Bridging the Semantic Gap in OLAP Models: Platform-independent Queries*

Jesús Pardillo
Lucentia Research Group
Department of Software and Computing Systems
University of Alicante, Spain
jesuspv@dlsi.ua.es

Jose-Norberto Mazón
Lucentia Research Group
Department of Software and Computing Systems
University of Alicante, Spain
jnmazon@dlsi.ua.es

Juan Trujillo
Lucentia Research Group
Department of Software and Computing Systems
University of Alicante, Spain
jtrujillo@dlsi.ua.es

ABSTRACT

The development of data warehouses is based on a three-stage process that starts specifying both the static and dynamic properties of *on-line analytical processing* (OLAP) applications by means of an intuitive, semantically rich abstraction, namely the conceptual model. Then, developers design its logical counterpart where platform-specific details such as performance or storage are also considered. Nevertheless, it is well known the existence of a semantic gap between the conceptual and logical levels that decreases the feasibility of their mapping. In order to bridge this gap, we propose the use of conceptual OLAP queries, *i.e.*, platform-independent, that can be automatically traced to their logical implementation in a coherent and integrated way. For this aim, in this paper, we focus on describing the specification of an OLAP algebra at the conceptual level by using the *object-constraint language* (OCL). Its operations are then translated into a particular OLAP system by using a *model-driven architecture* (MDA). The great advantage of our approach is that we allow analysts to query data warehouses without being aware of logical details.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*decision support*; H.2.1 [**Database Management**]: Logical Design—*data models*

General Terms

Design, Languages, Standardization

*This work has been supported by the ESPIA (TIN2007-67078) project from the Spanish Ministry of Education and Science, and QUASIMODO (PAC08-0157-0668) project from the Castilla-La Mancha Ministry of Education and Science. Jesús Pardillo and Jose-Norberto Mazón are funded by the Spanish Ministry of Education and Science under FPU grants AP2006-00332 and AP2005-1360, respectively.

DOLAP'08, October 30, 2008, Napa Valley, California, USA.

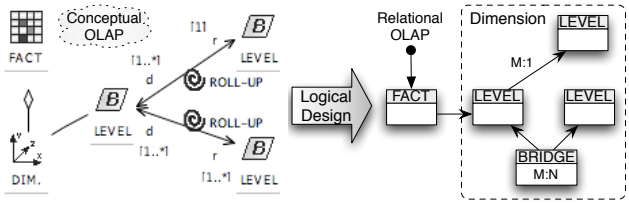

Figure 1: Aggregation-hierarchy logical designs

Keywords

MDA, OLAP, conceptual modelling, data warehouse

1. INTRODUCTION

The development of data warehouses [8] starts from their conceptual modelling by using intuitive, semantically rich abstractions [19] that are platform independent. Given a conceptual model, developers derive logical schemata tailored to a specific platform as result of particular design decisions. Nevertheless, it is well known the existence of a semantic gap between the conceptual and logical abstraction levels [19]. For instance, whereas conceptual models can represent complex aggregation hierarchies by means of their primitive modelling elements, in a relational deployment (see Fig. 1), they are usually mapped into many-to-one relationships [11] and bridge tables [8]. In this sense, it is difficult to properly establish the tool's metadata for querying them by using the *on-line analytical processing* (OLAP) technology (*e.g.*, being aware of bridge tables implementing many-to-many aggregation paths). Therefore, OLAP tools should take care of logical details such as the selected technology (*e.g.*, relational, multidimensional, etc.) or its logical variations (*e.g.*, star or snowflake schemata in relational deployments) for providing a conceptual view over them.

How to query such systems by overcoming the sketched pitfalls is the subject of our current work. We claim that the semantic gap between conceptual and logical models can be bridge by using a model-driven architecture [2, 15] where the design details are managed by the model-transformation scaffolding. Every semantic gap is thus removed when we provide a logical mapping for each multidimensional property captured at the conceptual level. This means that our proposal deals with not only the static part of a multidimensional model [14, 10] but also the dynamic one [16, 20, 3] in an integrated and automatic way. In this sense, we allow an-

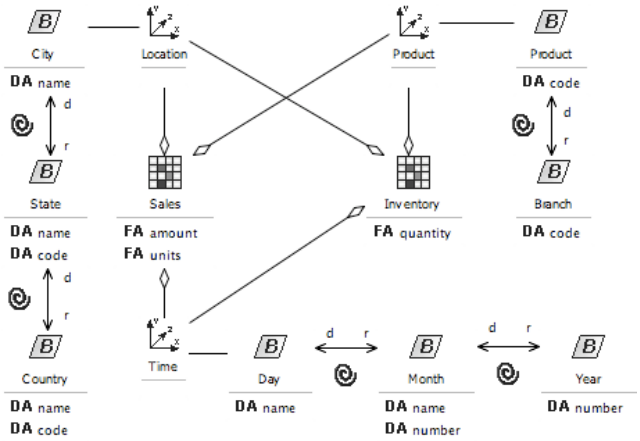

Figure 2: Scenario for sales and inventory analysis

alysts to express their queries at the conceptual level, which are mapped into the corresponding logical counterparts in order to response the analysts' information needs for a specific platform. This kind of approach has been successfully applied in our previous works on the repository [17, 14] and OLAP metadata [16]. Specifically, in this paper, we shall focus on the specification of an OLAP algebra in terms of the conceptual data model in order to provide analysts with query capabilities in a platform-independent manner. Thus, every information requirement can be expressed by means of a sequence of conceptual OLAP operations. The proposed solution uses widely-known standards from the software engineering [15], specifying both static and dynamic structures in a coherent and integrated framework that improves the understanding of both analysts and designers.

Next, in Section 2, we discuss the semantic gap and the involved assumptions in an example scenario. Then, we show how to define a conceptual algebra for the most common OLAP operations in Section 3, presenting the model-driven architecture in order to map them into their logical counterparts in Section 4. Section 5 reviews the related work and Section 6 exposes conclusions and future work.

2. BRIDGING THE SEMANTIC GAP

Due to their abstract nature, sometimes conceptual models suffer from no properly formalised semantics. This situation is not particular to multidimensional modelling, but it also appears in other well-known languages such as UML [15]. For this reason, a conceptual OLAP algebra that provides practical semantics in multidimensional data models requires to clearly specify them. In addition, there is a lack of standard for conceptual multidimensional models [19, 3]. Concerning literature, many conceptual frameworks have been presented [10, 1, 9, 6], whereas in industry, several commercial proposals have tried to become an OLAP standard, *e.g.*, the *common warehouse metamodel* (CWM) [15], *multidimensional expressions* (MDX) query language, OLAP4J, etc. However, such a lack is still present for data structures and OLAP operations at the conceptual level.

Our example scenario is based on the conceptual framework of [10] which is applied in Fig. 2. The presented data model is defined by means of two facts: *sales* and *inventory* (represented as ▦) with sales' *amount* and *units*, and inventory's *quantity* as measures (FA). These facts are described through three dimensions ( ): *time*, *location*, and *product*. In addition, each dimension contains several descriptors (DA) (that represent any kind of aggregation-level property) arranged through its aggregation hierarchies ( ) between different granularity levels ( ). For the sake of simplicity, our scenario omits advanced multidimensional properties such as inheritance or degenerate facts and dimensions, and also assumes that aggregation hierarchies are symmetric [11]. Nevertheless, in the next section, we extend the discussion for dealing with these elements.

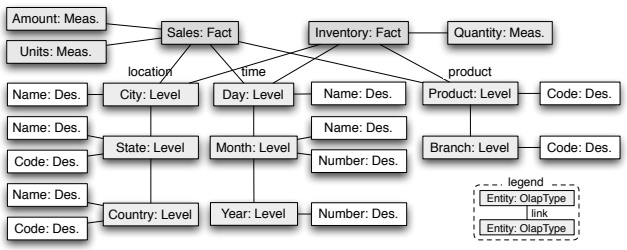

Figure 3: Object diagram of the abstract model

This conceptual modelling framework is implemented on the well-known *unified modelling language* (UML) [15]. This means that for each multidimensional property, a given UML metaclass, *i.e.*, language's primitive elements, has been extended by means of stereotypes and semantics constraints in order to host it in UML. Each one is also accompanied with a proper iconography to easily visualise it in UML diagrams.

Since the multidimensional modelling is interpreted as data models, its elements have a kind of semantics that allows their instantiation (because they are abstractions of a set of data entities). However, in conceptual models, there are usually modelling elements that violate these semantics in order to provide more suitable visual modelling elements for designers. For instance, [10] models dimensions that do not abstract any instance, due to they represent merely labels of the defining levels rooting aggregation hierarchies. In addition, other elements such as level descriptors are classified into categories (*e.g.*, object identifiers, descriptors, or dimension attributes) that do not have clear semantics related to an OLAP algebra. Even worse, sometimes these semantics are tailored to specific logical platforms. For these reasons, in order to be able to specify OLAP queries at the conceptual level, we should convert these visual-oriented models into a suitable intermediate representation.

2.1 Devisualisation of Conceptual Models

For providing conceptual modelling with proper OLAP semantics, every modelling element introduced as a visual decorator must be removed. In the resulting conceptual framework, only *classes* remain, *i.e.*, abstractions that model data occurrences from the domain problem. This process can be easily done by adapting every metaclass which cannot be related to any of the metaclasses involved in the UML class diagram [15]. Notice that since our approach [10] is already based on UML, this task can be additionally simplified.

Given the conceptual model of Fig. 2, its *devisualised* counterpart is sketched in Fig. 3 by using an object diagram [15]. This representation is useful for visualising oc-

currences from a given model without borrowing its concrete syntax or notation. In our scenario, we suppress the explicit representation of dimensions, thus, we shall use them as merely roles of the defining level regarding to the fact that they describe. In addition, each multidimensional element has been mapped into their abstract (devisualised) counterpart. For instance, every kind of dimension property in [10] is mapped into descriptors (abbreviated as *Des.* in Fig. 3, notice that measures are shown as *Meas.*). As it is stated, the remaining modelled elements conform the semantics of class diagrams. Then, it can be queried by OLAP algebras at the conceptual level as we discuss in the next section.

3. OLAP IN CONCEPTUAL MODELS

The main structural concept of an OLAP analysis is the data cube, that is in fact, what is multidimensionally modelled at the conceptual level. As a common practice, OLAP algebras thus operate data cubes. In an OLAP algebra, data cubes are represented as a set of occurrences from the fact class indexed by several occurrences through its related dimensions (specifically, their aggregation levels). In addition, the entire OLAP analysis is defined as a sequence of OLAP operations through these data cubes.

In our approach, we employ the *object-constraint language* (OCL) [15] that is a declarative language (despite of also providing some imperative constructions) for specifying restrictions on object-oriented models such as the UML-based multidimensional proposals [10, 1]. When OCL expressions do not impose boolean conditions on these models, they return data sets containing the modelled instances. Thus, we can employ OCL to specify OLAP queries at the conceptual level in an integrated way, where OCL expressions are equivalent to OLAP operations that manage data cubes as OCL data sets. Concerning their declaration, they are defined in the context of a given modelling element. From this context, we can *navigate* through the rest of the model in order to specify the required elements and restrictions.

The adopted data-cube definition is the following: a data cube is represented as a set of tuples where each one has a measure value and a coordinate system from its respective dimension levels. Its OCL type is:

$$Set(Tuple(V : T, CO : Tuple(D_1 : L_1, \ldots, D_n : L_n)))$$

where V is the measure under analysis over the domain T, CO is its coordinate system, $\{D_1, \ldots, D_n\}$ are the names of the involved dimensions, and $\{L_1, \ldots, L_n\}$ the actual aggregation levels. Notice that since we define data-cube's cells and their axes as tuples, we can rewrite every tuple's *named part* in a particular order without changing the tuple itself (because they are identified by name instead of position). With this definition, two cubes are equals iff they share the same (concrete) signature and they contain the same instances (of the coordinate system and measure). Despite of the OCL types are explicitly shown herein, in practice, they are not required, due to the expression type can be deduced by the OCL engine by evaluating its subexpressions' types. In this way, OCL can provide a very compact and, interestingly, directly executable definition of OLAP queries. Let us expose the following expression as example (the bodies of expressions are suppressed):

```
let cube1: Set(Tuple(V: Integer,              -- static types
  CO: Tuple(time: Year, product: Brach)))) = ...
let cube2 = ...                               -- dynamic types
```

The operation signature for the *ad-hoc* creation of data cubes is given next (due to the space constraints the operation's body is omitted). We also define an *ad hoc* OLAP query for *today's sales amount per city*. Notice that when we define a starting query with *Olap::query* its result can be *undefined* if we *query* cells by using non-indexed levels.

```
context Olap def: query(measure: OclAny, levels: Tuple(OclAny))
  : Set(V: OclAny, Tuple(CO: Tuple(OclAny))) = ...

let cube: Set(Tuple(V: OclAny, CO: Tuple(OclAny))) =
  Olap.query(Sales::amount, Tuple {
      time=Sales::day = Olap::today(), location=Sales::city})
in -- remaining OLAP analysis
```

Starting from a given data cube, during OLAP analyses, we can apply several OLAP operations in order to obtain the required information or gain insight about data. As we stated, there is no commitment of the OLAP operations that should be present in an OLAP algebra [20]. Nevertheless, [20] also exposes the backbone operations that every OLAP algebra specifies. Thus, for illustrative purposes, we can assume that there is a set of representative operations in OLAP algebras that we have extracted from [20]. Next, we show its specification in OCL at the conceptual level.

Change Base. This operation is borrowed from [1] where authors define it as an operation for modifying the dimensions of a data cell. This operation serves us in fact for establishing a new data cube from a given coordinate system (we specify the coordinate system before operation usage). For instance, we can obtain *sales amount per state and month*:

```
let coords: Set(Tuple(location: State, time: Month)) =
  let tmp: Set(Tuple(first: State, second: Month)) =
    State.allInstances()->product(Month.allInstances())
  in
  tmp->collect(t | Tuple {location=t.first, time=t.second})
in
let cubeCB: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Month)) =
  coords->collect(coord | Tuple {CO = coord,
      V = Sales->select(s |
          s.city.state = coord.location and
          s.day.month  = coord.time)->collect(amount)->sum()})
in -- remaining OLAP analysis
```

As it is shown, we employ the *product* operation for defining the coordinate systems by means of the Cartesian product of the involved dimensions. Then, we rename tuple's parts to the dimension names. From it, we check every point in the coordinate system and obtain the aggregated sales amount (applying the *sum* aggregation function) from the most granular data cube (notice that *allInstances* OCL operation gives us all the modelled data for a given OLAP element). It is worth noting that *change base* cannot reuse a particular data cube when the required base is more granular than the current one, *e.g.*, by adding additional dimensions or drilling down the current aggregation levels. Due to the space constraints, we omit a further explanation of the OCL syntax which can be referred in [15].

Dimension Addition & Removal. From the previous operation, we can extract other more atomic operations, specifically the dimension addition/removal and roll-up/drill-down through aggregation hierarchies. Now, we discuss the first ones. Adding a new dimension to an existing data cube implies its creation from a more granular version (due to the lack of required detail level in the current data cube). From

now on, let us refer our previous cubes (currently, *cubeCB*) in order to successively apply the discussed operations (notice the axis renaming after applying the Cartesian product between the *cubeCB*'s coordinate system and the new one):

```
-- adding products to cubeCB
let coords: Set(Tuple(location: State, time: Month,
    product: Branch)) =
  let tmp: Set(Tuple(first: Tuple(first: State, second: Month),
      second: Branch)) =
    cubeCB.CO->product(Branch.allInstances())
  in
  tmp->collect(t | Tuple {product = t.second,
      location = t.first.first, time = t.first.second})
in
let cubeDA: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Month))) =
  coords->collect(coord | Tuple {CO = coord,
      V = Sales->select(s |
          s.product.branch = coord.product and
          s.city.state     = coord.location and
          s.day.month = coord.time)->collect(amount)->sum()})
in -- remaining OLAP analysis
```

On the other hand, removing a given dimension implies that the current data-cube cells remain the same. Thus, we can completely reuse the source data cube. The next example is given from *cubeDA* to finally build a data cube that is equal to *cubeCB*:

```
-- removing products from cubeDA
let coords: Set(Tuple(location: State, time: Month)) =
  cubeDA.CO->collect(coord | Tuple
      {location = coord.location, time = coord.time})->asSet()
in
let cubeDR: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Month))) =
  coords->collect(coord | Tuple {CO = coord,
      V = cubeDA->select(cell |
          cell.CO.location = coord.location and
          cell.CO.time     = coord.time)->collect(V)->sum()})
in -- remaining OLAP analysis
```

Slice & Dice. These operations allow us to choose the subset of points of interest instead of managing the whole multidimensional space. Whilst *slice* means imposing restrictions on a given axis, *dice* imposes filters on several ones. The next example shows how to dice the data cube for *sales with an amount over* $5,000$ *located here*:

```
-- dicing by a particular dimension
let cubeSD: Tuple(V: Integer,
    CO: Tuple(location: State, time: Month)) =
  cubeDR->select(cell |
    cell.CO.locate = Olap.here() and cell.V > 5,000)
in -- remaining OLAP analysis
```

Alternative predicates can be applied by using the expressive power of OCL: all the boolean operators (*e.g.*, *and*, *or*, etc.), but also set-oriented operations such as *includes* [15].

Drill Across. This operation obtains the cells of a different data cube (in practice, for another fact) by holding the same coordinate system. For establishing the relationship between data cubes, it is required some kind of *semantic linkage* as [20] argues. These inter-schema relationships are explicitly modelled in some frameworks by using, for instance, flows or correlations [1]. Nevertheless, according to our previous work on data-marts development [17], we assume that these relationships are established by means of the *dimension sharing* between the facts that define data cubes. In our example, sales and inventory data cubes can be joined through all of their dimensions. This join links in fact the involved aggregation levels. If these levels correspond to the same entity, we will be able to drill across data cubes that employ them. This property is typically known as *conformity* among dimensions [8]. For instance, obtaining the inventory quantity from the sales' *cubeDR* requires a *conform* relation in order to check their semantic linkage. Usually, conformity on shared dimensions implies that $conform = equals$ is true. It is also worth noting that we employ a combination of *sum* and *avg* aggregation functions to assure inventory summarisability along time:

```
let cubeDAC: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Month))) =
  cubeSD->collect(cell | Tuple {CO = cell.CO,
      V = (let inv = Inventory.allInstances()->select(i |
          i.location.state.conform(cell.CO.location) and
          i.time.month.conform(cell.CO.time))
      in
      inv->collect(i.time.month)->asSet()->iterate(t; acc = 0 |
          acc + inv->select(i | i.time.month = t)
            ->collect(quantity)->sum())->avg())})
in -- remaining OLAP analysis
```

Multidimensional Projection. This operation selects some measures from those available in the related fact [20]. Due to our data-cube definition is oriented to consider different data cubes for each measure, this operation is then implemented by directly querying the most granular data. It is shown from the previous sales' *cubeSD*:

```
let cubeMP: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Month)) =
  cubeSD->collect(cell | Tuple {CO = cell.CO,
      V = Sales->select(s |
          s.location.state = cell.location and
          s.time.month = cell.time)->collect(quantity)->sum()})
in -- remaining OLAP analysis
```

Roll Up. It groups data-cube cells via aggregation hierarchies. This operation modifies the data granularity by moving from the more granular cells to the coarser ones:

```
-- months rolled up to years
let coords: Set(Tuple(location: State, time: Year)) =
  let tmp: Set(Tuple(first: State, second: Year)) =
    cubeMP.CO.location->product(cubeMP.CO.time.year->asSet())
  in
  tmp->collect(t | Tuple {location = t.first, time = t.second})
in
let cubeRU: Set(Tuple(V: Integer,
    CO: Tuple(location: State, time: Year)) =
  coords->collect(coord | Tuple {CO = coord,
      V = cubeMP->select(cell |
          cell.CO.location = coord.location,
          cell.CO.time     = coord.time)->collect(V)->sum()})
in -- remaining OLAP analysis
```

Drill Down. This operation is the opposite of the previous one: drilling down data cubes through an aggregation hierarchy results in a finer data cube. Differently from *roll up*, drilling down requires some kind of *undo* operation to recover the more granular data cube from a most granular version [1], *e.g.*, *cubeMP* (notice that each city only belongs to one state due to the symmetry of hierarchies [11]):

```
-- states drilled down to cities
let coords: Set(Tuple(V: Integer,
    CO: Tuple(location: City, time: Year)) =
  let tmp: Set(Tuple(first: City, second: Year)) =
```

```
    cubeRU.CO.location.city->product(cubeRU.CO.time)
  in
  tmp->collect(t | Tuple {location = t.first, time = t.second})
in
let cubeDD: Set(Tuple(V: Integer,
    CO: Tuple(location: City, time: Year)) =
  coords->collect(coord | Tuple {CO = coord,
      V = Sale.allInstances()->select(s |
          s.location        = coord.location,
          s.time.month.year = coord.time)->sum()})
in -- remaining OLAP analysis
```

Set-oriented Operations. They are primitive operations in OCL which are defined for the *Collection*s OCL type. Since an OLAP data cube is defined as a set of tuples in our framework, we can easily use operations such as the set's *union*, *intersection*, *difference*, etc. [15], but also user-defined operations without requiring any adaptation:

```
-- adding cells by applying set union
let cubeSO: Set(Tuple(V: Integer,
    CO: Tuple(location: City, time: Year))) =
  cubeDD->union(otherCube)
in -- remaining OLAP analysis
```

Until now, we have shown how to implement the operations that define well-known OLAP operations based on the backbone presented in [20]. Next, we sketch the OLAP-algebra treatment for some advance multidimensional properties at the conceptual level.

Dimension Inheritance. This semantic property is defined at conceptual level in order to represent a categorisation hierarchy between aggregation levels [10], contrasting to their typical aggregation hierarchy. These relationships imply particular information about the context of a given fact, when the specialised levels are involved in the OLAP analysis. In addition, they can be employed during data-cube *slice & dice* operations in order to take advantage of the level categories. In practice, due to data cubes are just a collection of data cells, inheritance involves additional mechanisms to adapt the employed coordinate system representation. Therefore, as other authors claim [13] in order to visually query the data warehouse, an additional data model and algebra related to visualisation should be required. Nevertheless, only concerning data manipulation, we can still use the OCL operator *oclIsKindOf* in order to check constraints or aggregate data cubes through the defined categories.

Push & Pull. These operations deal with the fact-dimension dichotomy in multidimensional models [20], converting measures into dimensions (*push*) or conversely (*pull*). In this way, conceptually speaking, we can change the focus of analysis from dimensions to measures and viceversa. As [20] states, these operations imply a navigation by (i) drilling across some kind of semantic relationship between measures and dimensions, and (ii) changing the base for the target data cube. Due to the space constraints we omit a particular example of this operation composition.

Degenerate Facts & Dimensions. These multidimensional elements are modelled when there are (i) interesting facts identified by another fact and some of its dimensions, or (ii) dimensions without an inner structure, respectively. In this sense, these elements are functionally equivalent to their respective canonical counterparts. An OLAP analysis should deal with these elements as simple fact and dimensions.

Derived Measures & Descriptors. Concerning multidimensional data, in the same way as almost any kind of data model, several mechanisms for deriving data can be conceptually modelled. In this cases, the OLAP algebra should remain unchanged, since derivation rules are already executed in the background. Nevertheless, it is interesting to take them into account when these rules are specifically executed respect to the entire data-cube aggregation process. If you do not take care of these details, summarisability inconsistencies could arise.

Summarisability & Complex Aggregation Functions. In a native manner, we can employ aggregation functions such as OLAP's count (*size* in OCL) and *sum* in our OCL-based OLAP algebra to assign a numeric value to a given data cube. Nevertheless, it is possible to define other functions in the same manner such as *avg* or even more complex functions. For instance, the following code defines the first one:

```
context Set(T) def: avg(): T =
  self->iterate(i: T; acc = 0: T | acc + i) / self->size()
```

Regarding to which aggregation function should be applied and whether summarisability constraints hold, it is required to deal with the specific semantics of the involved measures and dimensions apart from the actual needs of analysts. Due to the scope of the paper, we omit here an extended discussion about managing this kind of metadata.

As summary, we outline the sequence of operations involved during the OLAP analysis that we have carried out:

```
sales->changeBase(Tuple{location = City, time = Day}) -- cubeCB
  ->dimensionAddition(Tuple{product = Branch})         -- cubeDA
  ->dimensionRemoval(Tuple{product = Branch})          -- cubeDR
  ->sliceAndDice(Tuple{
      location = Tuple{op='equals', val=Olap.here()},
      V = Tuple{op='greater', val=5000})               -- cubeSD
 (a) ->drillAccross(Inventory::quantity)               -- cubeDA
 (b) ->multidimensionalProjection(Sales::quantity)     -- cubeMP
  ->rollUp(Tuple{time = Year})                         -- cubeRU
  ->drillDown(Tuple{location = City})                  -- cubeDD
  ->union(otherCube)                                   -- cubeSO
```

4. IMPLEMENTING OLAP QUERIES

During the transformation of a conceptual model to its logical counterpart, there are involved design decisions such as the related to platform suitability and performance assessment. As we stated, the solution of the semantic gap implies that, given a particular platform hosting and manipulating the logical schemata, there is a correspondence at the logical level of every multidimensional property defined at the conceptual one. It is worth noting that, cause several target platforms can be present, the semantic gap is always attached to a particular platform, *e.g.*, relational model in a ROLAP system. In these systems, we have to derive metadata for both database structures and OLAP metadata [16]. However, the standard mechanisms to specify OLAP metadata such as CWM [15] are commonly not expressive enough to deal with complex multidimensional properties. This drawback can be solved by their extension, but they then result in incompatibilities for the current OLAP tools. In this way, according to our proposal (see Fig. 5), directly querying the conceptual model overcomes this situation. Instead of providing OLAP metadata for a specific platform, we derive the required logical schemata and queries from the conceptual model, thus bridging the conceptual-logical semantic gap for the selected platform.

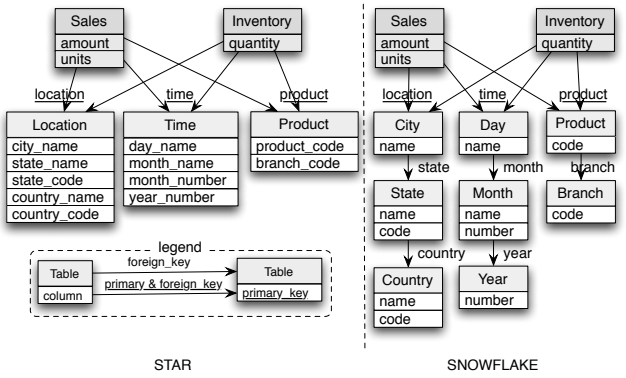

Figure 4: Schema variations for the logical design

Star & Snowflake Logical Variations. In order to illustrate the discussion, we show how to derive two of the most widely known logical representations of a data warehouse: the star and snowflake schemata [8]. First, given the intermediate model of Fig. 3, we use model-to-model transformations [2] in order to automatically derive both of these logical variations. The mapping involved in the star-schema transformation has been previously studied in [14]. Despite of this mapping is designed for the conceptual framework of [10] and exemplified with a particular design solution at the logical level, it can be easily adapted to our scenario. The resulting schemata are shown in Fig. 4.

After the conceptual model *devisualisation*, its semantics is equivalent to the class diagram's, and thus, its mapping into relational structures can be easily done. As Fig. 4 shows, whereas a snowflake schema maps every aggregation level to a different table, a star schema collapses aggregation hierarchies in a unique denormalised dimension table (notice that we omit the typical levels' surrogate keys [8] in this figure for the sake of simplicity). Depending on the preferred storage versus performance balance, the data-warehouse designer will be able to choose one of them. Given the data structure at the logical level, our conceptual OLAP algebra can be transformed into its relational calculus counterpart (see Fig. 5), thus articulating the conceptual query approach. As example, let us show the SQL template employed for querying these kinds of logical schemata:

```
-- star schema
SELECT DESCRIPTOR(d1.levelA_SK), ..., DESCRIPTOR(dN.levelZ_SK),
  AGGREGATION_FUNCTION(f.measure1), ...
FROM Fact f, Dimension1 d1, ..., DimensionN dN
WHERE f.FK1 = d1.PK AND ... f.FKN = dN.PK
  AND di.descriptorA = value AND ...
GROUP BY d1.levelA_SK, ..., dN.levelZ_SK

-- snowflake
SELECT DESCRIPTOR(d111.SK), ..., DESCRIPTOR(d211.SK), ...
  AGGREGATION_FUNCTION(f.measure1), ...
FROM Fact f,
  Dimension1Level1 d111, ..., Dimension1LevelN d11N,
  Dimension2Level1 d211, ..., Dimension2LevelM d21M, ...
WHERE f.FK1 = d111.SK AND ... f.FKN = d11N.SK
  AND d111.roll_FK = d112.SK AND ... d1LN-1.roll_FK = d11N.SK
  AND d211.roll_FK = d212.SK AND ... d2LM-1.roll_FK = d21M.SK
  AND di.descriptorA = value AND ...
GROUP BY d111.SK, ..., d211.SK, ...
```

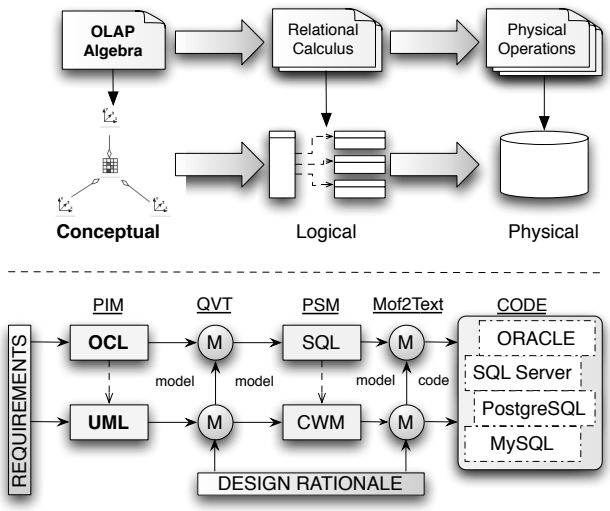

Figure 5: MDA for platform-independent queries

Model-transformation Architecture. The involved transformations are performed by using the architecture sketched in Fig. 5. After the selection of a logical platform and the data-model mapping from conceptual to logical level, each OCL operation defined in the OLAP algebra is transformed into their logical counterpart in the selected platform. For this aim, we employ the *model-driven architecture* (MDA) proposal [15] which was successfully employed for managing the data structures in [17, 14]. In MDA, the system implementation is conceived as result of a sequence of model-to-model and finally model-to-code transformations, from *platform-independent models* (PIM) –conceptual models in databases– to *platform-specific models* (PSM) –logical ones. These are designed by using standards transformation languages such as *query/view/transformation* (QVT) and *MOF (meta-object facility) models to text transformations* (Mof2Text) languages [15]. For instance, QVT includes a declarative sublanguage for specifying relations between the modelling elements of the two involved domain models. Given these relations, a compliant transformation engine can execute these mappings in order to automatically generate the final code. From the information requirements (see Fig. 5), we design our PIM by using UML (data-model static part) and OCL (dynamic) that actually depends on the specific data modelling framework, *e.g.*, [10, 1]. Then, these models are mapped into CWM and SQL standards as PSM where the algebra mapping depends on both the design rationale for the logical level and the actual conceptual-logical data mapping. Finally, these intermediate representations derive code implementing the designed OLAP system in a given commercial platform. It is worth noting that this architecture deal with the underlying metamodels [2] of each modelling language. Therefore, the PIM in Fig. 5 is just the presented in Fig. 3 instead of its visual counterpart (see Fig. 2) that results from conceptual modelling.

By using the OCL metamodel, we provide a representation of the conceptual query that is suitable to be allocated into a QVT mapping. The designed QVT mapping then relates the OCL query to the *standard query language* for relational

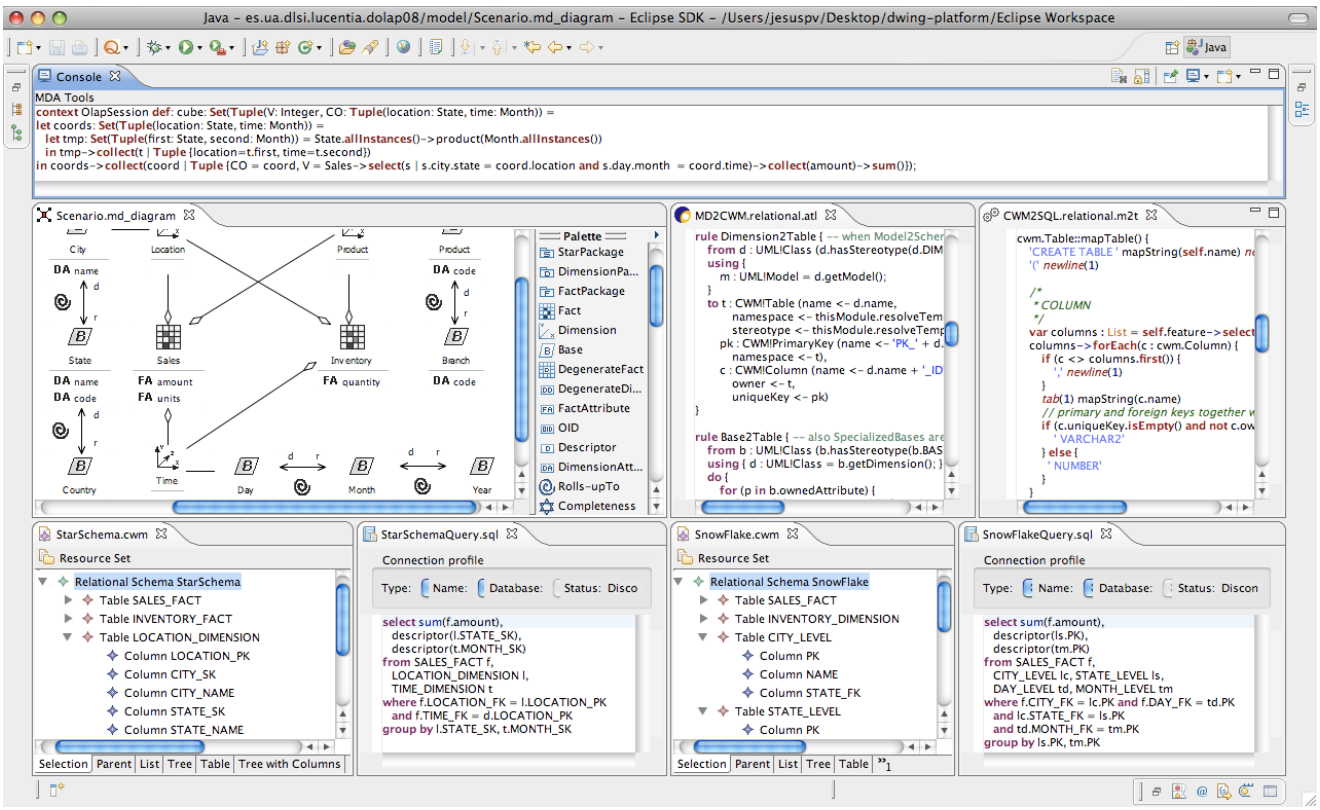

Figure 6: Implementation of the model-driven architecture in the *Eclipse* development platform

databases, namely SQL. This transformation is quite similar to the presented in [5], however, in our case, it is carried out by using well-known standards of the software industry for the model-driven development [2]. This process is assured by the fact that every OCL operation, due to their mathematical foundation based on the set theory [15], can be mapped into an equivalent SQL query on a relational schema [12]. All of these technologies have been implemented in the *Eclipse* development platform[1] by taking advantage of its modular architecture in order to provide the required implementation of the MDA standards. Fig. 6 shows this implementation for our example scenario. In the upper part of this figure, there is an example OCL query that is applied to the conceptual model of our scenario (on the left-hand side). By applying the designed set of model-to-model (in the middle) and model-to-code (right-hand side of figure) transformations, it is mapped into the star and snowflake logical schemata (on the lower part) as we have previously discussed. Given these schemata, the derived SQL queries (at the logical level) that properly response the same information requirements than the initial OCL query (at the conceptual) are also shown on the right-hand side of each logical schema. In this way, we provide a development environment for model-driven OLAP systems that enable analysts to query them in a platform-independent manner.

5. RELATED WORK

To the best of our knowledge, the main effort on studying OLAP algebras has been done in [20]. First, in [1], authors present a conceptual model with its two concerns, *i.e.*, as a static data model and as a dynamic analytical abstraction. They formalise both by using UML and a mathematical formulation, respectively. They provide an algebra for the OLAP operations that they state as backboned in every proposal [20]. Nevertheless, they do not provide suitable mechanisms for also taking advantage of the additional semantic relationships that they include in its model and also their formulation is not suitable for management. Importantly, as it occurs with many OLAP-algebra formulations [3], their mathematical scaffolding, mainly focused on studying their foundations, makes them difficult to understand and manipulate. Thus, they are not properly integrated in the whole conceptual modelling framework. On the other hand, it is worth noting that [7] formalises a powerful operator based on SQL for data-cube aggregation but at the logical level.

Concerning conceptual-logical equivalencies of OLAP algebras, [1] provides a mapping from their mathematical formalisation into SQL as the same manner that we have done with OCL. In the same line, several works related to software engineering [12, 5] expose quite similar mappings between OCL and relational calculus. On the other hand, it is well-known the necessity of providing visual algebras over OLAP (data-oriented) ones for visualising and interacting with data cubes in the manner of [13]. For example, [18] defines a graphical language to query data warehouses, whereas [4] directly represent queries by marking the conceptual model.

Interestingly, there is an analogy between the operations presented in an OLAP algebra and the identified data interactions in the visual-analytics discipline. Specifically, [21] ar-

[1] `http://www.eclipse.org`

gue about a widely-known *information seeking mantra* that summarises, in this sense, all human-computer interactions in a reduced set of operations. These operations are always "overview first, zoom & filter, details on demand". As readers can notice, this mantra can be also equally applicable to OLAP analyses where OLAP algebras implement these operations: *e.g.*, overviews, zooming, and details are implemented by OLAP's *roll-ups* and *drill-downs*, whereas the filtering's counterpart is done by the *slice & dice* operations.

6. CONCLUSIONS

In this work, we present how expressive conceptual models can be translated into their platform-specific logical counterparts by bridging their semantic gap [19]. Our solution comprises (i) the definition of an OLAP algebra that response analysts' information needs at the conceptual level, and (ii) a model-transformation architecture that automatically manages and derives from them the different logical designs, being aware of the platform-specific details. Our approach takes advantage of the well-known standards of the software engineering [15], *e.g.*, UML, OCL, or QVT, that enable an integrated solution for querying data warehouses. Its feasibility has been shown by specifying in OCL each of the most common OLAP operation in every OLAP algebra [20]. OCL is a well-known standard for querying object-oriented models, that has been successfully employed in order to automatically derive both data structures and OLAP queries for the well-known star and snowflake logical schemata in a SQL-based relational platform.

The main benefits of our approach are: it is the first approach that employs an standard declarative language, namely OCL, that is widely known and understandable in the software-engineering discipline, for textually specifying OLAP data cubes. Second, since it is difficult to intuitively represent the dynamic part of a multidimensional model within a pure mathematical notation, our approach provides an intuitive and easy way to model queries integrated into the conceptual data modelling. Importantly third, querying at the conceptual level implies that analysts do not need to be aware of which design decisions have been taken in order to implement the conceptual model to be queried. In fact, the model-transformation scaffolding performs the work of automatically managing and translating conceptual queries to their logical counterparts regarding to a specific platform.

Our future work includes additional mechanisms to enrich and improve the readability in OCL of conceptual OLAP algebras, the automation of the algebra mappings without requiring the explicit knowledge about logical platforms, enriching requirements elicitation by specifying conceptual OLAP queries for frequent user needs, and revisited normal forms [9] for alleviating anomalies in conceptual multidimensional models concerning their analysis capacity.

7. REFERENCES

[1] A. Abelló, J. Samos, and F. Saltor. YAM^2: a multidimensional conceptual model extending UML. *Inf. Syst.*, 31(6):541–567, 2006.

[2] J. Bézivin. Model Driven Engineering: An Emerging Technical Space. In *GTTSE*, pages 36–64, 2006.

[3] M. Blaschka, C. Sapia, G. Höfling, and B. Dinter. Finding Your Way through Multidimensional Data Models. In *DEXA Workshop*, pages 198–203, 1998.

[4] L. Cabibbo and R. Torlone. From a Procedural to a Visual Query Language for OLAP. In *SSDBM*, pages 74–83, 1998.

[5] B. Demuth and H. Hußmann. Using UML/OCL Constraints for Relational Database Design. In *UML*, pages 598–613, 1999.

[6] M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: A Conceptual Model for Data Warehouses. *Int. J. Cooperative Inf. Syst.*, 7(2-3):215–247, 1998.

[7] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *CoRR*, abs/cs/0701155, 2007.

[8] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. Wiley, 2002.

[9] J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Inf. Syst.*, 28(5):415–434, 2003.

[10] S. Luján-Mora, J. Trujillo, and I.-Y. Song. A UML profile for multidimensional modeling in data warehouses. *Data Knowl. Eng.*, 59(3):725–769, 2006.

[11] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: From conceptual modeling to logical representation. *Data Knowl. Eng.*, 59(2):348–377, 2006.

[12] L. Mandel and M. V. Cengarle. On the Expressive Power of OCL. In *World Congress on Formal Methods*, pages 854–874, 1999.

[13] A. S. Maniatis, P. Vassiliadis, S. Skiadopoulos, and Y. Vassiliou. Advanced visualization for OLAP. In *DOLAP*, pages 9–16, 2003.

[14] J.-N. Mazón and J. Trujillo. An MDA approach for the development of data warehouses. *Decis. Support Syst.*, 45(1):41–58, 2008.

[15] Object Management Group. Model Driven Architecture (MDA) proposal. `http://www.omg.org`.

[16] J. Pardillo, J.-N. Mazón, and J. Trujillo. Model-Driven Metadata for OLAP Cubes from the Conceptual Modelling of Data Warehouses. In *DaWaK*, pages 13–22, 2008.

[17] J. Pardillo and J. Trujillo. Integrated Model-driven Development of Goal-oriented Data Warehouses and Data Marts. In *ER*, pages 426–439, 2008.

[18] F. Ravat, O. Teste, R. Tournier, and G. Zurfluh. Graphical Querying of Multidimensional Databases. In *ADBIS*, pages 298–313, 2007.

[19] S. Rizzi, A. Abelló, J. Lechtenbörger, and J. Trujillo. Research in data warehouse modeling and design: dead or alive? In *DOLAP*, pages 3–10, 2006.

[20] O. Romero and A. Abelló. On the Need of a Reference Algebra for OLAP. In *DaWaK*, pages 99–110, 2007.

[21] B. Shneiderman. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *VL*, pages 336–343, 1996.