At the moment, the repository is implemented as a plain text file. This method has the advantage that its meaning can be easily understood without any difficulty, making it legible for anyone. Figure 6 shows part of the definition of the example fact and dimension classes, and the relationships between them. These scripts can be easily seen from the CASE tool.



```
factclass InventoryTransaction
{
# attributes
  inventory_key: atomic integer {OID}
  PO_number: atomic integer
  amount: atomic integer
# methods
  New()
  Destroy()
# description
Represents the transactions in the inventory
}

dimensionclass Product
{
# attributes
  product_key: atomic integer {OID}
  SKU_description: atomic char(10) {D}
  SKU_number: atomic integer
  weight: atomic integer
  weight_unit_of_measure: atomic integer
# methods
  New()
  Destroy()
# description
Represents all the products
}

relation ProductsInInventory
{
# from
InventoryTransaction
*
# to
Product
1
# description
}

factclass InventorySnapshot
{
```

Figure 6: Metadata of the example fact and dimension classes.

4.3 Export Model

The GMCT allows the designer to export the conceptual MD model to several external OLAP tools. Figure 7 shows the expanded menu option for exporting models.

The generation process into a target OLAP tool accomplished in this module is divided into three phases:

1. Loading the metadata corresponding to the conceptual schema from the repository. This metadata is internally stored using ad-hoc techniques introduced in the previous Section 4.2.

2. Applying specific transformations to adapt original semantics to the data model of the target system with the minimal loss of expressiveness.

3. Creating the output for the target OLAP tool by translating every element in the original

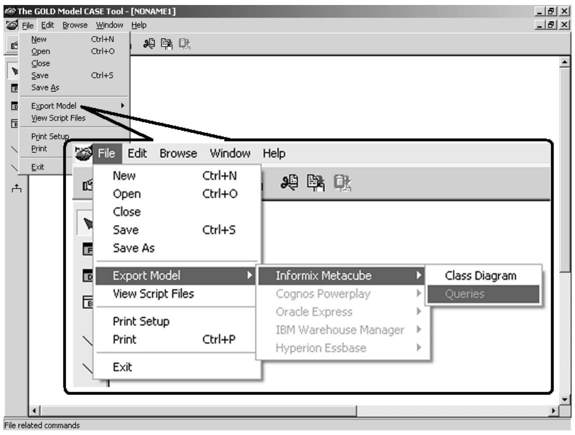


Figure 7: Export models in the GMCT.

schema to a corresponding construct in the target OLAP system. This logical schema is not directly written into the target OLAP repository, but written into an external file.

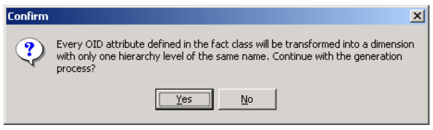
The transformation step handles the constraints of the target OLAP system. To accomplish this task, the target data model has to be inspected and studied in order to discover all the limitations with respect to the GOLD model and how they can be resolved with the application of specific transformation methods. These transformations are designed to preserve as much as possible of the original semantics, i.e. original expressiveness. In some cases, due to limitations in the target OLAP tool, some constraints cannot be resolved by the generation process and are simply ignored and discarded.

In order to check the validity of the generation process and demonstrate the applicability of our CASE tool, we focus on the generation process for Warehouse Manager (WM), the module for constructing DSS in Informix Metacube.

4.3.1 From the GOLD model into the Informix Warehouse Manager

Metacube, as a ROLAP tool, stores the data and the metadata in a relational database system. In order to create the OLAP schema for Metacube, our CASE tool generates two plain text files with SQL statements: the first file, containing the definition of the database elements (SQL Data Definition Language, DDL), e.g. tables for facts and dimensions with their attributes; the second file, defining the internal description of the model (SQL Data Manipulation Language, DML), i.e. the metadata (names of the dimension levels, default attributes, etc.) and information about the configuration of the target OLAP tool. The overall information generated by our CASE tool is then used by WM to build its own DSS.

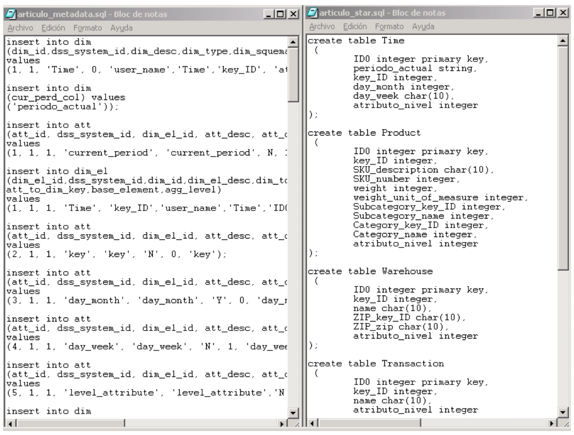
Following these considerations, we run the generation process for the example described in Section 4. During this interactive process, the user is requested to confirm the transformations made by this module. For example, as WM does not work with OID, a transformation is needed in order to generate the implementation. The acknowledgement for this transformation is requested from the user through the confirmation dialog illustrated in Figure 8.



Every OID attribute defined in the fact class will be transformed into a dimension with only one hierarchy level of the same name. Continue with the generation process?

Figure 8: Confirmation dialog in the generation process.

Once the generation process has finished, two plain text files have been created that contain SQL scripts that can be seen from the CASE tool as reflected in Figure 9.


articulo_metadata.sql - Bloc de notas
Archivo Edición Formato Ayuda
articulo_star.sql - Bloc de notas
Archivo Edición Formato Ayuda

```
insert into dim
(dim_id,dss_system_id,dim_desc,dim_type,dim_squema
values
(1, 1, 'Time', 0, 'user_name','Time','key_ID', 'at

insert into dim
(cur_perd_col) values
('periodo_actual'));

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_c
values
(1, 1, 1, 'current_period', 'current_period', N, 1

insert into dim_el
(dim_el_id,dss_system_id,dim_id,dim_el_desc,dim_to
att_to_dim_key,base_element,agg_level)
values
(1, 1, 1, 'Time', 'key_ID','user_name','Time','ID(

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_c
values
(2, 1, 1, 'key', 'key', 'N', 0, 'key');

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_c
values
(3, 1, 1, 'day_month', 'day_month', 'Y', 0, 'day_r

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_c
values
(4, 1, 1, 'day_week', 'day_week', 'N', 1, 'day_wee

insert into att
(att_id, dss_system_id, dim_el_id, att_desc, att_c
values
(5, 1, 1, 'level_attribute', 'level_attribute','N

insert into dim
```

```
create table Time
 (
        ID0 integer primary key,
        periodo_actual string,
        key_ID integer,
        day_month integer,
        day_week char(10),
        atributo_nivel integer
);

create table Product
 (
        ID0 integer primary key,
        key_ID integer,
        SKU_description char(10),
        SKU_number integer,
        weight integer,
        weight_unit_of_measure integer,
        Subcategory_key_ID integer,
        Subcategory_name integer,
        Category_key_ID integer,
        Category_name integer,
        atributo_nivel integer
);

create table Warehouse
 (
        ID0 integer primary key,
        key_ID integer,
        name char(10),
        ZIP_key_ID char(10),
        ZIP_zip char(10),
        atributo_nivel integer
);

create table Transaction
 (
        ID0 integer primary key,
        key_ID integer,
        name char(10),
        atributo_nivel integer
```

Figure 9: Scripts obtained in the generation process for Informix Metacube.

In Figure 9, two scripts are shown: on the left hand side, we can notice the SQL sentences that load the model into the Informix DSS; on the right hand side, we can see the SQL sentences that create the star schema in the database.

4.4 Export Queries

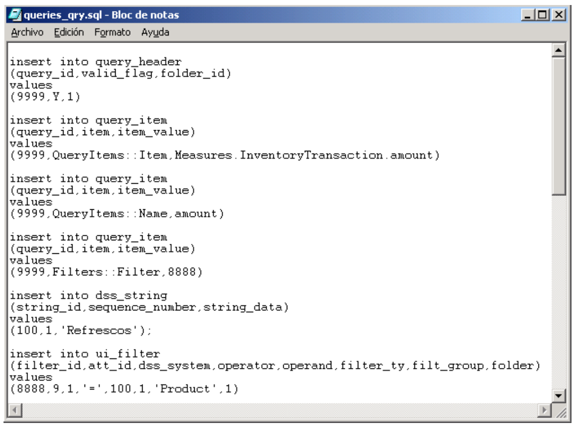
Cube classes defined in the query diagram can be also exported to an external query language, if supported by the target OLAP tool. The export process for a particular query is accomplished into three steps:

1. Loading the metadata corresponding to the query from the repository.
2. The query is then internally composed and analyzed in order to find inconsistencies within its own definition, e.g. non-allowed grouping conditions, constraints on different data types, etc.
3. Creating an external file that contains the mapping of the query for the corresponding external query language used by the target OLAP tool, e.g. SQL in the case of Informix Metacube.

As previously done, to check the validity of the export query process and demonstrate the applicability of our CASE tool, we focus on the export query process for Explorer, the query module in Informix Metacube.

4.4.1 From the GOLD model into the Informix Explorer

As introduced in Section 4.3.1, Metacube uses SQL as its query language. Figure 10 displays the script generated for our example cube class previously shown in Figure 5.



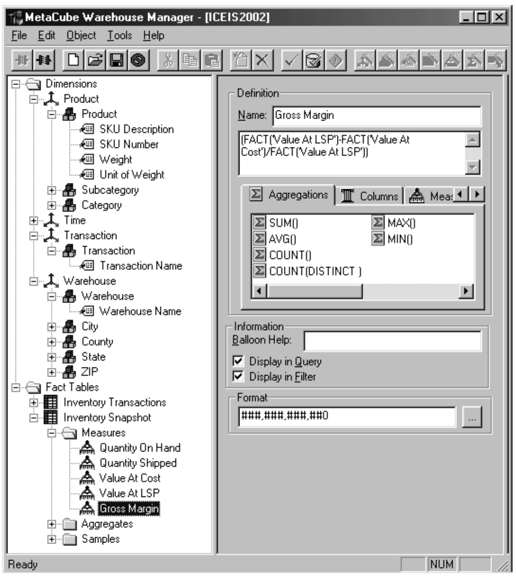
```
insert into query_header
(query_id,valid_flag,folder_id)
values
(9999,Y,1)

insert into query_item
(query_id,item,item_value)
values
(9999,QueryItems::Item,Measures.InventoryTransaction.amount)

insert into query_item
(query_id,item,item_value)
values
(9999,QueryItems::Name,amount)

insert into query_item
(query_id,item,item_value)
values
(9999,Filters::Filter,8888)

insert into dss_string
(string_id,sequence_number,string_data)
values
(100,1,'Refrescos');

insert into ui_filter
(filter_id,att_id,dss_system,operator,operand,filter_ty,filt_group,folder)
values
(8888,9,1,'=',100,1,'Product',1)
```

Figure 10: Query scripts obtained in the generation process for Informix Metacube.

The script represented in Figure 10 contains the statements needed to compose a query in Informix Metacube. Every SQL statement references one particular element of the internal query metadata of Explorer.

4.5 Using the exported model and queries

In order to check the validity of the generated implementation, both for the model and the queries, we build a DSS from the files presented in Figure 9. The resulting DSS in WM is shown in Figure 11.



Definition
Name: Gross Margin
(FACT('Value At LSP')-FACT('Value At Cost')/FACT('Value At LSP'))

Information
Balloon Help:
[✓] Display in Query
[✓] Display in Filter

Format
###,###,###,##0

Figure 11: DSS constructed from the generated SQL script.

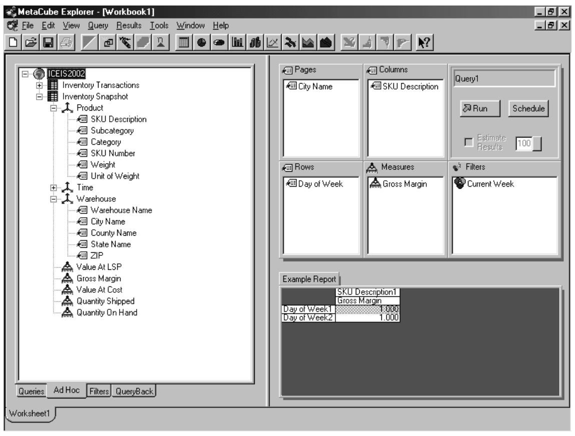
Figure 11 illustrates the DSS corresponding to the original GOLD model. We can notice that the definition of three levels (`Product`, `Subcategory` and `Category`) for the `Product` dimension and also the attributes within the `Product` level. Several levels and attributes are also shown for the rest of the dimensions (`Time`, `Transaction` and `Warehouse`). Furthermore, measures defined within the fact `InventorySnapshot` are illustrated with the particular definition of the `Gross Margin` derived measure.

On the other hand, the script that contains the queries is read from Explorer, so they are available to be executed. An example of a user query is represented in Figure 12.

Whenever we define a query in Explorer, a filter should be applied to constraint the set of data that the query will return as a result. Filters allow the user to limit the range of data returned for any given attribute or measure. As can be observed in Figure 12, we have filtered data for the current week period.

5 Conclusions and future works

In this paper, we have presented a CASE operational environment for OLAP applications that abstracts the conceptual design of implementation issues. The CASE tool gives support to the GOLD model, an object-oriented conceptual



	SKU Description1
	Gross Margin
Day of Week1	1.000
Day of Week2	1.000

Figure 12: An example of a user query taken from the generated SQL script

model based on a subset of the UML, which has successfully been checked for representing main MD properties at the conceptual level.

One of the main advances of this CASE tool is its capability for semi-automatically generating the corresponding implementation into a target commercial OLAP tool. This generation process involves a set of transformations from the modeling constructors used in the conceptual design (facts, dimension, classification hierarchies, etc.) into the target OLAP tool.

We are currently working on providing a standard metadata repository in XML format to assure us the portability of a conceptual design between different systems. We are also considering the integration of OLAP facilities in the CASE tool. This task involves data warehouse prototyping and sample data generation issues.

REFERENCES

Booch, G. (1994). *Object Oriented Analysis and Design with Applications.* Addison-Wesley, 2 edition.

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

Dinter, B., Sapia, C., Höfling, G., and Blaschka, M. (1998). The OLAP Market: State of the Art and Research Issues. In *Proc. of the ACM 1st Intl. Workshop on Data warehousing and OLAP (DOLAP'98)*, pages 22–27, Washington D.C., USA.

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

Golfarelli, M. and Rizzi, S. (1998). A methodological Framework for Data Warehouse Design. In *Proc. of the ACM 1st Intl. Workshop on Data warehousing and OLAP (DOLAP'98)*, pages 3–9, Washington D.C., USA.

Hahn, K., Sapia, C., and Blaschka, M. (2000). Automatically Generating OLAP Schemata from Conceptual Graphical Models. In *Proc. of the ACM 3rd Intl. Workshop on Data warehousing and OLAP (DOLAP'00)*, Washington D.C., USA.

Inmon, W. (1996). *Building the Data Warehouse.* John Wiley, 2 edition.

Kimball, R. (1996). *The data warehousing toolkit.* John Wiley, 2 edition.

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

Rational Software (2001). Rational Rose Family. http://www.rational.com.

Sapia, C., Blaschka, M., Hfling, G., and Dinter, B. (1998). Extending the E/R Model for the Multidimensional Paradigm. In *Proc. of the 1st Intl. Workshop on Data Warehouse and Data Mining (DWDM'98)*, volume 1552 of *LNCS*, pages 105–116. Springer-Verlag.

Trujillo, J. (2001). *The GOLD Model: An Object-Oriented Conceptual Model for the Design of OLAP Applications.* Ph.D. Thesis. DLSI. University of Alicante.

Trujillo, J., Gómez, J., and Palomar, M. (2000). Modeling the Behavior of OLAP Applications Using an UML Compilant Approach. In *Proc. of the 1st Intl. Conf. On Advances in Information Systems (ADVIS'00)*, volume 1909 of *LNCS*, pages 14–23. Springer-Verlag.

Trujillo, J., Palomar, M., Gómez, J., and Song, I.-Y. (2001). Designing Data Warehouses with OO Conceptual Models. *IEEE Computer, special issue on Data Warehouses*, 34(12).

Tryfona, N., Busborg, F., and Christiansen, J. (1999). starER: A Conceptual Model for Data Warehouse Design. In *Proc. of the ACM 2nd Intl. Workshop on Data warehousing and OLAP (DOLAP'99)*, pages 3–8, Kansas City, Missouri, USA.