Data Mapping Diagrams for Data Warehouse Design with UML

Sergio Luján-Mora[1], Panos Vassiliadis[2], and Juan Trujillo[1]

[1] Dept. of Software and Computing Systems
University of Alicante (Spain)
{slujan,jtrujillo}@dlsi.ua.es
[2] Dept. of Computer Science
University of Ioannina (Hellas)
pvassil@cs.uoi.gr

Abstract. In Data Warehouse (DW) scenarios, ETL (Extraction, Transformation, Loading) processes are responsible for the extraction of data from heterogeneous operational data sources, their transformation (conversion, cleaning, normalization, etc.) and their loading into the DW. In this paper, we present a framework for the design of the DW back-stage (and the respective ETL processes) based on the key observation that this task fundamentally involves dealing with the specificities of information at very low levels of granularity including transformation rules at the attribute level. Specifically, we present a disciplined framework for the modeling of the relationships between sources and targets in different levels of granularity (including coarse mappings at the database and table levels to detailed inter-attribute mappings at the attribute level). In order to accomplish this goal, we extend UML (Unified Modeling Language) to model attributes as first-class citizens. In our attempt to provide complementary views of the design artifacts in different levels of detail, our framework is based on a principled approach in the usage of UML packages, to allow zooming in and out the design of a scenario.

Keywords: data mapping, ETL, data warehouse, UML

1 Introduction

In Data Warehouse (DW) scenarios, ETL (Extraction, Transformation, Loading) processes are responsible for the extraction of data from heterogeneous operational data sources, their transformation (conversion, cleaning, normalization, etc.) and their loading into the DW. DWs are usually populated with data from different and heterogeneous operational data sources such as legacy systems, relational databases, COBOL files, Internet (XML, web logs) and so on. It is well recognized that the design and maintenance of these ETL processes (also called DW back stage) is a key factor of success in DW projects for several reasons, the most prominent of which is their critical mass; in fact, ETL development can take up as much as 80% of the development time in a DW project [1,2].

Despite the importance of designing the mapping of the data sources to the DW structures along with any necessary constraints and transformations, unfortunately, there are few models that can be used by the designers to this end. The front end of the DW has monopolized the research on the conceptual part of DW modeling, while few attempts have been made towards the conceptual modeling of the back stage [3,4]. Still, to this day, there is no model that can combine (a) the desired detail of modeling data integration at the attribute level and (b) a widely accepted modeling formalism such as the ER model or UML. One particular reason for this, is that both these formalisms are simply not designed for this task; on the contrary, they treat attributes as second-class, weak entities, with a descriptive role. Of particular importance is the problem that in both models attributes cannot serve as an end in an association or any other relationship.

One might argue that the current way of modeling is sufficient and there is no real need to extend it in order to capture mappings and transformations at the attribute level. There are certain reasons that we can list against this argument:

- The design artifacts are acting as blueprints for the subsequent stages of the DW project. If the important details of this design (e.g., attribute interrelationships) are not documented, the blueprint is problematic. Actually, one of the current issues in DW research involves the efficient documentation of the overall process. Since design artifacts are means of communicating ideas, it is best if the formalism adopted is a widely used one (e.g., UML or ER).
- The design should reflect the architecture of the system in a way that is formal, consistent and allows the what-if analysis of subsequent changes. Capturing attributes and their interrelations as *first-class modeling elements* (FCME, also known as first-class citizens) improves the design significantly with respect to all these goals. At the same time, the way this issue is handled now would involve a naive, informal documentation through UML notes.
- In previous lines of research [5], we have shown that by modeling attribute interrelationships, we can treat the design artifact as a graph and actually measure the aforementioned design goals. Again, this would be impossible with the current modeling formalisms.

To address all the aforementioned issues, in this paper, we present an approach that enables the tracing of the DW back-stage (ETL processes) particularities at various levels of detail, through a widely adopted formalism (UML). This is enabled by an additional view of a DW, called the *data mapping diagram*. In this new diagram, we treat attributes as FCME of the model. This gives us the flexibility of defining models at various levels of detail. Naturally, since UML is not initially prepared to support this behavior, we solve this problem thanks to the extension mechanisms that it provides. Specifically, we employ a formal, strict mechanism that maps attributes to proxy classes that represent them. Once mapped to classes, attributes can participate in associations that determine the inter-attribute mappings, along with any necessary transformations and constraints. We adopt UML as our modeling language due to its wide acceptance and the possibility of using various complementary diagrams for modeling

different system aspects. Actually, from our point of view, one of the main advantages of the approach presented in this paper is that it is totally integrated in a global approach that allows us to accomplish the conceptual, logical and the corresponding physical design of all DW components by using the same notation ([6,7,8]).

The rest of the paper is structured as follows. In Section 2, we briefly describe the general framework for our DW design approach and introduce a motivating example that will be followed throughout the paper. In Section 3, we show how attributes can be represented as FCME in UML. In Section 4, we present our approach to model data mappings in ETL processes at the attribute level. In Section 5, we review related work and finally, in Section 6 we present the main conclusions and future work.

2 Framework and Motivation

In this section we discuss our general assumptions around the DW environment to be modelled and briefly give the main terminology. Moreover, we define a motivating example that we will consistently follow through the rest of the paper.

The architecture of a DW is usually depicted as various layers of data in which data from one layer is derived from data of the previous layer [9]. Following this consideration, we consider that the development of a DW can be structured into an integrated framework with five stages and three levels that define different diagrams for the DW model, as explained in Table 1.

- **Phases**: we distinguish five stages in the definition of a DW:
  - Source: it defines the data sources of the DW, such as OLTP systems, external data sources (syndicated data, census data), etc.
  - Integration: it defines the mapping between the data sources and the DW.
  - Data Warehouse: it defines the structure of the DW.
  - Customization: it defines the mapping between the DW and the clients' structures.
  - Client: it defines special structures that are used by the clients to access the DW, such as data marts or OLAP applications.
- **Levels**: each stage can be analyzed at three levels or perspectives:
  - Conceptual: it defines the DW from a conceptual point of view.
  - Logical: it addresses logical aspects of the DW design, such as the definition of the ETL processes.
  - Physical: it defines physical aspects of the DW, such as the storage of the logical structures in different disks, or the configuration of the database servers that support the DW.
- **Diagrams**: each stage or level require different modeling formalisms. Therefore, our approach is composed of 15 diagrams, but the DW designer does not need to define all the diagrams in each DW project. In our approach, we use UML [10] as the modeling language, because its expressive power is sufficient for the modeling of all the diagrams of the framework. But as UML is a general modeling language, we need to use the UML extension mechanisms to adapt UML to specific domains.

Table 1. Data warehouse development framework

In previous works, we have presented some of the diagrams (and the corresponding profiles), such as the *Multidimensional Profile* [6,7] and the *ETL Profile* [4]. In this paper, we introduce the *Data Mapping Profile*.

To motivate our discussion we will introduce a running example where the designer wants to build a DW from the retail system of a company. Naturally, we consider only a small part of the DW, where the target fact table has to contain only the quarterly sales of the products belonging to the computer category, whereas the rest of the products are discarded.

In Fig. 1, we zoom-in the definition of the SCS (Source Conceptual Schema), which represents the sources that feed the DW with data. In this example, the data source is composed of four entities represented as UML classes: Cities, Customers, Orders, and Products. The meaning of the classes and their attributes, as depicted in Fig. 1 is straightforward. The "..." shown in this figure simply indicates that other attributes of these classes exist, but they are not displayed for the sake of simplicity (this use of "..." is not a UML notation).

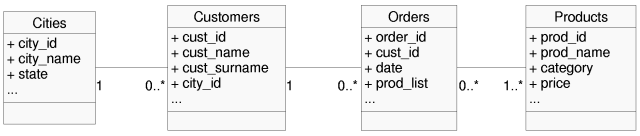

Fig. 1. Source Conceptual Schema (SCS)

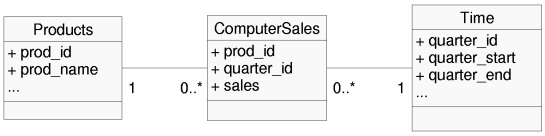

Fig. 2. Data Warehouse Conceptual Schema (DWCS)

Finally, the DWCS (Data Warehouse Conceptual Schema) of our motivating example is shown in Fig. 2. The DW is composed of one fact (ComputerSales) and two dimensions (Products and Time).

In this paper, we present an additional view of a DW, called the *Data Mapping* that shows the relationships between the data sources and the DW and between the DW and the clients' structures. In this new diagram, we need to treat attributes as FCME of the models, since we need to depict their relationships at attribute level. Therefore, we also propose a UML extension to accomplish

this goal in this paper. To the best of our knowledge, this is the first proposal of representing attributes as FCME in UML diagrams.

3 Attributes as First-Class Modeling Elements in UML

Both in the Entity-Relationship (ER) model and in UML, attributes are embedded in the definition of their comprising "element" (an entity in the ER or a class in UML), and it is not possible to create a relationship between two attributes. As we have already explained in the introduction, in some situations (e.g., data integration, constraints over attributes, etc.) it is desirable to represent attributes as FCME. Therefore, in this section we will present an extension of UML to accommodate attributes as FCME. We have chosen UML instead of ER on the grounds of its higher flexibility in terms of employing complementary diagrams for the design of a certain system.

Throughout this paper, we frequently use the term *first-class modeling elements* or *first-class citizens* for elements of our modeling languages. Conceptually, FCME refer to fundamental modeling concepts, on the basis of which our models are built. Technically, FCME involve an identity of their own, and they are possibly governed by integrity constraints (e.g., relationships must have at least two ends refering to classes.). In a UML class diagram, two kinds of modeling elements are treated as FCME. Classes, as abstract representations of real-world entities are naturally found in the center of the modeling effort. Being FCME, classes stand-alone entities also acting as attribute containers. The relationships between classes are captured by associations. Associations can be also FCME, called association classes. Even though an association class is drawn as an association and a class, it is really just a single model element [10]. An association class can contain attributes or can be connected to other classes. However, the same is not possible with attributes.

Naturally, in order to allow attributes to play the same role in certain cases, we propose the representation of attributes as FCME in UML. In our approach, classes and attributes are defined as normally in UML. However, in those cases where it is necessary to treat attributes as FCME, classes are imported to the *attribute/class diagram*, where attributes are automatically represented as classes; in this way, the user only has to define the classes and the attributes once. In the importing process from the class diagram to the attribute/class diagram, we refer to the class that contains the attributes as the *container class* and to the class that represents an attribute as the *attribute class*. In Table 2, we formally define attribute/class diagrams, along with the new stereotypes, ≪Attribute≫ and ≪Contain≫.

4 The Data Mapping Diagram

Once we have introduced the extension mechanism that enables UML to treat attributes as FCME, we can proceed in defining a framework on its usage. In

Definition 1 *Attribute classes are materializations of the ≪Attribute≫ stereotype, introduced specifically for representing the attributes of a class.* The following constraints apply for the correct definition of an attribute class as a materialization of an ≪Attribute≫ stereotype:
- *Naming convention*: the name of the attribute class is the name of the corresponding container class, followed by a dot and the name of the attribute.
- *No features*: an attribute class can contain neither attributes nor methods.
- *Tag definitions*: an attribute class contains the following tag definitions that represent the properties of an attribute model element (according to the UML Specification [10]): changeability, initialValue, multiplicty, ordering, ownerScope, property-string, stereotype, type, and visibility.

Definition 2 *A contain relationship is a composite aggregation between a container class and its corresponding attribute classes, originated at the end near the container class and highlighted with the ≪Contain≫ stereotype.*

Definition 3 *An attribute/class diagram is a regular UML class diagram extended with ≪Attribute≫ classes and ≪Contain≫ relationships.*

Table 2. Definitions

this section, we will introduce the *data mapping diagram*, which is a new kind of diagram, particularly customized for the tracing of the data flow, at various degrees of detail, in a DW environment. Data mapping diagrams are complementary to the typical class and interaction diagrams of UML and focus on the particularities of the data flow and the interconnections of the involved data stores. A special characteristic of data mapping diagrams is that a certain DW scenario is practically described by a set of complementary data mapping diagrams, each defined at a different level of detail. In this section, we will introduce a principled approach to deal with such complementary data mapping diagrams.

To capture the interconnections between design elements, in terms of data, we employ the notion of *mapping*. Broadly speaking, when two design elements (e.g., two tables, or two attributes) share the same piece of information, possibly through some kind of filtering or transformation, this constitutes a semantic relationship between them. In the DW context, this relationship, involves three logical parties: (a) the *provider* entity (schema, table, or attribute), responsible for generating the data to be further propagated, (b) the *consumer*, that receives the data from the provider and (c) their intermediate *matching* that involves the way the mapping is done, along with any transformation and filtering.

Since a data mapping diagram can be very complex, our approach offers the possibility to organize it in different levels thanks to the use of UML packages. Our layered proposal consists of four levels (see Fig. 3), as it is explained in Table 3.

At the leftmost part of Fig. 3, a simple relationship among the DWCS and the SCS exists: this is captured by a single Data Mapping package and these three design elements constitute the data mapping diagram of the database level (or Level 0). Assuming that there are three particular tables in the DW that we would like to populate, this particular Data Mapping package abstracts the fact that there are three main scenarios for the population of the DW, one for each of this tables. In the dataflow level (or Level 1) of our framework, the data

Database Level (or **Level 0**). In this level, each schema of the DW environment (e.g., data sources at the conceptual level in the SCS, conceptual schema of the DW in the DWCS, etc.) is represented as a package [8]. The mappings among the different schemata are modeled in a single mapping package, encapsulating all the lower-level mappings among different schemata.

Dataflow Level (or **Level 1**). This level describes the data relationship among the individual source tables of the involved schemata towards the respective targets in the DW. Practically, a data mapping diagram at the database level is zoomed-in to a set of more detailed data mapping diagrams, each capturing how a target table is related to source tables in terms of data.

Table Level (or **Level 2**).Whereas the mapping diagram of the dataflow level describes the data relationships among sources and targets using a single package, the data mapping diagram at the table level, details all the intermediate transformations and checks that take place during this flow. Practically, if a data mapping is simple, a single package that represents the data mapping can be used at this level; otherwise, a set of packages is used to segment complex data mappings in sequential steps.

Attribute Level (or **Level 3**). In this level, the data mapping diagram involves the capturing of inter-attribute mappings. Practically, this means that the diagram of the table is zoomed-in and the mapping of provider to consumer attributes is traced, along with any intermediate transformation and cleaning. As we shall describe later, we provide two variants for this level.

Table 3. Data mapping levels

relationships among the sources and the targets in the context of each of the scenarios, is practically modeled by the respective package. If we zoom in one of these scenarios, e.g., Mapping 1, we can observe its particularities in terms of data transformation and cleaning: the data of Source 1 are transformed in two steps (i.e., they have undergone two different transformations), as shown in Fig. 3. Observe also that there is an Intermediate data store employed, to hold the output of the first transformation (Step 1), before passed on to the second one (Step 2). Finally, at the right lower part of Fig. 3, the way the attributes are mapped to each other for the data stores Source 1 and Intermediate is depicted. Let us point out that in case we are modeling a complex and huge DW, the attribute transformation modelled at level 3 is hidden within a package definition, thereby avoiding the use of cluttered diagrams.

The constructs that we employ for the data mapping diagrams at different levels are as follows:

- The database and dataflow diagrams (Levels 0 and 1) use traditional UML structures for their purpose. Specifically, in these diagrams we employ (a) packages for the modeling of data relationships and (b) simple dependencies among the involved entities. The dependencies state that the mapping packages are dependent upon the changes of the employed data stores.
- The table level (Level 2) diagram extends UML with three stereotypes: (a) ≪Mapping≫, used as a package that encapsulates the data interrelationships among data stores, (b) ≪Input≫ and ≪Output≫ which explain the roles of providers and consumers for the ≪Mapping≫.
- The diagram at the attribute level (Level 3) is also using several newly introduced stereotypes, namely ≪Map≫, ≪MapObj≫, ≪Domain≫, ≪Range≫,

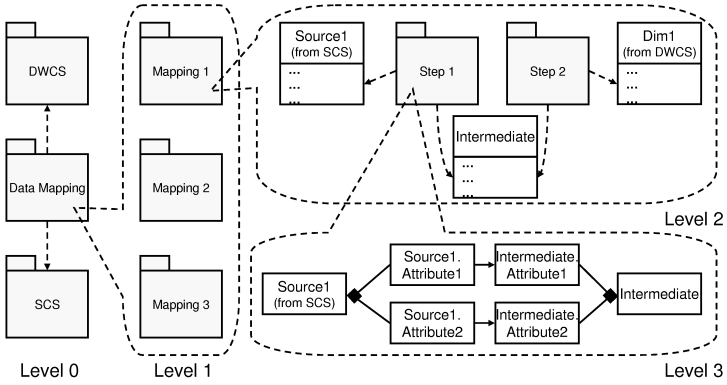

Fig. 3. Data mapping levels

≪Input≫, ≪Output≫, and ≪Intermediate≫ for the definition of data mappings.

We will detail the stereotypes of the table level in the next section and defer the discussion for the stereotypes of the attribute level to subsection 4.2.

4.1 The Data Mapping Diagram at the Table Level

During the integration process from data sources into the DW, source data may undergo a series of transformations, which may vary from simple algebraic operations or aggregations to complex procedures. In our approach, the designer can segment a long and complex transformation process into simple and small parts represented by means of UML packages that are materialization of a ≪Mapping≫ stereotype and contain an attribute/class diagram. Moreover, ≪Mapping≫ packages are linked by ≪Input≫ and ≪Output≫ dependencies that represent the flow of data. During this process, the designer can create *intermediate classes*, represented by the ≪Intermediate≫ stereotype, in order to simplify or clarify the models. These classes represent intermediate storage that may or may not exist actually, but they help to understand the mappings.

In Fig. 4, a schematic representation of a data mapping diagram at the table level is shown. This level specifies data sources and target sources, to which these data are directed. At this level, the classes are represented as usually in UML with the attributes depicted inside the container class. Since all the classes are imported from other packages, the legend (from ...) appears below the name of each class. The mapping diagram is shown as a package decorated with the ≪Mapping≫ stereotype and hides the complexity of the mapping, because a vast number of attributes can be involved in a data mapping. This package presents two kinds of stereotyped dependencies: ≪Input≫ to the data providers (i.e., the data sources) and ≪Output≫ to the data consumers (i.e., the tables of the DW).

4.2 The Data Mapping Diagram at the Attribute Level

As already mentioned, in the attribute level, the diagram includes the relationships between the attributes of the classes involved in a data mapping. At this level, we offer two design variants:

- Compact variant: the relationship between the attributes is represented as an *association*, and the semantic of the mapping is described in a UML *note* attached to the target attribute of the mapping.
- Formal variant: the relationship between the attributes is represented by means of a *mapping object*, and the semantic of the mapping is described in a *tag definition* of the mapping object.

With the first variant, the data mapping diagrams are less cluttered, with less modeling elements, but the data mapping semantics are expressed as UML notes that are simple comments that have no semantic impact. On the other hand, the size of the data mapping diagrams obtained with the second variant is larger, with more modeling elements and relationships, but the semantics are better defined as tag definitions. Due to the lack of space, in this paper we will only focus on the compact variant. In this variant, the relationship between the attributes is represented as an association decorated with the stereotype ≪Map≫, and the semantic of the mapping is described in a UML note attached to the target attribute of the mapping.

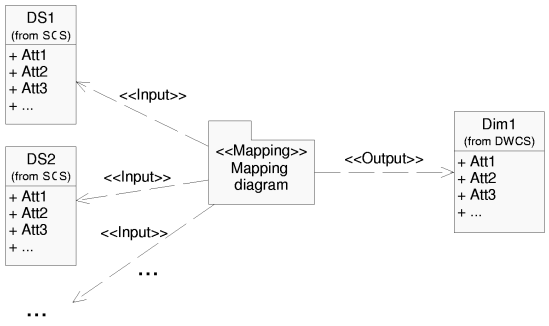

Fig. 4. Level 2 of a data mapping diagram

The content of the package Mapping diagram from Fig. 4 is defined in the following way (recall that Mapping diagram is a ≪Mapping≫ package that contains an attribute/class diagram):

- The classes DS1, DS2, ..., and Dim1 are imported in Mapping diagram.
- The attributes of these classes are suppressed because they are shown as ≪Attribute≫ classes in this package.

– The ≪Attribute≫ classes are connected by means of association relationships and we use the navigability property to specify the flow of data from the data sources to the DW.
– The association relationships are adorned with the stereotype ≪Map≫ to highlight the meaning of this relationship.
– A UML note can be attached to each one of the target attributes to specify how the target attribute is obtained from the source attributes. The language for the expression is a choice of the designer (e.g., a LAV vs. a GAV approach [11] can be equally followed).

4.3 Motivating Example Revisited

From the DW example shown in Fig.s 1 and 2, we define the corresponding data mapping diagram shown in Fig. 5. The goal of this data mapping is to calculate the quarterly sales of the products belonging to the computer category. The result of this transformation is stored in ComputerSales from the DWCS. The transformation process has been segmented in three parts: Dividing, Filtering, and Aggregating; moreover, DividedOrders and FilteredOrders, two ≪Intermediate≫ classes, have been defined.

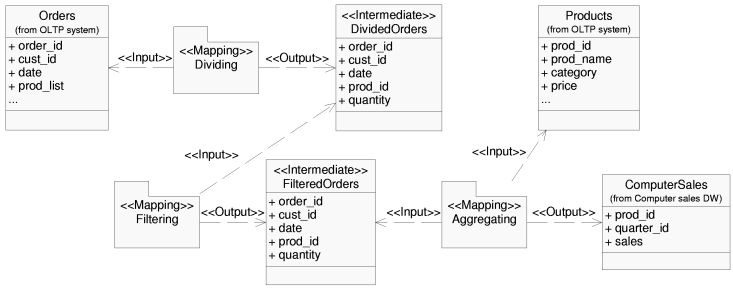

Fig. 5. Level 2 of a data mapping diagram

Following with the data mapping example shown in Fig. 5, attribute prod_list from Orders table contains the list of ordered products with product ID and (parenthesized) quantity for each. Therefore, Dividing splits each input order according to its prod_list into multiple orders, each with a single ordered product (prod_id) and quantity (quantity), as shown in Fig. 6. Note that in a data mapping diagram the designer does not specify the processes, but only the data relationships. We use the one-to-many cardinality in the association relationships between Orders.prod_list and DividedOrders.prod_id and DividedOrders.quantity to indicate that one input order produces multiple output orders. We do not attach any note in this diagram because the data are not transformed, so the mapping is direct.

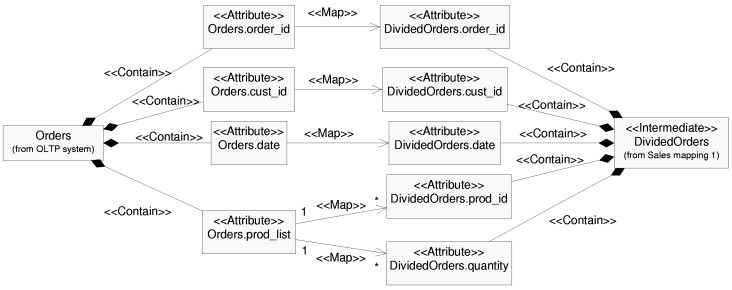

Fig. 6. Dividing Mapping

Filtering (Fig. 7) filters out products not belonging to the computer category. We indicate this action with a UML note attached to the prod_id mapping, because it is supposed that this attribute is going to be used in the filtering process.

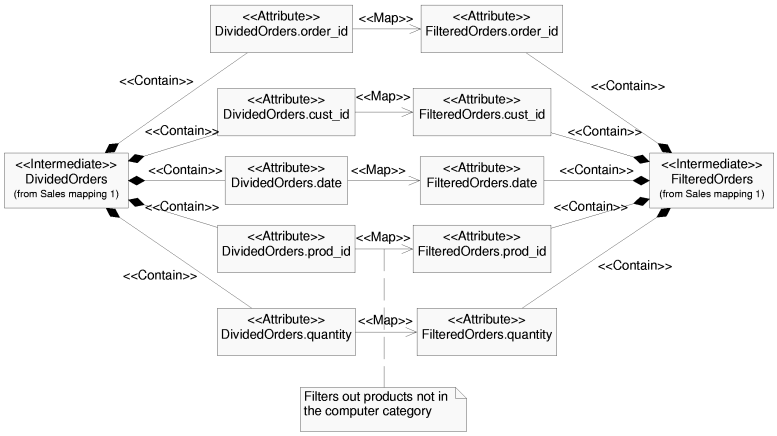

Fig. 7. Filtering Mapping

Finally, Aggregating (Fig. 8) computes the quarterly sales for each product. We use the many-to-one cardinality to indicate that many input items are needed to calculate a single output item. Moreover, a UML note indicates how the ComputerSales.sales attribute is calculated from FilteredOrders.quantity and Products.price. The cardinality of the association relationship between Products.price and ComputerSales.sales is one-to-many because the same price is used in different quarters, but to calculate the total sales of a particular product in

a quarter we only need one price (we consider that the price of a product never changes along time).

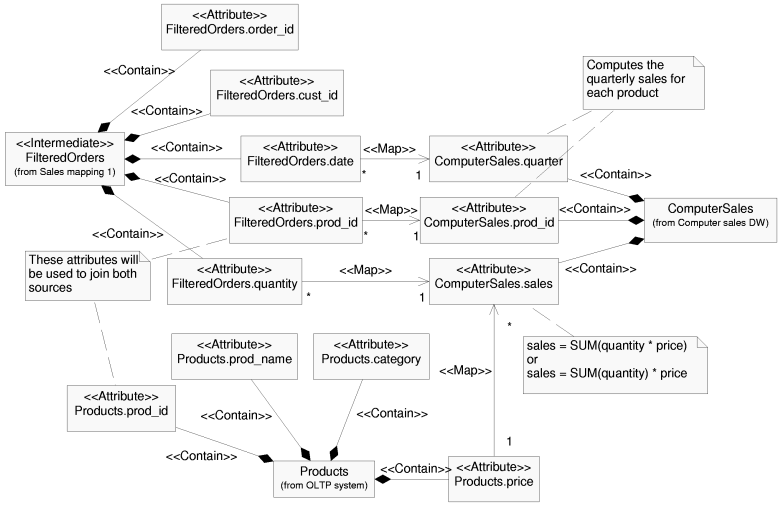

Fig. 8. Aggregating Mapping

5 Related Work

There is a relatively small body of research efforts around the issue of conceptual modeling of the DW back-stage.

In [12,13], the *model management*, a framework for supporting meta-data related applications where models and mappings are manipulated is proposed. In [13], two scenarios related to loading DWs are presented as case studies: on the one hand, the mapping between the data sources and the DW, on the other hand, the mapping between the DW and a data mart. In this approach, a mapping is a model that relates the objects (attributes) of two other models; each object in a mapping is called a *mapping object* and has three properties: *domain* and *range*, which point to objects in the source and the target respectively, and *expr*, which is an expression that defines the semantics of that mapping object. This is an isolated approach in which authors propose their own graphical notation for representing data mappings. Therefore, from our point of view, there is a lack of integration with the design of other parts of a DW.

In [3] the authors attempt to provide a first model towards the conceptual modeling of the DW back-stage. The notion of provider mapping among attributes is introduced. In order to avoid the problems caused by the specific

nature of ER and UML, the authors adopt a generic approach. The static conceptual model of [3] is complemented in [5] with the logical design of ETL processes as data-centric workflows. ETL processes are modeled as graphs composed of activities that include attributes as FCME. Moreover, different kinds of relationships capture the data flow between the sources and the targets.

Regarding data mapping, in [14] authors discuss issues related to the data mapping in the integration of data. A set of mapping operators is introduced and a classification of possible mapping cases is presented. However, no graphical representation of data mapping scenarios is provided, thereby making difficult using it in real world projects.

The issue of treating attributes as FCME has generated several debates from the beginning of the conceptual modeling field [15]. More recently, some object-oriented modeling approaches such as OSM (Object Oriented System Model) [16] or ORM (Object Role Modeling) [17] reject the use of attributes (*attribute-free models*) mainly because of their inherent instability. In these approaches, attributes are represented with entities (objects) and relationships. Although an ORM diagram can be transformed into a UML diagram, our *data mapping diagram* is coherently integrated in a global approach for the modeling of DW's [6,7], and particularly, of ETL processes [4]. In this approach, we have used the extension mechanisms provided by UML to adapt it to our particular needs for the modeling of DW's. In this case, we always use formal extensions of the UML for modeling all parts of DWs.

6 Conclusions and future work

In this paper, we have presented a framework for the design of the DW backstage (and the respective ETL processes) based on the key observation that this task fundamentally involves dealing with the specificities of information at very low levels of granularity. Specifically, we have presented a disciplined framework for the modeling of the relationships between sources and targets in different levels of granularity (i.e., from coarse mappings at the database level to detailed inter-attribute mappings at the attribute level). Unfortunately, standard modeling languages like the ER model or UML are fundamentally handicapped in treating low granule entities (i.e., attributes) as FCME. Therefore, in order to formally accomplish the aforementioned goal, we have extended UML to model attributes as FCME. In our attempt to provide complementary views of the design artifacts in different levels of detail, we have based our framework on a principled approach in the usage of UML packages, to allow zooming in and out the design of a scenario.

Although we have developed the representation of attributes as FCME in UML in the context of DW, we believe that our solution can be applied in other application domains as well, e.g., definition of indexes and materialized views in databases, modeling of XML documents, specification of web services, etc. Currently, we are extending our proposal in order to represent attribute constraints such as uniqueness or disjunctive values.

References

1. SQL Power Group: How do I ensure the success of my DW? Internet: http://www.sqlpower.ca/page/dw_best_practices (2002)
2. Strange, K.: ETL Was the Key to this Data Warehouse's Success. Technical Report CS-15-3143, Gartner (2002)
3. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Conceptual Modeling for ETL Processes. In: Proc. of 5th Intl. Workshop on Data Warehousing and OLAP (DOLAP 2002), McLean, USA (2002) 14–21
4. Trujillo, J., Luján-Mora, S.: 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., Chicago, USA (2003) 307–320
5. Vassiliadis, P., Simitsis, A., Skiadopoulos, S.: Modeling ETL Activities as Graphs. In: Proc. of 4th Intl. Workshop on the Design and Management of Data Warehouses (DMDW'02), Toronto, Canada (2002) 52–61
6. Luján-Mora, S., Trujillo, J., Song, I.: Extending UML for Multidimensional Modeling. In: Proc. of the 5th Intl. Conf. on the Unified Modeling Language (UML'02). Volume 2460 of LNCS., Dresden, Germany (2002) 290–304
7. Luján-Mora, S., Trujillo, J., Song, I.: Multidimensional Modeling with UML Package Diagrams. In: Proc. of the 21st Intl. Conf. on Conceptual Modeling (ER'02). Volume 2503 of LNCS., Tampere, Finland (2002) 199–213
8. Luján-Mora, S., Trujillo, J.: A Comprehensive Method for Data Warehouse Design. In: Proc. of the 5th Intl. Workshop on Design and Management of Data Warehouses (DMDW'03), Berlin, Germany (2003) 1.1–1.14
9. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. 2 edn. Springer-Verlag (2003)
10. Object Management Group (OMG): Unified Modeling Language Specification 1.4. Internet: http://www.omg.org/cgi-bin/doc?formal/01-09-67 (2001)
11. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Madison, Wisconsin, USA (2002) 233–246
12. Bernstein, P., Levy, A., Pottinger, R.: A Vision for Management of Complex Models. Technical Report MSR-TR-2000-53, Microsoft Research (2000)
13. Bernstein, P., Rahm, E.: Data Warehouse Scenarios for Model Management. In: Proc. of the 19th Intl. Conf. on Conceptual Modeling (ER'00). Volume 1920 of LNCS., Salt Lake City, USA (2000) 1–15
14. Dobre, A., Hakimpour, F., Dittrich, K.R.: Operators and Classification for Data Mapping in Semantic Integration. In: Proc. of the 22nd Intl. Conf. on Conceptual Modeling (ER'03). Volume 2813 of LNCS., Chicago, USA (2003) 534–547
15. Falkenberg, E.: Concepts for modelling information. In: Proc. of the IFIP Conference on Modelling in Data Base Management Systems, Amsterdam, Holland (1976) 95–109
16. Embley, D., Kurtz, B., Woodfield, S.: Object-oriented Systems Analysis: A Model-Driven Approach. Prentice-Hall (1992)
17. Halpin, T., Bloesch, A.: Data modeling in UML and ORM: a comparison. Journal of Database Management **10** (1999) 4–13