A UML Profile for Multidimensional Modeling in Data Warehouses

Sergio Luján-Mora, Juan Trujillo
Department of Software and Computing Systems
University of Alicante (Spain)
{slujan,jtrujillo}@dlsi.ua.es

Il-Yeol Song
College of Information Science and Technology
Drexel University (USA)
songiy@drexel.edu

Abstract

The Multidimensional (MD) modeling, which is the foundation of data warehouses (DWs), MD databases, and On-Line Analytical Processing (OLAP) applications, is based on several properties different from those in traditional database modeling. In the past few years, there have been some proposals, providing their own formal and graphical notations, for representing the main MD properties at the conceptual level. However, unfortunately none of them has been accepted as a standard for conceptual MD modeling.

In this paper, we present an extension of the Unified Modeling Language (UML) using a UML profile. This profile is defined by a set of stereotypes, constraints and tagged values to elegantly represent main MD properties at the conceptual level. We make use of the Object Constraint Language (OCL) to specify the constraints attached to the defined stereotypes, thereby avoiding an arbitrary use of these stereotypes. We have based our proposal in UML for two main reasons: (i) UML is a well known standard modeling language known by most database designers, thereby designers can avoid learning a new notation, and (ii) UML can be easily extended so that it can be tailored for a specific domain with concrete peculiarities such as the multidimensional modeling for data warehouses. Moreover, our proposal is Model Driven Architecture (MDA) compliant and we use the Query View Transformation (QVT) approach for an automatic generation of the implementation in a target platform. Throughout the paper, we will describe how to easily accomplish the MD modeling of DWs at the conceptual level. Finally, we show how to use our extension in Rational Rose for MD modeling.

Keywords: UML, multidimensional modeling, data warehouses, UML extension, UML profile

1 Introduction

Data warehouses (DW), multidimensional (MD) databases, and On-Line Analytical Processing (OLAP) applications provide companies with many years of historical information for decision making process. It is widely accepted that these systems are based on multidimensional (MD) modeling. MD modeling structures information into facts and dimensions. A fact contains interesting measures of a business process (sales, deliveries, etc.), whereas a dimension (product, customer, time, etc.) represents the context for analyzing a fact. The benefit of using this MD modeling is two-fold. On one hand, the MD model is close to the way of thinking of data analyzers and, therefore, helps users understand data better. On the other hand, the MD model supports performance improvement as its simple structure allows us to predict end users' intentions[1].

---

[1] We distinguish between *DW developers* (technical users, users who design and build the DW) and *DW end users* (users who are only interested in the business content or users who query the DW).

Some approaches have been proposed lately (presented in Section 3) to accomplish the conceptual design of these systems. Unfortunately, none of them has been accepted as a standard for DW conceptual modeling. These proposals try to represent main MD properties at the conceptual level with special emphasis on MD data structures (i.e. facts and dimensions). However, from our point of view, none of them considers all the main properties of MD systems at the conceptual level. Furthermore, these approaches provide their own graphical notations, which forces designers to learn a new specific model together with its corresponding MD modeling notation.

On the other hand, the Unified Modeling Language[2] (UML) [6, 29, 31] has been widely accepted as the standard object-oriented (OO) modeling language for modeling various aspects of software systems. Therefore, any approach using the UML will minimize the effort of developers in learning new notations or methodologies for every subsystem to be modeled. Another outstanding feature of the UML is that it is an extensible language in the sense that it provides mechanisms (stereotypes, tagged values, and constraints) to introduce new elements for specific domains if necessary, such as web applications, database applications, business modeling, software development processes, etc. [8, 27]. A collection of enhancements that extend an existing diagram type to support a new purpose is called a *profile*. On the other hand, UML is highly scalable as a new tailored UML profile can further be enriched by adding new properties to adapt it to new situations. Furthermore, the UML follows the OO paradigm, which has been proved to be semantically richer than other paradigms for MD modeling [3].

Following these considerations, we have previously proposed in [40] an OO conceptual MD modeling approach, based on the UML, for a powerful conceptual modeling of MD systems. This proposal considers major relevant MD properties at the conceptual level in an elegant and easy way. Furthermore, in [26] we applied the grouping mechanism called *package* provided by the UML. In this way, when modeling complex and large DW systems, we are not restricted to use flat UML class diagrams. Moreover, in [25] we presented a UML profile for MD modeling based on our previously proposed approaches.

In this paper, we present a UML profile for a coherent and unified conceptual MD modeling that joins our previous approaches presented in separate works. This profile expresses for each measure its MD context in terms of relevant dimensions and their hierarchies and allows us to easily and elegantly consider main MD properties at the conceptual level, such as the many-to-many relationships between facts and dimensions, degenerate dimensions and facts, multiple and alternative path classification hierarchies, and non-strict and complete hierarchies. Our extension uses the Object Constraint Language (OCL) [31, 42] for expressing well-formedness rules of the new defined elements, thereby avoiding an arbitrary use of this extension. Moreover, we program this extension in a well known model-driven development tool such as Rational Rose [34] to show its applicability.

Furthermore, in this paper we have enriched our previous works with new properties to obtain an improved proposal. The major important new issues considered in this profile are as follows:

- Roles and cycles: we explicitly specify roles on the associations to define classification hierarchies along dimensions to help us avoid the existence of cycles. We set the constraint DAG attached to a dimension to make sure that no cycles were defined.

- Dimensions: we have re-defined the concept of dimension to make it more understandable and readable by end users. In previous versions, the notion of dimension was defined as a dimension class in which attributes belonging to the lowest classification hierarchy level had to be defined in it. However, we consider it is more intuitive to have a class just to define the dimension to which a fact is related, and then, the required classification hierarchy levels are defined from the dimension.

- Different roles of one dimension: we can use the same dimension in different roles related to one fact by connecting it through different associations to the same fact. By using different roles, we make the specification of different roles more technical and formal, thereby facilitating

---

[2]We base our proposal in UML 2.0 [31]. Adoption of the UML 2.0 Superstructure is complete, but adoption of the other three parts of UML 2.0 (Infrastructure, OCL, Diagram Interchange) is nearly complete.

further design phases such as an automatic generation of the corresponding implementation or the definition of different views of cubes for different users.

- Association classes and degenerate facts: we can specify degenerate facts in many-to-many relationships by using association classes provided by the UML. In previous versions, we could specify these many-to-many relationships by means of the cardinality between classes. However, we could not explicitly define attributes for these relationships, which is frequently occurring for many many-to-many relationships. Therefore, the formalization of this fact provides a significant advantage compared to the previous version.

- Navigability: we can explicitly define the navigability of an association in a classification hierarchy to define a roll-up or drill-down (Roll-up is the presentation of data at a higher level of detail, whereas drill-down is the presentation of data at a lower level of detail) path by default when there exists different paths.

- Implementation: we intend to automatically generate the corresponding implementation of a MD model. To accomplish this goal, we use the Query View Transformation (QVT) approach for expressing model transformations. However, with QVT we are able to specify complex one-to-many transformations from a conceptual model into several logical models by composing simpler transformations using composition functions.

In summary, with the old and new properties, we intend to achieve a proposal with the following properties to obtain better models [37]:

- Accurate: a profile that allows us to represent all major important features of MD modeling at the conceptual level.

- Non-Redundant: we allow us to import a previously-defined element in our model whenever possible so that we avoid having different definitions and properties for the same concept throughout a model.

- Consistent: our approach is founded in a set of constraints that avoid any logical contradictions within a model.

- Simple: as simple as possible. We limit our graphical notation to a minimal subset of UML elements that allows us to correctly describe main MD properties at the conceptual level.

- Understandable: we attempt to make a proposal understandable for the intended audience (both DW designers and end users). When we build complex and huge DW systems, it is highly important to have a modeling approach that can successfully communicate with different actors who take part in the DW design. We provide this feature by using the UML grouping mechanism called *package*, thereby allowing us to define three different levels of abstraction and to avoid the use of flat diagrams when modeling large DW systems. In case that the DW is not too complex, the designer can merge the three levels into only one, and therefore, use the classical flat diagram approach.

The remainder of this paper is structured as follows: Section 2 introduces the main properties and aspects that a conceptual approach for MD modeling should take into consideration. Section 3 summarizes the different UML Extensibility Mechanisms that we can use to adapt the UML to a particular domain, context or model. Section 4 describes how we make use of the UML to consider all major properties of MD modeling at the conceptual level. Section 5 describes the transformation of MD models based on Query/View/Transformation (QVT) approach. Section 6 introduces a case study to illustrate the use of our approach and shows how to use our *profile* for MD modeling in Rational Rose. Section 7 summarizes the most relevant conceptual approaches proposed so far by the research community and provides a comparison framework between them. Section 8 presents the main conclusions and introduces our immediate future works. Finally, Appendix A formally defines the new UML extension (*profile*) we propose for MD modeling and Appendix B presents a list of acronyms used in this paper.

2 Multidimensional Modeling

In MD modeling, information is structured into **facts** and **dimensions**[3]. A fact is an item of interest for an enterprise, and is described through a set of attributes called **measures** or **fact attributes** (atomic or **derived**), which are contained in cells or points in the data cube. This set of measures is based on a set of dimensions that determine the granularity adopted for representing facts. On the other hand, dimensions provide the context in which facts are to be analyzed. Moreover, dimensions are also characterized by attributes, which are usually called **dimension attributes**.

Let us introduce a DW modeling example inspired by a case study presented by Giovinazzo in [14], which will be used throughout the rest of the paper. This example relates to a company that comprises different dealerships that sell automobiles (cars and vans) across several states. The DW contains three data marts[4], such as *automobile sales*, *part sales* and *service works* (they are separated because they are going to be used by different end users). However, these data marts share some common dimensions[5] such as *dealership* or *time*, although they also have their own particular dimensions, such as *salesperson* or *service*:

- Automobile sales (AS): considers the sales of automobiles.

- Part sales (PS): represents the sales of parts of automobiles such as spare wheels or light bulbs.

- Service works (SW): considers the services realized by dealerships such as change of lubricating oil or brake oil.

Every one of these models has corresponding fact which contains the specific measures to be analyzed. Furthermore, they consider the following dimensions to analyze measures: *dealership*, *time*, *customer*, *salesperson* and *auto* for the AS; *dealership*, *time*, *service*, *mechanic* and *parts* for the PS; and *dealership*, *time*, *service*, *mechanic* and *parts* for the SW. On the left hand side of Figure 1, we can observe a data cube typically used for representing a MD model. In this particular case, we have defined a cube for the AS for analyzing measures along the *auto*, *customer* and *time* dimensions.

We note that *many-to-one* relationships exist between the fact and every particular dimension, and thus facts are usually considered to have *many-to-many* relationships between any of two dimensions. In the previous AS, an *autosales* fact is related to only one *auto* that is sold by one *dealership* and purchased by just one *customer* at one *time*.

Nevertheless, there are some cases in which ***many-to-many*** relationships may exist between the fact and some particular dimensions. For example, the *autosales* fact of AS is considered to have a particular *many-to-many* relationship to the *salesperson* dimension, as more than one *salesperson* may have participated in selling one *auto* (although every auto is still purchased by only one customer in just one *dealership* store and at one *time*).

When having a *many-to-many* relationship with a particular dimension as previously-described, we usually need to describe specific attributes to provide further features for every instance combination in this particular relationship. In doing so, the measures provided are usually called ***degenerated facts*** [19, 14]. In the previous example, we may be interested in recording the specific *commission* that a *salesperson* obtains for every particular auto sales he/she participates.

There are some cases in which we do not consider a dimension explicitly because we believe that most of its properties is already represented throughout other elements (facts and dimensions) in our MD model. However, we still believe that we need some attribute or property in the fact to uniquely identify fact instances. When this occurs, we usually call these dimensions as ***degenerated dimensions*** [19, 14]. Therefore, a degenerate dimension is one whose identifier exists only in a fact,

---

[3]We avoid the terms *fact table* or *dimension table* during conceptual modeling, as a *table* suggests logical storage in a relational database management system (RDBMS).

[4]A data mart is a type of data warehouse primarily designed for addressing a specific function or department's needs: whereas a data warehouse combines databases across an entire enterprise, a data mart is usually smaller and focus on a particular subject or department. According to [14], there are two kinds of data marts: "dependent data marts receive their data from the enterprise data warehouse; independent data marts receive data directly from the operational environment".

[5]Common dimensions used in different data marts are usually called *conformed dimensions*[19].

but which is not materialized as an actual dimension. This provides other fact features in addition to the measures for analysis. In our example, instead of considering the *autosales*, we could had represented the bill of an *autosales* and consider the *bill* and *bill line numbers* as other bill features (while not having a *bill* dimension materialized).

With reference to measures, the concept of **additivity** or summaribility [5, 15, 19, 40, 41] on measures along dimensions is crucial for MD data modeling. A measure is additive along a dimension if the SUM operator can be used to aggregate attribute values along all hierarchies defined on that dimension. The aggregation of some fact attributes (roll-up in OLAP terminology), however, might not be semantically meaningful along all dimensions. For example, all measures that record a static level, such as inventory levels, financial account balances or temperatures, are not inherently additive along the *time* dimension. In our particular warehouse example, the measure *quantity* from that records the quantity of a specific *auto* in a *sale* at a given *time* is not additive along the *salesperson* dimension. However, other aggregation operators (e.g. MAX, MIN and AVG) could still be used along the same *salesperson* dimension. Moreover, *quantity* can be additive along the *auto* dimension. Thus, a measure such as *quantity* is called semiadditive since it is additive along one dimension, but non-additive along another dimension.

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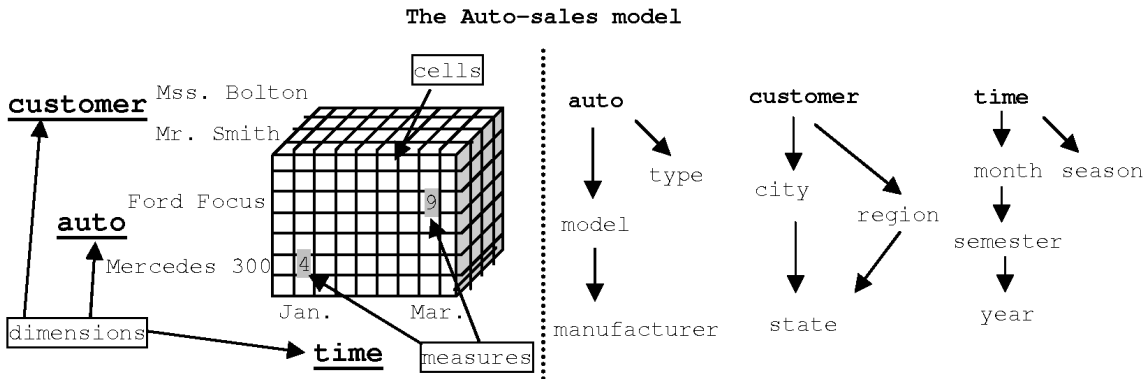

Figure 1: A data cube and classification hierarchies defined on dimensions

On the right hand side of Figure 1, we can observe different classification hierarchies defined on the *auto*, *customer* and *time* dimensions from the AS[6]. On the *auto* dimension, we have considered a multiple classification hierarchy to be able to aggregate data values along two different hierarchy paths: (i) *auto*, *model*, *manufacturer* and (ii) *auto*, *type*. There may exist other attributes that are not used for aggregating purposes and provide features for other dimension attributes (e.g. *auto description*). On the *customer* dimension, we have defined an alternative path classification hierarchy with two different paths that converge into the same hierarchy level: (i) *customer*, *city*, *state* and (ii) *customer*, *region* and *state*. Finally, we have also defined another multiple classification hierarchy with the following paths on the *time* dimension: (i) *time*, *month*, *semester*, *year* and (ii) *time* and *season*.

Nevertheless, classification hierarchies are not so simple in most cases. The concepts of **strictness** and **completeness** are important, not only for conceptual purposes, but also for further design steps of MD modeling [41]. "Strictness" means that an object of a lower level of a hierarchy belongs to *only*

---

[6]These classification hierarchies are different from those specifically presented by Giovinnazo in [14] as ours will allow us to consider more peculiarities.
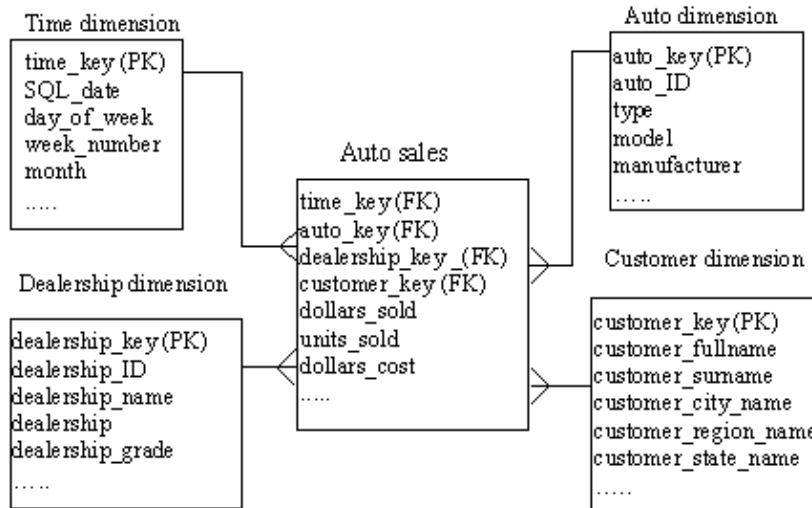
Figure 2: An example of the star schema suggested by R. Kimball [19]

one of a higher level, e.g. a *city* is related to only one *state*. "Completeness" means that all members belong to one higher-class object and that object consists of those members only. For example, suppose we say that the classification hierarchy between the *state* and *city* levels is "complete". In this case, a *state* is formed by *all* the *cities* recorded and all the *cities* that form the *state* are recorded.

OLAP scenarios sometimes become very large as the number of dimensions increases significantly, and therefore, this fact may lead to extremely sparse dimensions and data cubes. In this way, there are attributes that are normally valid for all elements within a dimension while others are only valid for a subset of elements (also known as the **categorization of dimensions** [21, 41]). For example, attributes *number of passengers* and *number of airbags* would only be valid for *cars* and will be "null" for *trucks*. Thus, a proper MD data model should be able to consider attributes only when necessary, depending on the categorization of dimensions.

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

Once the structure of the MD model has been defined, end users usually identify a set of initial user requirements as a starting point for the subsequent data analysis phase. From these initial requirements, users can apply a set of operations (usually called OLAP operations [7, 19]) to the MD view of data for further data analysis. These OLAP operations are usually as follows: roll-up (increasing the level of aggregation) and drill-down (decreasing the level of aggregation) along one or more classification hierarchies, slice-dice (selection and projection) and pivoting (re-orienting the MD view of data which also allows us to exchange dimensions for facts; i.e., symmetric treatment of facts and dimensions).

Let us now conclude this section by providing a brief reference to the star schema proposed by R. Kimball [19] as this star schema (and its variants fact constellations and snowflake) is widely known and used for implementing a data warehouse in relational systems. Basically, the star schema represents each dimension as a *dimension table* and each fact as a *fact table* with a many-to-many relationship with all the dimensions. Figure 2 shows an example of a star schema. In this particular schema, the fact is the name of the middle box, *Auto sales* fact table. Measures are the non-foreign keys in the *Auto sales* fact table. Dimensions (*Time*, *Customer*, *Auto* and *Dealership*) are each of the boxes connected to the fact table in a one-to-many relationship. Each dimension contains relevant
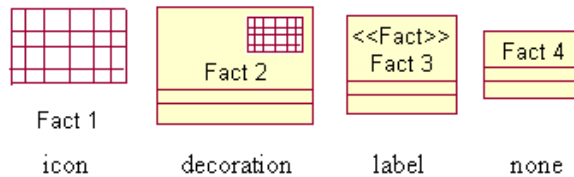
6

Figure 3: Different representations for a stereotyped class

attributes for analyzing the context of measures. As the star schema is at the logical level, its main purpose is not to represent main multidimensional properties such as classification hierarchies and their different types as above-mentioned or the additivity of measures.

3 UML Extensibility Mechanism

The OMG proposes two mechanisms to extend UML [31]: (i) the "heavyweight extension mechanism" and (ii) the "lightweight extension mechanism". A heavyweight extension mechanism allows us to adapt the UML semantics by extending the standard UML metamodel. In contrast, a lightweight extension mechanism allows us to adapt the UML semantics without changing the UML metamodel. This latter extension mechanism is supported by the UML Extensibility Mechanism package, which is the subpackage from the UML metamodel that specifies how particular UML model elements are customized and extended with new semantics by using stereotypes, tagged values, and constraints. A coherent set of such extensions, defined for specific purposes, constitutes a UML *profile* [13]. For example, [29] includes a standard profile for modeling software development processes and another one for business modeling.

A UML extension defined using a profile must be strictly additive to the standard UML semantics. This means that such extension must not conflict with or contradict the standard semantics. Therefore, there are restrictions on how a profile can extend the UML metamodel [31].

A *stereotype* is a model element that defines additional values (based on tagged values), additional constraints, and optionally a new graphical representation (an icon). A stereotype allows us to attach a new semantic meaning to a model element. A stereotype is either represented as a string between a pair of guillemets ($\ll$ $\gg$) or rendered as a new icon. In Figure 3, we show the different possible representations of a stereotype: (i) *icon*, in which the stereotype is represented with the new defined icon, (ii) *decoration*, where the icon is placed in the upper right hand side of the corresponding usual UML icon, (iii) *label*, in which the stereotype name is shown between a pair of guillemets ($\ll$ $\gg$), and (iv) *none*, where the new stereotype is represented by the normal UML element for the corresponding stereotype type and no external evidence of the stereotype is shown.

A *tagged value*[7] specifies a new kind of property that may be attached to a model element. A tagged value is rendered as a string enclosed by a pair of braces ({ }) and placed below the name of another element. A tagged value has the form name = value where name is the name of the tagged value and value is an arbitrary string that denotes its value.

A *constraint* can be attached to any model element to refine its semantics. As it is stated in [42], "A constraint is a restriction on one or more values of (part of) an object-oriented model or system". In the UML, a constraint is rendered as a string between a pair of braces ({ }) and placed near the associated model element. A constraint on a stereotype is interpreted as a constraint on all types to which the stereotype is applied. A constraint can be defined by means of an informal explanation in Natural Language and/or by means of OCL [31, 42] expressions. The OCL is a declarative language that allows software developers to write constraints over object models.

---

[7]There exists a confusion between tag definition and tagged value: a *tag definition* specifies the tagged values that can be attached to a kind of model element, whereas a *tagged value* is the actual value of a tag definition in a particular model.

4   Object-Oriented Multidimensional Modeling

Throughout this section, we will use a running example to illustrate the basics and the applicability of our OO MD approach. We use the same example presented in Section 2 and inspired by a case study from [14].

As our proposal addresses the DW design at a conceptual level, some implementation issues such as primary and foreign keys or data types are not our first priority. Therefore, the goal of our proposal is the representation of the main structural aspects of MD modeling at the conceptual level.

In our approach, the main structural properties of MD models are specified by means of a UML class diagram in which the information is clearly separated into facts and dimensions. The main features considered are the many-to-many relationships between facts and dimensions, degenerate facts and dimensions, multiple and alternative path classification hierarchies, and non-strict and complete hierarchies. Our approach proposes the use of UML packages in order to group classes together into higher level units creating different levels of abstraction, and therefore, simplifying the final model. In this way, when modeling complex and large DW systems, the designer is not restricted to use flat UML class diagrams.

Our proposal is formally defined as a UML extension by means of a UML profile. Although we provide the complete formal definition of our extension in the next section, we introduce the main stereotypes and some tagged values in this section (highlighted in the text using a SMALL CAPS font). In a diagram, UML allows us to represent a stereotype in four different ways. In Figure 3, we show four possible representations of a class with the FACT stereotype (one of the stereotypes we propose): icon (the stereotype icon is displayed), decoration (the stereotype decoration is displayed inside the element), label (the stereotype name is displayed and appears inside guillemets), and none (the stereotype is not indicated).

4.1   Different Levels of Detail

Based on our experience in real-world cases, we have developed a set of design guidelines for using UML packages[8] in MD modeling. In UML, a package defines a *namespace*, so that two distinct elements contained in two distinct packages may have the same name. We summarize all the design guidelines in Table 1.
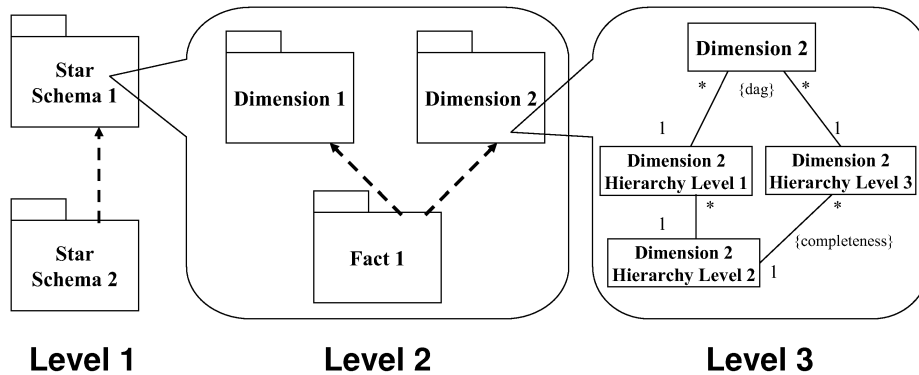

Figure 4: The three levels of a MD model explosion using packages

`Guideline 0a` is the foundation of the rest of the guidelines and summarizes our overall approach. This guideline closely resembles how data analyzers understand MD modeling. We have divided the design process into three levels. In Figure 4, we show a summary of our proposal by showing the main issues we define in each level and in Table 1 we indicate in which level each guideline is applied). The icons we use in levels one or two for packages are the *none* representation. The different levels show

---

[8]Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique.

how one package can be further exploded by defining their corresponding elements into the next level as we describe as follows:

Level 1 : Model definition. A package represents a star schema[9] of a conceptual MD model. A dependency between two packages at this level indicates that the star schemas share at least one dimension, allowing us to consider *conformed dimensions*.

Level 2 : Star schema definition. A package represents a fact or a dimension of a star schema. A dependency between two dimension packages at this level indicates that the packages share at least one level of a dimension hierarchy.

Level 3 : Dimension/fact definition. A package is exploded into a set of classes that represent the hierarchy levels defined in a dimension package, or the whole star schema in the case of the fact package.

The MD model is designed in a top-down fashion by further decomposing a package. We have limited our proposal to three levels because "deep hierarchies tend to be difficult to understand, since each level carries its own meanings" [8].

`Guidelines 2b` and `2c` make sure that cross-package dependencies result only in acyclic graphs[10] in order to keep things simple. Circular dependencies may be reduced by:

- Splitting one of the questionable packages into two smaller packages.

- Introducing a third intermediate package (try to factor the common elements out into a third package).

- Merging the questionable packages.

For example, in Figure 5 (a) the two STARPACKAGEs (stereotyped packages represented by means of icons) form a cycle that has been broken in Figure 5 (b) by the introduction of a third STARPACKAGE that contains the shared dimensions; this new package, that we call *utility package*, does not contain a FACTPACKAGE, just the definition of the common elements to both packages. In Figure 5 (c) we show an alternative solution: the two STARPACKAGEs have been merged into a single one called StarPackage1-2, eliminating the shared elements, and therefore, avoiding repeating already-defined elements.

4.1.1   Applying Package Design Guidelines

Figure 6 shows the first level of a MD model: on the left hand side, the packages are displayed with the common UML presentation and the corresponding stereotype icon is placed in the upper right corner of the package symbol; on the right hand side, the entire package symbol has been "collapsed" into the corresponding stereotype icon. Through the rest of the paper, we have adopted the second form of representing the stereotypes, because we consider it more expressive and symbolic, as well as it is also more understandable for the end users.

In the example shown in Figure 6, the first level is formed by three STARPACKAGEs that represent the different data marts that form the DW (`G.1`). A dashed arrow from one package to another one denotes a dependency between packages, i.e., the packages have some dimensions in common (`G.2a`). The direction of the dependency indicates that the common dimensions shared by the two packages were first defined in the package pointed to by the arrow (to start with, we have to choose a STARPACKAGE to define the dimensions, and then, the other STARPACKAGEs can use them with no need to define them again). If the common dimensions had been first defined in another package, the direction of the arrow would have been different. In any case, it is highly recommended to group together the definition of the common dimensions in order to reduce the number of dependencies (`G.2b`) and also to avoid circular dependencies (`G.2c`).

---

[9]Although we use the concept star schema, it does not imply any relational implementation of the DW. We prefer to use a well known concept instead of inventing a new term.

[10]Fowler states: "As a rule of thumb, it is a good idea to remove cycles in the dependency structure" [11].

N°	Level	Guideline
0a		At the end of the design process, the MD model will be divided into three levels: model definition, star schema definition, and dimension/fact definition
0b		Before starting the modeling, define facts and dimensions and remark the shared dimensions and dimensions that share some hierarchy levels
1	1	Draw a package for each star schema, i.e., for every fact considered
2a	1	Decide which star schemas will host the definition of the shared dimensions; according to this decision, draw the corresponding dependencies
2b	1	Group together the definition of the shared dimensions in order to minimize the number of dependencies
2c	1	Avoid cycles in the dependency structure
3	2	Draw a package for the fact (only one in a star package) and a package for each dimension of the star schema
4a	2	Draw a dependency from the fact package to each one of the dimension packages
4b	2	Never draw a dependency from a dimension package to a fact package
5	2	Do not define a dimension twice; if a dimension has been previously defined, import it
6	2	Draw a dependency between dimension packages in order to indicate that the dimensions share hierarchy levels
7	3	In a dimension package, draw a class for the dimension class (only one in a dimension package) and a class for every classification hierarchy level (the base classes)
8	3	In a fact package, draw a class for the fact class (only one in a fact package) and import the dimension classes with their corresponding hierarchy levels
9	3	In a dimension package, if a dependency from the current package has been defined at level 2, import the corresponding shared hierarchy levels
10	3	In a dimension package, when importing hierarchy levels from another package, it is not necessary to import all the levels

Table 1: Multidimensional modeling guidelines for using packages

StarPackage1

StarPackage1

StarPackage2

StarPackage2

StarPackage Aux

StarPackage1-2
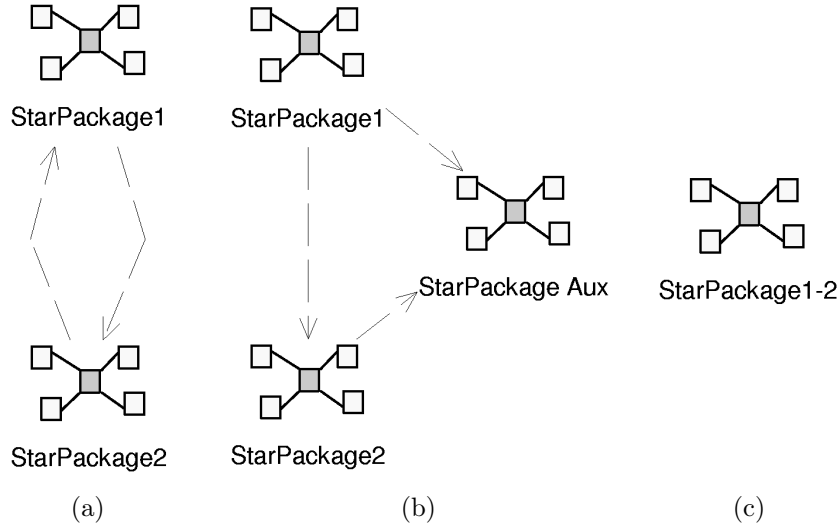
(a)　　　　　　　　　　　(b)　　　　　　　　　　　(c)

Figure 5: Model definition with and without cycles

At any level of our proposal, the DW designer can use UML notes to add more information, remark some characteristic, clarify some ambiguous situation, or describe some concept in end users' terms. For example, in Figure 6, we have used a UML note to remark the content of a package.

A package that represents a star schema is shown as a simple icon with names. The content of a package can be dynamically accessed by "zooming-in" to a detailed view. For example, Figure 7 shows the content of the package StarPackage1 (level 2). The package FactPackage1 is represented in the middle of Figure 7, while the different dimension packages are placed around the fact package (G.3). As seen in Figure 7, a dependency is drawn from the FACTPACKAGE to each one of the DI-MENSIONPACKAGES, because the FACTPACKAGE comprises the whole definition of the star schema, and therefore, uses the definitions of dimensions related to the fact (G.4a). At level 2, it is possible to create a dependency from a FACTPACKAGE to a DIMENSIONPACKAGE or between DIMENSIONPACK-AGEs, but we do not allow a dependency from a DIMENSIONPACKAGE to a FACTPACKAGE, since it is not semantically correct in our proposal (G.4b).

Figure 8 shows the content of the package StarPackage3 (level 2). The package FactPackage3 is placed in the middle of Figure 8 and the dimension packages are placed around the fact package in a star fashion. The package DimensionPackage1.1 has been previously defined in the StarPackage1 (Figure 7), and DimensionPackage2.1 has been previously defined in the StarPackage2 (not shown in this paper), so all of them are imported in this package (G.5). Our approach does not forbid to define another dimension with or without the same name or properties in different STARPACKAGEs. However, we highly recommend not to do it as we believe that this situation can lead us to a confusing or misleading diagram. Therefore, the name of the STARPACKAGEs where they have been previously defined appears below the package name (from StarPackage1 and from StarPackage2 respectively). In our proposal, it is possible to import packages defined in different STARPACKAGEs. On the other hand, DimensionPackage3.1 has been defined in the current package, therefore, it does not show a package name.

At level 2, a dependency between DIMENSIONPACKAGEs indicates that they share some hierarchy levels (G.6). For example, in Figure 8, a dependency between DimensionPackage3.1 and Dimension-Package1.1 is represented because there is a shared hierarchy.

The benefit of the UML importing mechanism is twofold. On one hand, the DW designer only needs to define the different MD elements once, and therefore, they can be used anywhere in the model. On the other hand, as the MD elements are defined only once, any possibility of duplication and ambiguity is removed.

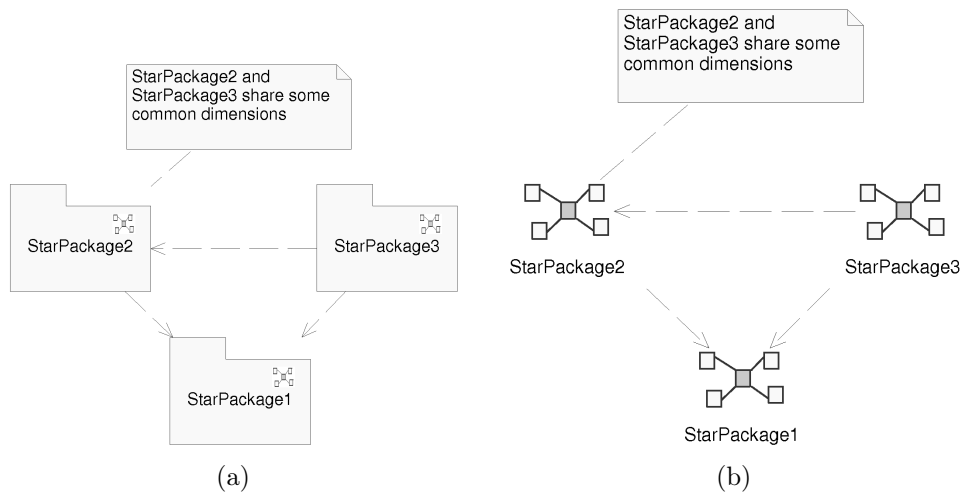The content of the DIMENSIONPACKAGE and FACTPACKAGE is represented at level 3. The dia-

StarPackage2 and StarPackage3 share some common dimensions

StarPackage2

StarPackage3

StarPackage1

(a)

StarPackage2 and StarPackage3 share some common dimensions

StarPackage2

StarPackage3

StarPackage1

(b)

Figure 6: Level 1: different representations

DimensionPackage1.1
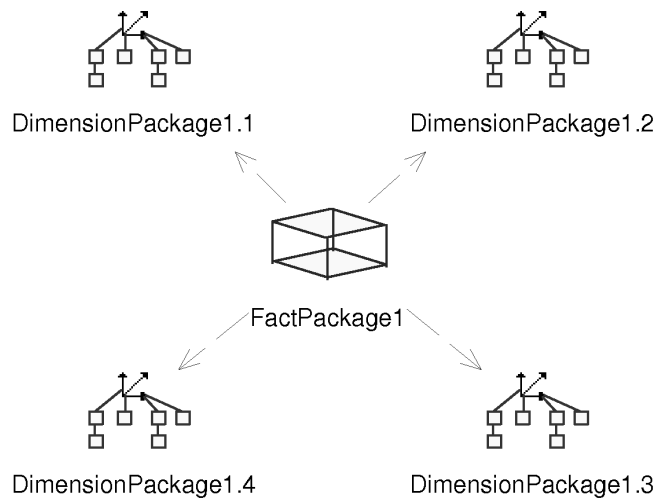
DimensionPackage1.2

FactPackage1

DimensionPackage1.4

DimensionPackage1.3

Figure 7: Level 2: content of StarPackage1
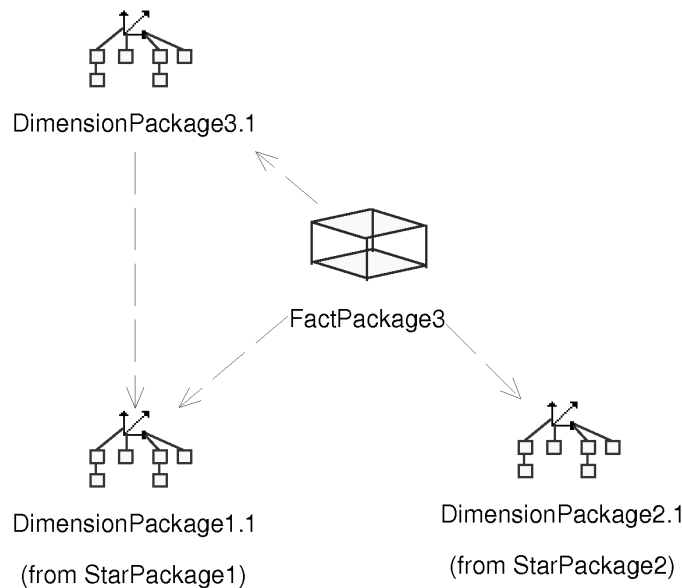
Figure 8: Level 2: content of StarPackage3

grams at this level are only comprised of classes and associations among them. For example, Figure 23 shows the content of the package Customer dimension (level 3), that contains the definition of the Dimension class (Customer) and the different hierarchy levels (Customer personal data, City, Region, and State) that are represented by Base classes (G.7). The hierarchy of a dimension defines how the different OLAP operations (roll-up, drill-down, etc.) can be applied [19].

4.1.2 Advantages of multi-fact schemas

Our approach allows DW designers to define a MD model that comprises multiple facts (multiple StarPackages) linked between them by shared dimensions. This feature, commonly known as *fact constellation*, provides the structure to allow the end user to traverse the schema to perform an analysis known as *drill-across*. Moreover, keeping common dimensions will facilitate the future implementation of the MD model, e.g., in the case of a DW, the loading and refreshment processes will be simpler.

4.2 Facts and Dimensions

Facts and dimensions are represented by Fact and Dimension classes, respectively. Then, Fact classes are specified as composed classes by means of aggregation relationships of n Dimension classes, represented by a hollow diamond attached to the end of the relationship next to the Fact class. The flexibility of the aggregation in the UML allows us to represent *many-to-many* relationships between Facts and particular Dimensions by indicating the 1..* cardinality at the end of the aggregation near the Dimension.

In our example shown in Figure 9, we can see how the Fact class Fact1 has a many-to-one relationship with Dimension1.1, Dimension1.2 and Dimension1.3, but a many-to-many relationship with Dimension1.4. This level may become very complex because the dimensions may be very complex and of a considerable size due to a high number of dimension levels. However, the DW designer only has to import them from the corresponding DimensionPackages.
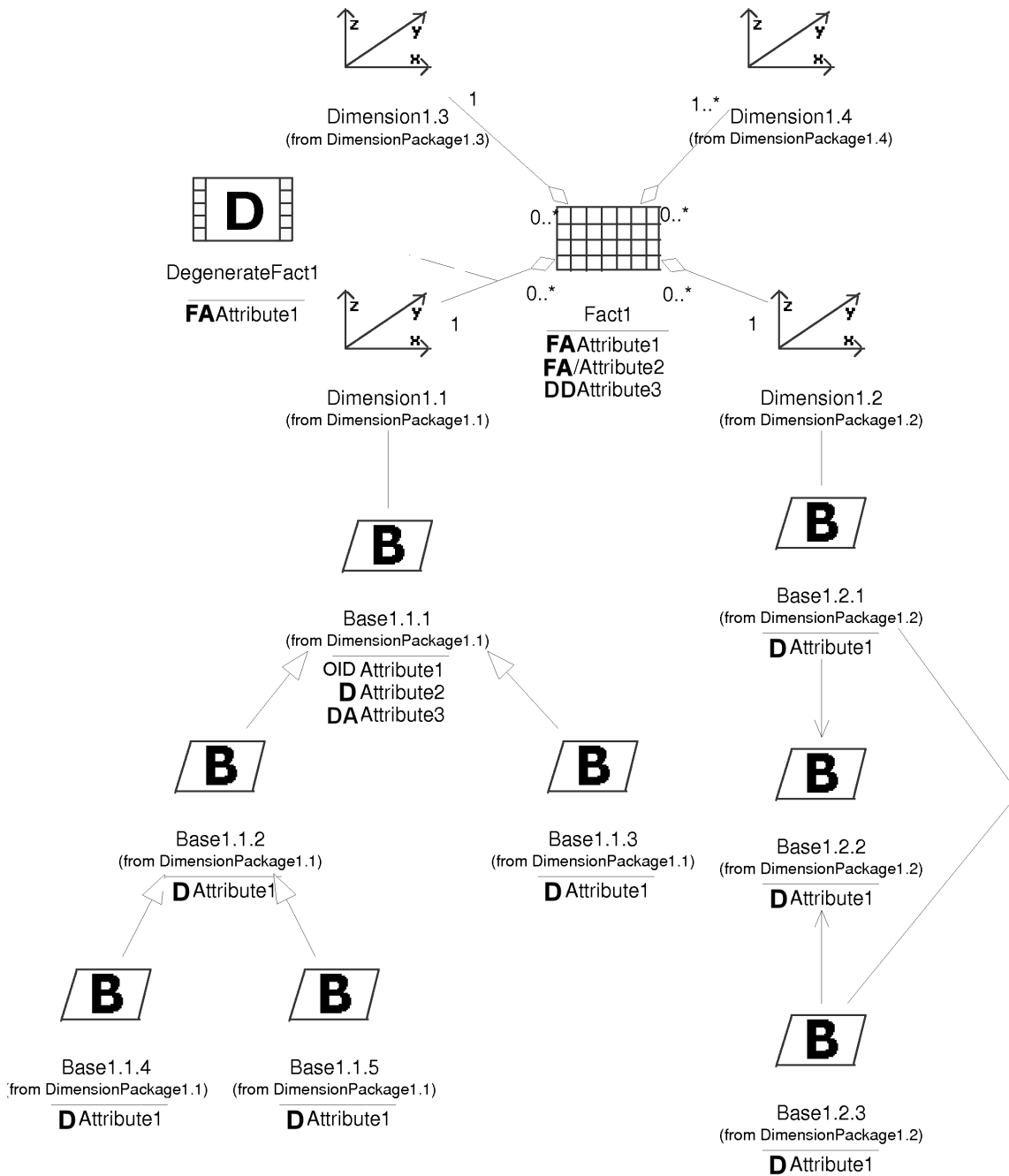
13

Dimension1.3
(from DimensionPackage1.3)

Dimension1.4
(from DimensionPackage1.4)

1

1..*

D

0..1

0..*

DegenerateFact1

FA Attribute1

z
y
x

z
y
x

0..*

0..*

Dimension1.1
(from DimensionPackage1.1)

1

Fact1

FA Attribute1
FA /Attribute2
DD Attribute3

1

Dimension1.2
(from DimensionPackage1.2)

B

B

Base1.1.1
(from DimensionPackage1.1)

OID Attribute1
D Attribute2
DA Attribute3

Base1.2.1
(from DimensionPackage1.2)

D Attribute1

B

B

B

Base1.1.2
(from DimensionPackage1.1)

D Attribute1

Base1.1.3
(from DimensionPackage1.1)

D Attribute1

Base1.2.2
(from DimensionPackage1.2)

D Attribute1

B

B

B

Base1.1.4
(from DimensionPackage1.1)

D Attribute1

Base1.1.5
(from DimensionPackage1.1)

D Attribute1

Base1.2.3
(from DimensionPackage1.2)

D Attribute1

Figure 9: Level 3: content of FactPackage1
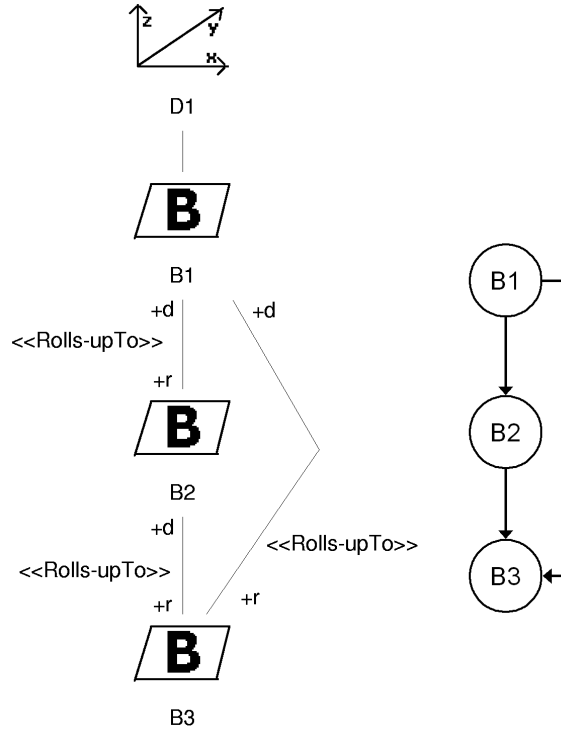
Figure 10: Classification hierarchy without cycles

4.3 Dimensions and Classification Hierarchy Levels

DIMENSION classes are composed of *classification hierarchy levels*; every classification hierarchy level is specified by a class called BASE class. An association (represented by a stereotype called ROLLS-UPTO) between BASE classes specifies the relationship between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the DIMENSION class. The DAG structure can represent both alternative path and multiple classification hierarchies.

Following Husemann's definitions [16], a DIMENSION contains a unique first hierarchy (or dimension) level called *terminal dimension level*. A roll-up path (or aggregation path in [16]) is a subsequence of dimension levels, which starts in a terminal dimension level (lower detail level) and ends in an implicit dimension level (not graphically represented) that represents all the dimension levels.

We use roles to represent the way the two classes see each other in a ROLLS-UPTO association: role R represents the direction in which the hierarchy rolls-up, whereas role D represents the direction in which the hierarchy drills-down. Moreover, we use roles to detect and avoid cycles in a classification hierarchy, and therefore, help us to achieve the DAG condition. For example, on the left hand side of Figure 10, a classification hierarchy composed of three BASE classes is represented. On the right hand side of Figure 10, a graph that symbolizes the classification hierarchy is shown and the direction of the arrows is based on the roles of the ROLLS-UPTO associations: from role D to role R (in the direction of rolling-up). As we can see in this figure, this classification hierarchy does not contain any cycle. However, the classification hierarchy shown in Figure 11 presents a cycle (BASE classes B2, B3, and B4), and therefore, this classification hierarchy is absolutely incorrect in our model.

In UML, an arrow may be attached to the end of an association to indicate that navigation is supported toward the class attached to the arrow. In our proposal, the navigation is always supported toward both ends of an association (it is always possible to roll-up or drill-down on both directions), but the DW designer can use the UML navigability to deliberately represent a default roll-up or
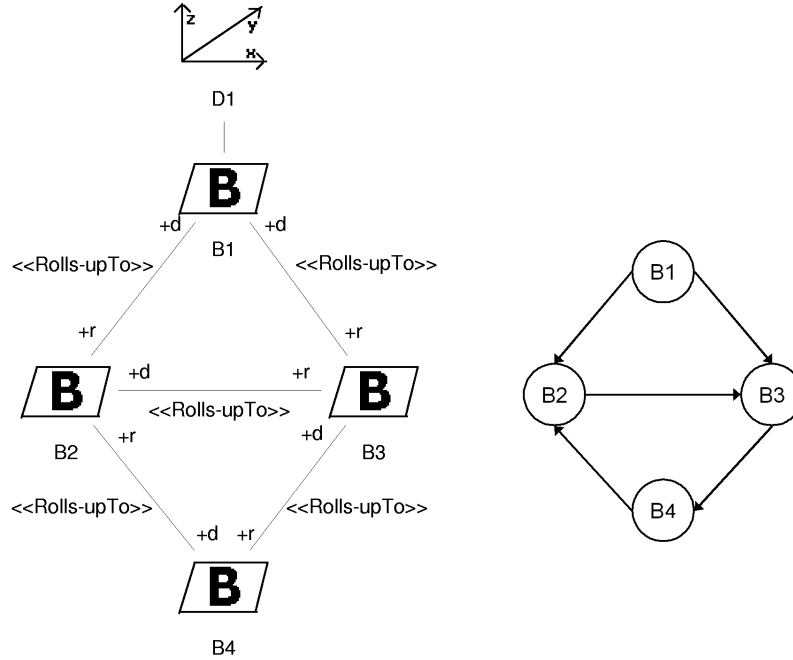
15

Figure 11: Classification hierarchy with one cycle

drill-down path when a BASE class participates in multiple classification hierarchies[11]. However, only one default roll-up and one default drill-down path can start from a BASE class. For example, in Figure 12 (a) we have represented a classification hierarchy that is incorrect because two default roll-up paths start from B1 (B1 rolls-up to B2 and B1 rolls-up to B3), and two default drill-down paths start from B4 (B4 drills-down to B2 and B4 drills-down to B3). The default roll-up and drill-down paths that are in conflict have been remarked with a dashed circle. This same classification hierarchy is correctly represented in Figure 12 (b); note that it would have also been possible to define other default paths, such as B1 rolls-up to B3 and B4 drills-down to B3.

The multiplicity *1* and *1..** defined in the role R of a classification hierarchy level addresses the concepts of *strictness* and *non-strictness*, respectively. Strictness means that an object at a hierarchy's lower level belongs to only one higher-level object (e.g., as one month can be related to more than one season, the relationship between them is non-strict). In a DW, it is very important to identify and define non-strict hierarchies, because if they are not correctly treated, some problems such as double-counting can appear when aggregations are calculated in further design steps.

Moreover, defining an association as COMPLETENESS addresses the completeness of a classification hierarchy. By completeness we mean that all members belong to one higher-class object and that object consists of those members only; for example, all the recorded seasons form a year, and all the seasons that form the year have been recorded. Our approach assumes all classification hierarchies are non-complete by default.

In a DW, time is the dominant dimension. Many forms of analysis involve either trends or inter-period comparisons. Inmon [17] defines "A data warehouse is a subject-oriented, integrated, **time-variant**, nonvolatile collection of data in support of management's decisions", and Kimball [19] says that "The **time dimension** is the one dimension virtually guaranteed to be present in every data warehouse, because virtually every data warehouse is a time series". Due to this important fact, in our proposal a DIMENSION class includes a boolean tagged value called ISTIME that indicates whether

---

[11]Please, note that the navigability is an optional feature of our approach and it is not mandatory to always specify a roll-up or drill-down default path. Moreover, it is not necessary to draw the navigability when there is only one roll-up or drill-down path.
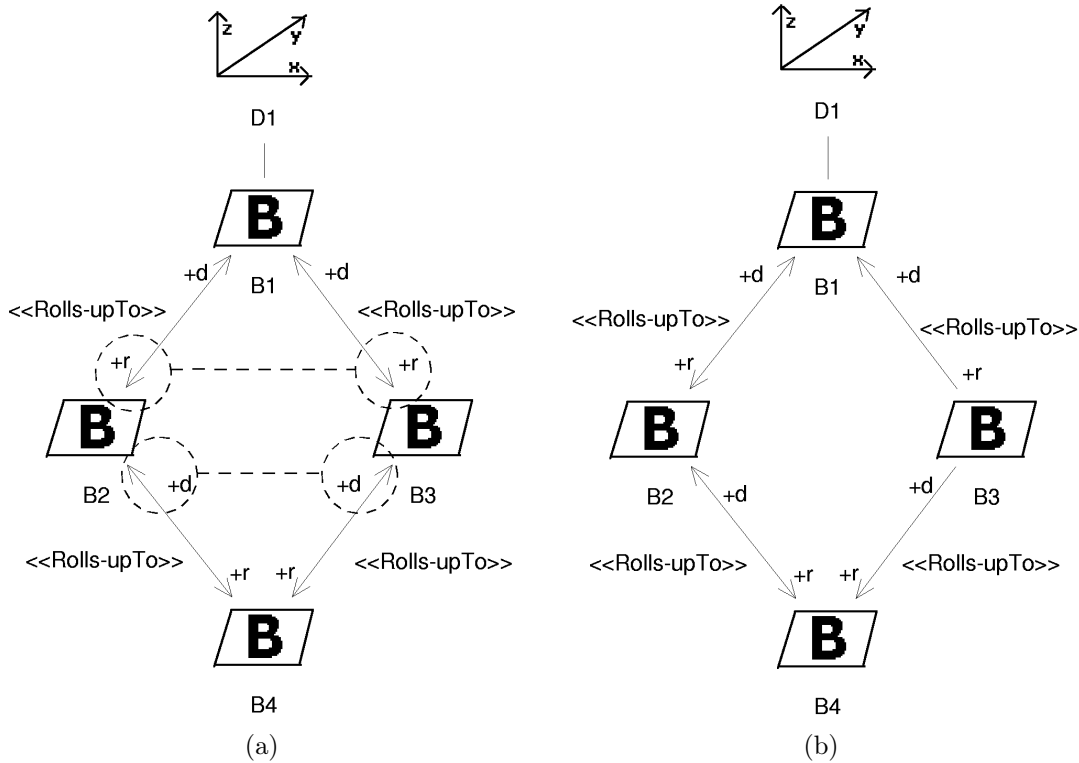
Figure 12: Classification hierarchy with wrong and right navigability

it is a time dimension or not[12].

4.4 Categorization of Dimensions

The *categorization of dimensions*, used to model additional features for a class's subtypes, is represented by means of generalization-specialization relationships in our approach. However, only the parent of a categorization can belong to both a classification and generalization-specialization hierarchy at the same time. Moreover, multiple inheritance is not allowed in our approach. In UML, generalization is shown as a solid-line from the subclass to the superclass, with a small hollow triangle at the end of the line near the superclass. In our approach, the categorization of dimensions does not require any additional UML extension.

For example, Figure 9 shows an example of categorization for the Dimension1.1. This categorization presents two levels: (a) Base1.1.2 and Base1.1.3 to Base1.1.1, and (b) Base1.1.4 and Base1.1.5 to Base1.1.2. The different levels of a categorization are represented by Base classes that are shown with the corresponding icon (B).

4.5 Attributes

Only Fact, Base, and DegenerateFact (see Section 4.7) classes can have attributes. Dimension classes do not contain attributes, because they represent the concept of dimension and they are used as "anchorage points": the information about a dimension is represented in the corresponding hierarchy levels (Base classes).

Fact classes consist of two kinds of attributes: FactAttributes, which represent measures (the transactions or values being analyzed), and DegenerateDimensions (see Section 4.6).

---

[12]This will allow us an automatically generation of particular time structures in a target commercial OLAP tool.

On the other hand, BASE classes consist of three kinds of attributes: OIDs, DESCRIPTORs, and/or DIMENSIONATTRIBUTEs. Every BASE class can have one OID attribute (an identifying attribute) and must have one DESCRIPTOR attribute[13]. These attributes are necessary for an automatic exportation process into commercial OLAP tools, as these tools store this information in their metadata (if the OID attribute is not provided, then it will be automatically created in the exportation process). A DIMENSIONATTRIBUTE provides descriptive information about dimension instances. A DIMENSION-ATTRIBUTE can be optional: it does not need to be specified for each element of the corresponding level and therefore may contain *null* values. As a DIMENSIONATTRIBUTE can be used to delimit the resulting set of a query, it is important to know if an attribute is optional (considered in our approach as *tagged values*, see Section A.3.4), because then the results may be incomplete [16].

FACTATTRIBUTE, DESCRIPTOR, and DIMENSIONATTRIBUTE can also be derived attributes. This situation is indicated by placing / before the corresponding name, and the derivation rule is defined as a tagged value called DERIVATIONRULE of the corresponding stereotype.

For example, Figure 9 shows FACT and BASE classes with different types of attributes. The attributes are decorated with an icon that indicates the stereotype that applies to. In the FACT class, attributes are defined as FACTATTRIBUTE (FA) and DEGENERATEDIMENSION (DD). On the other hand, in BASE classes attributes are defined as OID, DESCRIPTOR (D), and DIMENSIONATTRIBUTE (DA).

4.6 Degenerate Dimensions

Our approach also allows the DW designer to define degenerate dimensions in the FACT class, by using the stereotype DEGENERATEDIMENSION for an attribute. A *degenerate dimension* is a DIMENSION that is stored as an attribute of the FACT class, but we do not explicitly represent it as a dimension in our diagram. *Degenerated dimensions* are useful when attempting to associate the facts in the DW with the original data sources [14, 19].

In Figure 9, Attribute3 is a DEGENERATEDIMENSION of Fact1. A DEGENERATEDIMENSION is represented by an icon with the letters DD.

4.7 Degenerate Facts

In [14], the *degenerate fact* concept is defined as a measure recorded in the intersection table of a *many-to-many* relationship between the fact table and a dimension table. In our approach, we represent a DEGENERATEFACT as a UML association class attached to a many-to-many aggregation relationship between a FACT class and a DIMENSION class[14]. This DEGENERATEFACT class can contain FACTATTRIBUTEs and DEGENERATEDIMENSIONs.

For example, in Figure 9, we show DegenerateFact1 attached to the aggregation relationship between Fact1 and Dimension1.1. Although DegenerateFact1 resembles an independent element, it is actually part of the aggregation relationship.

4.8 Additivity

We consider all measures as additive by default, i.e. the SUM operator can be applied to aggregate their measure values along all dimensions. Non-additivity and semi-additivity are considered by defining constraints on measures between brackets and placing them somewhere around the fact class. These constraints are represented in a property tag of the UML notation for clarity reasons, although they have formal underlying formulae and contain the allowed operators, if any, along the dimension that the measure is not additive. However, in large MD models, the readability can be reduced due to a great amount of additivity rules shown in a diagram. In these cases, we use *summarizability appendices*, as described in [16].

---

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

[14]Actually, an association class is an association that also has class properties (or a class that has association properties). Even though it is drawn as an association or a class, it is really just a single model element containing attributes.
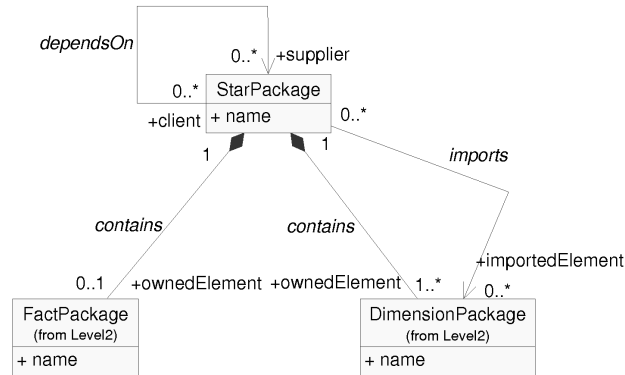
Figure 13: Metamodel

Figure 14: Metamodel: level 1

4.9 Merged Level 2 and 3

In some occasions, the DW designer needs to have a general overview of all facts, dimensions, and dependencies that a DW comprises. In our approach, this can be achieved if all the star schema definitions (level 2) are merged into one diagram. In a CASE tool that supports our proposal, this diagram can be automatically built; the DW designer should be allowed to make changes in this aggregated view and the changes should be propagated through related levels 2 and 3.

The same can be done at level 3, but the resulting diagram can be extraordinary complex in a big real DW with tens of dimensions and hundreds of hierarchy levels.

4.10 Metamodel

In this section, we present the metamodel of our OO conceptual MD approach using a UML class diagram. In order to simplify this diagram, we have divided this diagram into tree packages, as it is shown in Figure 13.

In Figure 14 we show the content of the package Level1. This package specifies the modeling elements that can be applied in the level 1 of our approach. In this level, only the STARPACKAGE model element is allowed. We use the navigability of an association to denote the direction of a dependency or an importation. For example, a STARPACKAGE may import DIMENSIONPACKAGEs from another STARPACKAGE. We also show the modeling elements that a STARPACKAGE can contain: FACTPACKAGE and DIMENSIONPACKAGE.

In Figure 15 we show the content of the package Level2. In this level, the modeling elements that can be used are FACTPACKAGE and DIMENSIONPACKAGE. A FACTPACKAGE may contain only one FACT and various DEGENERATEFACTS, whereas a DIMENSIONPACKAGE may contain only one DIMENSION and various BASEs.

Finally, in Figure 16 we show the content of the package Level3. This diagram represents the main MD properties of our modeling approach. In this way, dimensions and facts are represented using the classes DIMENSION and FACT, respectively.
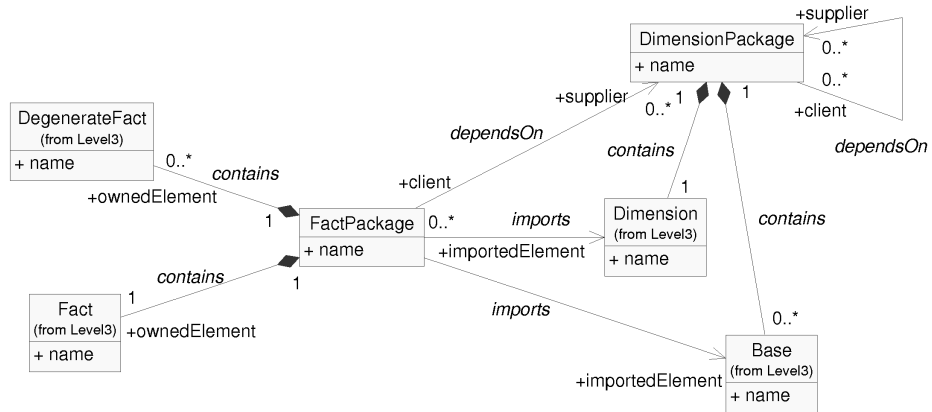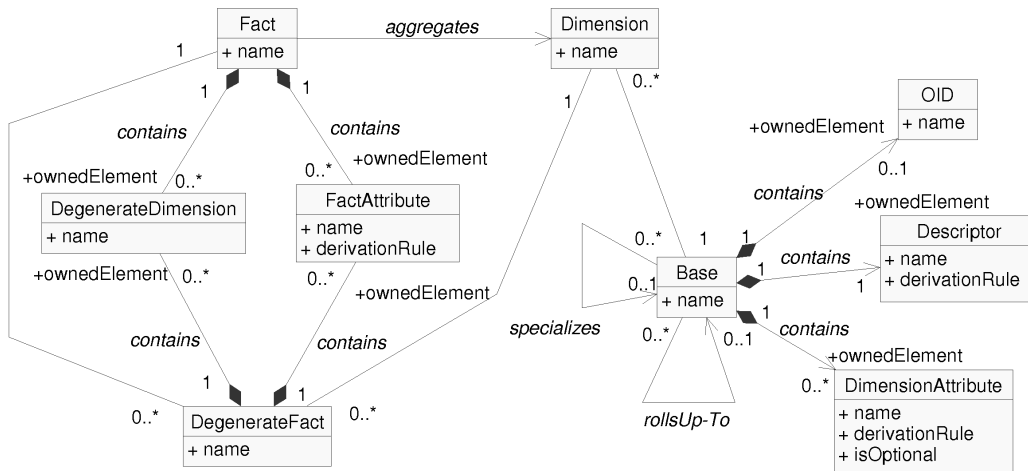
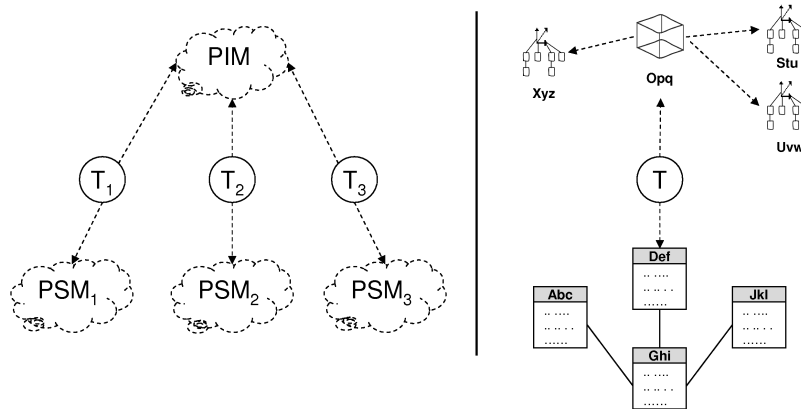Figure 15: Metamodel: level 2

Figure 16: Metamodel: level 3

Figure 17: Transformation of a Multidimensional Model

5 Implementation of Multidimensional Models

Model Driven Architecture (MDA) [30] is an Object Management Group (OMG) standard that addresses the complete life cycle of designing, deploying, integrating, and managing applications. MDA separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform, i.e. a Platform Independent Model (PIM) can be transformed into multiple Platform Specific Models (PSM) in order to execute on a concrete platform (see left hand side of Figure 17).

MOF (Meta-Object Facility) 2.0 QVT (Query, View, Transformation) is an under-developing standard for expressing model transformations, which can define transformation rules between two MOF-compliant models. In response to the Request for Proposal (RFP) of QVT, different transformation approaches have been proposed [9]. One of the most remarkable approaches is QVT-Partners [33].

QVT-Partners proposes a possibly extended version of OCL 2.0 as the query language and provides a standard language called MTL[15] (Model Transformation Language) for relations and mappings. In QVT-Partners, complex transformations can be built by composing simpler transformations using composition functions. Moreover, QVT-Partners suggests a sequence of steps that lead to an executable transformation that can be executed by means of a model transformation engine (e.g. Inria MTL Engine [18]).

Model transformation is the process of converting one model to another model. In [12], model transformations are categorized along vertical (a source model is transformed into a target model at a different level of abstraction) and horizontal (a source model is transformed into a target model that is at the same level of abstraction) dimensions.

We have aligned our MD proposal with the MDA approach; thus, as presented through the paper, we accomplish the conceptual modeling of a DW without considering any aspect of the implementation in a concrete target platform, thereby providing a PIM. We have developed an algorithm that, from the MD models accomplished by using our UML profile, generates the corresponding implementation in different platforms (relational and object-relational) through a vertical transformation, thereby allowing different PSM. In this section, we present the transformation process from an MD model to a relational one; on the right hand side of Figure 17, we show a high-level view of a transformation process from an MD model to the relational model, in which we generate the specific platform structures according to the modeling elements.

For example, the next code represents the QVT implementation of the mapping for the FACT class into a table. The body of a mapping contains an object expression that creates an object (method `new`) and produces the output. In the body of a mapping, OCL is used to select and filter the model elements.

---

[15]The syntax of this language resembles C, C++ and Java language family.

21

```
mapping FactToTable {
  domain {
    (Fact)[name = fn, attibutes = atts, associations = ass]
  }
  body {
    ta = new Table()
    ta.name = fn
    ta.columns = atts->iterate(a cols = {} | cols + FactAttributeToColumn(a))
    ta.keys = atts->forAll(a keys = {} | keys + DDToKey(a))
    ta.foreignKeys = ass->forAll(a fkeys = {} | fkeys + AggregationToForeignKey(a))
  }
}

mapping FactAttributeToColumn {
  domain {
    (FactAttribute)[name = fn, type = ty]
  }
  body {
    co = new Column()
    co.name = fn
    co.type = ty
  }
}

mapping DDToKey {
  domain {
    (DegenerateDimension)[name = fn, type = ty]
  }
  body {
    ke = new Key()
    ke.name = fn
    ke.type = ty
  }
}

mapping AggregationToForeignKey {
  domain {
    (Association)[name = fn, source = aggS , destination = aggD ]
  }
  body {
    fk = new ForeignKey()
    fk.name = fn
    // ForeignKey is autoincrement
    fk.type = 'auto'
  }
}
```

6   A Case Study

Throughout this section, we will use a running example to illustrate the use of our OO MD approach. We use the same example presented in Section 2 and inspired by a case study from [14].

6.1   Requirements

During this step, what the end users expect to do with the data warehouse is captured: the end users should specify the most interesting measures and aggregations, the analysis dimensions, the queries used to generate periodical reports, the update frequency of the data, etc. We model the requirements with use cases. The UML provides the use case diagram for visual modeling of uses cases. Nevertheless, there is no UML standard for a use case specification. However, we follow the common template defined in [4], which specifies for every use case a name, a unique identifier, the actor involved in the use case, the system state before the use can begin (*preconditions*), the actual steps of the use case (*flow of events*), and the system state when the use case is over (*postconditions*).

In this section, we model a couple of requirements for our case study. From interviews to the end users, we have detected two requirements related to sales managers, so the resulting schema will be able to achieve these requirements:

- A sales manager needs to analyze the sales rates per month regarding the city where the customer lives.
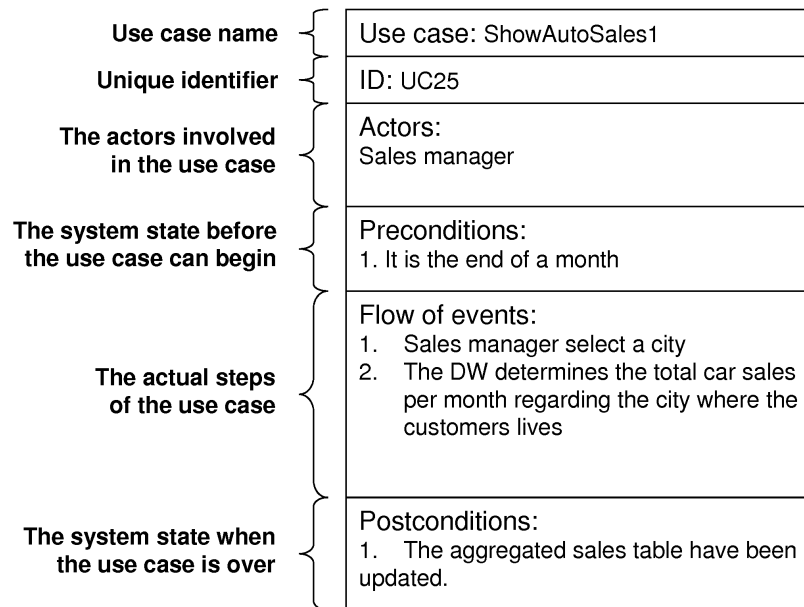
Figure 18: UML use case template

- A sales manager needs to know the different car models with higher sales rates in cities with more than 100.000 inhabitants.

For example, in Figure 18, we show the template we apply to describe a use case. Each use case has a name, a unique identifier, a list of actors involved in the use case, and a specification. The specification consists of preconditions (conditions that must be true before the use case), the flow of events (the steps in the use case, captured as prose or pseudocode), and postconditions (conditions that must be true at the end of the use case). In the example shown in Figure 18, the use case is about sales managers making a query about the quarterly sales of the products in the computer category.

On the other hand, in Figure 19 we describe the second use case. The preconditions state the conditions that must be true before the sales managers can perform the analysis: cities with more than 100.000 inhabitants and autos belonging to the car type.

6.2 Levels 1 and 2: Package Design

The DW of our running example consists of three data marts: automobile sales, part sales, and service works. Figure 20 shows the first level of the model. The first level is formed by three STARPACKAGEs that represent the different data marts that form the DW (G.1: Draw a package for each star schema, i.e., for every fact considered). We have grouped the definition of the common dimensions in Auto-sales schema in order to avoid some design problems (G.2b: Group together the definition of the shared dimensions in order to minimize the number of dependencies and G.2c: Avoid cycles in the dependency structure). Moreover, in order to clarify the model, we have used three UML notes to remark the content of each package in Figure 20.

Regarding level 2, Figure 21 shows the content of the package Auto-sales schema (level 2). The FACTPACKAGE Auto-sales fact is represented in the middle of Figure 21, while the DIMEN-SIONPACKAGEs are placed around the FACTPACKAGE (G.3: Draw a package for the fact and a package for each dimension of the star schema). As seen in Figure 21, a dependency is drawn from the FACTPACKAGE to each one of the DIMENSIONPACKAGEs, because the FACTPACK-AGE comprises the whole definition of the star schema, and therefore, uses the definitions of di-

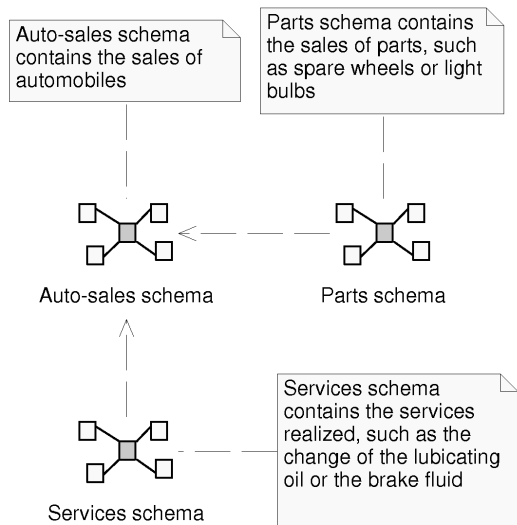| |
|---|
| Use case: ShowAutoSales2 |
| ID: UC26 |
| Actors:
Sales manager |
| Preconditions:
1. City must have more than 100.000 inhabitants
2. Auto is a car |
| Flow of events:
1. Sales manager select a city
2. The DW determines the different car models with higher sale rates in cities |
| Postconditions:
1. The aggregated sales table have been updated. |

Figure 19: Example of use case

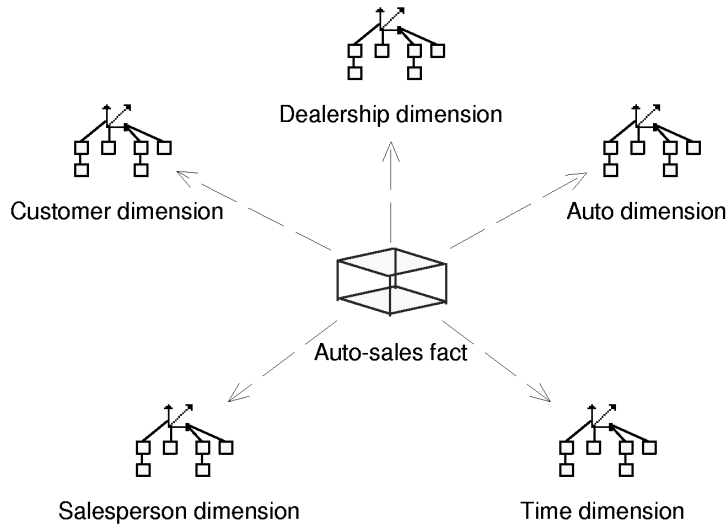Figure 20: Level 1: different star schemas of the running example

Figure 21: Level 2: Auto-sales schema

mensions related to the fact (G.4a: Draw a dependency form the fact package to each one of the dimension packages).

Figure 22 shows the content of the package Services schema (level 2). As in the Auto-sales schema, the FACTPACKAGE is placed in the middle of Figure 22 and the DIMENSIONPACKAGEs are placed around the FACTPACKAGE in a star fashion. The three DIMENSIONPACKAGEs (Customer dimension, Dealership dimension, and Time dimension) have been previously defined in the Auto-sales schema (Figure 21), and Parts dimension has been previously defined in the Parts schema (not shown in this paper), so all of them are imported in this package (G.5: Do not define a dimension twice; if a dimension has been previously defined, import it). When dimensions are imported, the name of the STARPACKAGEs where they have been previously defined appears below the package name (from Auto-sales schema and from Parts schema respectively). On the other hand, since Mechanic dimension and Service dimension have been defined in the current package, they do not show a package name.

Moreover, at level 2, the dependencies between DIMENSIONPACKAGEs represent shared hierarchy levels.   For example, a dependency between Mechanic dimension and Customer dimension is represented because there is a shared hierarchy[16] between these two DIMENSIONPACKAGEs (G.6: Draw a dependency between dimension packages in order to indicate that the dimensions share hierarchy levels). The shared hierarchy levels (City, Region, and State) can be observed in Figure 23 and Figure 24.

6.3   Level 3: Dimension Classes

The content of the DIMENSIONPACKAGE and FACTPACKAGE is represented at level 3. The diagrams at this level are only comprised of classes and associations among them. Figure 23 shows the content of the package Customer dimension (level 3), that contains the definition of the DIMENSION class (Customer) and the different hierarchy levels (Customer personal data, City, Region, and State) that are represented by BASE classes (G.7: In a dimension package, draw a class for the dimension class and a class for every classification hierarchy level). FullName attribute of Customer personal data is derived, because it is obtained by joining Name and Surname attributes (the derivation rule is not shown in order to avoid a cluttered diagram).

---

[16]We have decided to share a hierarchy for both dimensions to obtain a clearer design, although the designer may have decided not to do it if such sharing is not totally feasible.
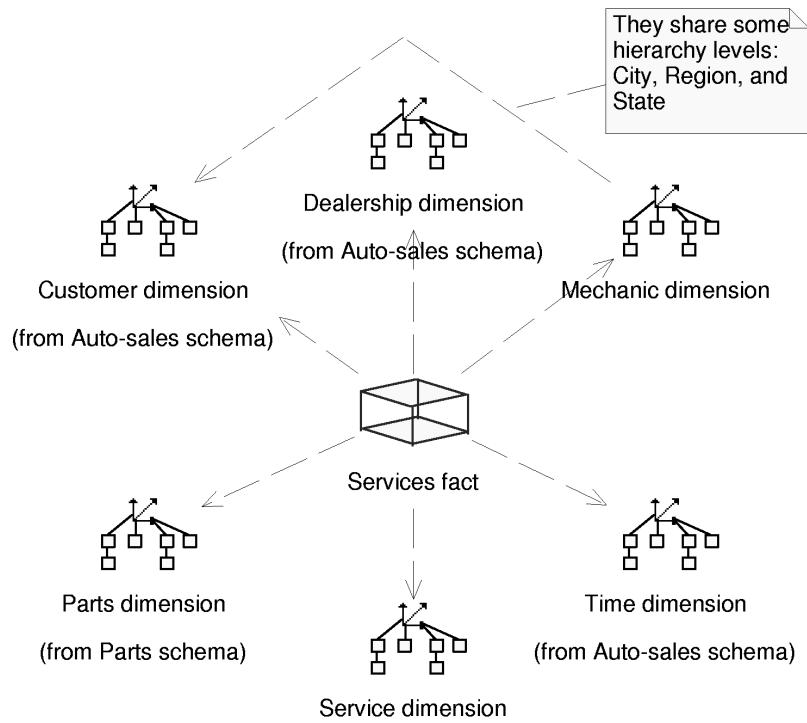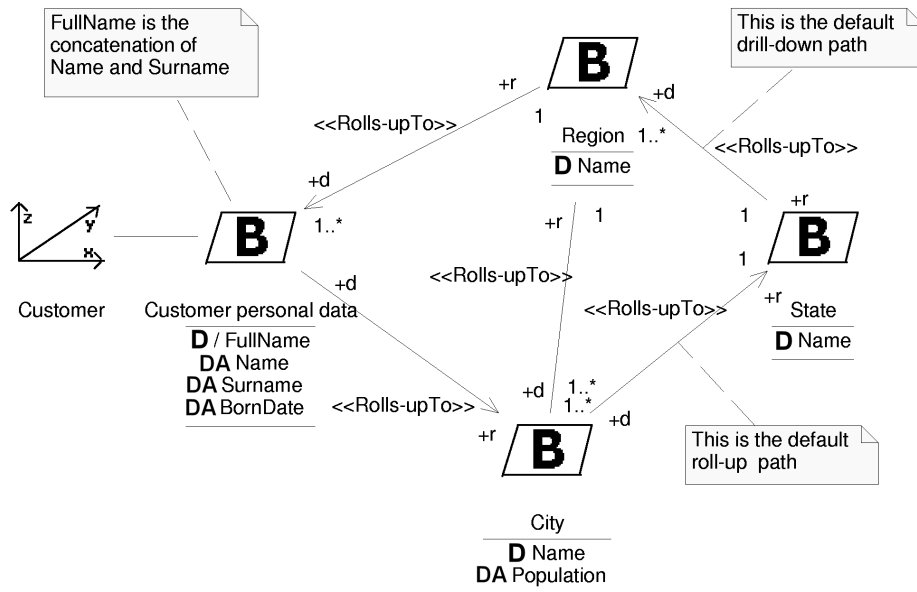
Figure 22: Level 2: Services schema

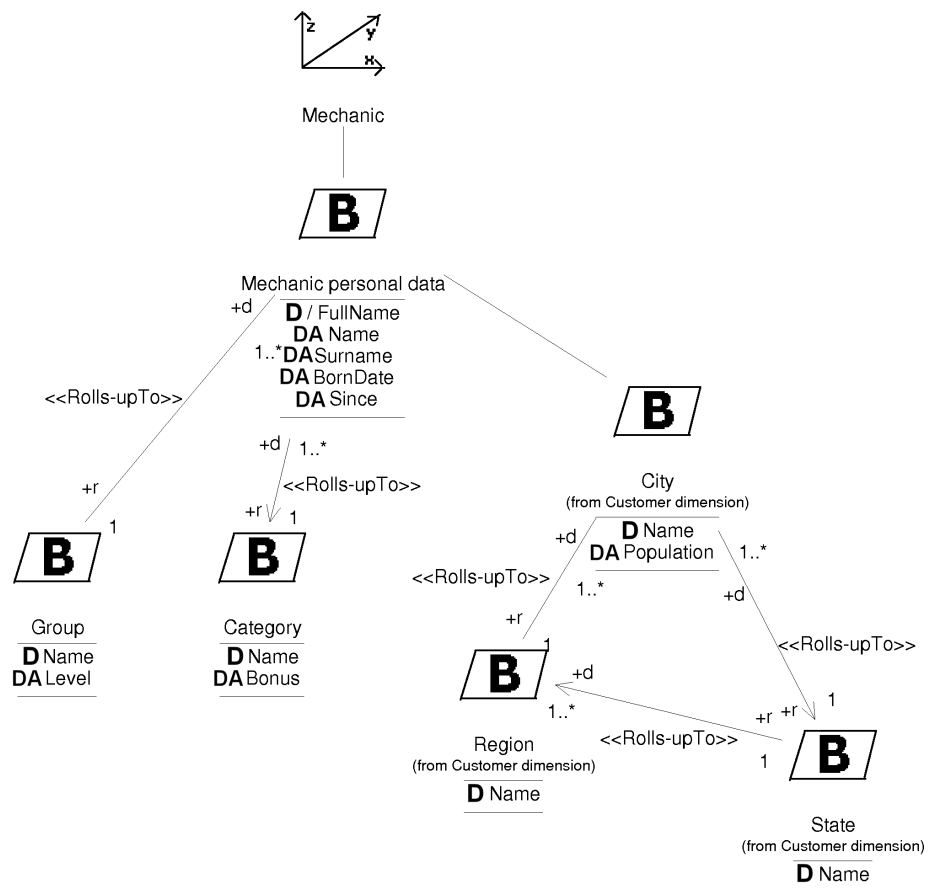Figure 23: Level 3: Customer dimension

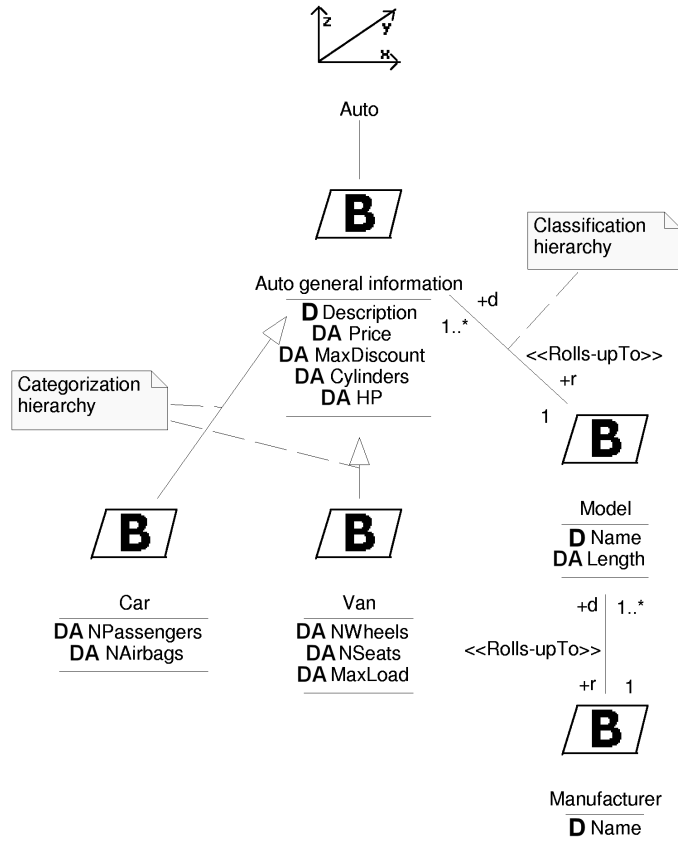Figure 24: Level 3: Mechanic dimension

Figure 25: Level 3: Auto dimension

As previously commented, Mechanic dimension and Customer dimension share some hierarchy levels, and therefore, there is a dependency between them (see Figure 22). Figure 24 shows the content of Mechanic dimension: this dimension contains six hierarchy levels, but three of them (City, Region and State) have been imported from another dimension.

An example of categorization for the Auto dimension is shown in Figure 25: Car and Van belong to a generalization-specialization relationship rooted in Auto general information; we have created this categorization because Car and Van contain different attributes.

Finally, we can see the use of the navigability (default path) and roles in Figure 23, 24, and 25.

6.4 Level 3: Fact Class

Regarding FACTPACKAGE, Figure 26 shows the content of the package Auto-sales fact (level 3). In this package, the whole star schema is displayed: the FACT class is defined with the corresponding measures: Commission, Quantity, Price, and Total. Total is derived measure (the derivation rules are not shown in order to avoid a cluttered diagram). Moreover, ContractN performs much the same function as a dimension but it is stored in the FACT class, therefore, ContractN is a DEGENERATEDIMENSION of Auto-sales that represents the identification number of the sale contract.

The FACT class Auto-sales has a many-to-one relationship with Auto, Dealership, Time, and Customer dimensions, but a many-to-many relationship with the Salesperson dimension. Moreover, the DIMENSION classes with their corresponding hierarchy levels are imported (G.8: In a fact package, draw a class for the fact class and import the dimension classes with their corresponding hierarchy levels). This level may become very

complex because the dimensions may be very complex and of a considerable size due to a high number of dimension levels. However, the DW designer only has to import them from the corresponding DimensionPackages. In Figure 26, we have hidden a part of the attributes and methods of some Base classes for the sake of simplicity.

On the other hand, in Figure 26, SP commision is a DegenerateFact attached to the aggregation relationship between Auto-sales fact and Salesperson dimension. This DegenerateFact is the commission percentage that a salesperson received for a particular sale. The relationship between Auto-sales and Salesperson is many-to-many because different salespersons can take part in the same sale (they share the total commission), and a salesperson can also take part in different sales.

Regarding the additivity of the measures of the Fact class, in Figure 26, the additivity rule {Quantity Is Not SUM Along Salesperson} states that the attribute Quantity in the Auto-sales class cannot be aggregated along the Salesperson dimension by using the SUM operator. However, the AVG, MIN and MAX aggregation operators can still be applied to aggregate this attribute along the Salesperson dimension.

Finally, in Figure 27 we show the merged level 2 of our running example with the three FactPackages and the different DimensionPackages; for each one of them, the legend (from ...) indicates in which StarPackage it has been defined. Moreover, the dependencies show where each DimensionPackage is used.

6.5 CASE Tool Support

All the MD models presented in this case study have been designed by using Rational Rose 2003. Instead of creating our own CASE (*Computer-Aided Software Engineering*) tool, we have chosen to extend a well known CASE tool available in the market, such as Rational Rose. In this way, we believe that our contribution can reach a greater number of people.

Rational Rose is one of the most well known visual modeling tools. Rational Rose is extensible by means of add-ins, which allows to package customizations and automation of several Rational Rose features through the Rose Extensibility Interface (REI) [34] into one component. An add-in is a collection of some combination of the following: main menu items, shortcut menu items, custom specifications, properties (UML tagged values), data types, UML stereotypes, online help, context-sensitive help, and event handling.

We have developed an add-in, which allows us to use our MD modeling approach in Rational Rose. Therefore, we can use this tool to easily accomplish MD conceptual models. Our add-in customizes the following elements:

- Stereotypes: We have defined the stereotypes by means of a stereotype configuration file.

- Properties: We have defined the tagged values by means of a property configuration file.

- Menu item: We have added the new menu item MD Validate in the menu Tools by means of a menu configuration file. This menu item runs a Rose script that validates a MD model: our script checks all the constraints that our UML profile defines.

In Figure 28, we can see a screenshot from Rational Rose that shows the definition of Mechanic dimension (see Figure 24) from the running example used in Section 4. Some comments have been added to the screenshot in order to remark some important elements: the hierarchy structure of our proposal (Level 1, Level 2, and Level 3), the new buttons added to Rational Rose, the stereotyped attributes, and the different ways of displaying a stereotype (Icon, Decoration, and Label). In this screenshot, the content of Mechanic dimension from Services schema is shown; this dimension shares some hierarchy levels (City, Region, and State) with Customer dimension, the place where the shared hierarchy levels have been defined firstly (see Figure 23), and because of this, they are imported into this dimension.

Auto
(from Auto dimension)

Dealership
(from Dealership dimension)

Time
(from Time dimension)

{Quantity Is Not SUM Along Salesperson}

1

1

1

0..*   0..*

0..*

Salesperson
(from Salesperson dimension)

B

1..*

0..*

0..*

1

Customer
(from Customer dimension)

B

D

SP commission

FA Commission

Auto-sales

FA Commission
FA Quatity
FA Price
FA / Total
DD ContractN

SP personal data
(from Salesperson dimension)

D / FullName
DA Name
DA Surname
DA Borndate

+d   +d

<<Rolls-upTo>>   1..*

1..*   <<Rolls-upTo>>

+r   1

1   +r

B

B

Position
(from Salesperson dimension)

D Name
DA Description

Group
(from Salesperson dimension)

D Name
DA Description

Customer personal data
(from Customer dimension)

D / FullName
DA Name
DA Surname
DA BornDate

+d

1..*

<<Rolls-upTo>>

<<Rolls-upTo>>

+r

+r   1

B

<<Rolls-upTo>>
+d   +r

1..*   1

B

City
(from Customer dimension)

D Name
DA Population

+d
1..*

Region
(from Customer dimension)

D Name

+d
1..*

<<Rolls-upTo>>
+r

1   1

<<Rolls-upTo>>
+r

B

State
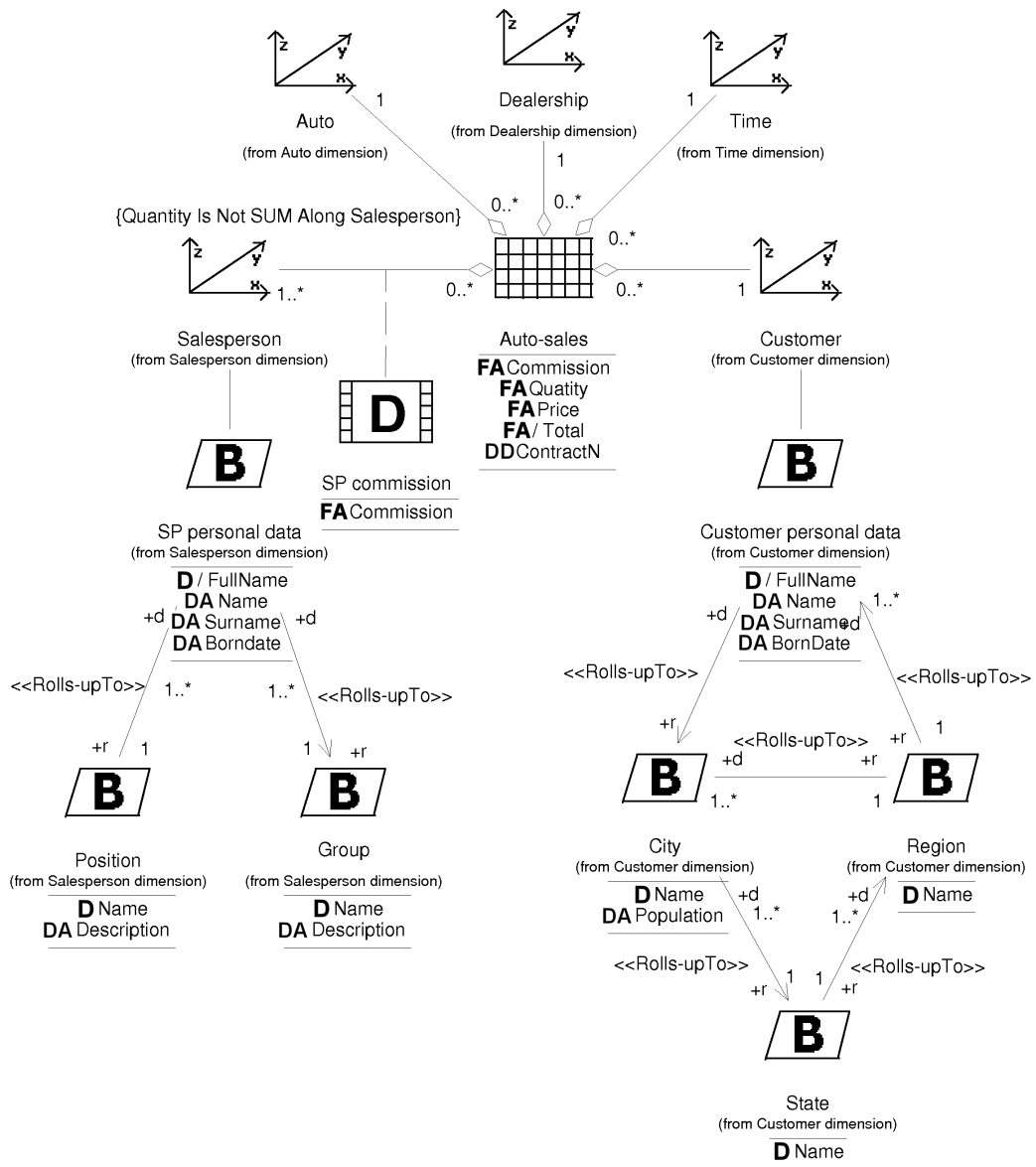(from Customer dimension)

D Name
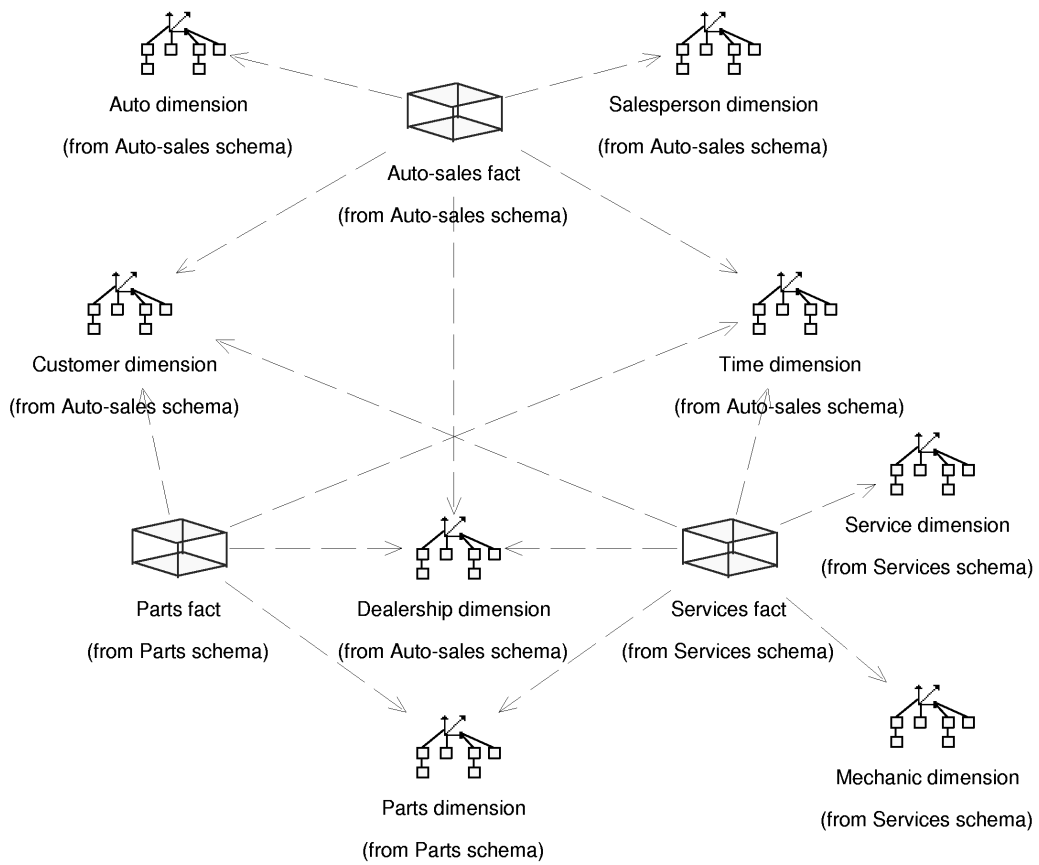
Figure 26: Level 3: Auto-sales fact

Figure 27: Merged level 2: representation of all the fact and dimension packages together
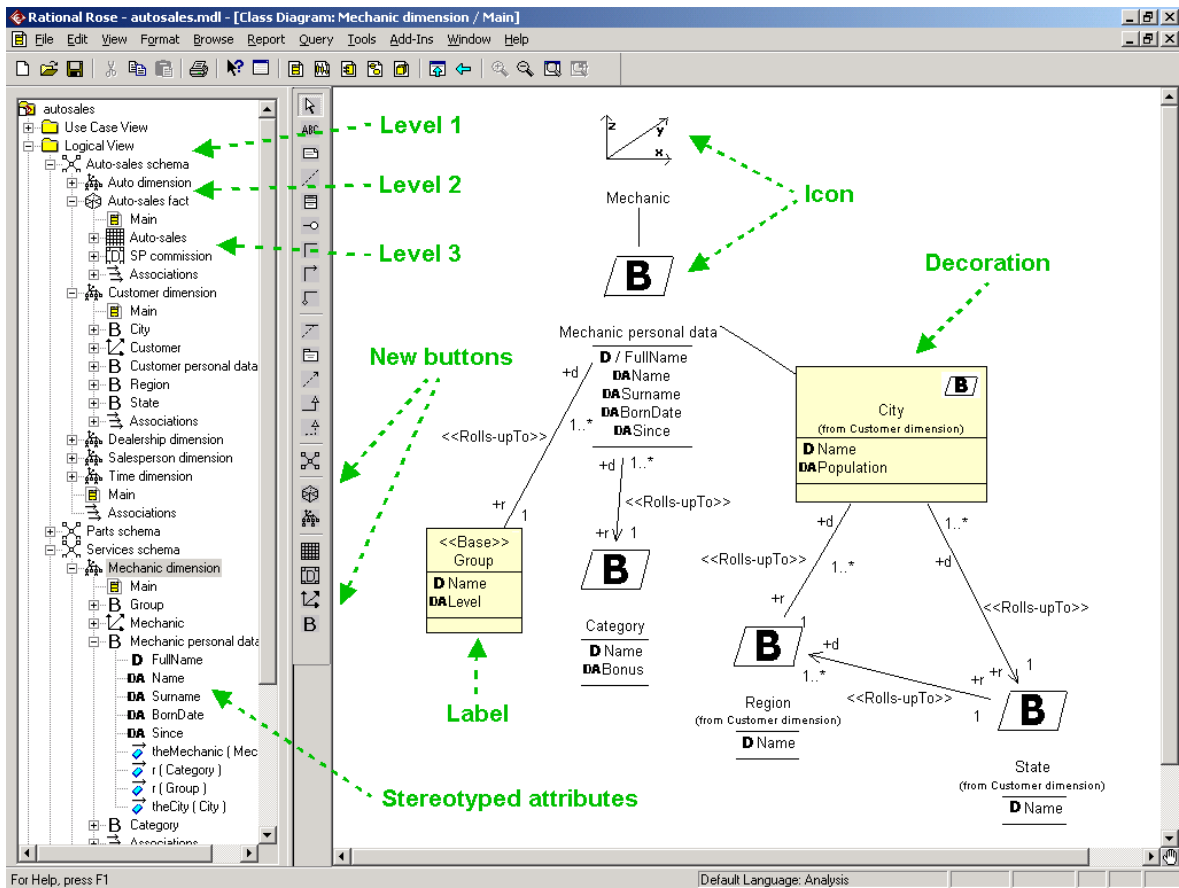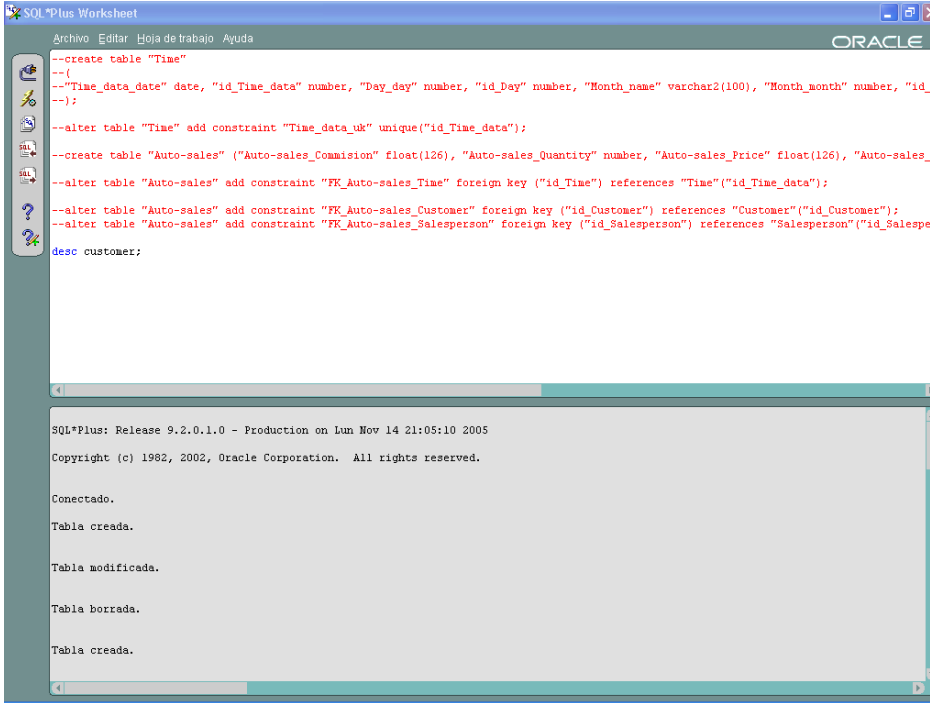
Figure 28: A screenshot from Rational Rose

6.6 Implementation in Oracle

In Oracle, creating a database basically consists of the following steps: (i) Creating the database's datafiles[17], its control files[18] and its redo log files[19], (ii) Creating the system tablespaces and (iii) Creating the data dictionary (tables, views, etc.). The steps (i) and (ii) are normally accomplished by the database administrator together with other issues such as defining the group of users or implementing permission issues. Therefore, we consider out of the scope of this paper to provide further detail on these steps. Instead, we will briefly mention the step (iii), i.e. how the data warehouse repository has been created.

Figure 29: A screenshot from SQL worksheet to define the relational tables of the DW repository

Regarding data warehouses, many issues are considered in real world projects such as the modeling of data sources, specifying the Extraction-Transformation-Loading (ETL) processes, defining and maintaining the metadata, the multidimensional modeling of the data warehouse repository or the modeling of the different OLAP tools to query the data warehouse. As this paper is focused on the multidimensional modeling of the DW repository (and mainly at the conceptual level), we will only focus on the definition of the relational tables that will correspond to the multidimensional model (accomplished with our UML proposal) and the use of a commercial OLAP tool to query the implemented multidimensional model. For more details on how to implement other data warehouse parts, we refer the reader to [23, 22, 24, 39].

Let us remind here that the CASE tool presented in the previous section generates an SQL script that contains the required SQL sentences to create the database tables and relations needed to specify the data warehouse repository according to the defined conceptual schema. In concrete, we have

---

[17]Files where the database server will host the data in the database structures, such as tables, materialized views, etc.

[18]The control files of a database store the status of the physical structure of the database. It contains (but is not limited to) the following types of information: database name and creation date, tablespace and datafile records, etc.

[19]The redo log files record all changes made in datafiles. If something happens to one of the datafiles of a database, a backed up datafile can be restored and the redo, that was written since, which brings the datafile to the state it had before it became unavailable (crash recovery).

implemented this case study in Oracle 9i. In Figure 29, we can see a screenshot from SQL worksheet. With this tool we open the SQL script file generated by our CASE tool and we execute it in order to create the required tables and relations that will host the data warehouse data.
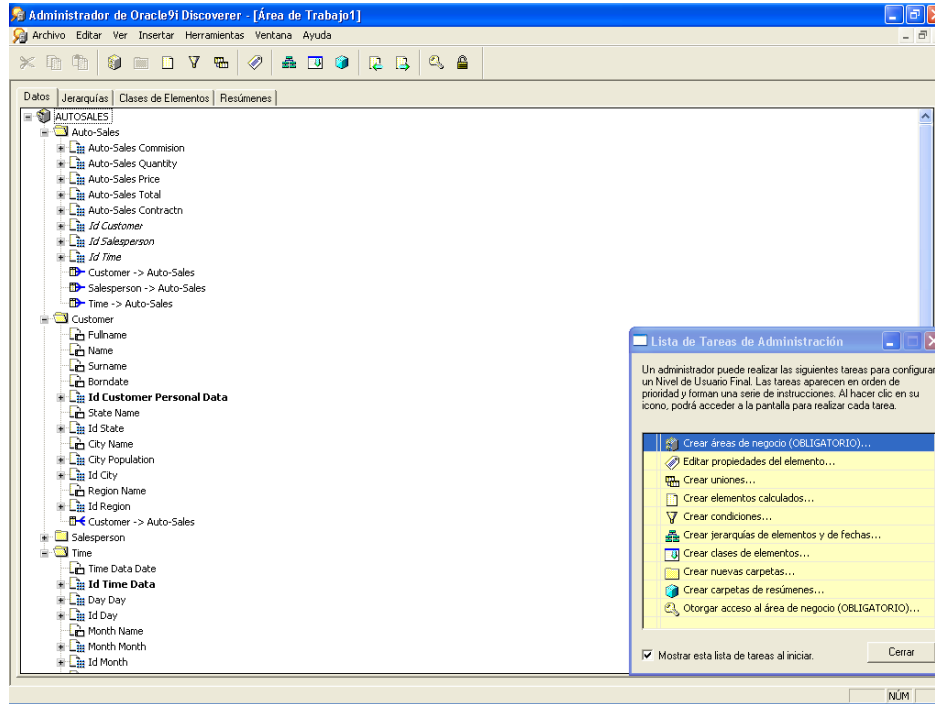

Figure 30: A screenshot from Discoverer Administrator showing the MD model of our case study

However, this is not enough if we wish to query a DW by using a commercial OLAP tool. To accomplish this, we need to upload and specify the multidimensional model of our data warehouse in the corresponding commercial OLAP tool we wish to use. In this case study, we have used Oracle Discoverer to query the data warehouse [32]. Discoverer provides two different tools: Discoverer Administrator and Discoverer Desktop. With the former, the data warehouse designer uploads, updates and maintains the multidimensional model of the implemented data warehouse. With this tool, we are also able to define the authorized users to query the data warehouse and the different parts of the MD model that can be accessed by the different group of users. In Figure 30, we can see a screenshot from the Discoverer Administrator tool with the MD model of the case study. We can easily see the fact *Auto-sales* and the dimensions *Salesperson*, *Customer*, *Time* and their corresponding attributes. Due to space constraints, we do not provide further detail about the dimensions and all their attributes. Once this MD model has been define with the Discoverer Administrator, an end user can query the data warehouse by using the Discoverer Desktop tool.

Finally, in Figure 31, we show a screenshot of a cube that provides data according to the requirement and use case 1 presented in Section 6.1. As easily seen, data is classified into two different classification hierarchy levels: *cities* where *customers* (Customer dimension) who bought *cars* live are placed in the horizontal header and the *months* (Time dimension) in which they were sold are placed in the vertical header. In the central cells of the cube, we can observe the figures for the measure *car sales* of the fact Auto-Sales.

7 Related Work

Lately, several MD data models have been proposed. Some of them fall into the logical level (such as the well known star-schema by R. Kimball [19]). Others may be considered as formal models as
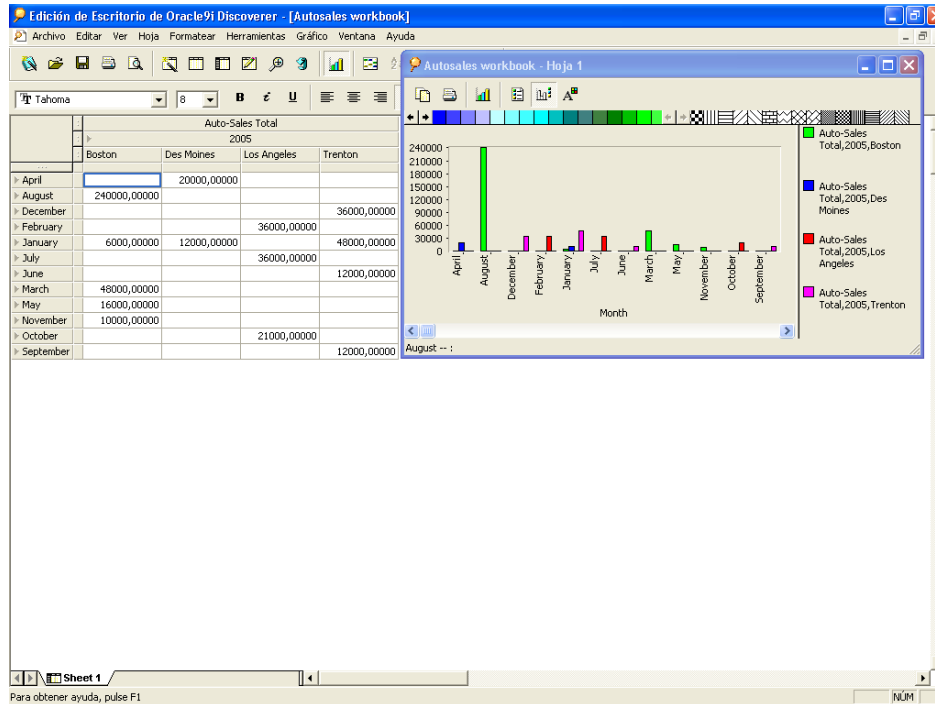
Figure 31: A screenshot from Discoverer Desktop showing a cube to provide data for the use case 1

they provide a formalism to consider main MD properties. A review of the most relevant logical and formal models can be found in [5] and [1].

In this section, we will make brief reference to the most relevant models that we consider "pure" conceptual MD models. These models provide a high level of abstraction for the main MD modeling properties presented in Section 2 and are totally independent from implementation issues. One outstanding feature provided by these models is that they provide a set of graphical notations (such as the classical and well known EER model) that facilitates their use and reading. These are as follows: *The Dimensional-Fact (DF) Model* by Golfarelli *et al.* [15], *The Multidimensional/ER (M/ER) Model* by Sapia *et al.* [36, 35], *The starER Model* by Tryfona *et al.* [41], the model proposed by Hüseman *et al.* [16], *The Yet Another Multidimensional Model (YAM^2)* by Abelló *et al.* [2], and *The Object Oriented Multidimensional Model* (OOMD) by Trujillo *et al.* [40].

In Table 2, we provide the coverage degree of each above-mentioned conceptual model regarding the main MD properties described in the previous section. To start with, to the best of our knowledge, only YAM^2 provides a grouping mechanism to avoid flat diagrams and simplify the conceptual design when a system becomes complex due to a high number of dimensions and facts sharing dimensions and their corresponding hierarchies. In particular, this model structures the MD modeling into different levels of complexity considering facts and dimensions at the first level, then classification hierarchies, and finally, the whole model. However, from our point of view, even though these different levels try to make the MD modeling easier, YAM^2 is a complex model not only for end users, but also for DW designers; mainly due to the high number of relations and classes that are needed in the design.

Regarding facts, only YAM^2 explicitly manages the term of multistar, which means that we are able to represent more than one fact in the same MD model (i.e. a star schema with more than one fact). Only the starER model, YAM^2, and OOMD consider *many-to-many* relationships between facts and particular dimensions by indicating the exact cardinality (multiplicity) between them. However, none of these models explicitly represents the term *degenerate facts*. We understand by degenerate facts the measures recorded in a "intersection table" of *many-to-many* relationships [14]. Only YAM^2 and OOMD consider derived measures together with their derivation rules as part of the conceptual

35

schema. The DF and the M/ER models represent derived measures with the provided query patterns, but not as part of the conceptual schema itself. The DF, the starER, YAM^2, and OOMD models consider the additivity of measures by explicitly representing the set of aggregation operators that can be applied on non-additive measures.

With reference to dimensions, only YAM^2 is able to have only one definition of a dimension and share it by different facts in *multistar* schemas, thereby avoiding defining the same dimension more than once and allowing the use of *conformed dimensions*. Moreover, only YAM^2 is able to define more than one role for a dimension regarding the same fact by connecting them through different associations. Only the OOMD model allows us to share only few classification hierarchy levels from dimensions, whereas the rest of the approaches force us to share the whole classification hierarchy path including all levels. All of the models consider multiple and alternative path classification hierarchies by means of Directed Acyclic Graphs (DAG) defined on certain dimension attributes. However, only the starER, YAM^2, and OOMD models consider non-strict classification hierarchies by specifying the exact cardinality between classification hierarchy levels; moreover, only the starER and OOMD models consider adequate to represent complete classification hierarchies. As both the M/ER and the starER models are extensions of the Entity Relationship (ER) model, they easily consider the categorization of dimensions by means of *Is-a* relationships. The YAM^2 and OOMD models represent the categorization of dimensions by means of generalization relationships of the OO paradigm.

Multidimensional modeling properties	Model					
	DF	M/ER	starER	Hüsem.	YAM^2	OOMD
Structural level						
Grouping mechanisms to avoid flat diagrams	No	No	No	No	Yes	No
Facts						
Multi-stars	No	No	No	No	Yes	No
many-to-many relations with particular dimensions	No	No	Yes	No	Yes	Yes
Degenerate facts	No	No	No	No	No	No
Atomic measures	Yes	Yes	Yes	Yes	Yes	Yes
Derived measures	No	No	No	No	Yes	Yes
Additivity	Yes	No	Yes	No	Yes	Yes
Dimensions						
Degenerate dimensions	No	No	No	No	No	Yes
Sharing dimensions (Conformed dimensions)	No	No	No	No	Yes	Yes
Different roles of a dimension with the same fact	No	No	No	No	Yes	No
Sharing few hierarchy levels	No	No	No	No	No	Yes
Multiple and alternative path classification hierarchies	Yes	Yes	Yes	Yes	Yes	Yes
Non-strict classification hierarchies	No	No	Yes	No	Yes	Yes
Complete classification hierarchies	No	No	Yes	No	No	Yes
Categorization of dimensions	No	Yes	Yes	Yes	Yes	Yes
Dynamic level						Yes
Specifying initial user requirements	Yes	Yes	No	No	Yes	Yes
OLAP operations	No	Yes	No	No	Yes	Yes
Modeling the behavior of the system	No	Yes	No	No	No	Yes
Graphical notation	Yes	Yes	Yes	Yes	Yes	Yes
Automatic generation into a target commercial OLAP tool	No	Yes	No	No	No	Yes

Table 2: Comparison of conceptual multidimensional models

With reference to the dynamic level of MD modeling, the starER model is the only one that does not provide an explicit mechanism to represent initial user requirements. On the other hand, only the M/ER, YAM^2, and OOMD models provide a set of basic OLAP operations to be applied from these initial user requirements. Instead, only the M/ER and OOMD models consider the behavior of the system by modeling the evolution of initial user requirements with state diagrams.

Finally, we note that all the models provide a graphical notation that facilitates the conceptual modeling task to the designer. On the other hand, only the M/ER and OOMD models provide a framework for an automatic generation of the database schema into a target commercial OLAP tool

(particularly into Informix Metacube and Cognos Powerplay).

From Table 2, one may conclude that none of the current conceptual modeling approaches considers all MD properties at both the structural and dynamic levels. From our point of view, the YAM^2 and OOMD models are the richest approaches as they consider most of the major MD properties, mainly because they are based on the OO paradigm. Therefore, we claim that a standard conceptual model is needed to consider all MD modeling properties at both the structural and dynamic levels. We argue that an OO approach with the UML is the right way of linking structural and dynamic level properties in an elegant way at the conceptual level.

8   Conclusions and Future Work

In this paper, we have presented an extension of the UML as a *profile* that allows us to accomplish the conceptual modeling of data warehouses by representing the major relevant MD properties at the conceptual level. We have invested some effort on ensuring that all the concepts have a well defined semantic basis. Therefore, this extension contains the needed stereotypes, tagged values and constraints for a complete and powerful MD modeling. In order to make our proposal as simple as possible, we have defined a minimal set, but powerful enough, of new UML elements that enable us to represent main MD modeling at the conceptual level. Moreover, we have used the OCL to specify the constraints attached to these new defined elements, thereby avoiding an arbitrary use of them. On the other hand, a frequent criticism highlighted at diagrammatic notations is their scalability; in our approach, thanks to the use of packages, we can elegantly represent huge and complex models at different levels of complexity. Therefore our UML extension allows us to avoid complex conceptual schemas and obtain understandable and easy to read conceptual schemas. On the other hand, packages also allow us to import elements defined in one package into another one, thereby avoiding redundancy as no different definitions for the same concept are permitted.

We have based our approach in UML for reasons: (i) UML is a widely-accepted object-oriented modeling language, which saves developers from learning a new model and its corresponding notations for specific MD modeling and (ii) UML can be easily extended so that it can be tailored for a specific domain with concrete peculiarities such as the multidimensional modeling for data warehouses.

Finally, we have defined concrete model transformations by following the standard Query/View/Transformations (QVT) that allow us to automatically generate the corresponding implementation into target platforms. We have also programmed this extension in a well known visual modeling tool, Rational Rose, from implementations can automatically be obtained. Finally, in order to show the applicability of our approach, we have applied our approach to a case study and implemented in Oracle.

We are currently working on defining a method (based on the Unified Process) for MD modeling based on the extension presented in this paper. This method will explicitly consider all underlying design guidelines that are hidden under every defined new MD element. Furthermore, in this UML extension we are also considering new stereotypes regarding object-oriented and object-relational databases for an automatic generation of the database schema into these kinds of databases. Other topics that we are also considering are (i) proposing a group of metrics as a means to describe good MD models based on more objective criteria and (ii) extending the current profile for considering the conceptual modeling of secure data warehouses. As a further future work, we will consider the specification of dynamic aspects of the multidimensional modeling such as the modeling of end user requirements for the current profile version.

Acknowledgments

References

[1] A. Abelló, J. Samos, and F. Saltor. A Framework for the Classification and Description of Multidimensional Data Models. In *Proc. of the 12th Intl. Conference on Database and Expert Systems Applications (DEXA'01)*, pages 668–677, Munich, Germany, September 2001.

[2] A. Abelló, J. Samos, and F. Saltor. YAM2 (Yet Another Multidimensional Model): An Extension of UML. In *Intl. Database Engineering & Applications Symposium (IDEAS 2002)*, pages 172–181, Edmonton, Canada, July 2002.

[3] A. Abelló, J. Samos, and F. Saltor. Benefits of an Object-Oriented Multidimensional Data Model. In *Proc. of the Symposium on Objects and Databases in 14th European Conference on Object-Oriented Programming (ECOOP'00)*, volume 1944 of *LNCS*, pages 141–152, Sophia Antipolis and Cannes, France, June 2000. Springer-Verlag.

[4] J. Arlow and I. Neustadt. *UML and the Unified Process. Practical Object-Oriented Analysis & Design.* Object Technology Series. Addison-Wesley, 2002.

[5] M. Blaschka, C. Sapia, G. Höfling, and B. Dinter. Finding your way through multidimensional data models. In *Proc. of the 9th Intl. Conf. on Database and Expert Systems Applications (DEXA'98)*, volume 1460 of *LNCS*, pages 198–203, Vienna, Austria, August 1998. Springer-Verlag.

[6] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language: User Guide.* Object Technology Series. Addison-Wesley, 1999.

[7] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM Sigmod Record*, 26(1):65–74, 1997.

[8] J. Conallen. *Building Web Applications with UML.* Object Technology Series. Addison-Wesley, 2000.

[9] K. Czarnecki and S. Helsen. Classification of Model Transformation Approaches. In *Proc. of the OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, Anaheim, USA, October 2003.

[10] EmPowerTec. Oclarity. Internet: http://www.empowertec.de/products/rational-rose-ocl.htm.

[11] M. Fowler. *UML Distilled. Applying the Standard Object Modeling Language.* Object Technology Series. Addison-Wesley, 1998.

[12] R. France and J. Bieman. Multi-View Software Evolution: A UML-based Framework for Evolving Object-Oriented Software. In *Proc. of the International Conference on Software Maintenance (ICSM 2001)*, pages 386–397, Florence, Italy, November 2001.

[13] L. Fuentes-Fernández and A. Vallecillo-Moreno. An Introduction to UML Profiles. *UPGRADE*, V(2):6–13, April 2004.

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

[15] M. Golfarelli, D. Maio, and S. Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems (IJCIS)*, 7(2-3):215–247, 1998.

[16] B. Husemann, J. Lechtenborger, and G. Vossen. Conceptual Data Warehouse Design. In *Proc. of the 2nd. Intl. Workshop on Design and Management of Data Warehouses (DMDW'00)*, pages 3–9, Stockholm, Sweden, June 2000.

[17] W. H. Inmon. *Building the Data Warehouse.* QED Press/John Wiley, 1992. (Last edition: 3rd edition, John Wiley & Sons, 2002).

[18] Institut National de Recherche en Informatique et en Automatique (INRIA). Model transformation at Inria. Internet: http://modelware.inria.fr/, 2004.

[19] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, 1996. (Last edition: 2nd edition, John Wiley & Sons, 2002).

[20] Laboratorul de Cercetare in Informatica - BABES-BOLYAI University. Object Constraint Language Environment. Internet: http://lci.cs.ubbcluj.ro/ocle/index.htm.

[21] W. Lehner. Modelling Large Scale OLAP Scenarios. In *Proc. of the 6th Int'l Conference On Extending Database Technology (EDBT'98)*, volume 1377 of *LNCS*, pages 153–167, Valencia, Spain, March 1998. Springer-Verlag.

[22] S. Luján-Mora and J. Trujillo. A data warehouse engineering process. In *Advances in Information Systems, Third International Conference (ADVIS'04)*, volume 3261 of *LNCS*, pages 14–23. Springer-Verlag, 2004.

[23] S. Luján-Mora and J. Trujillo. Physical modeling of data warehouses using UML. In *Proc. of the ACM 7th Intl. Workshop on Data warehousing and OLAP (DOLAP'04)*, pages 48–57, Washington D.C., USA, 2004.

[24] S. Luján-Mora and J. Trujillo. Physical Modeling of Data Warehouses Using UML Component and Deployment Diagrams. *Journal of Database Management*, 17(To be confirmed), 2006.

[25] S. Luján-Mora, J. Trujillo, and I. Song. Extending UML for Multidimensional Modeling. In *Proc. of the 5th Intl. Conf. on the Unified Modeling Language (UML'02)*, volume 2460 of *LNCS*, pages 290–304, Dresden, Germany, September 2002. Springer-Verlag.

[26] S. Luján-Mora, J. Trujillo, and I. Song. Multidimensional Modeling with UML Package Diagrams. In *Proc. of the 21st Intl. Conf. on Conceptual Modeling (ER'02)*, volume 2503 of *LNCS*, pages 199–213, Tampere, Finland, October 2002. Springer-Verlag.

[27] E.J. Naiburg and R.A. Maksimchuk. *UML for Database Design.* Object Technology Series. Addison-Wesley, 2001.

[28] No Magic, Inc. MagicDraw UML. Internet: http://www.magicdraw.com/.

[29] Object Management Group (OMG). Unified Modeling Language Specification 1.5. Internet: http://www.omg.org/cgi-bin/doc?formal/03-03-01, March 2003.

[30] Object Management Group (OMG). Model Driven Architecture (MDA). Internet: http://www.omg.org/mda/, 2004.

[31] Object Management Group (OMG). UML Superstructure Specification, v2.0. Internet: http://www.omg.org/cgi-bin/doc?formal/05-07-04, August 2005.

[32] Oracle. Oracle Business Intelligence Discoverer. http://www.oracle.com/technology/products/-discoverer/index.html.

[33] QVT-Partners. Revised submission for MOF 2.0 Query / Views / Transformations RFP. Internet: http://qvtp.org/downloads/1.1/qvtpartners1.1.pdf, 2003.

[34] Rational Software Corporation. *Using the Rose Extensibility Interface.* Rational Software Corporation, 2001.

[35] C. Sapia. On Modeling and Predicting Query Behavior in OLAP Systems. In *Proc. of the Int'l Workshop on Design and Management of Data Warehouses (DMDW'99)*, pages 1–10, Heidelberg, Germany, June 1999.

[36] C. Sapia, M. Blaschka, G. Höfling, and B. Dinter. 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, Singapore, November 1998. Springer-Verlag.

[37] G. Simsion. Better Data Models - Today Understanding Data Model Quality. The Data Administration Newsletter (TDAN.com). Internet: http://www.tdan.com/i034ht01.htm, October 2005.

[38] Tigris.org. ArgoUML. Internet: http://argouml.tigris.org/features.html.

[39] J. Trujillo and S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *Proc. of the 22nd Intl. Conf. on Conceptual Modeling (ER'03)*, volume 2813 of *LNCS*, pages 307–320, Chicago, USA, October 2003. Springer-Verlag.

[40] J. Trujillo, M. Palomar, J. Gómez, and I.Y. Song. Designing Data Warehouses with OO Conceptual Models. *IEEE Computer, special issue on Data Warehouses*, 34(12):66–75, December 2001.

[41] N. Tryfona, F. Busborg, and J.G. Christiansen. 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, November 1999.

[42] J. Warmer and A. Kleppe. *The Object Constraint Language. Precise Modeling with UML.* Object Technology Series. Addison-Wesley, 1998.

A   A UML Profile for Multidimensional Modeling

In this section, we present an extension to the UML in the form of a profile. A UML profile is defined as a UML package stereotyped $\ll profile \gg$, that can extend either a metamodel or another profile [13]. Unfortunately, it does not exist a standard way for documenting a UML profile.

According to [8], "An extension to the UML begins with a brief description and then lists and describes all of the stereotypes, tagged values, and constraints of the extension. In addition to these elements, an extension contains a set of well-formedness rules. These rules are used to determine whether a model is semantically consistent with itself". Therefore, based on this quote and our personal experience, we define our UML extension for MD modeling following the schema shown in Table 3.

- **Description**: A little description of the extension in natural language.
- **Prerequisite Extensions**: It indicates whether the current extension needs the existence of previous extensions.
- **Stereotypes**: The definition of the stereotypes.
- **Well-Formedness Rules**: The static semantics of the metaclasses are defined as a set of invariants defined by means of OCL expressions.
- **Comments**: Any additional comment, decision or example, usually written in natural language.

Table 3: Extension definition schema

For the definition of the stereotypes and tagged values, we follow the structure of the examples included in the UML Specification [29]. In Table 4 and Table 5 we show the schemas followed in our definition of the stereotypes and the tagged values, respectively.

For the definition of well-formedness rules and constraints we use the OCL [31]. In this way, we avoid an arbitrary use of the profile. Moreover, using OCL has several benefits: it is a well known

- **Name**: The name of the stereotype.

- **Base class** (also called Model class): The UML metamodel element that serves as the base for the stereotype.

- **Description**: An informal description with possible explanatory comments.

- **Icon**: It is possible to define a distinctive visual cue (an icon).

- **Constraints**: A list of constraints defined by means of OCL expressions associated with the stereotype, with an informal explanation of the expressions.

- **Tagged values**: A list of all tagged values that are associated with the stereotype.

Table 4: Stereotype definition schema

- **Name**: The name of the tagged value.

- **Type**: The name of the type of the values that can be associated with the tagged value.

- **Multiplicity**: The maximum number of values that may be associated with the tagged value.

- **Description**: An informal description with possible explanatory comments.

Table 5: Tagged value definition schema

constraint language, we do not need to invest effort on defining a new language, and there is tool support for OCL, both from the academic and open source movement [20, 38] and the software industry [10, 28]. In order to increase the readability of the constraint definitions, we use the conventions stated in the UML Specification:

- `self`, which can be omitted as a reference to the element defining the context of the invariant, has been kept for clarity. For example, we write:
  self.feature
  instead of only
  feature.

- In expressions where a collection is iterated, an iterator is used for clarity, even when formally unnecessary. However, the type of the iterator is usually omitted. For example, we write:
  self.contents->forAll(me | not me.oclIsKindOf(Package))
  instead of
  self.contents->forAll(not me.oclIsKindOf(Package)).

- The `collect` operation is left implicit where possible. For example, we write:
  self.connection.participant
  instead of
  self.connection->collect(participant).

We have defined fourteen stereotypes: three specialize in the Package model element[20], three specialize in the Class model element, one specializes in the AssociationClass model element, five specialize in the Property model element, and two specialize in the Association model element. In Figure 32, we have represented a portion of the UML metamodel to show where our stereotypes fit. We have only represented the specialization hierarchies, as the most important fact about a stereotype

---

[20]We have based our MD extension on the most semantically similar constructs in the UML metamodel.
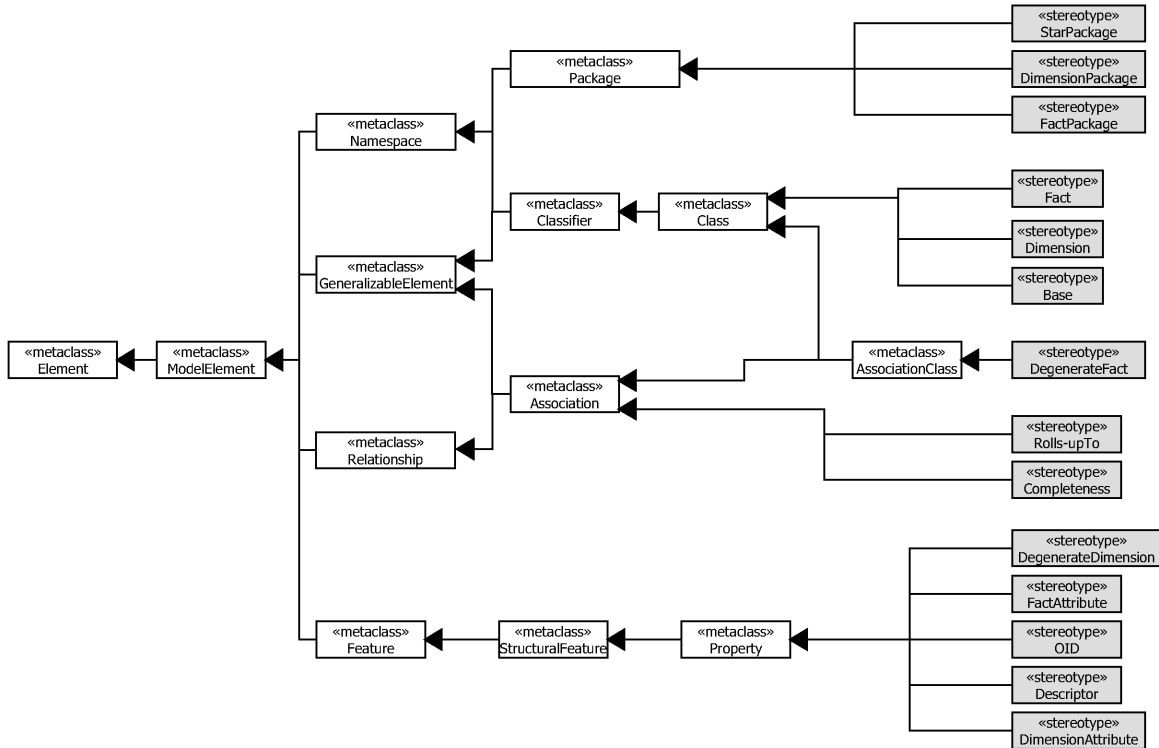
Figure 32: Extension of the UML with stereotypes

is the base class that the stereotype specializes. In this figure, new stereotypes are colored in grey, whereas classes from the UML metamodel remain white. The notation for an extension is an arrow pointing from a stereotype to the extended class, where the arrowhead is shown as a solid triangle.

Some issues of our MD approach, such as the derivation rule or the initial value of an attribute, are not defined in our stereotypes because these concepts have already been defined in the UML metamodel. We provide a list of these concepts in Table 6.

In the following, we present our extension following the extension definition schema shown in Table 3.

A.1   Description

This UML extension defines a set of stereotypes, tagged values, and constraints that enable us to design MD models. The stereotypes are applied to certain components that are particular to MD modeling, allowing us to represent them in the same model and on the same diagrams that describe the rest of the system. The MD models are divided into three levels: model definition (level 1), star schema definition (level 2), and dimension/fact definition (level 3)[21].

The major elements to MD modeling are the Fact class and the Dimension class. A Fact class consists of FactAttributes and DegenerateDimensions. The hierarchy levels of a Dimension are represented by means of Base classes. A Base class consists of OIDs, Descriptors, and DimensionAttributes. Finally, Rolls-upTo and Completeness association are also defined.

The correct use of this extension is assured by the definition of 51 constraints specified both in natural language and in OCL expressions (to avoid redundancy) in the definitions of the different stereotypes.

---

[21]Although we use the concept star schema, it does not imply any relational implementation of the DW. We prefer to use a well known concept instead of inventing a new term.

Concept	Comes from	Description	Used by
name	`ModelElement`	It is an identifier for the `ModelElement`	Base, Completeness, Descriptor, Dimension, DimensionAttribute, Fact, FactAttribute, OID
documentation	`Element`	It is a comment, description or explanation of the `Element` to which it is attached	Base, Completeness, Descriptor, Dimension, DimensionAttribute, Fact, FactAttribute, OID
type	`StructuralFeature`	Designates the classifier whose instances are values of the feature	Descriptor, Dimension-Attribute, Fact-Attribute, OID
initialValue	`Attribute`	An expression specifying the value of the `Attribute` upon initialization	Descriptor, Dimension-Attribute, Fact-Attribute, OID
derived	`ModelElement`	A true value indicates that the `ModelElement` can be completely derived from other model elements and is therefore logically redundant	Descriptor, Dimension-Attribute, FactAttribute

Table 6: Concepts inherited from the UML metamodel

A.2 Prerequisite Extensions

No other extension to the language is required for the definition of this extension.

A.3 Stereotypes

The stereotypes are presented depending on the base class that specializes: Package, Class, AssociationClass, Property, and Association.

A.3.1 Stereotypes of Package

Three stereotypes have been defined from the Package model element: StarPackage, DimensionPackage, and FactPackage.

- Name: **StarPackage**
- Base class: Package
- Description: Packages of this stereotype represent MD star schemas
- Icon: Figure 33 (a)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Package
  inv: self.isStereotyped(StarPackage) implies

  – A StarPackage can only contain FactPackages or DimensionPackages:[22]
    self.contents->forAll(me | me.oclIsTypeOf(FactPackage) or me.oclIsTypeOf(DimensionPackage))

  – A StarPackage can only contain one FactPackage:[23]
    self.ownedElement->select(me | me.oclIsTypeOf(FactPackage))->size <= 1

  – A StarPackage cannot import a FactPackage from another StarPackage (only DimensionPackage):
    self.importedElement->forAll(me | me.oclIsTypeOf(DimensionPackage))

  – There are no cycles in the dependency structure:[24]
    not self.allSuppliers->includes(self)

- Tagged values: None

---

[22]Some operations used in our extension are not from the OCL standard. For example, contents is an additional operation defined in the UML Specification [29]: "The operation contents results in a Set containing the ModelElements owned by or imported by the Package".

[23]It is not mandatory that every StarPackage has a FactPackage, because it is possible to have utility packages with only DimensionPackages for defining conformed dimensions to be imported by other packages.

[24]allSuppliers is an additional operation defined in the UML Specification [29]: "The operation allSuppliers results in a Set containing all the ModelElements that are suppliers of this ModelElement, including the suppliers of these ModelElements. This is the transitive closure".
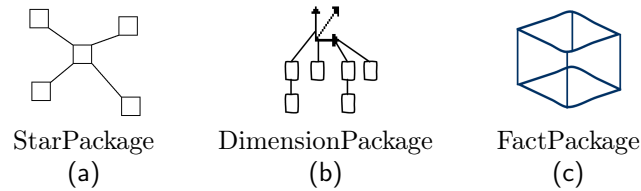
Figure 33: Stereotype icons of Package

- Name: **DimensionPackage**
- Base class: Package
- Description: Packages of this stereotype represent MD dimensions
- Icon: Figure 33 (b)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Package
  inv: self.isStereotyped(DimensionPackage) implies

  – It is not possible to create a dependency from a DimensionPackage to a FactPackage (only to another DimensionPackage):
  self.clientDependency->forAll(d | d.supplier->forAll(me | me.oclIsTypeOf(DimensionPackage)))

  – There are no cycles in the dependency structure:
  not self.allSuppliers->includes(self)

  – A DimensionPackage cannot contain Packages:
  self.contents->forAll(me | not me.oclIsKindOf(Package))

  – A DimensionPackage can only contain Dimension or Base classes:
  self.contents->select(co | co.oclIsKindOf(Class))->forAll(f | f.oclIsTypeOf(Dimension) or
  f.oclIsTypeOf(Base))

  – A DimensionPackage must have a Dimension class (and only one):
  self.contents->select(me | me.oclIsTypeOf(Dimension))->size = 1

- Tagged values: None

- Name: **FactPackage**
- Base class: Package
- Description: Packages of this stereotype represent MD facts
- Icon: Figure 33 (c)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Package
  inv: self.isStereotyped(FactPackage) implies

  – There are no cycles in the dependency structure:
  not self.allSuppliers->includes(self)

  – A FactPackage cannot contain Packages:
  self.contents->forAll(me | not me.oclIsKindOf(Package))

  – A FactPackage can only contain Fact, DegenerateFact, Dimension or Base classes:
  self.contents->select(co | co.oclIsKindOf(Class))->forAll(f | f.oclIsTypeOf(Fact) or
  f.oclIsTypeOf(DegenerateFact) or f.oclIsTypeOf(Dimension) or f.oclIsTypeOf(Base))

  – A FactPackage must have a Fact class (and only one):
  self.contents->select(me | me.oclIsTypeOf(Fact))->size = 1

- Tagged values: None

A.3.2 Stereotypes of Class

Three stereotypes have been defined from the Class model element: Fact, Dimension, and Base.

- Name: **Fact**
- Base class: Class
- Description: Classes of this stereotype represent facts in a MD model
- Icon: Figure 34 (a)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Class
  inv: self.isStereotyped(Fact) implies

  - All attributes of a Fact must be DegenerateDimension or FactAttribute:
    self.feature->select(fe | fe.oclIsKindOf(Property))->forAll(f | f.oclIsTypeOf(DegenerateDimension) or
    f.oclIsTypeOf(FactAttribute))
  - All associations of a Fact must be aggregations[25] (neither *none* nor *composite*):
    self.association->forAll(as | as.aggregation = #aggregate)
  - A Fact can only be associated with Dimension classes:[26]
    self.allOppositeAssociationEnds->forAll(ae | ae.participant.oclIsTypeOf(Dimension))

- Tagged values: None

- Name: **Dimension**
- Base class: Class
- Description: Classes of this stereotype represent dimensions in a MD model
- Icon: Figure 34 (b)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Class
  inv: self.isStereotyped(Dimension) implies

  - A Dimension cannot have neither attributes nor methods:
    self.feature->size = 0
  - All associations of a Dimension with a Fact must be aggregations at the end of the Fact (the opposite end):
    self.association.association->forAll(as | as.associationEnd.participant.oclIsTypeOf(Fact) implies
    as.associationEnd.aggregation = #aggregate)
  - All associations of a Dimension with a Fact must not be aggregations at the end of the Dimension (the current end):
    self.association.association->forAll(as | as.associationEnd.participant.oclIsTypeOf(Fact) implies as.aggregation
    <> #aggregate)
  - A Dimension can only be associated with Fact or Base classes:
    self.allOppositeAssociationEnds->forAll(ae | ae.participant.oclIsTypeOf(Fact) or
    ae.participant.oclIsTypeOf(Base))
  - A Dimension can only be associated with one Base class:
    self.allOppositeAssociationEnds->select(ae | ae.participant.oclIsTypeOf(Base))->size <= 1

- Tagged values:

  - isTime:
    * Type: UML::Datatypes::Boolean
    * Multiplicity: 1
    * Description: Indicates whether the dimension represents a time dimension or not[27]

- Name: **Base**
- Base class: Class
- Description: Classes of this stereotype represent dimension hierarchy levels in a MD model
- Icon: Figure 34 (c)

---

[25]The part may be contained in other aggregates.

[26]**allOppositeAssociationEnds** is an additional operation defined in the UML specification [29]: "The operation allOppositeAssociationEnds results in a set of all AssociationEnds, including the inherited ones, that are opposite to the Classifier".

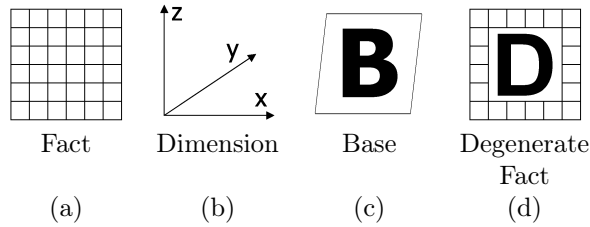[27]The "Time dimension" is treated differently from the others in OLAP tools.

Figure 34: Stereotype icons of Class and AssociationClass

- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Class
  inv: self.isStereotyped(Base) implies

  - All attributes of a Base must be OID, Descriptor, or DimensionAttribute:
    self.feature->select(fe | fe.oclIsKindOf(Property))->forAll(f | f.oclIsTypeOf(OID) or
    f.oclIsTypeOf(Descriptor) or f.oclIsTypeOf(DimensionAttribute))

  - A Base must have a Descriptor attribute (and only one):
    self.feature->select(fe | fe.oclIsKindOf(Property))->select(f | f.oclIsTypeOf(Descriptor))->size = 1

  - A Base may have an OID attribute:
    self.feature->select(fe | oclIsKindOf(Property))->select(f | f.oclIsTypeOf(OID))->size <= 1

  - A Base can only be associated with Dimension or Base classes:
    self.allOppositeAssociationEnds->forAll(ae | ae.participant.oclIsTypeOf(Dimension) or
    ae.participant.oclIsTypeOf(Base))

  - A Base cannot be associated with itself (in order to avoid cycles):
    self.allOppositeAssociationEnds->forAll(ae | ae.participant <> self)

  - A Base class may only inherit from another Base class:
    self.generalization->size > 0 implies self.generalization.parent->forAll(me | me.oclIsTypeOf(Base))

  - A Base class may only be parent of another Base class:
    self.specialization->size > 0 implies self.specialization.child->forAll(me | me.oclIsTypeOf(Base))

  - A Base can only be child in one generalization (no multiple inheritance):
    self.generalization->size <= 1

  - A Base cannot simultaneously be a child in a generalization/specialization hierarchy and belong to an
    association hierarchy:
    (self.generalization->size = 1) implies (self.association->size = 0)

- Tagged values: None

A.3.3 Stereotypes of AssociationClass

One stereotype has been defined from the AssociationClass model element: DegenerateFact.

- Name: **DegenerateFact**
- Base class: AssociationClass
- Description: Classes of this stereotype represent degenerate facts in a MD model
- Icon: Figure 34 (d)
- Constraints:
  context UML::Classes::AssociationClasses:AssociationClass
  inv: self.isStereotyped(DegenerateFact) implies

  - All attributes of a DegenerateFact class must be DegenerateDimension or FactAttribute:
    self.feature->select(fe | fe.oclIsKindOf(Property))->forAll(f | f.oclIsTypeOf(DegenerateDimension) or
    f.oclIsTypeOf(FactAttribute))

  - A DegenerateFact association can only be connected to two elements[28]:
    self.connection->size = 2

  - One of the ends of a DegenerateFact has to be a Fact and the other end has to be a Dimension:
    self.connection.participant->exist(me | me.oclIsTypeOf(Fact)) and self.connection.participant->exist(me |
    me.oclIsTypeOf(Dimension))

- Tagged values: None

---

[28]In UML, an association can be connected to two or more elements.

A.3.4 Stereotypes of Property

Five stereotypes have been defined from the Property model element: DegenerateDimension, Fact-Attribute, OID, Descriptor, and DimensionAttribute.

- Name: **DegenerateDimension**
- Base class: Property
- Description: Attributes of this stereotype represent degenerate dimensions in a MD model
- Icon: Figure 35 (a)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Property
  inv: self.isStereotyped(DegenerateDimension) implies

  - A DegenerateDimension cannot be derived:
    not self.derived

  - A DegenerateDimension can only belong to a Fact or a DegenerateFact:
    self.owner.oclIsTypeOf(Fact) or self.owner.oclIsTypeOf(DegenerateFact)

- Tagged values: None

- Name: **FactAttribute**
- Base class: Property
- Description: Attibutes of this stereotype represent attributes of Fact or DegenerateFact classes in a MD model
- Icon: Figure 35 (b)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Property
  inv: self.isStereotyped(FactAttribute) implies

  - A FactAttribute can only belong to a Fact or a DegenerateFact:
    self.owner.oclIsTypeOf(Fact) or self.owner.oclIsTypeOf(DegenerateFact)

  - If a FactAttribute is derived, then it needs a derivation rule (an OCL expression):
    self.derived implies self.derivationRule.oclIsTypeOf(OclExpression)

- Tagged values:

  - derivationRule:

    * Type: UML::Datatypes::String
    * Multiplicity: 1
    * Description: If the attribute is derived, this tagged value represents the derivation rule

---

- Name: **OID**
- Base class: Property
- Description: Attributes of this stereotype represent OID attributes of Base classes in a MD model[29]
- Icon: Figure 35 (c)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Property
  inv: self.isStereotyped(OID) implies

  - An OID can only belong to a Base:
    self.owner.oclIsTypeOf(Base)

  - An OID cannot be derived:
    not self.derived

- Tagged values: None

- Name: **Descriptor**
- Base class: Property
- Description: Attributes of this stereotype represent descriptor attributes of Base classes in a MD model
- Icon: Figure 35 (d)

---

[29]See Section 3 or [40] for further information on OID and Descriptor attributes.

<div align="center">

DD   **OID**   **FA**   **D**   **DA**

</div>

Degenerate Dimension	OID	Fact Attribute	Descriptor	Dimension Attribute
(a)	(b)	(c)	(d)	(e)

Figure 35: Stereotype icons of Property

- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Property
  inv: self.isStereotyped(Descriptor) implies

  – A Descriptor can only belong to a Base:
  self.owner.oclIsTypeOf(Base)

  – If a Descriptor is derived, then it needs a derivation rule (an OCL expression):
  self.derived implies self.derivationRule.oclIsTypeOf(OclExpression)

- Tagged values:

  – derivationRule:
    * Type: UML::Datatypes::String
    * Multiplicity: 1
    * Description: If the attribute is derived, this value represents the derivation rule

---

- Name: **DimensionAttribute**
- Base class: Property
- Description: Attributes of this stereotype represent attributes of Base classes in a MD model
- Icon: Figure 35 (e)
- Constraints:
  context UML::InfrastructureLibrary::Core::Constructs::Property
  inv: self.isStereotyped(DimensionAttribute) implies

  – A DimensionAttribute can only belong to a Base:
  self.owner.oclIsTypeOf(Base)

  – If a DimensionAttribute is derived, then it needs a derivation rule (an OCL expression):
  self.derived implies self.derivationRule.oclIsTypeOf(OclExpression)

- Tagged values:

  – derivationRule:
    * Type: UML::Datatypes::String
    * Multiplicity: 1
    * Description: If the attribute is derived, this value represents the derivation rule

  – isOptional:
    * Type: UML::Datatypes::Boolean
    * Multiplicity: 1
    * Description: An optional attribute needs not be specified for each element of the corresponding Base class and therefore may contain "null" values

A.3.5 Stereotype of Association

Two stereotypes have been defined from the Association model element: Rolls-upTo and Completeness.

- Name: **Rolls-upTo**
- Base class: Association
- Description: Associations of this stereotype represent associations between Base classes
- Icon: None

- Constraints:
context UML::InfrastructureLibrary::Core::Constructs::Association
inv: self.isStereotyped(Rolls-upTo) implies

  - The ends of a Rolls-upTo association can only be Base classes:
  self.connection.participant->forAll(pa | pa.oclIsTypeOf(Base))
  - A Rolls-upTo association can only be connected to two elements:[30]
  self.connection->size = 2
  - In a Rolls-upTo association, one of the ends contains the role r and the other end contains the role d:[31]
  self.associationEnd->exists(ae | ae.name = 'r') and self.associationEnd->exists(ae | ae.name = 'd')

- Tagged values: None

- Name: **Completeness**
- Base class: Association
- Description: Associations of this stereotype represent complete associations[32] between Base classes
- Icon: None
- Constraints:
context UML::InfrastructureLibrary::Core::Constructs::Association
inv: self.isStereotyped(Completeness) implies

  - The ends of a Completeness association can only be Base classes:
  self.connection.participant->forAll(pa | pa.oclIsTypeOf(Base))
  - A Completeness association can only be connected to two elements:
  self.connection->size = 2
  - In a Completeness association, one of the ends contains the role r and the other end contains the role d:
  self.associationEnd->exists(ae | ae.name = 'r') and self.associationEnd->exists(ae | ae.name = 'd')

- Tagged values: None

A.4 Well-Formedness Rules

There are only two rules for the Namespace element:

context UML::InfrastructureLibrary::Core::Constructs::Namespace
inv:

  - All the classes in a MD model must be Fact, Dimension, or Base:[33]
  self.allContents->forAll(oclIsKindOf(Class) implies (oclIsTypeOf(Fact) or oclIsTypeOf(Dimension) or oclIsTypeOf(Base)))
  - All the packages in a MD model must be StarPackage, FactPackage, or DimensionPackage:
  self.allContents->forAll(oclIsKindOf(Package) implies (oclIsTypeOf(StarPackage) or oclIsTypeOf(FactPackage) or oclIsTypeOf(DimensionPackage)))

A.5 Comments

Next, we summarize the UML elements we have just used or defined to consider the main relevant MD properties:

- Facts and dimensions: they are represented by means of Fact and Dimension stereotypes.

- Many-to-many relationships: thanks to the flexibility of the shared-aggregation relationships, we can consider many-to-many relationships between facts and particular dimensions by means of the 1..* cardinality on the dimension class role. In these cases, a DegenerateFact can be defined to add more information.

---

[30] In the UML, an association can have more that two association ends.

[31] The role is the name of the AssociationEnd.

[32] A complete association means that all members belong to one higher-class object and that object consists of those members only.

[33] allContents is an additional operation defined in the UML specification [29]: "The operation allContents results in a Set containing all ModelElements contained by the Namespace".

- Derived measures: they are represented by means of derived attributes from the UML metamodel and the tagged value derivationRule we have defined in the Descriptor, DimensionAttribute, and FactAttribute stereotypes.

- Classification hierarchies: they are considered by means of the association between Dimension and Base stereotypes.

- Strictness: the multiplicity 1 and 1..* defined in the target associated class role of a classificaton hierarchy address the concepts of strictness and nonstrictness.

- Completeness: the stereotype Completeness addresses the completeness of a classification hierarchy.

- Categorizing dimensions: we use generalization-specialization relationships to categorize a Dimension.

B   List of Acronyms

- CASE: Computer-Aided Software Engineering

- DAG: Directed Acyclic Graph

- DF: The Dimensional Fact Model

- DW: Data Warehouse

- MDA: Model Driven Architecture

- MD: Multidimensional Modeling

- M/ER: Multidimensional/Entity-Relationship model

- MOF: Meta-Object Facility

- MTL: Model Transformation Language

- OCL: Object Constraint Language

- OLAP: On-Line Analytical Processing

- OMG: Object Management Group

- OO: Object-Oriented

- PIM: Platform Independent Model

- PSM: Platform Specific Models

- QVT: Query, View, Transformation

- RDBMS: Relational Database Management System

- REI: Rose Extensibility Interface

- RFP: Request for Proposal

- UML: Unified Modeling Language

- YAM2: Yet Another Multidimensional Model

- Acronyms of some stereotypes defined in our UML profile

  - D: Descriptor attribute

- DA: Dimension Attribute
- DD: Degenerate Dimension
- F: Fact Attribute
- DC: Dimension Class
- FC: Fact Class
- BC: Base Class
- DF: Degenerated Fact