Multidimensional Modeling with UML Package Diagrams*

Sergio Luján-Mora[1], Juan Trujillo[1], and Il-Yeol Song[2]

[1] Dept. de Lenguajes y Sistemas Informáticos
Universidad de Alicante (Spain)
{slujan,jtrujillo}@dlsi.ua.es
[2] College of Information Science and Technology
Drexel University (USA)
songiy@drexel.edu

Abstract. The Unified Modeling Language (UML) has become the *de facto* standard for object-oriented analysis and design, providing different diagrams for modeling different aspects of a system. In this paper, we present the development of multidimensional (MD) models for data warehouses (DW) using UML package diagrams. In this way, when modeling complex and large DW systems, we are not restricted to use flat UML class diagrams. We present design guidelines and illustrate them with various examples. We show that the correct use of the package diagrams using our design guidelines will produce a very simple yet powerful design of MD models. Furthermore, we provide a UML extension by means of stereotypes of the particular package items we use. Finally, we show how to use these stereotypes in Rational Rose 2000 for MD modeling.

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

1 Introduction

Multidimensional (MD) modeling is the foundation of data warehouses (DW), MD databases, and OLAP applications. These systems provide companies with many years of historical information for the decision making process. MD modeling structures information into facts and dimensions. A fact table contains interesting measures of a business process (sales, deliveries, etc.), whereas a dimension table represents the context for analyzing a fact (product, customer, time, etc.). Various approaches for the conceptual design of MD systems have been proposed in the last few years [1][2][3][4] to represent main MD structural and dynamic properties. However, none of them has been accepted as a standard conceptual model for MD modeling. Due to space constraints, we refer the reader to [5] for a detailed comparison and discussion about most of these models.

* This paper has been partially supported by the Spanish Ministery of Science and Technology, project number TIC2001-3530-C02-02.

On the other hand, the Unified Modeling Language (UML) [6] has been widely accepted as the standard object-oriented (OO) modeling language for describing and designing various aspects of software systems. Therefore, any approach using the UML will minimize the efforts of developers in learning new diagrams or methodologies for every subsystem to be modeled. Following this consideration, we have previously proposed in [7] an OO conceptual MD modeling approach, based on the UML, for a powerful conceptual MD modeling. This proposal considers major relevant MD properties at the conceptual level in an elegant and easy way.

In this paper, we start from our previously presented approach [7] and show how to apply the grouping mechanism called *package* provided by the UML. A package groups 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 data warehouse systems, we are not restricted to use flat UML class diagrams.

Furthermore, based on our experience in developing real world cases, we provide design guidelines to properly and easily apply packages to MD modeling. These design guidelines are extremely relevant for two reasons. First, the UML Specification does not formally define how to apply packages, and therefore, different people may use them in different ways. Second, these guidelines are very close to the natural way both designers and analyzers understand and accomplish MD modeling. Our experience indicates that these guidelines produce a very simple yet powerful design of large and complex MD models.

To facilitate the MD modeling using our package diagrams, we provide a UML extension by using stereotypes of the particular package items we define. Our extension uses the Object Constraint Language (OCL) [6] for expressing well-formedness rules of the new defined elements, thereby avoiding an arbitrary use of our extension. Finally, we use these package stereotypes in Rational Rose 2000 for MD modeling to show the applicability of our proposal.

The remainder of the paper is structured as follows: Section 2 briefly presents other works related to grouping mechanisms in conceptual modeling. Section 3 summarizes how we have previously used the UML for proper conceptual MD modeling. Section 4 presents our design guidelines for using packages in MD modeling. Section 5 presents a case study to show how our guidelines are properly applied for MD modeling. Section 6 presents the definition of our UML extension in terms of package stereotypes. Section 7 shows how to apply our package extension in Rational Rose. Finally, Section 8 presents the main conclusions and introduces our immediate future work.

2 Related Work

The benefits of layering modeling diagrams have been widely recognized. Different modeling techniques, such as Data Flow Diagrams, Functional Modeling (IDEF0), Entity Relationship Model (ER) and the UML make use of some kind of layering mechanism.

Focusing on the aim of this paper, i.e. data modeling, several approaches have been proposed to provide grouping mechanisms to the ER to simplify complex diagrams. In [8], the *Clustered Entity Model*, one of the early attempts at layering ER diagrams, is presented. In this approach, an ER diagram at a lower level appears as an entity on the next level. In [9], a model and a technique for clustering entities in an ER diagram is described. This modeling technique refines ER diagrams into higher-level objects that lead to a description of the conceptual database on a single page. The great benefit of this proposal is that it increases the clarity of the database description, and therefore, facilitates a better communication between end-users and the database designer. In [10], the *Leveled Entity Relationship Model* presents another layering formalism for ER diagrams.

Regarding the UML, and as it is stated in [11], "There are different views about how to use UML to develop software systems". The UML is a complex modeling language and it lacks a systematic way of guiding the users through the development of systems. With respect to *packages*, the UML defines them as a mechanism to simplify complex UML diagrams. However, no formal design guidelines regarding how to properly use them are clearly stated. In this context, many text books and authors have provided guidelines to apply packages in general or in a specific domain. For example, Fowler states [12]: "I use the term package diagram for a diagram that shows packages of classes and the dependencies among them". In a specific domain such as web applications, Conallen states [13] that "A package is merely a mechanism to divide the model into more manageable pieces" and "Try to avoid making the package hierarchy match the semantics of the business, and instead use packages as a means of managing the model". The author proposes to make packages "*comprehensible*, *cohesive*, *loosely coupled* and *hierarchically shallow*". To the best of our knowledge, no work has been presented showing the benefits of using UML packages for MD modeling.

On the other hand, there have lately been proposed several approaches to accomplish the conceptual design of data warehouses following the MD paradigm. Due to space constraints, we will only make a brief reference to those works which are close to the research topic of this paper [1][2][3][4]. These MD models are mainly conceived to gain user requirements and provide an easy to be used but yet powerful set of graphical elements to facilitate the task of conceptual modeling as well as the specification of queries. Both [2] and [3] extend the ER model to use it for MD modeling providing specific items such as facts, dimension levels and so on. [1] and [4] propose different graphical notations for data warehouse conceptual design.

Motivation: All the above-commented MD approaches use "flat design" in the sense that all the elements that form a MD model (e.g. facts, dimensions, classification hierarchies and so on) are represented in the same diagram at the same level. Therefore, these approaches are not often suitable for huge and complex MD models in which several facts share many dimensions and their classification hierarchies, thereby leading to cluttered diagrams that are very difficult to read. Based on our experience in designing real-world cases, we argue

that in most cases we need an approach that structures the design of complex data warehouses at different levels.

3 Object-Oriented Multidimensional Modeling

In this section, we summarize[3] how an OO MD model can represent main structural aspects of MD modeling. The main features considered are the many-to-many relationships between facts and dimensions, degenerate dimensions, multiple and alternative path classification hierarchies, and non-strict and complete hierarchies. In this approach, the main structural properties of MD models are specified by means of a UML class diagram in which the information is clearly separated into facts and dimensions.

Dimensions and facts are represented by *dimension classes* and *fact classes*, respectively. Then, fact classes are specified as composite classes in shared aggregation relationships of n dimension classes. The flexibility of shared aggregation in the UML allows us to represent *many-to-many* relationships between facts and particular dimensions by indicating the 1..* cardinality on the dimension class role. In our example in Fig. 1 (a), we can see how the fact class Sales has a many-to-one relationship with both dimension classes.

By default, all measures in the fact class are considered additive. For non-additive measures, additive rules are defined as constraints and are included in the fact class. Furthermore, derived measures can also be explicitly considered (indicated by /) and their derivation rules are placed between braces near the fact class, as shown in Fig. 1 (a).

This OO approach also allows us to define identifying attributes in the fact class, by placing the constraint *{OID}* next to an attribute name. In this way we can represent *degenerate dimensions* [14][15], thereby representing other fact features in addition to the measures for analysis. For example, we could store the ticket number (ticket_num) and the line number (line_num) as degenerate dimensions, as reflected in Fig. 1 (a).

With respect to dimensions, every *classification hierarchy* level is specified by a class (called a *base class*). An association of classes specifies the relationships between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the dimension class (constraint *{dag}* placed next to every dimension class). The DAG structure can represent both alternative path and multiple classification hierarchies. Every classification hierarchy level must have an *identifying* attribute (constraint *{OID}*) and a *descriptor* attribute[4] (constraint *{D}*). These attributes are necessary for an automatic generation process into commercial OLAP tools, as these tools store this information in their metadata. The multiplicity *1* and *1..** defined in the target associated class role addresses the concepts of *strictness* and *non-strictness*, respectively. Strictness means that an object at a hierarchy's lower level belongs to only one higher-level object (e.g., as one month can be

[3] We refer the reader to [7] for a complete description of our approach.
[4] A descriptor attribute will be used as the default label in the data analysis.

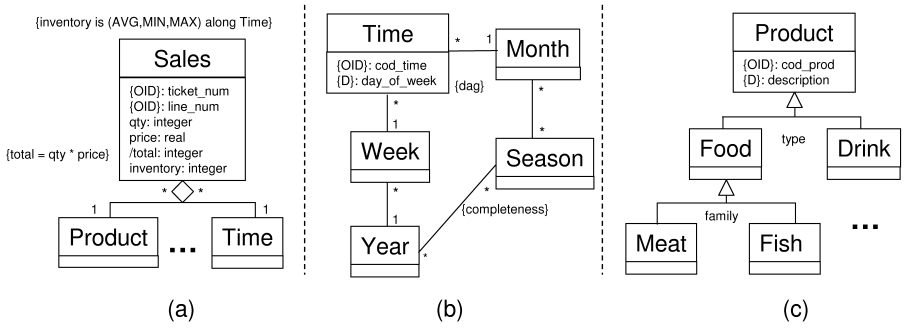

Fig. 1. Multidimensional modeling using UML

related to more than one season, the relationship between them is non-strict). Moreover, defining the *{completeness}* constraint in the target associated class role addresses the completeness of a classification hierarchy (see an example in Fig. 1 (b)). By completeness we mean that all members belong to one higher-class object and that object consists of those members only. For example, all the recorded seasons form a year, and all the seasons that form the year have been recorded. Our approach assumes all classification hierarchies are non-complete by default.

The *categorization of dimensions*, used to model additional features for a class's subtypes, is represented by means of generalization-specialization relationships. However, only the dimension class can belong to both a classification and specialization hierarchy at the same time. An example of categorization for the Product dimension is shown in Fig. 1 (c).

4 Package Design Guidelines for Multidimensional Modeling

In this section, based on our experience in real-world cases, we present our design guidelines for using UML packages in MD modeling following the approach presented in Section 3. We believe these guidelines are very close to the natural way that both designers and analyzers accomplish and understand MD modeling. Our experience indicates that these guidelines produce a very simple yet powerful design of MD models. We summarize all the design guidelines in Table 1.

`Guideline 0` 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 (Fig. 2 shows a summary of our proposal):

Level 1 : Model definition. A package represents a star schema of a conceptual MD model. A dependency between two packages at this level indicates that the star schemas share at least one dimension.

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 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" [13].

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
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
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 form another package, it is not necessary to import all the levels

Table 1. Multidimensional modeling guidelines

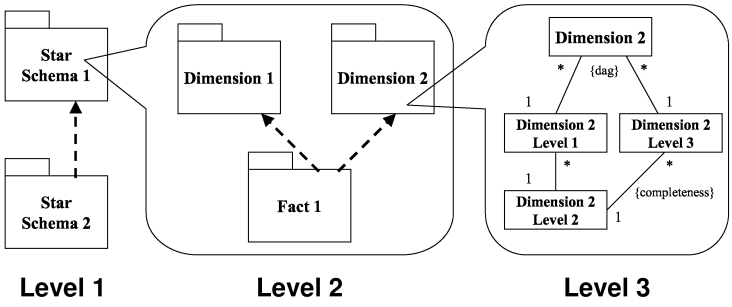

Fig. 2. The three levels of a MD model explosion using packages

5 Applying Package Design Guidelines: a Case Study

In this section, we use a case study to show how our guidelines[5] are properly applied to MD modeling. We use the *supply value chain* example taken from Chapter 5 of [15]. As Kimball states, "The supply side of the business consists of the steps needed to manufacture the products from original ingredients or parts...". Typical DWs that support the supply value chain include seven facts (Purchase Orders, Deliveries, Materials Inventory, Process Monitoring, Bill of Materials, Finished Goods Inventory, and Manufacturing Plans) and nine dimensions (Time, Ingredient, Supplier, Deal, Plant, Ship Mode, Process, Product, and Warehouse).

In Fig. 3, we show the supply value chain example modeled by our approach presented in [7]. The fact classes have been filled in a dark colour, while the dimension classes in a light colour, and the base classes of the dimension hierarchies in white. As seen in Fig. 3, the Time dimension is the only one connected to all the facts. For the sake of clearness, we have omitted the definition of the classification hierarchy levels for both the Process and Ship Mode dimensions. Moreover, we have not represented the attributes and methods of the classes either.

Even though we have tried to obtain a clear model, the model is confusing because there are a lot of lines that cross each other. Furthermore, it is difficult to see at a glance the different dimensions connected to a fact. In short, when the scale of a model is large and includes a large number of interconnections among its different elements, it may be very difficult to understand and manage it, especially for end users.

Guideline 1 and Guideline 2. Fig. 4 shows the first level of the model that is formed by seven packages that represent the different star schemas that form our

[5] For the sake of comprehensibility, we explicitly indicate when we apply each guideline.

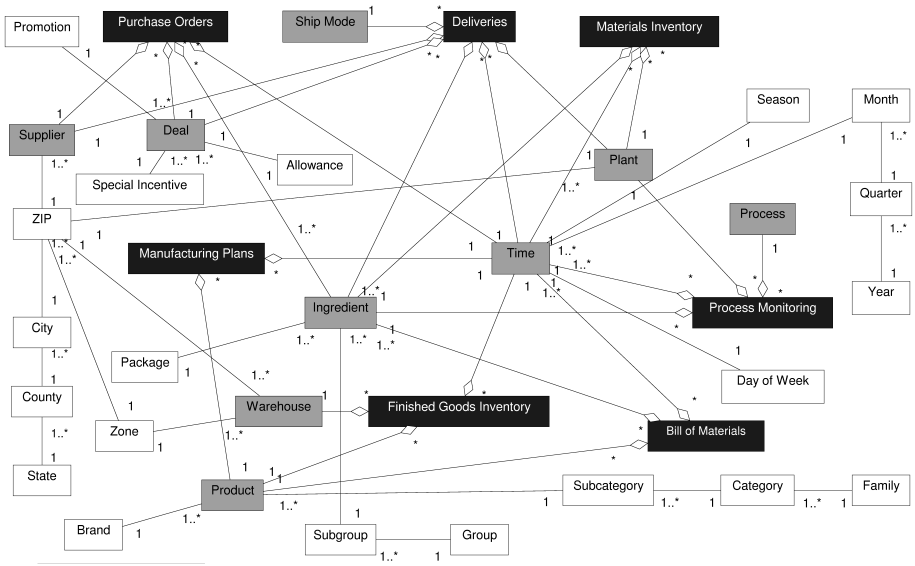

Fig. 3. A partial complex MD model

case study (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 star schema to define the dimensions, and then, the other schemas 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 better to group together the definition of the common dimensions in order to reduce the number of dependencies (G.2b).

Guideline 3 and Guideline 4. A package that represents a star schema is shown as a simple icon with names. The package contents can be dynamically accessed by "zooming" to a detailed view. For example, Fig. 5 shows the content of the package Purchase Orders Star (level 2). The fact package Purchase Orders Fact is represented in the middle of Fig. 5, while the dimension packages are placed around the fact package (G.3). As it can be seen, a dependency is drawn from the fact package to each one of the dimension packages, because the fact package comprises the whole definition of the star schema (G.4a). At level 2, it is possible to create a dependency from a fact package to a dimension package or between dimension packages, but we do not allow a dependency from a dimension package to a fact package, since it is not semantically correct in our proposal (G.4b).

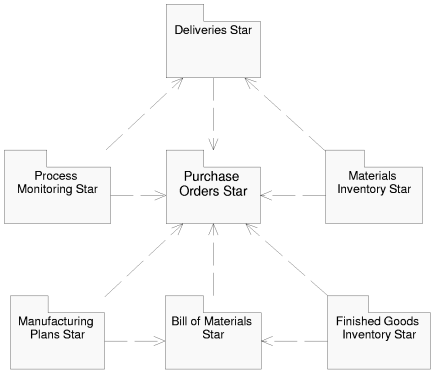

Fig. 4. Level 1: different star schemas of the supply value chain example

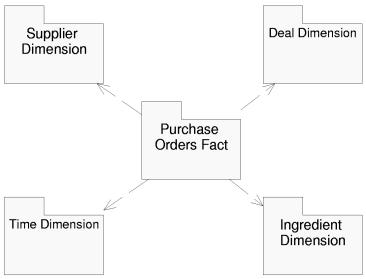

Fig. 5. Level 2: Purchase Orders Star

Guideline 5 and Guideline 6. Fig. 6 shows the content of the package Deliveries Star (level 2). As in the previous package, the fact package is placed in the middle of the figure and the dimension packages are placed around the fact package in a star fashion. The three dimension packages (Deal Dimension, Supplier Dimension, and Time Dimension) have been previously defined in the Purchase Orders Star (Fig. 5), so they are imported in this package[6] (G.5). Because of this, the name of the package where they have been previously defined appears below the package name (from Purchase Orders Star). Since Plant Dimension and Ship Mode Dimension have been defined in the current package, they do not show a package name. At this level, a dependency between dimension packages indicates that they share some hierarchy levels (G.6). For example, a dependency between Plant Dimension and Supplier Dimension is represented because there is a shared hierarchy[7] (ZIP, City, ...), as shown in Fig. 3.

In a similar way, Materials Inventory Star is further decomposed as shown in Fig. 7 (level 2). All the dimensions that this package contains have been previously defined in other packages. We can notice that it is possible to import packages defined in different star packages.

Guideline 7. The content of the dimension and fact packages is represented at level 3. The diagrams at this level are only comprised of classes and associations among them. For example, Fig. 8 shows the content of the package Supplier

[6] However, our approach does not forbid defining the same dimension twice but with different names defined by views. For example, the designer can define two dimensions, such as Shipment Date and Reception Date with the same structure instead of defining only one Date dimension.

[7] 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.

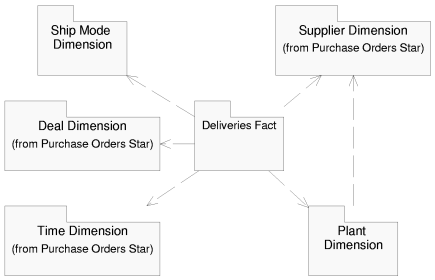

Fig. 6. Level 2: Deliveries Star

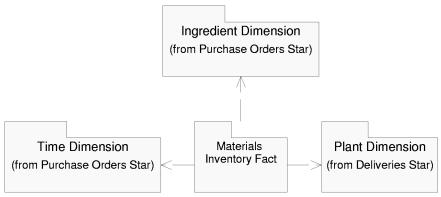

Fig. 7. Level 2: Materials Inventory Star

Dimension (level 3), that contains the definition of the dimension (Supplier) and the different hierarchy levels (ZIP, City, County, and State) (G.7). The hierarchy of a dimension defines how the different OLAP operations (roll up, drill down, etc.) can be applied [15].

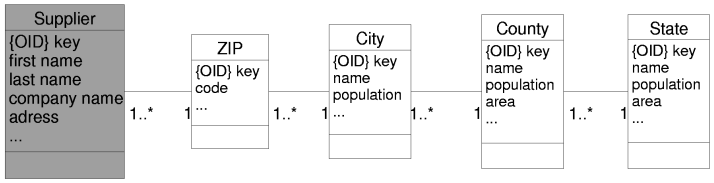

Fig. 8. Level 3: Supplier Dimension

Guideline 8. Regarding fact packages, Fig. 9 shows the content of the package Purchase Orders Fact (level 3). In this package, the whole star schema is displayed: the fact class (filled in a dark colour) is defined (we show some of its attributes) and the dimensions with their corresponding hierarchy levels are imported (G.8). To avoid unnecessary detail, we have hidden the attributes and methods of dimensions and hierarchy levels.

Guideline 9 and Guideline 10. Fig. 10 shows the content of the package Plant Dimension (level 3). This dimension shares some hierarchy levels with Supplier Dimension (Fig. 8). Therefore, we notice that the shared hierarchy levels have been imported (the name of the package where they have been defined appears below the class name) (G.9). Furthermore, we also notice a salient feature of our approach: two dimensions, that share hierarchy levels, do not need to share the

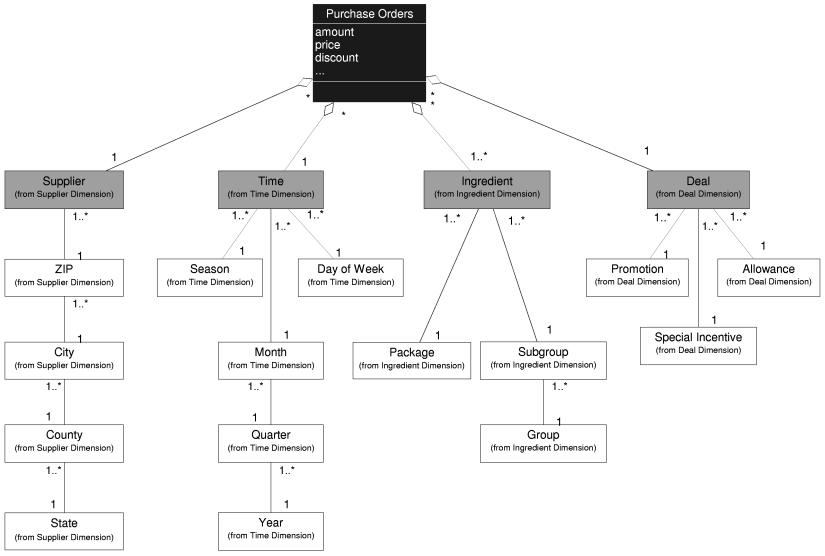

Fig. 9. Level 3: Purchase Orders Fact

whole hierarchy (`G.10`)[8]. For example, the hierarchy of Plant Dimension does not include State level defined in Supplier Dimension.

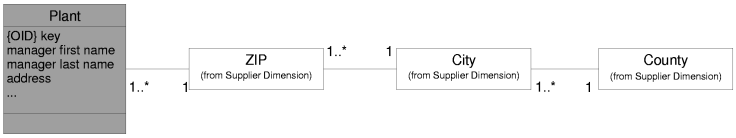

Fig. 10. Level 3: Plant Dimension

6 Definition of Package Stereotypes

In Section 4 and 5, we have shown how the complexity of a MD model is managed by organizing it into logical packages. We have used three different kinds of packages: star package, dimension package, and fact package. In this section, we provide a UML extension by means of stereotypes of the particular packages we have defined.

[8] Other proposals, such as [1][2], when sharing dimension hierarchy levels, share all the hierarchy path from the shared level. However, the package mechanism allows us to import only the required levels, thereby providing a higher level of flexibility.

For the definition of stereotypes, we follow the examples included in the UML Specification [6]:

- 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 for the stereotype.
- Constraints: A list of constraints defined by OCL expressions associated with the stereotype, with an informal explanation of the expressions.
- Tagged values: A list of all tagged values that may be associated with the stereotype.

We have defined three stereotypes that specialize the Package model element:

- Name: StarPackage
- Base class: Package
- Description: Packages of this stereotype represent MD star schemas
- Icon: Fig. 11 (a)
- Constraints:
  - A StarPackage can only contain FactPackages or DimensionPackages:[9]
    self.contents->forAll(oclIsTypeOf(FactPackage) or
    oclIsTypeOf(DimensionPackage))
  - A StarPackage can only contain one FactPackage:
    self.contents->select(oclIsTypeOf(FactPackage))->size <= 1
  - There are no cycles in the dependency structure:[10]
    not self.allSuppliers->includes(self)
- Tagged values: None

- Name: DimensionPackage
- Base class: Package
- Description: Packages of this stereotype represent MD dimensions
- Icon: Fig. 11 (b)
- Constraints:
  - It is not possible to create a dependency from a DimensionPackage to a FactPackage (only to a DimensionPackage):
    self.clientDependency->forAll(supplier->forAll(oclIsTypeOf(DimensionPackage)))
  - There are no cycles in the dependency structure:
    not self.allSuppliers->includes(self)
- Tagged values: None

---

[9] contents is an additional operation defined in the UML Specification [6]: "The operation contents results in a Set containing the ModelElements owned by or imported by the Package".

[10] allSuppliers is an additional operation defined in the UML Specification [6]: "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".

- Name: FactPackage
- Base class: Package
- Description: Packages of this stereotype represent MD facts
- Icon: Fig. 11 (c)
- Constraints:
  - There are no cycles in the dependency structure:
    not self.allSuppliers->includes(self)
- Tagged values: None

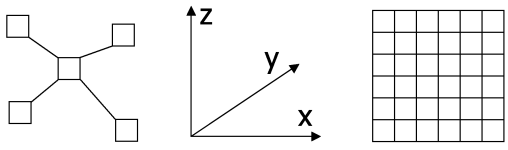

Fig. 11. Stereotype icons of the MD extension

7 Using Multidimensional Modeling in Rational Rose

Rational Rose (RR) is one of the most well-known visual modeling tools. As RR supports the UML, it is becoming the common modeling tool for OO modeling. RR is extensible by means of add-ins, that allows us to group together customizations and automation of several RR features through the Rose Extensibility Interface (REI) [16] into one component. An add-in allows us to customize main menu items, data types, stereotypes, etc. In this section, we present an add-in we have developed, that allows us to use the stereotypes we have previously presented in RR. Our add-in customizes the following elements:

- Menu item: We have added the new menu item MD Validate in the menu Tools. This menu item runs a Rose script that validates a MD model: our script checks all the constraints we have presented in Section 6.
- Stereotypes: We have defined the stereotypes we have previously presented in Section 6.

We use our stereotypes in RR by means of a stereotype configuration file. To graphically distinguish model elements from different stereotypes, each stereotype can have a graphical representation. Thus, for each stereotype, there may be four different icons: a diagram icon, a small diagram toolbar icon, a large diagram toolbar icon, and a list view icon.

The best way to understand our extension is to show a tangible example. Fig. 12 shows the level 1 of the supply value chain example (the same level is

displayed in Fig. 4 without stereotypes). We can notice the list view icons of our stereotypes in the list of the browser (left hand panel in Fig. 12). Besides, the vertical toolbar in the middle of Fig. 12 contains three new buttons that correspond to our stereotypes.

Furthermore, we also show how the level 2 of a MD model is displayed in RR. In Fig. 13, the content of Purchase Orders Star is shown (the same level is displayed in Fig. 5 without stereotypes). The icons for the FactPackage and DimensionPackage can be observed.

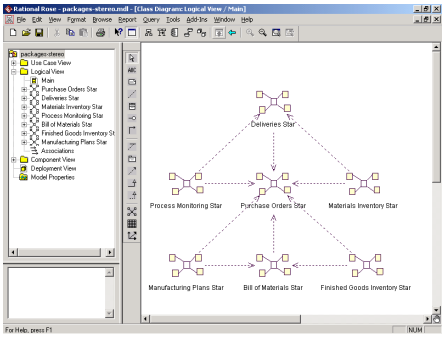

Fig. 12. Multidimensional modeling using Rational Rose (level 1)

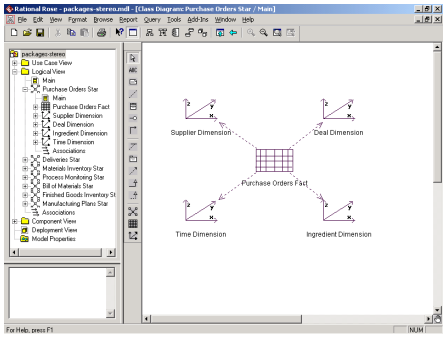

Fig. 13. Multidimensional modeling using Rational Rose (level 2)

8 Conclusions and Future Work

Existing multidimensional (MD) modeling approaches use "flat design" in that all the modeling elements are represented in the same diagram. These approaches are not often suitable for huge and complex MD models. Therefore, in this paper, we have presented how UML package mechanisms can be successfully used for MD modeling at three levels of complexity. We have also provided design guidelines, based on our experience in designing real cases, that allow the correct use of the UML packages for simplifying a conceptual design when modeling large and complex data warehouses. We have also illustrated the benefits of these guidelines by applying them to a case study. These guidelines are extremely useful as they allow us to obtain conceptual MD models that can be understood by both designers and analyzers, facilitating the communication between them. Furthermore, we have also provided a UML extension by means of stereotypes of the different package items we use. Finally, to show the applicability of our proposal, this UML extension has been defined for a well-known modeling tool such as Rational Rose 2000, which allows us to put in practice all ideas developed throughout the paper.

Future works are concerned with providing a UML extension including stereotypes for main structural properties of MD modeling (fact class, dimension class, etc.). Further future work refers to extending our approach to allow us to cover all life cycle of MD systems, which involves implementation of MD models into OO and object-relational databases.

References

1. Golfarelli, M., Rizzi, S.: A methodological Framework for Data Warehouse Design. In: Proc. of the ACM 1st Intl. Workshop on Data warehousing and OLAP (DOLAP'98), Washington D.C., USA (1998) 3–9
2. Sapia, C., Blaschka, M., Höfling, G., Dinter, B.: 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., Springer-Verlag (1998) 105–116
3. Tryfona, N., Busborg, F., Christiansen, J.: starER: A Conceptual Model for Data Warehouse Design. In: Proc. of the ACM 2nd Intl. Workshop on Data warehousing and OLAP (DOLAP'99), Kansas City, Missouri, USA (1999)
4. Husemann, B., Lechtenborger, J., Vossen, G.: Conceptual Data Warehouse Design. In: Proc. of the 2nd. Intl. Workshop on Design and Management of Data Warehouses (DMDW'2000), Stockholm, Sweden (2000) 3–9
5. Abelló, A., Samos, J., Saltor, F.: 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), Munich, Germany (2001) 668–677
6. Object Management Group (OMG): Unified Modeling Language Specification 1.4. Internet: http://www.omg.org/cgi-bin/doc?formal/01-09-67 (2001)
7. Trujillo, J., Palomar, M., Gómez, J., Song, I.: Designing Data Warehouses with OO Conceptual Models. IEEE Computer, special issue on Data Warehouses **34** (2001) 66–75
8. Feldman, P., Miller, D.: Entity Model Clustering: Structuring a Data Model by Abstraction. The Computer Journal **29** (1986) 348–360
9. Teorey, T., Wei, G., Bolton, D., Koenig, J.: ER Model Clustering as an Aid for User Communication and Documentation in Database Design. Communications of ACM **32** (1989) 975–987
10. Gandhi, M., Robertson, E., Gucht, D.V.: Leveled Entity Relationship Model. In: Proc. of the 13th Intl. Conference on Entity-Relationship Approach (ER'94). Volume 881 of LNCS., Springer-Verlag (1994) 420–436
11. Siau, K., Cao, Q.: Unified Modeling Language (UML) - A Complexity Analysis. Journal of Database Management **12** (2001) 26–34
12. Fowler, M.: UML Distilled. Applying the Standard Object Modeling Language. Object Technology Series. Addison-Wesley (1998)
13. Conallen, J.: Building Web Applications with UML. Object Technology Series. Addison-Wesley (2000)
14. Giovinazzo, W.: Object-Oriented Data Warehouse Design. Building a star schema. Prentice-Hall, New Jersey, USA (2000)
15. Kimball, R.: The Data Warehouse Toolkit. 2 edn. John Wiley & Sons (1996)
16. Rational Software Corporation: Using the Rose Extensibility Interface. Rational Software Corporation (2001)