YAM^2 : A Multidimensional Conceptual Model

PhD Thesis

Alberto Abelló

Advisors: Dr. José Samos and Dr. Fèlix Saltor

Programa de Doctorat de Software

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

2002

To Enry,
for her unlimited
support and love.

Foreword

> "I have no talent for making new friends, but oh, such a genius for fidelity to old ones."
>
> Peter Ibbetson, 1891

Let's say this work began several years ago when the Spanish army gave me a whole year of vacations in the North of Africa. Leaving aside making good friends like Jorge, I had nothing to do but reading books, playing chess, and think about my future. Paraphrasing G. Pólya, I thought, I am not good enough for mathematics and I am too good for the army. Computer science is in between.

Thus, I arrived to the Facultat d'Informàtica de Barcelona. There, I was lucky in enjoying great classmates like Alex and Xavi (something essential to get a degree). We found some good lecturers, but I guess that who made me fall in love with database design was Jaume Sistac.

Five years later, as I finished my undergraduate studies, I decided to try a doctorate. Fèlix Saltor gave me the opportunity of joining his research group, and the Generalitat de Catalunya the grant 1998FI-00228, which allowed me to write this thesis (I was also included in projects TIC96-0903, TIC1999-1078-C02 and TIC2000-1723-C02 from the Spanish Research Program PRONTIC). So, I became a member of the Departament de Llenguatges i Sistemes Informàtics (which resourced me, with special mention for the valuable work of the secretaries of the department), and I was kindly welcome by the members of the Secció de Sistemes d'Informació.

Since then, I've been sharing an office with Elena for nearly four lovely years. From time to time, we got the visit of Marta (the other member of the research group), always supportive from Lleida. What to say about them? Just a pleasure to work together.

Time arrived to find an advisor, and I got two instead of only one. Felix taught me how to do quality research and contributed his long experience. I should name a couple of important things I was not able to learn from him: write a correct bibliography and drink good wine instead of coke or kalimotxo. The other advisor was José Samos. I should thank him lots of things like being an inexhaustible fountain of optimism, but over the others, his almost infinite patience during our fruitful never-ending discussions.

The work with José was easier thanks to the Departamento de Lenguajes y Sistemas Informáticos of the Universidad the Granada, which offered me a place to work there. As a side

effect, being there allowed me to meet wonderful people like Cecilia, Eladio or Ventura who made me feel at home during my numerous stays in Granada.

Arriving to the end, I also thank Antoni Olivé, Ernest Teniente, Juan Carlos Trujillo, Pedro Blesa, Mohand-Said Hacid, A. Min Tjoa, and Panos Vassiliadis for revising this PhD thesis and accepting being part of the jury. I am also grateful to the anonymous reviewers of the thesis, and those anonymous referees of the different papers who sent useful, constructive, and instructive comments.

Two more things before I finish this words. I should not forget friends here, because chat, beer, playing role games and cycling is also important to write a thesis. And last but not least, an special acknowledgement for the women in my family: my mum, my dear aunt, Angelines and my grandmother. They brought me up.

Alberto

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> “ 'Where shall I begin, please your Majesty?' he asked.
> 'Begin at the beginning,', the King said, very gravely, 'and go on till you come to the end: then stop.”
>
> Lewis Carroll, “Alice's Adventures in Wonderland”

In this first chapter, general “Data Warehousing” and “On-Line Analytical Processing” (OLAP) concepts are defined. Afterwards, motivation and objectives of this thesis are established. In next section, its main contributions are briefly explained. The chapter finishes with the organization of the rest of the thesis, containing a summary of the other chapters.

1.1 General concepts

As it was defined by William Inmon in [Inm96], a “Data Warehouse” (DW) is a subject-oriented, integrated, non-volatile, and time variant collection of data in support of management's decisions. Other authors, like [Gar98] prefer to talk about “Data Warehousing”, and define it as a process, not a product, for assembling and managing data from various sources for the purpose of gaining a single, detailed view of part or all of a business. Whether collection of data or process, the point is that we are dealing with a huge amount of data aimed for analysis tasks, which presents challenges in its construction, management, and usage (see [Wid95] and [WB97] for two surveys of research issues in this field). [JLVV00] contains a wide overview of the area, and [Vas00b] compiles and classifies the papers published in three of the most significant database conferences (i.e. PODS, SIGMOD, and VLDB), from 1995 to 1999, related to the subject.

Figure 1.1 shows the “Corporate Information Factory” (CIF) architecture presented in [IIS98]. We can see that raw detailed data enters from the left side into the operational applications. These applications represent transactional systems that deal with day by day data. They could also get processed information from some external sources, if needed.

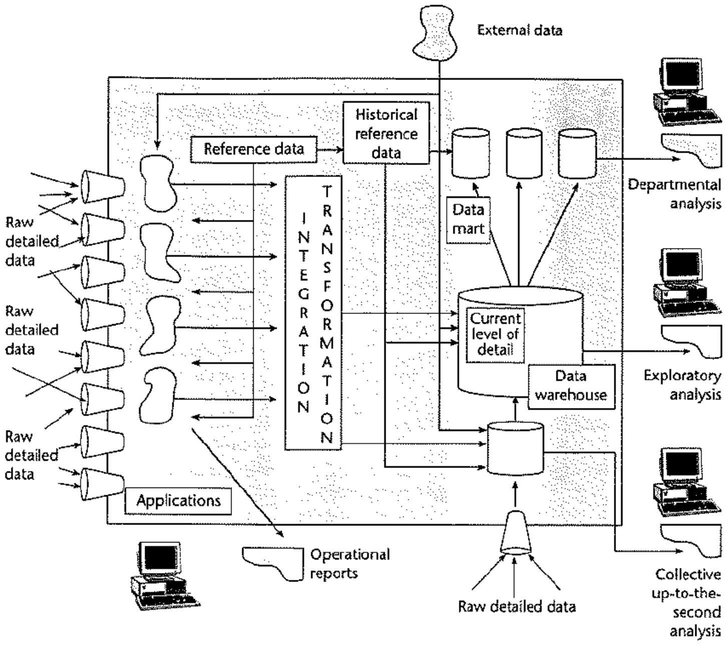


Figure 1.1: Corporate Information Factory [IIS98]

All data in the operational applications is time stamped, transformed, cleansed, integrated, and finally deployed into either the DW or the "Operational Data Store" (ODS). An ODS is an architectural construct that is subject-oriented, integrated, volatile, current-valued and contains only corporate detailed data, as defined in [IIB96]. It is used to support the up-to-the-second collective tactical decision-making process for the enterprise, and can contain data not coming from the operational systems. The ODS can be used as an intermediate step for the load of the DW.

Based on the analysis requirements of a department or set of users, "Data Marts" (DM) are built. As defined in [IIS98], a DM contains customized, summarized data from the DW tailored to support the specific analytical requirements of a given business unit.

The interactive querying of the DMs is known as "On-Line Analytical Processing" (OLAP). OLAP products, specially conceived for departmental analysis, were presented for the first time in [CCS93], where we can also find twelve evaluation rules for them. The first one of Codd's evaluation rules expresses the main characteristic of OLAP, namely multidimensionality. This characteristic is also outlined in [Pen01], which defines OLAP tools as "FASMI" (Fast Analysis of Shared Multidimensional Information). The OLAP Council, in [OLA97], gives the following definition:

> OLAP is a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a

> wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user. OLAP functionality is characterized by dynamic multi-dimensional analysis of consolidated enterprise data supporting end user analytical and navigational activities.

OLAP tools represent data as if these were placed in an n-dimensional space, allowing their study in terms of facts subject of analysis, and dimensions showing the different points of view according to which the subject can be analyzed. This conception gives rise to data schemas with star shape (i.e. a subject of analysis in the middle, and its analysis dimensions around it).

OLAP concepts are not completely new. As it was shown in [Sho97], most of them were already used in statistical databases. Nevertheless, in the last years the area has got the attention of the industry as well as the research community, giving rise to important advances. [DSHB99] surveys the OLAP market, while [CD97] gives an overview of DW and OLAP all together.

1.2 Motivation and objectives

In the last years, lots of work have been devoted to multidimensional modeling, and several models have been proposed. However, there is neither a well accepted model nor a standard terminology, yet.

Some of the existing models formalize multidimensional concepts in one way or another, and present calculus and/or algebras to operate on n-dimensional data cubes. Other models show how multidimensional data could be stored in either Relational, O-O, or pure Multidimensional DBMSs. Thus, out of all this work already done, few authors paid special attention to conceptual multidimensional modeling. What is more, only a couple of them studied the applicability of the O-O paradigm to this field. [TPGS01] focuses on software engeneering for OLAP tools rather than true data modeling, while [BTW00] actually only uses O-O concepts, namely "Unified Modeling Language" (UML), in the definition of metaclasses.

Multidimensional concepts and relationships are really useful for analysis tasks. Nevertheless, this should not imply that other data modeling concepts should be ignored. In the last years, the O-O paradigm proved to be close to the human way of thinking (an essential characteristic for a conceptual data model). Therefore, an objective of this thesis is to study the applicability of O-O concepts to multidimensional conceptual modeling.

Almost all existing multidimensional models are limited to model isolated stars (i.e. isolated subjects of analysis). At most, some of them allow to share analysis dimensions between different star schemas. However, it is easy to find semantic relationships (like *Generalization*, *Association*, etc.) that relate concepts in two such schemas. To utilize data used/obtained on analyzing a subject, during the analysis of another one, would be a powerful tool in analysis tasks. An architecture based on different levels of schemas needs to be studied to allow that. The "Data Warehouse Architecture" presented in [KRRT98] is, semantically, too poor. It just allows to share analysis dimensions.

Moreover, besides the lack of semantic relationships there is no agreement on the definition and properties of multidimensional concepts. All models merely impose the properties and structure of aggregation hierarchies in the analysis dimensions, and nobody discussed nor proved them.

Another important issue in multidimensional modeling is that of aggregability or summarizability. The data schemas should show how data of a given granularity can give rise to data of coarser granularity. [LS97] restricts summarizability problems to aggregating along the temporal dimension. However, other analysis dimensions can also be problematic, and it is important to reflect this kind of problems in the schema in order to warn analysts.

1.3 Main contributions

[SSSB98] proposed to consider the DW as part of the database architecture. It studied different "Data Warehousing" architectures and presented an integrated database architecture of schemas for "Federated Information Systems" (FIS) and "Data Warehousing". It is well known (see [SCG91]) that O-O data models are well suited to be used as canonical models for FIS. Therefore, from the inclusion of "Data Warehousing" schemas in the FIS, it follows that O-O models also have positive characteristics for "Data Warehousing".

Based on that previous work, the main contributions of this thesis are:

1. The work in [SSSB98] has been extended by studying an architecture based on different levels of schemas that facilitates the construction of the different CIF components. The characteristics of every level of schemas have been stated.

2. It is quite common in analysis tasks that information used or obtained from the study of a given subject is valuable for the analysis of another subject. However, existing models do not pay enough attention to this, and only allow to represent isolated star schemas. This thesis illustrates and exemplifies the usage of multi-star schemas. A variation of the three-levels ANSI-SPARC architecture is presented to facilitate it.

3. In the last years, several multidimensional models appeared. Each of those models uses a different nomenclature, and was conceived for a different purpose, so that their comparison becomes really difficult. A framework for their classification and comparison has been defined. Multidimensional models are classified into **Conceptual**, **Logical**, **Physical**, and **Formalism**. Moreover, their elements are characterized within three detail levels, namely **Upper**, **Intermediate**, and **Lower**.

4. The importance of aggregation hierarchies is recognized by almost all authors. Thus, most multidimensional models provide mechanisms to define them. Nevertheless, none of the authors proved nor justified the characteristics of those hierarchies. In this thesis, from the assumption that those hierarchies are defined by part-whole relationships, mereology axioms have been used to demonstrate some of their properties.

5. Based on the structure of aggregation hierarchies and data dependencies, the structure of the facts subject of analysis has also been studied.

6. UML is becoming a standard language for conceptual modeling. Thus, its metaclasses have been extended in this thesis to encompass multidimensional concepts. This has given rise to **YAM^2** (Yet Another Multidimensional Model).

7. The usage of different O-O relationships (i.e. in UML terminology *Generalization*, *Association*, *Aggregation*, *Derivation*, and *Flow*) has been studied for multidimensional schemas.

8. A closed and complete algebra of operations on data cubes has been defined for **YAM^2** .

9. Integrity constrains have been defined for **YAM^2** . They focus on identification, and aggregability of data. **YAM^2** provides a flexible set of mechanisms to show summarizability of the different kinds of user measures.

1.4 Organization of the thesis

This thesis has been organized into six chapters (including this one), and three appendixes. Chapters from two to five contain the contributions of the thesis. A brief overview of each chapter and appendixes is shown below.

1.4.1 Second chapter: Multidimensional modeling and the O-O paradigm

This chapter begins explaining some basic multidimensional concepts that will be needed to understand the rest of the thesis. The duality fact-dimension is introduced, besides the notion of data cube, and the well known multidimensional operations over data cubes. Then, an original analysis framework for the classification and comparison of multidimensional models is introduced, so that related work can be clearly presented and compared. Most existing multidimensional data models are described here with regard to the analysis framework.

The last part of the chapter introduces the notion of O-O dimension, as explained in [Sal96], in order to be used as a basis for the presentation of some basic ideas of the thesis. The usage of the Generalization/Specialization, Aggregation/Decomposition, Instantiation/Classification, Derivability or Point of view, Dynamicity, and Behavioural O-O dimensions in multidimensional modeling is briefly explained by examples.

1.4.2 Third chapter: Multi-level schemas architecture

Out of the four characteristics of a DW defined by W. Inmon, we can see that one of them (i.e. "integrated") is also present in a FIS. This chapter presents how the 5-levels schemas architecture for FIS of [SL90], extended to 7-levels in [ROSC97], has been modified to include "Data Warehousing" schemas, by continuing the work done in [SSSB98]. **Data Warehouse Schema**s, **Operational Data Store Schema**s, and **Data Mart Schema**s are placed in the architecture. The characteristics of these new schemas are analyzed. The design of the DW is presented as data-driven, versus the query-driven design of the DMs' star shape schemas.

Most multidimensional models are restricted to isolated stars. However, a semantically rich set of abstractions in the data model, like those in **YAM^2** , and an appropriate architecture of schemas can facilitate the modelization of related stars. The second part of the chapter pays attention to this issue, focusing on the **Data Mart Schema**s in the architecture. Useful semantic relationships that can be used to relate different stars are shown, and their usability to drill across different data cubes is discussed. Specifically, *Generalization*, *Association*, *Derivation*, and *Flow* relationships (in UML sense) are studied. Moreover, three schema levels, based on the ANSI/SPARC architecture, are defined to facilitate the management of these more complex schemas.

1.4.3 Fourth chapter: Elements of a multidimensional model

Multidimensionality is marked by the duality fact-dimension. That is, factual and dimensional data drive the modeling, implementation and usage of OLAP tools. In this chapter these kinds of data are analyzed separately.

Firstly, **Dimension**s are studied. In the literature, we can find different definitions and conceptions of relationships between aggregation levels. This section contends that they are part-whole relationships. Thus, mereology axioms can be used on the study of **Dimension**s. From a simple definition and those axioms, some properties of **Dimension**s are proved, addressing several problems (or controversial points) detected in existing multidimensional models. Moreover, the consequences of *Generalization*, and *Aggregation* relationships between **Dimension**s are also studied.

The second half of the chapter studies **Fact**s. Their components are defined, and their structure is analyzed with regard to that of the **Dimension**s and data dependencies. A **Cube** is defined as a function from the cartesian product of **Level**s in orthogonal **Dimension**s to the domain of a **Fact**. A new operation, i.e. **ChangeBase**, is presented to allow the modification of the n-dimensional space were data cells are placed. The possibility of having *Generalization*, *Association*, and *Derivation* relationships between **Fact**s is studied.

1.4.4 Fifth chapter: YAM^2 (Yet Another Multidimensional Model)

This chapter presents a multidimensional conceptual O-O model, its structures, integrity constraints and query operations. It has been developed as an extension of UML core metaclasses to facilitate its usage, as well as to avoid the introduction of already existing general concepts. **YAM^2** allows the representation of several semantically related stars, as well as summarizability and identification constraints.

The first section outlines the main differences between this and other models (i.e. usability, "Semantic Power", "Semantic Relativism", and the possibility of expressing summarizability and identification constraints). Then, data structures of the model are defined and exemplified in terms of nodes and arcs of a graph. The applicability of all UML relationships is systematically studied. In next section, the inherent integrity constraints of the model are presented. Another section is devoted to multidimensional operations over **Cube**s. Finally, to summarize the model, its metaclasses are presented. Each **YAM^2** metaclass have been defined as a subclass of a UML metaclass. The chapter concludes with a comparison of multidimensional data models.

1.4.5 Sixth chapter: Conclusions

The last chapter of the thesis contains some conclusions and future work.

1.4.6 Appendixes

There are three appendixes to this thesis. The first one contains the formal extension of UML, i.e. a *Profile* with all **YAM**^{2} modeling elements as *Stereotype*s, and the corresponding integrity constraints in OCL. Another appendix shows several multidimensional design examples of the usage of **YAM**^{2} . Some schemas in other models are translated to **YAM**^{2} , and some original design cases are also presented. Finally, a list of papers published as the result of this thesis work, sorted by chapter, is included.

1.5 Typographic conventions

Several typographic conventions have been taken to improve the readability of the document:

- **Bold Face** is used for terms defined along this thesis. For instance, the term **Dimension**, in spite of being used by other authors, has been carefully studied and defined in pages 74, and 95, so that it is used in exactly that sense.
- "Quotation" indicates terms defined by other authors. If the term is considered well known, it is only quoted the first time it appears.
- `Times Font` marks words and concepts in the figures or examples.
- *Italics* is used for UML terms.

Moreover, UML notation, as defined in [OMG01b], has been used in the figures.

Chapter 2

Multidimensional modeling and the O-O paradigm

"To be is to be related."

C.J. Keyser

The words "On-Line Analytical Processing" bring together a set of tools, that use multidimensional modeling in the management of information to improve the decision making process. Lately, a lot of work has been devoted to modeling the multidimensional space. Thus, the next sections relate this thesis to other work.

Firstly, section 2.1 introduces main multidimensional concepts like "analysis dimension", "facts", "star", etc. Then, section 2.2 presents an original framework to classify and describe multidimensional models. They are divided based on the design phase for which they seem more appropriate (i.e. **Conceptual**, **Logical**, and **Physical**), or if not used on designing (i.e. **Formalism**). Moreover, this section also explains how the elements of each model can be placed at three different detail levels (i.e. **Upper**, **Intermediate**, and **Lower**) so that they can be easily compared. These detail levels refer to the containment of multidimensional elements into one another (for instance, an analysis dimension is composed by different aggregation levels).

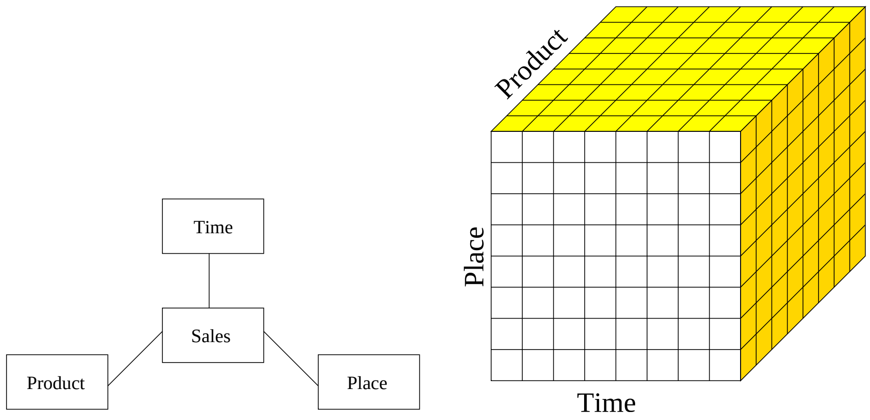
Section 2.3 corresponds to the state of the art of the thesis. Existing multidimensional models are classified and described there with regard to the above mentioned framework.

Finally, section 2.4 outlines the advantages of using an O-O model in multidimensional design. It is argued that multidimensional modeling is lacking in semantics, which can be obtained by using the O-O paradigm. Some benefits that could be obtained by doing this are classified in six O-O-Dimensions (i.e. Classification/Instantiation, Generalization/Specialization, Aggregation/Decomposition, Behavioural, Derivability, and Dynamicity), and exemplified with specific cases.

2.1 Multidimensional modeling

Along its years of existence, SQL proved to be really useful and well accepted in "On-Line Transactional Processing" environments. However, as time went by, due to the wide spread of computers, databases arrived to analysis tasks in the form of "Data Warehousing" systems. In this kind of environments, because of the huge amount of data, complexity of queries and unskillfulness of users, SQL has proved not to be the best solution.

To bring data near analysts, "Data Marts" (DMs) appeared. They are small "Data Warehouses" devoted to satisfy the needs of a reduced set of users. They are customized to obtain good query performance (most of times by means of a query-driven design). Closely related to DMs are "On-Line Analytical Processing" (OLAP) tools. By means of multidimensionality, this kind of tools allow non-expert users to formulate their own queries and obtain the results interactively (without the assistance of the IT department).



(a) Star schema (b) Cube metaphor

Figure 2.1: Multidimensional modeling

Multidimensionality is based on the duality fact-dimensions, i.e. facts are analyzed with regard to data in the dimensions. A fact represents a subject of analysis, while its analysis dimensions show the different points of view we can use to study it. This gives rise to schemas with star shape (like that one depicted in figure 2.1(a)), having the abstraction representing the facts in the middle, and the analysis dimensions around it. The fact in a multidimensional schema represents the set of measurements to be analyzed (mainly numeric attributes). On the other hand, the analysis dimensions mainly contain descriptive attributes that describe the points in the space.

Frequently, the "Data Cube" metaphor (depicted in figure 2.1(b)) is used to explain multidimensionality. Each cell in the cube represents a unit of data (for instance, in the example above, `Sales` as the intersection of a `Product`, `Place`, and `Time`). By defining a single position in every dimension of the analysis space, we select exactly one of those cells. In general, since we could have several (more than three) analysis dimensions, the "Cube" should actually be called "Hypercube" (from here on, the term "Cube" will be misused).

Benefits of multidimensional modeling are two fold. On the one hand, it makes the data schemas more understandable to final users, and on the other hand, it allows to use specific storage and access techniques that improve query performance. The way to obtain these benefits is by simplifying the data schemas, so that they only contain the essential things (i.e. a fact to be analyzed and its analysis dimensions). These schemas are close to the analysts conception of data, and suggest a specific kind of queries, so that the system can be easyly customized to solve them with good response times.

Specific operations have also been defined in the multidimensional world. However, there is no agreement on a standard set of such operations. Often, the process of navigating through multidimensional data is called "Slice and Dice". Just to cite here those navigation operations defined in [OLA97]:

Consolidate/Aggregate/Roll-up Multidimensional databases generally have hierarchies or formula-based relationships of data within each dimension. Consolidation involves computing all of these data relationships for one or more dimensions. While such relationships are normally summations, any type of computational relationship or formula might be defined.

Drill-down It is a specific analytical technique whereby the user navigates among levels of data ranging from the most summarized (up) to the most detailed (down). The drilling paths may be defined by the hierarchies within dimensions or other relationships that may be dynamic within or between dimensions.

Rotate/Pivot This operation changes the dimensional orientation of a report or page display. For example, rotating may consist of swapping the rows and columns, or moving one of the row dimensions into the column dimension, or swapping an off-spreadsheet dimension with one of the dimensions in the page display (either to become one of the new rows or columns), etc.

Selection A selection is a process whereby a criterion is evaluated against the data or members of a dimension in order to restrict the set of data retrieved.

Another generic definition of operations over data cubes can be seen in [Gio00]: "Slice" reduces the dimensionality of a cube; "Dice" selects a set of data; "Roll-up" aggregates data along the hierarchy in an analysis dimension; "Drill-down" gives more detail in a dimension, by descending along its aggregation hierarchy; and "Drill-across" travels from a data cube to another one. SQL syntax was also extended to support some multidimensional operations as can be seen in [ISO99].

An essential characteristic of multidimensional analysis is the study of data summarized at different granularities. Thus, out of these operations, it is essential to remark the importance of "Roll-up" and "Drill-down". They imply moving up and down aggregation hierarchies, which define the aggregation levels of interest for every analysis dimension.

2.2 An analysis framework

This section presents the original analysis framework that will be used in section 2.3 to classify the huge amount of efforts in the area, devoted to modeling the data cube. The models will be divided into four groups based on the design phase for which they are more suitable. Moreover, different detail levels are also defined to be able to compare their modeling constructs. Firstly, section 2.2.1 briefly reviews previous work on classifying and describing multidimensional models. Then, section 2.2.2 define the framework that will allow to describe and classify the different models.

2.2.1 Other frameworks

In [BSHD98], a list of requirements for a multidimensional model in order to be suitable for OLAP applications, is used to analyze seven models, which are chosen because they contain some kind of formalism. Among those seven we find [AGS97], [GL97], [CT98a], [Vas98], and [Leh98]. Those requirements (derived from general design principles, and from characteristics of OLAP applications) are the following:

- Explicit separation of cube structure and its contents
- Complex dimensions
  - Level structure
  - Member (i.e. level instance) structure
  - Formalism (mathematical construct) for level structure
  - Dimension attributes (those not defining hierarchies)
- Symmetry of measures and dimension members
- Complex measures
  - Support of structured measures
  - Support of derived measures
  - Additivity of measures
- Query formalism
  - Type of formalism (i.e. algebra or calculus)
  - Ad-hoc hierarchies
  - User defined aggregates

[PJ98] and [Ped00] present eleven requirements (found in clinical data warehousing) for multidimensional data models, and evaluates twelve preexisting data models against them. Those presented in [AGS97], [Dyr96], [Kim96], [GL97], [CT98a], [Leh98], and [Vas98] are among those twelve. An statistical model, and a commercial system are also included. Moreover, it presents a data model which does address all those requirements. The requirements are:

1. Explicit hierarchies in dimensions
2. Symmetric treatment of dimensions and measures
3. Multiple hierarchies in each dimension (different aggregation paths)
4. Support for aggregation semantics (applicability of aggregation functions)
5. Non-strict hierarchies (overlapping classifications)
6. Non-onto hierarchies (non-balanced trees of instances)
7. Non-covering hierarchies
8. Many-to-many relationships between facts and dimensions
9. Handling change and time
10. Handling different levels of granularity
11. Handling uncertainty

[VS99] and [Vas00a] give yet another classification of multidimensional models. In this case, the discussion is said to be placed at "logical" level. Among others, it pays attention to [GL97], [AGS97], [CT98a], [Leh98], some industrial standards, and a couple of statistical models. The requirements studied in this case are:

- Representation of the multidimensional space
  - Cubes/Tables
  - Explicit/Implicit hierarchies
- Language issues
  - Character of the query language (Procedural/Declarative/Visual)
  - Support of sequences of operations
  - Naturality of OLAP operations modeled
- Mappings offered to
  - Relations
  - Multidimensional arrays

The difference between these sets of comparison criteria and the framework proposed in this section is that the former aim to discover weaknesses in the existing models, while the latter treats to facilitate the comparison of the different work and terminology. Each one of the three papers, begins by defining a list of specific requirements for a multidimensional model in order to evaluate all those models already existing. In this section, there is not such a list. Each one of the multidimensional models compiled in next section uses its own terminology and defines a specific set of design elements. In this sense, different detail levels are used to classify the constructs of the models, in order to be able to compare them, and examine the expressive power of every model.

2.2.2 A classification and description framework

This section introduces two sets of classification and description levels for multidimensional models. Both sets of levels are orthogonal. Thus, a model can be classified as either **Conceptual**, **Logical**, **Physical**, or **Formalism**; and contain constructs at any of the three detail levels, i.e. **Upper**, **Intermediate**, or **Lower**.

Design levels

As defined in [EN00], a "data model" is a set of concepts that can be used to describe the structure of a database. In the same book, we also find a categorization of "data models" into "High-level" or "Conceptual", if they provide concepts that are close to the way users perceive data; "Low-level" or "Physical", if they provide concepts that describe the details of how data is stored in the computer; and "Implementation", if they provide concepts that can be understood by end users, but that are not too far removed from the way data is organized within the computer.

Also [BCN92] describes those three groups of models. Adopting its terminology, from here on, three different kinds of multidimensional data models are distinguished, based on the constructs/concepts they provide and the "Data Mart" design phase they help: Those at **Conceptual** level that are close to the user and independent of the implementation; those at **Logical** level depending on the kind of Database Management System (DBMS) used in the implementation, but still understandable by end users; and finally, those at **Physical** level depending on the specific DBMS used, and conceived to describe how data is actually stored.

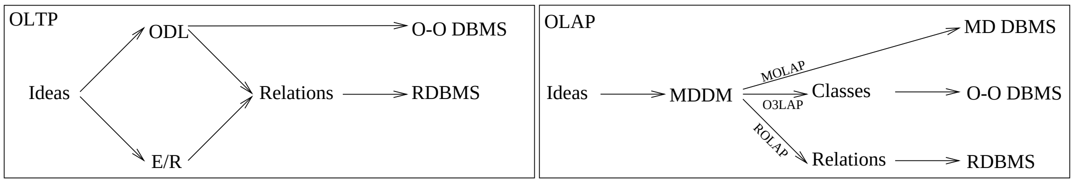


Figure 2.2: Modeling and implementation process in OLAP vs OLTP environments

As shown in figure 2.2 (left), from [UW97], in an On-Line Transactional Processing (OLTP) environment, during the first design step, at **Conceptual** level, we would use Object Definition Language (ODL) or Entity-Relationship (E/R) to represent user ideas; in the next step, at **Logical** level, we would usually use the Relational model, but we could also use Hierarchical, or Network models (not depicted in the figure); and in the last step, at **Physical** level, the implementation would depend on a specific DBMS (i.e. Oracle, Informix, ObjectStore, etc.). In a similar way, in the proposal of this thesis for an OLAP environment, in figure 2.2 (right), we would have the Multidimensional Data Model (MDDM) at **Conceptual** level, and, depending on the approach (i.e. Relational -ROLAP-, Object-Oriented -O3LAP-, or pure Multidimensional -MOLAP-), we would use a different model at **Logical** level, and a different DBMS for the implementation.

Besides these three mentioned above, there is another set of models (which will be referred along this thesis as **Formalisms**) whose concepts would not be used at any database design phase, but on giving a theoretical framework. Their stress is on formalizing multidimensionality rather than on database modeling. They include an algebra or calculus. In an OLTP environment, a formalism would be the Relational Algebra.

These four design levels are not ad hoc, they are based on the well known design phases of OLTP systems. Thus, they can be used to classify any kind of data model. A fourth group of models has been added to cluster those data models that do not seem well suited for design, but emphasize the formalization of the domain.

Detail levels

In a multidimensional model, several detail levels can be distinguished. Thus, we can see a schema with coarser or more detailed elements. It is similar to show the attributes, methods, and constraints for every class in a schema, or just show the name of the classes. By looking to the names of the classes, we get an idea of the modeled reality, but if we do not look to the more detailed information, we cannot completely understand the data. Three different detail levels can be found in multidimensional modeling:

Upper: At this level, we find **Dimension**s and **Fact**s. The **Dimension**s are used to characterize the **Fact**s, and show the viewpoints the **Fact**s will be analyzed from. By relating a set of **Dimension**s to a **Fact**, we obtain a star shape schema. The possibility of navigating from one of such star shape schemas to another one uses to be shown by the share of **Dimension**s.

Intermediate: **Dimension**s and **Fact**s are decomposed into **Level**s, and **Cell**s respectively. The different **Level**s in a **Dimension** form an aggregation hierarchy. Each **Cell** contains data at a given **Level** for each **Dimension** its **Fact** is related to.

Lower: The most detailed level shows the attributes of the **Level**s and **Cell**s. That is **Descriptor**s, and **Measure**s respectively.

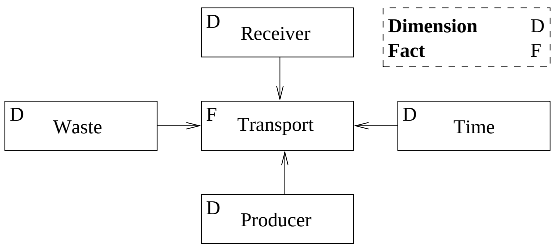


Figure 2.3: Example of multidimensional schema at **Upper** detail level

Figure 2.3 represents a multidimensional schema at **Upper** detail level. If we are talking about a waste transport business, we could be interested in analyzing `Transport` involving the transported `Waste`, the `Time` the transport takes place, the `Producer` it is transported from, and the `Receiver` it is transported to. Therefore, we would have a 4-dimensional space where each point represents transport data, and is identified by a waste, a point in time, a waste producer, and a waste receiver.

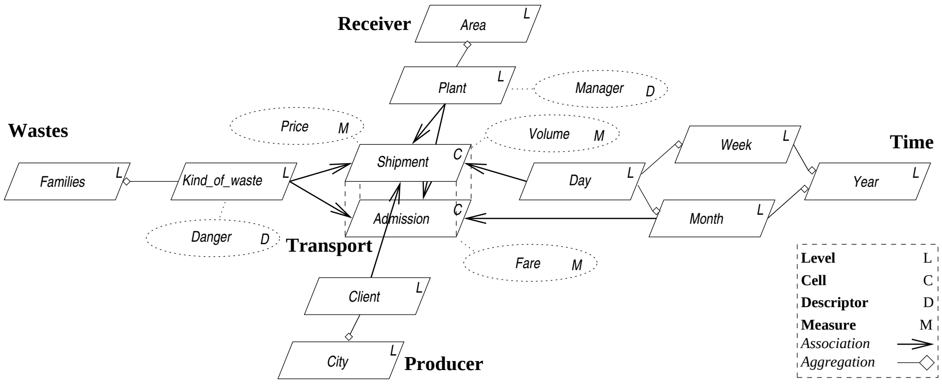


Figure 2.4: Example of multidimensional schema at **Intermediate** and **Lower** detail levels

The same multidimensional schema is depicted in figure 2.4 in more detail. Each one of the **Dimension**s is further described by a hierarchy of different **Level**s. For instance, `Time` **Dimension** contains `Day`, `Week`, `Month`, and `Year` **Level**s. Moreover, in a similar way, we could decompose the **Fact**. If we were interested in the benefits of a transport, we would need to analyze data that would belong to different kinds of data cells (price of the shipment that we charge to our client, minus admission fare that a processing plant charges to us). On one hand, we can see a `Shipment` **Cell** containing data about our shipments which depends on the lower **Level** of each one of the four **Dimension**s. On the other hand, data about admission of waste in a plant do not depend on our clients, nor on `Day` **Level** of `Time` **Dimension** (it depends on `Month` **Level**). Therefore, `Admission` and `Shipment` are different kinds of **Cell**s, but belong to the same **Fact** we want to analyze.

Finally, drawn with dotted lines, we can see constructs at **Lower** detail level. Some **Level**s have **Descriptor**s associated (for instance, a `Plant` has a `Manager`). Besides, **Cell**s have associated **Measures** (for instance, an `Admission` has a `Fare`) that we want to analyze.

2.3 Classification and description of existing multidimensional models

This section contains the state of the art of the research in the field of multidimensional modeling. The analysis framework presented in the previous section is used here to classify and describe existing multidimensional models.

Models are grouped into four different sets based on the multidimensional database design phase they are conceived for. Some of the publications considered in those three classifications in section 2.2.1 are not included in this one, because they do not fit at any of those sets (i.e. papers about statistical models, which relevant contributions are incorporated into more recent multidimensional ones, or papers whose main subject, despite being devoted to multidimensionality, is not really multidimensional modeling). The different elements and relationships between them that are provided at each one of the detail levels are studied for each and every model.

Sections 2.3.1, 2.3.2, 2.3.3, and 2.3.4 contain the four groups of models. Moreover, a section on miscellaneous issues has been added. Each one of these four subsections contains proposals at a given design level (i.e. **Conceptual**, **Logical**, **Physical**, and **Formalisms** respectively). Inside the subsections, models are chronologically ordered by year. Section 2.3.5 contains contributions to multidimensional modeling that were not classified at any of the previous ones.

At the beginning of each one of those subsections, there is a table showing the constructs of each model at the corresponding level with regard to the description framework. As pointed out by [BSHD98], some multidimensional models do not separate cube structure and contents. Only those concepts represented at the schema level are considered (relationships among instances are not taken into account).

A tick ("$\surd$") means something is captured by the model, while a hyphen ("-") means that the authors of the model either say something not to be modeled or just do not say how to model it. A hyphen in the column corresponding to:

Measures (M) means that nothing can be represented in the schema about **Measure**s. Maybe, only pure numerical values are considered (without any meaningful domain).

Descriptors (d) means dimensional entities do not have attributes describing their instances. All the information is kept in the form of classification hierarchies at the most.

Relationships (Lower detail level) means there is not any way in the model to represent relationships among **Measure**s and/or **Descriptor**s.

Levels (L) means there are not explicit aggregation levels in the **Dimension**s.

Cells (C) means that either the **Measure**s are not grouped, or they are not related to a specific set of **Level**s, but to **Dimension**s as a whole (usually reflected as relating the **Measure**s to the lowest level in the classification hierarchy).

Relationships (Intermediate detail level) means there is not any way in the model to represent relationships among **Cell**s and/or **Level**s. For instance, it implies that there is not the possibility of explicit dimension hierarchies.

Facts (F) means that the different **Cell**s can not be grouped with the intention to relate **Measure**s that, even though are defined at different granularities, are used together in a given decision making process.

Dimension (D) means that either there is only the possibility of modeling one **Level**, or if it is possible to model more than one, they cannot be grouped into another construct of the model.

Relationships (Upper detail level) means there is not any way in the model to represent relationships among **Fact**s and/or **Dimension**s.

2.3.1 Research efforts at Conceptual level

This section collects those models that contain concepts which are closer to the user than to the actual computer implementation (i.e. those in [Leh98], [CT98a], [GMR98b], [TP98], [SBHD99], [SCdMM99], [TBC99], [BTW00], and [HLV00]). These efforts try to represent how users perceive a multidimensional cube without paying special attention to formalisms.

	Upper detail level			**Intermediate** detail level			**Lower** detail level		
Author	F	D	Relationships	C	L	Relationships	M	d	Relationships
Lehner	–	✓	–	–	✓	Linear hierarchy of Ls $Ls \in cube$	–[1]	✓	ds associated to instances of L
Cabibbo & Torlone	✓	✓	F and Ds ∈ "cube" F is a set of Ms	–[2]	✓	Ls form a roll-up hierarchy (partial order)	✓	✓	d ∈ L $f : L^n \rightarrow M$
Golfarelli et al.	–	✓	–	✓	✓	Ds and C ∈ "cube" Aggregation hierarchy of Ls Compatibility between Cs	✓	✓	"to-one" between d and L Aggregability between M and D
Trujillo et al.	✓	✓	Part-whole between F and Ds	–	✓	Classification hierarchy of Ls	✓[3]	✓	Aggregability between M and D
Sapia et al.	✓	✓[4]	Cs in F share Ls in some D	✓	✓	"rolls-up to" between Ls "fact" relates n Ls	✓	✓	M ∈ C d ∈ L
Sanchez et al.	✓	✓	Cs in F use Ds in F	✓	✓	Aggregation functions between Ls	✓	✓	M ∈ C d ∈ L
Tryfona et al.	–	✓	–	✓	✓	Membership hierarchy of Ls Aggregation between Ls Specialization between Ls Allows M:N between C and L	✓	✓	M ∈ C d ∈ L
Nguyen et al.	✓	✓	Ds and Cs ∈ "cube"	✓	✓	Aggregability of Cs Partially ordered set of Ls "groupby" relates Cs and Ls	–[1]	–	–
Hüsemann et al.	–	✓	Ds ∈ "cube"	✓	✓	$f : Ls \rightarrow C$ Aggregation path between Ls	✓	✓	d ∈ L M ∈ C Aggregability between M and L

[1] Only domains over N, Z, and R are allowed.
[2] Even though **Measures** are related to **Level**, they are not grouped into **Cell**.
[3] Possibly derived.
[4] Implicit within the structure of the "rolls-up to" graph.

Table 2.1: Schema constructs in the different models at **Conceptual** level

Lehner (Nested Multidimensional Data Model - NMDM)

This is rather a presentation-oriented model, conceived to ease navigation through data. [Leh98] emphasizes the presentation of data at two different ("Nested") levels, and the operations offered to the user in order to accomplish this (i.e. "slicing", "drill-down", "roll-up", "split", "merge", "aggregation", and other cell-oriented operators like "max", "min", "+", etc.). The existence of two levels is said to improve the power and flexibility of the whole analysis process.

One of the most interesting features of NMDM (besides the existence of two levels) is the way it qualifies **Dimension** instances by means of different sets of attributes (i.e. different instances in the same class might have different attributes). Thus, at the bottom of every "classification hierarchy" is placed a "primary attribute" (PA) whose instances are called "dimensional elements". Those "dimensional elements" are the leaf nodes of a balanced tree-structured "classification hierarchy". Each tree level is called "classification attribute" (CA), whose instances are "classification nodes" (CNs). "dimensional attributes" (DA) are associated to every CN (notice that CNs are instances in the hierarchy).

A "Primary Multidimensional Object" (PMO) consists of an unique cell identifier; a set of CAs and PAs (one per **Dimension**), denoting the granularity of the cell; a set containing one instance per CA/PA, specifying the selection criteria; an aggregation type, describing the aggregation operations applicable; and a data type (i.e. domains over N, Z, or R). In turn, a "Secondary Multidimensional Object" (SMO) consists of a set of CNs; and a set of DAs applicable to them. Thus, a "Multidimensional Object" is a PMO, and a set of DAs for defining the corresponding nested SMOs.

All the schema constructs in this model refer to the **Dimension**s. They are defined as a linear hierarchy of **Level**s (called "classification attributes") at **Intermediate** detail level, and the instances of each **Level** have associated **Descriptor**s (called "classification nodes") at **Lower** detail level.

Cabibbo and Torlone (*MD*)

Cabibbo and Torlone, in [CT98b], [CT98a], and [CT97], qualify their model *MD* as "logical". However, they say that it is independent of any specific implementation, and present a design methodology to obtain an *MD* schema from an E/R one. Moreover, the authors argue that *MD* is at a higher level of abstraction than a star schema consisting of relational tables. Therefore, it should be classified as **Conceptual**, even though it provides a strong formal foundation (including a calculus).

The main constructs in the model are "dimension" and "f-table". Each "dimension" is organized in a hierarchy of "levels" corresponding to data domains at different granularity. In turn, a "level" can have "descriptors" associated with it. The "f-tables" are functions from "levels" to "measures".

We can clearly identify the data about **Dimension**s at the three different levels: "dimension", "levels", and "descriptors". About facts, there are only "measures" at **Lower Detail Level**, and a set of "f-tables" at **Upper** detail level. However, "measures" are not grouped regarding the **Level**s they are defined at.

Golfarelli, Maio, and Rizzi (Dimensional Fact Model)

[GR98], [GMR98a], [GMR98b], and [GR99] present a graphical conceptual model (DFM) for data warehousing, besides a methodology to obtain a multidimensional schema from the operational schemas (either E/R or Relational).

Contrary to what is said for some formal models, the authors claim that it is important to clearly distinguish between dimensional and factual data. Thus, a "dimensional scheme" consist of a set of "fact schemes", and each one of these contains a "fact", "measures", "dimensions", and "hierarchies". A "fact" is a focus of interest, and its attributes are "measures". The "dimensions" are discrete attributes which determine the minimum level of granularity chosen to represent the "fact". Finally, a "hierarchy" is a set of dimensional attributes linked by "to-one" relationships (i.e. 1:1, or N:1), which form a "quasi-tree". Hierarchies may also include "non-dimension attributes" that contain additional information which can not be used for aggregation (but just for selection). Moreover, aggregability can also be expressed by relationships between a "measure" and a "dimension", tagged by the allowed aggregation functions.

A special relation between two schemas is also defined (called "compatibility" and "strict compatibility"), which indicates and restricts when a query can be formulated including measures in both schemas. Roughly, two schemas are "compatible" when they have, at least, one common "dimension attribute".

Placing those constructs in the analysis framework of the previous section, we obtain **Descriptor**s (called "non-dimension attributes") and **Measure**s at **Lower** detail level, grouped respectively into **Level**s (called "dimension attributes") and **Cell**s. The **Level**s form dimension hierarchies. Furthermore, **Cell**s are related by "compatibility", and **Measure**s and **Dimension**s by aggregability.

Trujillo, Palomar, and Gómez (GOLD)

[TP98], [TPG00], and [TPGS01] describe an Object-Oriented conceptual model based on a subset of UML. A query notation is also presented.

A "fact" (represented as a basic class) is described through a set of "fact attributes" (either atomic or derived) representing **Measure**s. By mean of part-whole relationships, a "fact" is related to a set of "dimensions" (also represented as basic classes) that show the granularity adopted for representing facts. Those "dimensions" are also described by "dimension attributes". A "classification hierarchy" is defined as a Directed Acyclic Graph of "level" classes, rooted in the "dimension" class. Multiple classification hierarchies are allowed; and strictness, and completeness explicited. Aggregability of **Measure**s along each analysis dimension can be represented, as well as derived **Measure**s.

In this model, information at **Lower** detail level is represented in the form of **Measure**s (called "fact attributes"), and **Descriptor**s (called "dimension attributes"). The former are attributes of a **Fact** at **Upper** detail level, while the later are attributes of a **Level** at **Intermediate** detail level. A **Dimension** is defined as a classification hierarchy of **Level**s.

Sapia, Blaschka, Höfling, and Dinter (Multidimensional Entity Relationship Model)

[SBHD99] argues that the E/R model is not suited for multidimensional conceptual modeling. Thus, a specialization is defined, and its usage exemplified.

The design of this model was driven by the following ideas:

- Specialization of the E/R model
- Minimal extension of the E/R model
- Representation of the multidimensional semantics

Following those guidelines, these specializations are introduced:

- A special entity set: "dimension level"

- Two special relationship sets connecting "dimension levels":
  - "fact" relationship set (n-ary)
  - "rolls-up to" relationship set (binary)

A "rolls-up to" relates two "dimension levels" where the second one represents a "higher level of abstraction". This kind of relationships define a Directed Acyclic Graph. Multiple hierarchies, alternative paths, and shared hierarchy levels for different analysis dimensions are allowed. A "fact" relates n different "dimension level" entities. There is not any restriction to different "facts" being related to the same "dimension level".

Since this model is based on E/R, "dimension levels" and "facts" would have attributes, which are identified as **Descriptor**s and **Measure**s at **Lower** detail level. The "dimension levels" are clearly placed at **Intermediate** detail level, as well as "facts". Finally, a **Fact** would correspond to what is called a "multi-cube model". At this level we also find implicit **Dimension**s (a hierarchy of "dimension levels").

Sánchez, Cavero, de Miguel, and Martínez (IDEA)

Their authors claim that the aim of [SCdMM99] is to present a conceptual multidimensional model allowing to design multidimensional databases independently of the specific product used in their implementation. Besides the model, a closed algebra is defined with the following operations: "roll-up", "join", "destroy dimension", "slice and dice", and "select". A methodology and CASE tool are also mentioned.

A multidimensional schema is defined as a non empty set of "domains", set of "domain aggregations", set of "hierarchies", and non empty set of "fact schemas". Three different kinds of "domains" are distinguished (i.e. "dimension domain", "synthesis domain", and "description domain"). Furthermore, a "hierarchy" is a set of "domain aggregations" between "category domains" (a subclass of "dimension domain") linked to shape a directed graph. A "fact schema" is a set of "dimension attributes", set of "dimensions" (i.e. a subset of that of "dimension attributes"), structure of the cell, and predicate (showing the selected cells). Every "cell structure" is described as a list of "synthesis attributes" plus an attached list of applyable "synthesis functions".

Measures correspond to attributes defined on "synthesis domains", while **Descriptor**s are those attributes defined on "description domains". At **Intermediate** detail level, we find that every **Level** corresponds to a "dimension attribute", and **Cell**s are called "cell structure". Different "dimension attributes" are related by "aggregation functions" giving rise to **Dimension**s. Each "fact schema" contains exactly one "cell structure". However, different "fact schemas" are related. We could identify a **Fact** as a "multidimensional schema" containing a set of related "fact schemas" sharing **Dimension**s.

Tryfona, Bushorg, and Christiansen (starER)

In [TBC99], firstly a set of user requirements for a "data warehouse conceptual model" is listed. Then, a data model (based on the well known E/R model) addressing those requirements is defined. The requirements are:

1. Represent "facts" and their "properties". Three different kinds or "properties" are considered (i.e. "stock", "flow", and "value-per-unit")
2. Connect the temporal dimension to "facts"
3. Represent "objects", capture their "properties" and "associations" among them. Three different kinds of associations are highlighted:
   (a) Specialization/Generalization
   (b) Aggregation
   (c) Membership, characterized by strictness (or, not) and completeness (or, not)
4. Record the "associations" between "objects" and "facts"
5. Distinguish "dimensions" and categorize them into "hierarchies" ("dimensions" are those "objects" connected by an "association" relationship to a "fact")

Based on those requirements, the constructs of the model are "Fact set" that represents a set of real-world facts sharing the same characteristics or properties; "Entity set" which represents a set of real-world objects with similar properties; "Relationship set" that represents a set of associations (of any kind out of the three aforementioned, namely "Specialization/Generalization", "Aggregation", and "Membership") among "entity sets" and "fact sets" (any cardinality is allowed - i.e. 1:N, N:1, and N:M); and "Attribute" which represents a static property of "entity sets", "relationship sets", or "facts sets", which can be of any of the three kinds mentioned above, namely "stock", "flow", and "value-per-unit".

Placing those constructs in the three detail levels, we can see implicitly defined a **Dimension** at **Upper** detail level as a set of related "entity sets". Aggregation hierarchies in **Dimension**s are defined by means of "Membership" relationships. Those "entity sets", besides "fact sets", would respectively play **Level**s and **Cell**s roles at **Intermediate** level. Finally, their "attributes" would be **Measure**s and **Descriptor** at **Lower** level. Three different kinds of relationships are allowed between **Level**s at **Intermediate**: "Specialization", "Aggregation", and "Membership". Moreover, any cardinality is allowed for the relationship between a **Level** and a **Cell**.

Nguyen, Tjoa, and Wagner conceptual multidimensional data model

The multidimensional model, presented in [BTW00], uses the Object-Oriented paradigm to represent its metamodel. Specifically, UML is used in a schema which models multidimensional data and metadata all together. For instance, this schema contains a class "Dimension", and another class "MeasureValue".

The "dimension members" form a "hierarchical domain" which partitions them into "dimension levels", that belong to a "dimension". In turn, "measures" are integer or float values grouped into "cells", grouped into "groupbys", where every cell conforms with a "groupby schema". Each "groupby schema" refers to a set of "measure schemas", and "dimension levels"; where "measure schemes" indicate aggregability of "measures", and "dimension levels" show the granularity of the "measures".

Since data and metadata are defined at the same level, multidimensional schemas have a predefined structure, and **Measure** domains and **Descriptor**s cannot be defined. Therefore, it can be considered that this model does not allow the representation of any kind of information at **Lower** detail level. However, at **Intermediate**, we find **Level**s and **Cell**s (i.e. "groupbys"); and at **Upper** we find **Fact**s (as the "groupby schemas" associated to a cube schema) and **Dimension**s (called "dimension schemas").

Hüsemann, Lechtenbörger, and Vossen conceptual warehouse design

[HLV00] presents a phase-oriented Data Warehouse design methodology, which systematically derives schemas in "generalized multidimensional normal form".

Those schemas contain "dimensions" structured in terms of one or more "aggregation paths" (which could be "alternative" or "optional") that share the same terminal "dimension level"; and a "fact", which is a set of "measures" determined by terminal "dimension levels" ("measures" functionally depend on "dimension levels"). The sets of "dimension levels" of different "dimensions" are assumed to be disjoint. Each one of those levels has a set of "property attributes" associated. A "fact schema" represents the dimensional context for a set of "facts" that share the same terminal "dimension levels". Summarizability is also shown by relating "measures" and "dimension levels" to a "restriction level" indicating the aggregation functions allowed.

This model has **Measure**s and **Descriptor**s at **Lower** detail level, which are respectively grouped into **Cell**s and **Level**s at **Intermediate**. However, while **Level**s are grouped into **Dimension**s based on the meaningful "aggregation paths", **Cell**s are not grouped if they are not sharing the same terminal **Level**s. It is important to remark that summarizability is shown at **Lower** detail level (for each **Measure**).

2.3.2 Research efforts at Logical level

This section contains the work of those authors describing a model which is neither **Conceptual**, nor **Physical**. Their constructs are clearly oriented to a given kind of DBMS. Nevertheless, they are not that far from users conceptions. At this level, we can find the following papers: [Kim96], [BSH98], [MTW99], [GLK99], and [MK00].

Kimball multidimensional model

Doubtless, the most prominent work at this design level is [Kim96]. It describes the implementation of the multidimensional model on a Relational DBMS. Its explanations are not specific of any DBMS, like could be Microsoft SQL Server, nor discusses subject such the most appropriate kind of indexes, partitions of a table, or retrieve algorithms. Therefore, it should not be considered a **Physical** model.

Author	**Upper** detail level F	D	Relationships	**Intermediate** detail level C	L	Relationships	**Lower** detail level M	d	Relationships
Kimball	–	✓	Ds shared by Cs	✓	–[1]	FK between C and Ds	✓	✓	d ∈ D M ∈ C
Buzydlowski et al.	–	✓	Ds shared by Cs	✓	✓	FK between C and Ds FK between Ls	✓	✓	d ∈ L M ∈ C
Mangisengi et al. (NR)	✓	–	–	✓	–	C ∈ F	✓	–	M ∈ C
Mangisengi et al. (ER)	–	✓	Ds ∈ C	✓	✓	Ls ∈ D	✓	✓	d ∈ L M ∈ F
Gopalkrishnan et al.	–	✓	Ds shared by Cs	✓	✓	Pointers from C to Ds Pointers between Ls	✓	✓	d ∈ L M ∈ C
Moody et al.	✓[2]	✓[2]	Star schemas share Ds	✓	✓	Ls form hierarchies Cs form hierarchies "one-to-many" between Cs and Ls	✓	✓	d ∈ L M ∈ C

[1] They are implicitly defined by **Descriptors** in each **Dimension**.
[2] Implicitly defined by existing hierarchies.

Table 2.2: Schema constructs in the different models at **Logical** level

In this book, Ralph Kimball presents some multidimensional design patterns, and describes how they could be tackled. Some efforts have been done to improve Kimball's work. [BSH98], or [GLK99] show two Object-Oriented approaches.

The "star join schema" is defined as composed by a huge central "fact table", and a set of usually smaller "dimension tables" surrounding it. The primary key of the "fact table" is composed by a foreign key to each one of the primary keys of the "dimension tables". The "fact table" contains "numerical measures" (usually continuously valued, and additive), while "dimension tables" have "attributes" (usually textual, and discrete). The "dimension tables" can be shared by different "fact tables" giving rise to a "data warehouse bus" architecture, as explained in [KRRT98].

The possibility of normalizing the "dimension tables" (obtaining an "snowflake schema") is presented as an option that should be avoided. It would allow to explicit dimension hierarchies. However, the saved space is irrelevant, while query performance is really worsened (a series of joins become necessary), and browsing into dimension attribute values is more difficult.

Kimball's model does not define any explicit aggregation hierarchy or **Level**s, but they are implicit in the **Descriptor**s. Moreover, the "fact table" represents a given **Cell** related to its **Dimension**s by foreign keys. At **Lower** detail, we find **Descriptor**s, as well as **Measure**s.

Buzydlowski, Song, and Hassell (O3LAP)

[BSH98] draws the advantages of an O3LAP approach as opposed to ROLAP and MOLAP. It presents a direct translation from Kimball's model into the Object-Oriented paradigm. Instead of using relational tables, the usage of object classes is proposed. Only two new concepts are introduced (i.e. "dimension non-associative classes", and "dimension associative classes") in order to distinguish those analysis dimensions with and without an explicit hierarchy, respectively.

Thus, its constructs are those of Kimball's model plus the possibility of expliciting **Level**s within a **Dimension**.

Mangisengi, Tjoa, and Wagner (Nested Relations and Extended Relational)

[MTW99] introduces and compares two different approaches to multidimensional modeling (notice that there are two entries in the summary table for these authors). The ideas of those approaches are based on "nested relations" (Non-First Normal Form Relations), and the extension to the Relational model introduced in [Cod79].

A "nested relation" is a Relation whose attributes may be other Relations. By nesting Relations, we can reflect the different detail levels in the fact measurements. Therefore, we will obtain a **Fact**, at **Upper** level, as a Relation; different "nested relations" corresponding to **Cell**s at different detail levels; and finally, the **Measure**s for each **Cell**.

On the other hand, Codd's extension to the Relational model uses concepts like "object identifiers" (OIDs), "associations", or "object types". Moreover, it allows new operations like "patt", which partitions a Relation based on a given attribute. A "fact relation" can be modeled as an association relation with participating "dimension relation" types, containing OIDs of dimension tuples. Each "dimension relation" type could further be refined by other characteristics (expliciting the aggregation hierarchy) in the same way (having OIDs as attributes). Thus, at **Upper** detail level, we would have the **Dimension**s. At **Intermediate** level, each **Dimension**, contains identifiers of its **Level**s, which contain identifiers of finer levels, and so on. A "fact relation" would correspond to a **Cell** at this level. Finally, every **Level** contains **Descriptor**s, and every **Cell** contains **Measure**s.

Gopalkrishnan, Li, and Karlapalem (Object-Relational View)

[GLK99] also presents an Object-Oriented approach to multidimensional modeling. It not only describes a data model, but a methodology to build a Data Warehouse from Relational data sources.

A translation from Kimball's "snowflake schemas" to an Object-Oriented model is provided. The poor browsing performance in this kind of schemas, outlined by Kimball, is avoided here by using a "Structural Join Index Hierarchy" mechanism. A one-to-one mapping from Kimball's tables to object classes is defined. Foreign keys are translated to "Object Identifier pointers".

By these means, we obtain **Dimension**s as a hierarchy of **Level**s related by object pointers. At **Intermediate** detail level, we also have **Cell**s, related to **Dimension**s by object pointers, too. **Cell**s as well as **Level**s contain attributes (i.e. **Measure**s, and **Descriptor**s respectively).

Moody, and Kortink design methodology

[MK00] describes a methodology to develop multidimensional "models" from E/R "models". The idea behind this work is to benefit the multidimensional design from the information already in the operational schemas. Different kinds of schemas can be obtained as result of the different steps (i.e. "flat", "terraced", "star", "constellation", "galaxy", "snowflake", or "star cluster"). All those kinds of schemas contain Relational tables and are based upon the duality fact-dimension. They are characterized by different levels of denormalization in either fact or dimension tables. Thus, for instance, one chooses whether to explicit **Level**s or not, by normalizing **Dimension**s, and place the information about aggregation levels in different tables. Different topologies are offered:

- "Flat schemas" contain the minimum number of "fact tables". They do not have any "dimension table", because they are collapsed (denormalized) into the corresponding "fact table". Moreover, some "fact tables" are also collapsed into more detailed ones if possible. They keep all possible joins precalculated.

- "Terraced schemas" contain all the "fact tables" without any "dimension table" (all them are collapsed). These schemas only precalculate star joins (i.e. those involving a "fact table" and a "dimension table").

- "Star schemas" contain "fact" as well as "dimension tables". However, they do not explicit dimension hierarchies, since they are collapsed into a single "dimension table".

- "Constellation schemas" consist of a set of "star schemas" with hierarchically linked "fact tables".

- "Galaxy schemas" consist of "star schemas" sharing "dimension tables".

- "Snowflake schemas" are "star schemas" with explicit dimension hierarchies obtained by normalization of "dimension tables".

- "Star cluster schemas" are "snowflake schemas" were we collapse those "dimension tables" that do not have a multiple hierarchy.

This methodology, in addition to **Measure**s and **Descriptor**s (at **Lower** detail level) being members of **Level**s and **Cell**s respectively, considers constructs to relate those **Level**s and **Cell**s. Different **Level**s can be related to form possibly multiple dimension hierarchies. Moreover, **Cell**s can be related to show fact hierarchies (i.e. different levels of detail). It is not explicitly said in the methodology, but, at **Upper** detail level, we can identify a **Dimension** as a set of "dimension tables" in the same dimension hierarchy; and a **Fact** as a set of "fact tables" in the same fact hierarchy. It is also explained what to do with "many-to-many" relationships, and "subtypes", since they could be found in a E/R model, but can not exist in a multidimensional one.

2.3.3 Research efforts at Physical level

In this section, those proposals that explain how a data cube could be implemented (i.e. stored, and/or retrieved) are placed. The proposals at this level do not only depend on the kind of DBMS, but also present which specific mechanisms it should implement.

At this level, only one paper about modeling was found: [Dyr96]. It could be surprising that there is only one paper in this section. However, at this level, proposals must be devoted to specific storage techniques instead of providing a true data model. Since modeling is a conceptualization by means of a given set of constructs, it is more suitable when we consider notions closer to the user. Thus, we could expect not to find any work in this section, but this one expresses how data should be stored besides some concepts to understand it.

Author	Upper detail level F	D	Relationships	Intermediate detail level C	L	Relationships	Lower detail level M	d	Relationships
Dyreson	–	✓	–	–	✓	"finer that" between Ls	–	–	–

Table 2.3: Schema constructs in the different models at **Physical** level

Dyreson

[Dyr96] explains how a sparse cube could be implemented in a MOLAP database by means of disjoint, complete "cubettes". An algorithm to retrieve an aggregate value from the incomplete data cube is described, besides another algorithm to remove redundant "cubettes".

A "measure" is defined as a system of measurement, and a "unit" as a subset chosen from the domain of interest. Thus, a set of disjoint "units", chosen from the same domain, form a "measure". A partial order is defined among "measures" based on their granularity or precision. A "cubette" is defined as containing data about a given "unit", at a given detail level (i.e. "measure").

Levels (called "measures") and hierarchies (defined as graphs of "finer that" relationships between "measures") are the only constructs provided in this framework, both at **Intermediate** detail level. There is nothing said about factual information.

2.3.4 Research efforts on Formalisms

In this section, those models mainly devoted to the definition of a multidimensional algebra and/or calculus are placed. Their stress is on formalization of multidimensional concepts rather than data modeling. These models do not pay too much attention to facilitate the capture of the specific user concepts. Since their focus is not in conceptualizing users ideas, we can see, in the summary table, that they do not offer as much constructs as other models. However, if we would take into account the expressiveness of the algebras, they might be as semantically rich as **Conceptual** models are. Modeling constructs are not taken into account, since studying the expressiveness of the operations is out of the scope of this work. At this level, we find the following models: [AGS97], [LW96], [DT97], [HS97], [GL97], [Vas00a], and [Ped00].

Agrawal, Gupta, and Sarawagi logical model

[AGS97] presents one of the first multidimensional models, and probably, one of the most referenced ones. In spite of its qualification as "logical" by the authors, since its focus is on presenting an algebra as powerful as Relational algebra, it can be considered a **Formalism**. The main characteristics of this model are the following:

- Symmetric treatment of factual and dimensional data by providing conversion operations from one to another.
- A minimal, closed set of operations (i.e. "push", "pull", "destroy dimension", "restriction", and "join") which can be directly translated to SQL.
- Support for multiple (non-explicit) hierarchies along each analysis dimension.

Author	Upper detail level			Intermediate detail level			Lower detail level		
	F	D	Relationships	C	L	Relationships	M	d	Relationships
Agrawal et al.	–	–	–	√	–	–	–[1]	√	C = tuple of ds or C = boolean
Li & Wang	–	√	cube=<D,set of d>^{n} Cubes share Ds	–	–	Aggregation hierarchies defined at query time	–[2]	√	–
Datta & Thomas	–	–	–	√[3]	√	set of M and D ∈ "cube"	√	√	$f : D \rightarrow$ set of ds
Hacid & Sattler	–	√	–	–	√	Part-whole between Ls	√	√	$f : d^n \rightarrow$ M Aggregated concepts
Gyssens & Lakshmanan	–	–	–	√[4]	√	set of Ls ∈ "cube"	√	√	$f : D \rightarrow$ set of ds
Vassiliadis	–	√	"basic cube" uses Ds	√	√	Ls form a lattice	√	–	C = tuple of Ms
Pedersen	√	√	F and Ds ∈ "cube" **Cube**s sharing **Dimension**s form a "multidimensional object family"	–	√	Ls form a lattice Applicability of aggregation functions per L	√[5]	–	–

[1] Due to the desired symmetry fact-dimension, everything is considered a **Dimension**, and the function from the cartesian product of the **Dimension** domains is defined on the booleans rather than on **Measures**.
[2] There is not any information about **Measures** in the schema. A function is defined from **Dimension**s to a set of scalar values.
[3] Implicit on defining a cube as containing a set of **Measure**s.
[4] Those attributes that are not at any **Level**, must be in the **Cell**.
[5] **Dimension**s are treated as **Measure**s.

Table 2.4: Schema constructs in the different **Formalisms**

This model distinguishes a "cube" composed by k analysis dimensions, a function from k parameters to the booleans (or tuple of values), and a name for each analysis dimension. It does not provide any means to explicit dimension hierarchies. Moreover, the only way to show that there are several values in the cells of a data cube is by defining tuples. However, the model does allow to show which tuple of values is available depending on the selected dimension values.

This approach does not offer too many conceptual elements to model a multidimensional schema. Actually, it just provides **Descriptor**s (in the form of dimension values) without any possibility of even grouping them into different **Dimension**s. At most, we could consider that it allows to group **Measure**s into tuples giving rise to **Cell**s.

Li and Wang (Multidimensional Data model)

In [LW96], its authors define a "Formal Multidimensional Data" (MDD) model for OLAP systems. At the center of their approach is the notion of "multidimensional cube". They also define a "Multidimensional Database" (MDDB) as a set of "multidimensional cubes" and a finite set of Relations.

A "multidimensional cube schema" is a set of pairs "dimension name", "set of attribute names". Thus, a "multidimensional cube" is a "multidimensional cube schema" and a mapping from a combination of tuples containing the attribute values (one for each analysis dimension) to a scalar value. There is not any kind of information in the schema at **Intermediate** detail level, and aggregation hierarchies are not explicitly defined, but dynamically fixed at query time by means of ordering operations. However, "multidimensional cubes" in the same MDDB share dimension Relations. This means that, if two "multidimensional cubes" have the same dimension name, they are using the same dimension Relation.

Besides a formalism for "multidimensional cubes", they also present a "grouping algebra", which is used to query the MDDB; and a "multidimensional cube algebra", used to query a MDDB and generate views. A novel feature of the "grouping algebra" is that it includes order-related operations. The set of operations provided by this algebra are those of the Relational algebra, plus some order-oriented operations, and an aggregation operation. The multidimensional cube algebra offers six operations that are mappings from "multidimensional cubes" to "multidimensional cubes" (i.e. "add dimension", "transfer", "union", "cube aggregation", "rc-join", and "construct").

We can see the conceptual elements provided by this model as **Dimension**s at **Upper** detail level, stating an implicit relation among different "multidimensional cubes" possibly sharing a **Dimension**; and deaggregating **Dimension**s into **Descriptor**s at **Lower** detail level. Since the mapping function between **Dimension**s and **Measure**s is defined on a scalar value (without any kind of semantic domain), we could say that the proposed model does not provide any means to represent **Measure**s.

Datta and Thomas

The model of [DT97] resembles that of [AGS97]. The three goals of the authors on offering their model are to:

1. Allow symmetric treatment of dimensional and factual data.
2. Separate structure and contents.
3. Provide comprehensive OLAP functionality.

The authors define a "data cube" as a set of "dimensions"; a set of "measures"; a set of "attributes"; and a mapping function corresponding to each "dimension", a set of "attributes". So, they neither define explicit hierarchies, nor the set of **Measure**s available at each aggregation level.

By defining "cube-instances", they accomplish their second goal. A "cube-instance" is a "data cube" plus a set of values, plus a mapping from the cartesian product of the "dimension" domains to the values. Moreover, a set of operations (i.e. "restriction", "aggregation", "cartesian product", "join", "union", "difference", "pull", and "push") is defined on "cube-instances". Operations "push" and "pull" are used to accomplish the first goal.

In this case, we can clearly see elements at different detail levels. At **Lower** level, we find **Descriptor**s and **Measure**s. While at **Intermediate** level, we find the set of **Dimension**s (as sets of **Descriptor**s), each corresponding to exactly one **Level**; and the set of **Measure**s in the "data cube" (implicitly, the **Cell** corresponding to the unique aggregation level in the "data cube").

Hacid and Sattler description logics framework

[HS97] propose an object-centered, logical framework (i.e. Description Logics) for multidimensional data models. Their aim is to facilitate comparison or evaluation of different multidimensional models, provide well defined semantics, and allow precise definition of problems. A translation between an Extended E/R diagram and Description Logics is given in [FS99].

By means of Description Logics, the authors represent a data cube as a relationship among cells, which keep the coordinates and measures. Every cell in a data cube must have the same structure. The functional dependency between coordinates and measures is explicitly shown. Beside data cubes, dimension hierarchies can also be modeled. A "hierarchically structured dimension" is a set of objects interrelated by part-whole relationships. Thus, a hierarchy is represented as a finite partially ordered set.

Moreover, a set of operations on data cubes is also defined. In this case, those operations are "restrict", "destroy", "join", "rename", "Join" (which offers more parameters than "join"), "aggr", and "roll-up". Furthermore, a whole section is devoted to the problems of the "drill-down" operation.

This is a semantically powerful model. However, some multidimensional modeling mechanisms are not explicitly explained or exemplified (i.e. the participation of dimension hierarchies in the definition of a data cube, or the usage of complex concepts). At **Upper** detail level, **Dimension**s could be modeled as the set of concepts participating in classification hierarchies, in spite of it is not explicitly said. At **Intermediate** level, those hierarchies are decomposed into different aggregation concepts at different levels, related by part-whole relationships. Finally, at **Lower** level, we find **Measure**s and **Descriptor**s that can be aggregated into more complex concepts.

Gyssens, and Lakshmanan multi-dimensional database model

As some authors before, [GL97] also define some required functionalities, and drive their model to fulfill them:

1. Ability to pose powerful ad-hoc queries through a simple and declarative interface
2. Ability to restructure information
3. Ability to classify or group data sets
4. Ability to summarize values

To accomplish these goals, the authors propose a Relational approach, and define an "n-dimensional table schema", as a triple containing a "dimension name" set; an "attribute" set; and a function from "dimension names" to "attribute" set, showing the attributes of each analysis dimension. From this definition, they develop an algebra based on the Relational algebra. Apart from the redefinition of classical Relational operators, the authors add other operators like "fold" and "unfold" (in order to remove and add a **Dimension** to the schema, respectively); and a summarization and aggregation functions. Operators "fold" and "unfold" allow to convert **Measure**s into **Descriptor**s and vice versa, since the attributes of the disappeared **Dimension** remain in the data cube as **Measure**s. Therefore, the model allows a symmetric treatment of both of them. Moreover, it shows that every multidimensional table can be represented by a classical Relation and vice versa.

With regard to the modeling elements provided, we can distinguish **Descriptor**s and **Measures** at **Lower** level. At **Intermediate** level, the **Descriptor**s form **Level**s. If we subtract the **Descriptor**s from the set of all attributes in the data cube, we could also consider implicitly defined a **Cell**.

Vassiliadis

[Vas98], and [Vas00a] present another formal model for multidimensional data, besides its mapping to ROLAP and MOLAP databases. Here, the cube algebra (demonstrated to be complete and sound) consists of just three operations (i.e. "navigate", "selection", and "split measures").

For each dimension of analysis, a set of "levels" is defined, forming a lattice (bounded by "All" at top, and the "detailed level" at bottom). A "dimension" consists of a set of "dimension paths", which are totally ordered lists of "dimension levels". A "dimension level" belongs exactly to one "dimension", and has an associated space of values. The "dimension levels" can be monovalued or multivalued, whether their domain is a set or a power set of the space of values.

A "MDDB" is defined as a set of "dimensions", "dimension levels" and a "basic cube". A "basic cube" contains the data cells at the maximum level of detail. Over this, by mean of the cube algebra, other cubes (we could call views) are defined. The existence of the "basic cube" is justified by the impossibility of performing the drill-down operation without it.

In this model, a **Dimension** (at **Upper** detail) is composed by a lattice of **Level**s (at **Intermediate**). However, the "dimension levels" do not contain further details at **Lower**. If we look at **Cell**s, which are not explicitly defined, we see that they are a tuple of **Measure**s identified by the bottom levels of the different dimension lattices.

Pedersen (Extended Multidimensional Data Model)

Besides a classification of multidimensional models, [PJ98], [PJ99], and [Ped00], already referenced in section 2.2.1, also present an "Extended Multidimensional Data Model" (EMDM). After the definition of the requirements (most of them refer to semantics) for the usage of a multidimensional model in a clinical context, and the verification that none of the existing models addresses all of them, this new model was defined.

EMDM provides a formalism and algebra that is closed and, at least, as strong as Relational algebra with aggregation functions. The operations in the algebra are "selection", "projection", "rename", "union", "difference", "identity-based join", "aggregate formation", "value-base join", "duplicate removal", "SQL-like aggregation", "star-join", "drill-down", and "roll-up". The implementation of the model using Relational databases is also explained.

An "n-dimensional fact schema" consists of a "fact type", and n "dimension types". In turn, a "dimension type" consists of a set of partially ordered "category types" forming a lattice. To each "category type", an "aggregation type" has been associated, indicating the aggregate functions applicable at that level. The model treats dimensional and factual data symmetrically. Multiple hierarchies per analysis dimension, non-strict hierarchies, non-onto hierarchies, non-covering hierarchies, or many-to-many relations between facts and dimensions are allowed. However, there is no way to reflect such information in the schema. Instead, it is deduced from data instances. Moreover, relating values that represent the "same" concept along time is also possible thanks to temporal constructs.

The semantic constructs offered by the model are **Dimension**s and **Fact**s at **Upper** detail level, and **Level**s at **Intermediate**. It cannot be considered that the model allows to show

Cells. Data in the **Fact**s can be related to any **Level**, however this information cannot be shown in the schema. At **Lower** level, we find that **Descriptor**s do not exist. **Cell**s do not have attributes neither, however, **Dimension** values are used as **Measure**s.

2.3.5 Other work

There are other papers about multidimensional interfaces, multidimensional query languages, etc. ([GJJ97], [GL98], [GBLP96], and [BPT97] among others) that also treat, as a minor subject, some kind of multidimensional model. These were left out of the classification, because the models have not any new or improved characteristics, neither was in the aim of the authors to present a multidimensional model.

Moreover, there is a lot of literature devoted to either ROLAP or MOLAP implementation ([HRU96], and [TS97] among others). For instance, they present different kinds of indexing techniques or partition strategies. They were not included in this survey (in the section about models at physical level), because they do not model the multidimensional data, but just give useful hints to obtain good storage or query performance.

Metadata standards, either de jure (like "Common Warehouse Metamodel") or de facto (like OLE DB for OLAP), have neither been considered, because they are not true data models. They do not aim to model the data cube, but to provide an interface that facilitates metadata interchange among OLAP applications.

2.3.6 Summary

Table 2.5 contains a summary of elements and relationships among them found at the schema level of each model (see section 2.2.2 and the beginning of 2.3 for the meaning of the different columns). Notice that information about either instances or instantiation relations is not shown. As outlined in [BSHD98], some models do not separate cube structure and contents. In these cases, only that information contained in the schema has been taken into account. A cell containing a hyphen means the corresponding model does not provide any construct in that context, while a tick implies the model does provide some kind of construct.

It seems that **Conceptual** models offer the possibility of representing much more semantics that models at other levels. Indeed, **Conceptual** models do have to provide a rich set of semantic constructs in order to capture user ideas. In turn, **Formalisms** are those that offer less conceptual constructs. However, notice they do offer an algebra whose expressiveness was not considered in this work, because the focus was on modeling constructs. At **Physical** level, we find storage techniques instead of true data models. Thus, just one **Physical** model was reviewed. Moreover, there was not found a great variety of models at **Logical** level.

Looking at the table we can appreciate that the more recent the models are (they are ordered chronologically into each design level), they use to capture more semantics. This can be interpreted as a trend to semantically enrich multidimensional models. However, having models that provide constructs at every heading does not mean they capture all possible semantics. There is neither a model encompassing the semantic constructs of the rest, nor a consensus or standard stating what should be represented in a multidimensional schema.

Authors (Model)	Design Level	**Upper**			**Intermediate**			**Lower**		
		F	D	Rel.	C	L	Rel.	M	d	Rel.
Lehner (NMDM)	C	–	√	–	–	√	√	–	√	√
Cabibbo and Torlone (*MD*)	C/F	√	√	√	–	√	√	√	√	√
Golfarelli et al. (DFM)	C	–	√	–	√	√	√	√	√	√
Trujillo et al. (GOLD)	C	√	√	√	–	√	√	√	√	√
Sapia et al. (MERM)	C	√	√	√	√	√	√	√	√	√
Sánchez et al. (IDEA)	C	√	√	√	√	√	√	√	√	√
Tryfona et al. (starER)	C	–	√	–	√	√	√	√	√	√
Nguyen et al.	C	√	√	√	√	√	√	–	–	–
Hüsemann et al.	C	–	√	√	√	√	√	√	√	√
Kimball	L	–	√	√	√	–	√	√	√	√
Buzydlowski et al. (O3LAP)	L	–	√	√	√	√	√	√	√	√
Mangisengi et al. (NR)	L	√	–	–	√	–	√	√	–	√
Mangisengi et al. (ER)	L	–	√	√	√	√	√	√	√	√
Gopalkrishnan et al. (ORV)	L/P	–	√	√	√	–	√	√	√	√
Moody and Kortink	L	√	√	√	√	√	√	√	√	√
Dyreson	P	–	√	–	–	√	√	–	–	–
Agrawal, Gupta, and Sarawagi	F	–	–	–	√	–	–	–	√	√
Li and Wang (MDD)	F	–	√	√	–	–	√	–	√	–
Datta and Thomas	F	–	–	–	√	√	√	√	√	√
Hacid and Sattler	F	–	√	–	–	√	√	√	√	√
Gyssens and Lakshmanan	F	–	–	–	√	√	√	√	√	√
Vassiliadis	F	–	√	√	√	√	√	√	–	√
Pedersen (EMDM)	F	√	√	√	–	√	√	√	–	–

Table 2.5: Summary table of the different multidimensional models

2.4 How multidimensional analysis benefits from O-O

Probably, the most important advantage of modeling the UoD (Universe of Discourse) by means of an O-O model is that the result is closer to the user conception, i.e. it naturally reflects people's way of thinking. Every object or class modeled will have a correspondence with some real entity, making it quite easy to be understood. We can also find other, not that abstract, benefits in the O-O paradigm:

Object-Oriented Software Engineering Since the O-O paradigm is widely used and well accepted in Software Engineering, it does not seem a good idea to break it by using a non-OO approach in data modeling. Moreover, an O-O data model eases some specific tasks like designing a Distributed Object System.

Non-First Normal Form (NF^2) It is not mandatory to have flat (normalized) entities. We can design objects containing non-atomic values. In some cases, this can be found really useful due to performance reasons, or just because, conceptually, it is not necessary to create an unrealistic entity only to normalize the schema.

Object Identifier (OID) The existence of an OID solves the identification problem. A key is not enough to identify an entity. We must consider the case when the primary key changes, and the identity of the object remains the same. In that case, an internal identifier, without any real meaning, is needed. It would keep the same value along the whole life of an object in the database, independently of any change in the represented entity. It is important to remember here two characteristics in the definition of W. Inmon, i.e. "non-volatile" and "time variant". Our data will evolve, and OIDs will be a useful tool keeping them consistent.

Semantics "Expressiveness" or "semantic power", as it is defined in [SCG91], is the degree to which a model can express or represent a conception of the real world. It measures the power of the structures of the model to represent conceptual structures, and to be interpreted as such conceptual structures. The most expressive a model is, the better it represents the real world, and the more information about the data gives to the user. An O-O model is semantically richer than others (for instance E/R or Relational). It is true we can enrich any of those others with O-O features, but why should we do that if we can use a true O-O model?

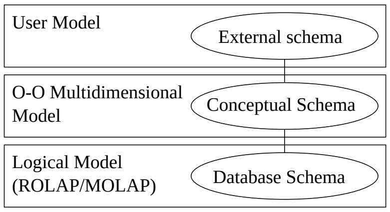


Figure 2.5: Database schemas at three levels

The aim of this section is just to outline how the O-O paradigm could be used to help multidimensional modeling by giving some examples. In figure 2.5, one can see the level where the discussion is placed. The interest is neither in the best user model, nor the best kind of database to use (either ROLAP -Relational OLAP-, MOLAP -Multidimensional OLAP-, HOLAP -Hybrid OLAP-, or even O3LAP -Object-Oriented OLAP- presented in [BSH98] could be good). The benefits of an O-O data model to integrate the different multidimensional views and keep the semantics of the data at the conceptual level are highlighted (this is shown in more depth in section 3.2.2). If the user wishes to use a different one, it could always be translated to the desired model. The same can be said about the internal level, the usage of an O-O multidimensional model does not imply we are storing the data in an O-O database.

The stress is on showing the need of using O-O semantic concepts. Six OO-Dimensions (i.e. Classification/Instantiation, Generalization/Specialization, Aggregation/Decomposition, Behavioural, Derivability, and Dynamicity) were enumerated in [Sal96]. Each one of these OO-Dimension adds a little of semantic power to a data model. We are going to see how each one of them helps multidimensional modeling, by allowing to represent different relationships among data.

Along this section, **nexus** stands for any relationship (tagged or not) between two objects. The **nexus** are specialized for every one of the OO-Dimension to obtain the different meanings. As can be seen in [AR00], **nexus** roughly corresponds to *Relationship* in UML terminology, which is not used here to be more general.

2.4.1 Classification/Instantiation

This OO-Dimension distinguishes between the occurrences and the schema. Every instance is related to, at least, a class in the schema by **nexus** in this OO-Dimension. All instances sharing some attributes, and representing related concepts are grouped into a given class. In the same way, all elements in a schema (i.e. classes, **nexus**, ...) representing related concepts in a data model are grouped into a metaclass. To finish the recurrence, all metaclasses can be grouped into exactly one metametaclass, which is instance of itself. Of special interest in this OO-Dimension, present in all data models in one way or another, is the dynamic and multiple classification, explained in [MO96].

Dynamic classification refers to the ability of the instances to change the class they belong to. If we want to analyze the sales depending on how good our clients are, and we have them classified into different classes, it will be a matter of time that we want to move a given client from a class to a different (hopefully better) one. We cannot delete the instance of `Client` in the database and create a new one in the new desired class, because we would get a new identity (OID) for it. That is not what we want to represent, since we did not lose a client and found a new one. It was just our consideration (classification) about a client that actually changed, and that is exactly what the data model should be able to represent. This is one case of "Slowly Changing Dimensions", where the change affects the classification of the object. The general problem is explained in section 2.4.6.

On the other hand, multiple classification refers to the possibility of having an instance classified in more than one class (not related by Generalization/Specialization **nexus**) at the same time. For instance, it is absolutely possible to have a client as provider at the same time. Since there is not any relationship between the `Client` and `Provider` classes, we need to have the same instance classified at both of them (multiply classified).

These characteristics are always desirable. Specifically in the field of data warehousing, the words "non-volatile" and "time variant", together with the OLAP need of analyzing relatively long periods of time, emphasizes their importance. Dynamic and multiple classification are really interesting due to the flexibility needed to represent the big amount of changes present along the long period of time that uses to be taken into account in analysis tasks.

2.4.2 Generalization/Specialization

Another OO-Dimension is that of Generalization/Specialization relationships (represented in UML by means of *Generalization*). The **nexus** in this OO-Dimension relate two classes (or metaclasses). One of those classes has a more specific meaning than the other. The more general class is called "superclass" with regard to the specific one, referred as "subclass". As a consequence of this kind of **nexus**, we obtain inheritance. That is, the subclass inherits the properties and methods of its superclass (or superclasses). If it is allowed to have more than one superclass, we gain multiple inheritance (a class inherits from all its superclasses at a time). Every class will have (besides its own attributes) the attributes and relationships of each one of its superclasses. Note this is absolutely different from multiple classification where an instance is classified in multiple classes.

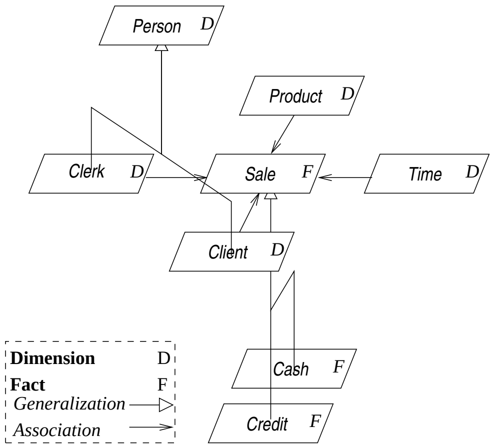


Figure 2.6: Example of Generalization/Specialization

In figure 2.6, we can see an example of a multidimensional schema. It has `Sales` as **Fact**, and `Clerk`, `Time`, `Product`, and `Client` as analysis dimensions. Thus, the subject of analysis is `Sales`, and we want to analyze it depending on the clerk who sold, the moment it was done, the product sold and the client who bought. Besides that basic information, other details are also represented by means of **nexus** in this OO-Dimension:

- the `Sales` **Fact** is specialized in two different **Fact**s (i.e. `Cash`, and `Credit`) depending on the kind of payment, and
- two **Dimension**s (i.e. `Clerk`, and `Client`) are related by generalizing them in the same class (i.e. `Person`).

Specializing **Fact**s, you can generate new data cubes (if they contain any different data), or, at least, show a criterion to select the facts involved in the analysis. In the example, if `Sales` would have different attributes depending on the kind of payment, we would obtain three different data cubes to be analyzed i.e. two containing the **Measure**s specific to each kind of payment, and another one with those **Measure**s shared by both of them. Conversely, if it would not have any other attribute but those common to both kinds of payment, we could analyze the `Sales` depending on whether the payment was done by cash, or by credit card. This could also be achieved by just adding an attribute to the facts, but it would give a slightly different tint.

With regard to relating two **Dimension**s, it shows a common domain between them, so that it is allowed to compare the instances, or restrict both *Classes* at the same time. In the example, the analysts could formulate queries comparing instances of `Client` and `Clerk`, because the data schema shows both as subclasses of the same class (i.e. `Person`). Moreover, we could consider the possibility of class `Person` being used in a different multidimensional schema, which would become directly related to that of `Sales` by means of the **nexus** between the **Dimension**s. This would point out the relationship between facts, easing the navigation through the data.

2.4.3 Aggregation/Decomposition

By means of this OO-Dimension, it is possible to build new objects as a result of the aggregation of others, which in turn can be aggregations, as well. Two different kinds of **nexus** can be distinguished belonging to it. Based on their strength, **nexus** in the Aggregation/Decomposition OO-Dimension can denote:

Part-Whole, if the new object is conceived as composed by others, which are its parts. This is called "part-whole" relationship by some authors, and implies an existence dependency between both sides of the **nexus** (i.e. the whole cannot exist without its parts). This is called *Aggregation* in UML terminology.

Simple aggregation, if the aggregating objects are just characteristics of the new one. They could have an existence dependency too, but it is not an implication of the existence of the **nexus** itself. This is called *Association* in UML terminology.

The usage of this OO-Dimension in multidimensional design is mandatory, since it helps to represent some of the most common situations, and other maybe not so common:

- Firstly, it helps to define the analysis dimension hierarchies by means of part-whole links. A dimension hierarchy can be defined as a lattice with the class corresponding to the maximum level of detail in the facts at the bottom, and a class representing the whole set of points in the **Dimension** at the top (see section 4.1 for an explanation of the properties of aggregation hierarchies). In between, we have other **Level**s corresponding to different data granularities. For instance, if we collect data hourly, the `Time` **Dimension** would have `Hour` class at the bottom, which would compose `Day` above it, which would give raise to `Week` and `Month`, and so forth. The lattice would be closed at the top by an `All` class containing exactly one instance representing all time points in the database. These hierarchies are used to roll-up the data in the database, augmenting its granularity.

Moving up (e.g. rolling-up from days to months) or down (e.g. drilling-down from years to months) along a hierarchy we obtain more or less detail in the data.

- On the other hand, using any kind of **nexus** in this OO-Dimension, we can relate either **Level**s or **Cell**s to their attributes. These attributes will be used to ease the selection of facts to be considered in a given analysis by allowing to select them depending on the values.

- **Nexus** between the **Cell**s and the **Level**s in every dimension hierarchy are in this OO-Dimension, as well. They could be part-whole or simple aggregation **nexus**, but whether denoting part-whole or not, a fact will be identified by one object at each linked analysis dimension (or more than one if the **Dimension** has more than one **nexus** with the facts class). Thus, the **nexus** with the analysis dimensions will form the class-key of the facts, and that is what really distinguishes them from other attributes. Sales can be identified by the product sold, the clerk who sold it, the time when it was sold, and the client who bought it. Therefore, in figure 2.6, we associate `Sales` with **Dimension**s `Clerk`, `Time`, `Product`, and `Client`.

- Finally, part-whole **nexus** can be found between classes of facts. By reflecting these **nexus** in the schema, we will also allow the navigation between different stars.

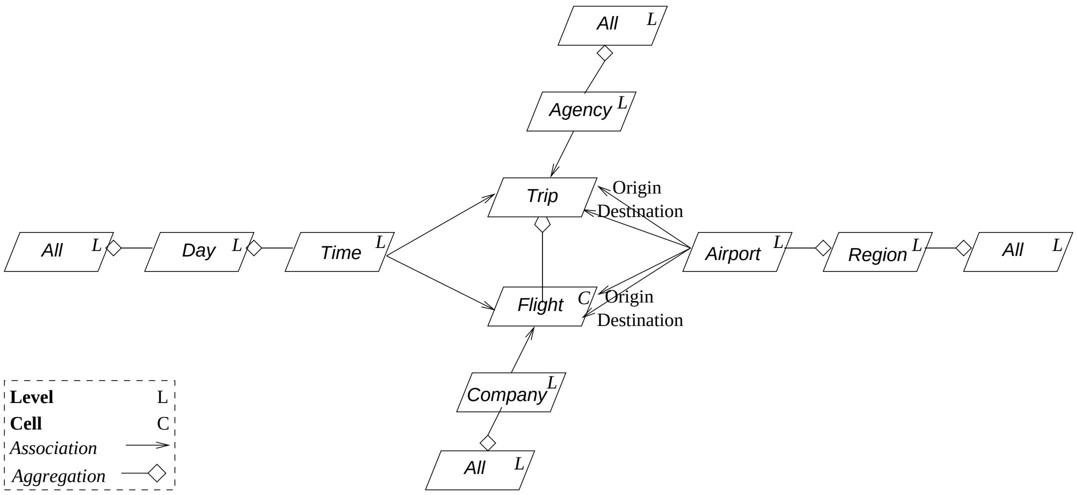


Figure 2.7: Example of Aggregation/Decomposition

The example in figure 2.7 depicts two classes of facts, sharing some analysis dimensions, and related by a part-whole **nexus**. The first class of facts is `Flight`. We are interested in analyzing each flight depending on the time it takes place, the airline company that owns the plane, and its origin and destination airports (it is related to the corresponding analysis dimensions by simple aggregation **nexus**). At the same time, we want to analyze the sequences of flights that give rise to whole trips sold by travel agencies. The fact that an instance of `Trip` is composed of a set of instances of `Flight` is represented by connecting `Trip` and `Flight` by means of a part-whole **nexus**. `Trip` is also connected to the corresponding analysis dimensions by simple aggregation **nexus**. Moreover, it is important to notice that two of those **Dimension**s contain more than one class, connected again by part-whole **nexus**. A region is composed of a set of airports, in the same way that a day is a set of time points.

In order to keep it simple and understandable, the example does not contain the **nexus** representing the attributes of the facts and dimension classes, which would belong to the Aggregation/Decomposition OO-Dimension, as well. The four classes at the top of each one of the analysis dimension hierarchies (i.e. `All`) always exist and contain exactly one instance corresponding to the whole set of instances in the lowest granularity level of the **Dimension**.

2.4.4 Behavioural (Caller/Called)

In O-O, the objects interchange messages. A class accepts certain kinds of messages from instances of other classes, which trigger the execution of methods (i.e. queries, updates, calculations, etc.). The **nexus** in this Behavioural OO-Dimension, also known as "Caller/Called", show when a class is allowed to invoke a given method in another class. This concept could be identified as a kind of *Permission* in UML terminology.

As pointed out in [Fir97], Relational entities represent tables, purely passive containers for data, and since they are not real objects, are independent of behaviours. The inclusion of methods in the data model helps to model the behaviour together with the data. It looks like a bad idea to have two different, separated models for statics and dynamics. Specifically, in multidimensional modeling, by associating operations to a domain, we would be able to know which aggregation functions can be used on a given **Measure**. For instance, as explained in [GMR98a], we can find semi-additive attributes (those that are not additive along one or more **Dimension**s), or non-additive attributes (which are additive along no **Dimension**). `Temperature` should be marked as non-additive (nobody could call an additive method on it), and `InventoryLevel` as semi-additive, since it cannot always be added (e.g. along `Time` dimension). It does not imply that other aggregation operations could be applied on those **Measure**s. Therefore, we need to show the applicability of every different operation.

Moreover, methods facilitate the implementation of complex aggregate functions. In an analysis environment, it is important to keep track of the way the **Measure**s are obtained. It is not advisable to allow the users to implement their own ad hoc functions. It is error prone, and drive to misunderstandings. O-O concepts such as inheritance, polymorphism, or encapsulation perfectly fit at this point. For instance, suppose we would like to obtain the delay of a flight, defined as the difference between the expected and real durations (actually, not a complex function). The problem could arise if the expected duration of the flight were kept as a time interval. If this is the case, the difference could be done by subtracting the minimum, maximum, or even midpoint expected duration, which result in completely different values. Probably, it does not matter how the result is obtained, but we must ensure it is always calculated in the same, easy to change way to be able to compare the obtained values among different users or even sessions.

Leaving those considerations aside, this OO-Dimension is also important because of security reasons, but that is completely out of the scope of this thesis.

2.4.5 Derivability

"Semantic Relativism" of a data model is defined in [SCG91] as the degree to which the model can accommodate not only one, but many different conceptions. It is really important because since different persons perceive and conceive the world in different ways, the data model should be able to capture all of them. This is represented in UML by means of *Derivation* relationships.

The Derivability OO-Dimension, also known as "Point of View", helps to represent the relationships between abstractions in different conceptions of the UoD. The database does not need to physically keep all those conceptions, but only their definitions and different relationships among them. In general, it is not good to store derived data (unless because of performance reasons, not considered by now). What we do really need to store is that derived data exists and how it is obtained. Here is the importance of this OO-Dimension. Derivation mechanisms can be used to easily restructure the schemas to show them in the way the user wants, in order to be closer to his/her thoughts. Summing up, Derivability OO-Dimension is used to define derived data.

Some analysts do not mind whether data are atomically stored in the database or not. In this sense, it is desirable that either derived or atomic **Measure**s are treated equally. However, others would like to know how **Measure**s are obtained. Therefore, the definition of the derived **Measure**s should be in the schema of the database, as Relational views are. It allows either to hide the complexity, or to know where something comes from, depending on the user needs. At the same time, as in the Behavioural OO-Dimension, this also makes possible that groups of users have available the same definitions.

In multidimensional modeling, it is specially important to have the powerful possibilities offered by this OO-Dimension. When a fact is being analyzed, what really matters is to be able to see it from as many points of view as possible. Therefore, it is crucial to have the mechanisms to define those different views of the data. For instance, all summarized data are related to their detail data by a **nexus** in this OO-Dimension. If we did not have it, we would not have any kind of summarized data. It is necessary to show such **nexus** at conceptual level to understand the real meaning of data and where they come from.

Going back to the example in figure 2.7, we can see that the `origin` of a `Trip` would be derived from the `origin`s of the `Flight`s that compose it, by taking the first one; `destination` would be defined in the same way; the duration of a `Trip` would be function of the duration and taking off times of the different `Flight`s; and so on.

2.4.6 Dynamicity

This OO-Dimension refers to changes along time. These changes can be considered at three different levels:

Object Objects are created, deleted, and also updated. Keeping the history of those updates is often referred as "Versioning".

Class As well as the objects, the data schema can be updated, too. New classes are created, old ones are deleted, and others just modified in what is called "Schema Evolution".

Metaclass In the same way we can modify classes, we can add new metaclasses (notice we can neither modify nor delete them). This means having an "Extensible Data Model".

If we just wanted to represent the current reality, we would not need to consider Dynamicity OO-Dimension. However, it is common to need past states. Therefore, changes need to be kept, and often stamped with some kind of time tag to know when they happened.

In multidimensional analysis tasks, time is an omnipresent dimension. Moreover, to make things worse, analysts frequently consider a scale of years. If we add how fast things change nowadays, we can see the importance of this OO-Dimension for multidimensional modeling. It is almost impossible to find a business that has not changed at all in the last three or five years, and those changes must be reflected in the corresponding information system. Leaving aside changes in metaclasses, we want to see the need of considering the other two kinds of changes (i.e. those in objects and classes).

The importance of user requirements makes schema evolution an important issue. When the user requirements or conceptions change, it is advisable to change the data schema in accordance with them. A change in a class or **nexus** should be shown in the schema by connecting the old and the new version with a Dynamicity **nexus**. By doing it, the analysts can easily see the available data, and the meaning of the results they are obtaining. For instance, when the definition of a derived **Measure** changes, the analyst is able to compare the results using the new and the old definitions. Moreover, if some attribute is not kept any more, or a new one is added, the analist can know whether it can be queried or not at a given point in time.

A special case of changes in the data is referred in [Kim96] as "Slowly Changing Dimensions". It arises when attributes in analysis dimension classes are modified. The old values must be kept, because the facts previous to the change are probably still related to them, while the new ones will be referred by the facts occurring from now on. However, both instances represent the same entity in reality, and it has to be outlined by a **nexus** between them. Clearly, if an airport increases its number of tracks, it would be incorrect to analyze the air traffic previous to the enlargement with regard to the new number of tracks. Therefore, we need to have two instances of the same airport related by a Dynamicity **nexus** showing that they represent the same object.

Studying the storage of versions of data is completely out of the scope of this thesis. The point here is to outline the importance of reflecting changes in the schema, so that they can be taken into account by analists. This is similar to the schema evolution problem, but worsened because we need to keep the old schema.

2.5 Conclusions

In this chapter, a framework that allows us to classify and compare multidimensional models has been presented, and exemplified by studying some representative models. There exist previous studies comparing different multidimensional models (see section 2.2.1). However, those studies intended to show their lacks against a given list of requirements, and models for absolutely different purposes were put into the same bag. On the contrary, here, research efforts have been classified into different levels (i.e. **Conceptual**, **Logical**, **Physical**, and **Formalisms**) based on their usage in the multidimensional design process, or if they are not conceived for such process. Furthermore, a framework was given to compare the terminology used by different authors for the constructs of their models.

Along this chapter, the questions of multidimensional modeling have been introduced. Probably because of the interest of the industry in the subject, it is being mainly developed in a specially commercial way. This means stressing performance, and passing over semantics and conceptual modeling. Multidimensional semantics are really important because of their proximity to the inherent structure of the problem domain, but they are not the only ones to be represented. Other semantics should not be overlooked. It is not enough having an isolated multidimensional schema reflecting how the user will access the information, leaving aside the representation of other data relationships.

The applicability of six OO-Dimensions (i.e. Classification/Instantiation, Generalization/Specialization, Aggregation/Decomposition, Behavioural, Derivability, and Dynamicity) to semantically enrich multidimensional schemas has been shown by exemplifying it. This is a really important point since, as shown in this chapter, most authors consider isolated stars composed by a central fact table, and different flat, denormalized dimension tables arranged around it, each one related to the central table by a foreign key. That is not a bad idea at all, but there is much more information about the data subject of analysis that the schema could contain, which would be really useful to the analysts, users of the multidimensional system.

For the sake of simplicity and understandability, the stars use to be represented in an isolated manner. The necessity of providing an overall view of the data has been stressed. Multidimensional analysis is used in decision making processes. Therefore, the most global view is provided, the more the schema helps the users. It is really important to offer an integrated vision of the business or subject of analysis, in order to give the analists a unified set of data instead of lots of puzzle pieces. The proposal is to relate the puzzle pieces by means of **nexus** in the different OO-Dimensions. Thus, existing multidimensional models are not enough.

Chapter 3

Multi-level schemas architecture

> "For the fashion of Minas Tirith was such that it was built on seven levels, each delved into a hill, and about each was set a wall, and in each wall was a gate."
>
> J.R.R. Tolkien, "The Return of the King"

In this chapter, an architecture of seven schema levels for "Federated Information Systems" (FIS) is related to "Data Warehousing" schemas, which allows to provide better understanding to the characteristics of every schema, as well as the way they should be defined. Because of the confidentiality of data used to make decisions, and the federated architecture used, data protection issues are also mentioned.

Navigation among different stars is usually overlooked in literature. Thus, this chapter studies different kinds of conceptual relationships between stars (i.e. Derivability, Generalization/Specialization, Aggregation/Decomposition, and Dynamicity), and analyzes how they fit into the schemas architecture. The aim is to ease the implementation and usage of multi-star schemas.

Firstly, section 3.1 presents the integrated schemas architecture for FIS and DW. Then, section 3.2 shows different semantic possibilities to relate stars, and how this affects the schemas architecture.

3.1 Extending a schemas architecture for "Data warehousing"

DW is a relatively new area of study. The idea behind this section is to use the advances already done in other subjects to benefit it. The presence of the integration concept in the definition of a DW given in [Inm96] invites to choose FIS as a first class candidate to contribute its advances. Specifically, the location of DW schemas in an architecture for FIS is proposed, which allows to study them from this point of view.

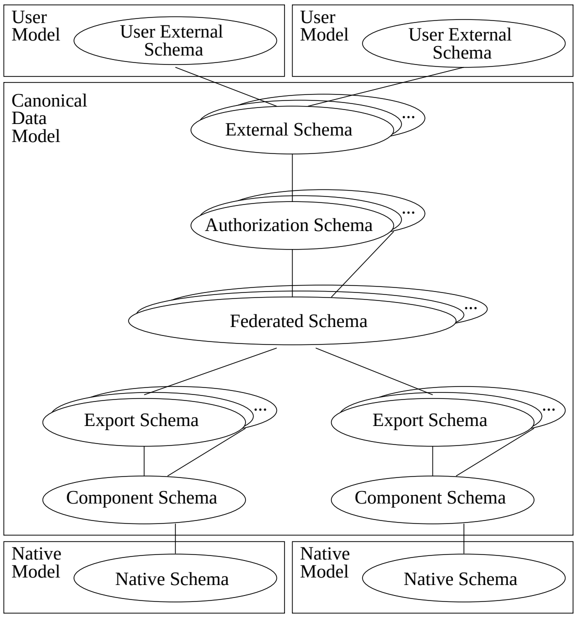


Figure 3.1: 7-levels schemas architecture [ROSC97]

The architecture of seven schema levels depicted in figure 3.1 will be used for that purpose. This architecture was presented in [ROSC97], as an extension of that in [SL90], in order to separate different issues in the process of obtaining the user schemas from the federated ones. Its different schema levels, bottom-up, are:

"Native Schema" is the conceptual schema of a "Component Database" (CDB) expressed in the native data model.

"Component Schema" is the conversion of the "Native Schema" into the "Canonical Data Model" (CDM), BLOOM (defined in [CSG94]) in this architecture.

"Export Schema" represents the part of the "Component Schema" that is available to a class of federated users.

"Federated Schema" is the integration of multiple "Export Schemas". Each "Federated Schema" supports exactly one semantics.

"Authorization Schema" is defined to apply a "Multilevel Security" (MLS) policy. MLS is a "Mandatory Access Control" (MAC) mechanism to protect data where access right authorizations are not used but access decisions depend on security levels, organized as a partial ordered set, associated to each subject and each protected object. The security level associated to a subject is named "Clearance Level". Each one of the "Authorization Schemas" represents a subset of the "Federated Schema", which is accessible by a class of federated users (subjects) with a certain "Clearance Level". The set of data included in an "Authorization Schema" is classified at the same level, or at a level smaller, than the level corresponding to the "Authorization Schema".

"External Schema" defines a schema for a class of users and/or applications. It is still expressed in the CDM.

"User External Schema" is the conversion of an "External Schema" to the user data model.

The idea of relating "Data Warehousing" and federated databases was already presented in [SSSB98]. In this section, the architecture is studied in more depth, paying special attention to the new schemas appearing to achieve "Data Warehousing". Section 3.1.1 presents the schemas used to exemplify the architecture; section 3.1.2 explains the new schema levels one by one; and in section 3.1.3, the schema levels are presented all together. Finally, section 3.1.4 lists the different kinds of operations over schemas necessary to achieve the whole transformation process (from "Native Schemas" to "User External Schemas").

3.1.1 An example

An example, previously introduced in [ROSC97], is used to illustrate the architecture. The "Federated Schema" is obtained from the integration of two CDB that belong to an enterprise of industrial waste transports.

shipment[P]	sh_code	waste_code	prod_code	rec_code	TC
	0111	060105	123	321	P

waste[P]	waste_code	waste_name	risk[c]	precautions[R]	TC
	060105	Nitric Acid	3	Explosive; Keep cool	C
	060105	Acid	-	Explosive: Keep cool	R
	060105	Liquid	-	-	P

customers[P]	customer_code	customer_type	company_name	address	TC
	123	P	C.I.Q.S.A.	Pi, 5, Reus	R
	321	R	T.R.I.S.A.	Major, 2, Vich	P

```
class receivers[L]{
  aggregation_of
    company_name:strings
    treatment_type:tr_codes
    rec_code:rec_codes
      ...
    class_key rec_code
    oc:{L,H}
}
```

```
class producer[H] {
  aggregation_of
    company_name:strings
    pr_code:pr_codes
     ...
    class_key pr_code
    oc:{H}
}
```

```
class shipment[L]{
  aggregation_of
    wa_code:wa_codes
    wa_name:strings
    risk:integres[H]
    precaution:strings[H]
    producer:producers[H]
    receiver:receivers
    sh_code:sh_codes
    ...
    class_key sh_code
    oc:{L,H}
}
```

Figure 3.2: Examples of "Component Schemas" of CDB_1 and CDB_2

In figure 3.2, we can see the conceptual schema of CDB_1 and some data in it (expressed in the Relational data model), as well as the conceptual schema of CDB_2 (expressed in the BLOOM data model, which syntax can be found in [AORS99]).

The five classes of the "Federated Schema" showed in figure 3.3 (in the CDM, i.e. BLOOM), as well as the partial ordered set of security levels of the FIS, and the classification of all components of the "Federated Schema", are obtained through data schema integration and security

```
class f_customers [U] {
 alte_graliz_of customers_db1, customers_db2
          by db_discr delete_effect propagate
 alte_graliz_of f_producers [C], f_receivers by
          f_customer_type delete_effect propagate
 aggregation_of
   db_discr: {db1, db2}
   f_cu_code: cu_codes
   f_customer_type: strings
   f_telephone_number: strings
   ...
   class_key f_cu_code, db_discr
}
```

```
class f_producers [C] {
 alte_graliz_of producers_db1, producers_db2
          by db_discr delete_effect propagate
 alte_spaliz_of f_customers by f_customer_type
          delete_effect propagate
 aggregation_of
   db_discr: {db1, db2} [inherited]
   f_prod_code: pr_codes [inherited]
   f_customer_type: strings [inherited]
   f_telephone_number: strings [inherited]
   ...
   class_key f_prod_code, db_discr
}
```

```
class f_receivers [U] {
 alte_graliz_of receivers_db1, receivers_db2
          by db_discr delete_effect propagate
 alte_spaliz_of f_customers by f_customer_type
          delete_effect propagate
 aggregation_of
   db_discr: {db1, db2} [inherited]
   f_rec_code: rec_codes [inherited]
   f_customer_type: strings [inherited]
   f_telephone_number: strings [inherited]
   ...
   class_key f_rec_code, db_discr
}
```

```
class f_wastes [U] {
 alte_graliz_of wastes_db1, wastes_db2 by
          db_discr delete_effect propagate

 aggregation_of
   db_discr: {db1, db2}
   f_wa_code: wa_codes
   f_risk: integers [S]
   f_precautions: strings [C]
   ...
   class_key f_wa_code, db_discr
}
```

```
class f_shipments [U] {
 alte_graliz_of shipments_db1, shipments_db2
          by db_discr delete_effect propagate

 aggregation_of
   db_discr: {db1, db2}
   f_sh_code: sh_codes
   f_waste: f_wastes
   f_producer: f_producers [C]
   f_receiver: f_receivers
   ...
   class_key f_prod_code, db_discr
}
```

Figure 3.3: Example of "Federated Schema"

policies integration processes respectively. These integration processes are out of the scope of this thesis. For further information the reader can see [GSC95] (data schema integration) and [OS01] (security policies integration).

3.1.2 The parts

In this section, each one of the schemas helping on "Data Warehousing" is dissected. As it is shown in figure 3.4, all of them are obtained from the "Federated Schema", which means that the construction of the DW does not start from scratch. Instead, the integration work is assumed as already done.

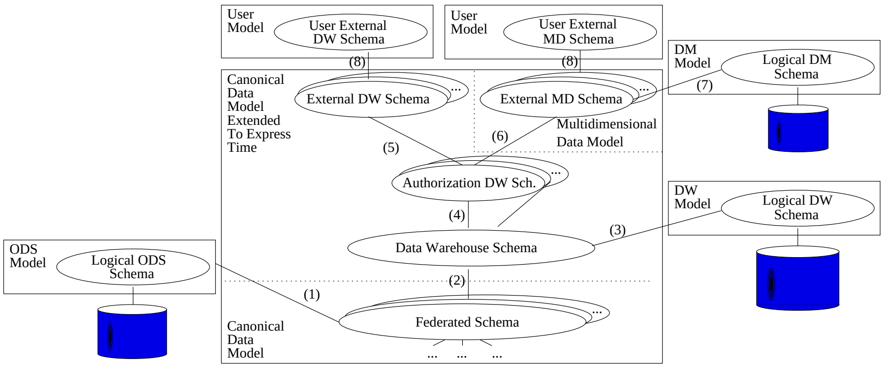


Figure 3.4: "Data Warehousing" schemas architecture from the "Federated Schema"

The Operational Data Store Schema

Once we have a "Federated Schema", the first thing we could do is to materialize it (tag (1) in figure 3.4). This means physically storing data to improve query response times.How that materialization is performed, is completely out of the scope of this thesis. What really matters is what we obtain with that materialization, i.e. the "Operational Data Store" (ODS).

If we analyze the definition of an ODS, we can see that a "Federated Schema" also satisfies the "collective, integrated, operational needs" demanded in [IIS98]. Concerning the characteristics of an ODS enumerated in its definition (i.e. subject-oriented, integrated, current-valued, and volatile), the federated data is obviously integrated, volatile, current-valued, and detailed. With regard to "subject-oriented", this does not come from the integration mechanisms but from the purpose of who integrates (that should choose a "subject-oriented" schema out of the multiple possibilities that integrate the data sources). Thus, it is always possible to obtain a subject-oriented "Federated Schema" with a small extra design effort.

Therefore, to obtain an ODS, if better response times are required, we just have to physically store federated data (solving problems related to CDBs interdependencies, polyinstantiations, etc.). Its schema will be exactly one of the "Federated Schemas". Notice that we could obtain many different "Federated Schemas", from different integration and negotiation processes. Not all these schemas can be used for the ODS. The implication goes the other way: The ODS schema is one of the possible "Federated Schemas".

The Data Warehouse Schema

The second thing we could do with the "Federated Schema" is to define the historic storage schema needed to support the decision-making process. A decision about which data is going to be stored needs to be made. We need to choose a set of integrated data that could be interesting to analyze. Most of the literature seems to suggest the usage of star shape schemas at this point.

The main advantages of star shape schemas are their simplicity and proximity to the business analysis concepts. It makes them quite easy to be understood by the final users. However, even more important than that is the fact that they imply a given kind of queries. Their structure is quite concrete, and allows to propose specific optimizations, access paths, and storage methods. Stars are probably the best way to study some isolated facts with regard to the desired analysis dimensions. However, they are not as good at keeping the data of the whole business. In the seven levels architecture, the DW is what [KRRT98] calls the "Storage Structure", and it is only accessed to solve a small number of specific queries. As it is outlined in [IIS98], there is not an homogeneous access pattern in the DW, and that is why isolated star shape schemas do not fit well.

We do not want to have little knowledge islands but a huge, fully connected continent to travel around. Star shape schemas do not seem semantically rich enough to represent the business process all in once, and accomplish that goal. The DW is used to represent the data all together. Thus, its strength does not have to be in easy querying, but in good integration and data semantics representation. Precisely because of that, the CDM of the federation could be the basis for a good data model for the DW. O-O models were found as good CDM in [SCG91], thereby we could think of having an O-O data model for the DW. Moreover, the importance of the time dimension in analysis tasks, as well as in the DW (notice the presence of the words "time-variant" and "non-volatile" in its definition) suggests a temporal extension of the model.

The process to obtain the **DW Schema**, in figure 3.4 tag (2), should not be query-driven, but data-driven. We need to choose a "Federated Schema" containing the data of interest for the analysis, and represent time in it. It means a semantic enrichment of the "Federated Schema" along temporal dimensions, reflecting new, temporal integrity constraints and security restrictions. It is not as simple as extending the keys (OIDs in our case) with an element of time. [BFG97] contains a temporal extension of an O-O data model.

As in [SA86], we should use two different kinds of time: "Transaction Time", and "Valid Time" (as defined in [DGK+94]). The storage of the time data enters the system (i.e. "Transaction Time") is mandatory and always possible. At least, two different times could be considered in this temporal dimension. The first one would be the time when the data was introduced in the operational applications, and the other one would be the time of entrance in the DW itself. When talking about "Valid Time", we could consider two different times, as well. The first one would be applied to the objects, and represented by exactly one continuous time interval. About the other "Valid Time" (represented by a set of non-contiguous, disjoint intervals), it will be used to tag the relationships between objects, indicating when they are valid.

"Transaction Time" will always be present in the DW (because we will always be able to register, at least, the "Transaction Time" in the DW, if the CDBs do not support it), while the "Valid Time" will completely depend on its availability in the sources. A good data model for the DW should take both into account, and ease their representation. "Transaction Time" could be implicit and the "Valid Time" explicit.

In the example, for the definition of the **DW Schema**, we should take another "Federated Schema" containing only that data interesting for the analysis (i.e. `f_telephone_number` attribute in `f_customers` should not be in it, because it does not seem interesting to be analyzed). Once we have that schema, we must modify it to reflect the different times of interest. Classes `f_producers` and `f_receivers` could have an associated "Valid Time" depending on the dates of their licenses to handle wastes. Each shipment should have a timestamp indicating the day it takes place. Moreover, every object would have a "Transaction Time" saying when it was introduced/modified in the DW, and maybe another one with the entrance date to the CDB.

All this does not mean that the users need to learn any new data model, later translation from the canonical model to any other user model is always possible (in figure 3.4 tag (8)), if desired. Notice, this architecture does not force us to use an Object-Oriented Database to perform the historic storage, either. Any kind of system could be used, defining its schema as in figure 3.4 tag (3).

Authorization DW Schemas

Authorized access in a DW is scantily studied. However, the set of data, stored in a DW, that is needed to support the decision-making process has to be protected from unauthorized accesses, just as any other information system, because data helping to make decisions is probably very confidential.

"Authorization Schema" in figure 3.1 helps federated databases on data protection, because each "Authorization Schema" defines the subset of information that a class of users/applications can access. **Authorization DW Schema** helps the DW just as "Authorization Schema" helps federated databases.

The process to obtain an **Authorization DW Schema** from the **DW Schema** (in figure 3.4 tag (4)) takes into account the security policy of the federation itself. The mechanism should be similar to that of "Authorization Schema". Nevertheless, it is out of the scope of this thesis work.

External DW and Multidimensional Schemas

Besides reflecting the security aspects of the DW, we can also define the subsets of data of interest depending on the classes of users and/or applications (tags (5) and (6) in figure 3.4). The external schemas are expressed in the CDM. However, they can be translated (tag (8) in figure 3.4) to any other model.

At this point, the strength is not in the data itself, but in the needs of the users. Here, we will have a query-driven design, where what really matters is the vision users have. We could define, by (5) in figure 3.4, **External DW Schemas** in the same data model of the DW. However, if the users have a multidimensional vision of the data, we will obtain star shape **External Multidimensional Schemas** (by (6) in figure 3.4). Notice that this transformation includes two different actions. On the one hand, we are deriving the desired view of data, and on the other hand, we are translating the schema to a multidimensional data model. As we can see in section 3.2.2, this introduces a modification in the architecture to obtain processors that perform exactly one task. At this moment, this modification is not included in the discussion, because it is only important if we allow several **Star**s in one **External Multidimensional Schema**. With only one **Star**, translation and definition of the required multidimensional view can be easily done in one step.

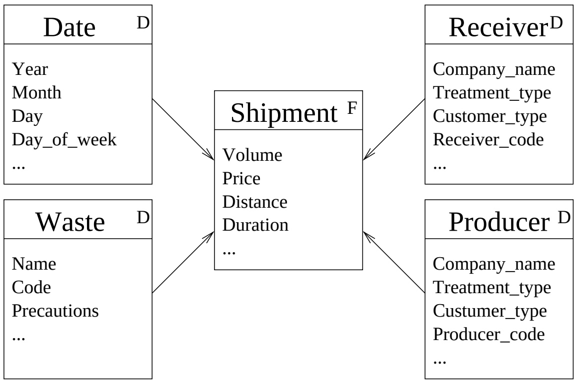


Figure 3.5: Example of **External Multidimensional Schema**

In the industrial wastes example, the **External Multidimensional Schema** depicted in figure 3.5 could be defined from the **Authorization DW Schema**. In this case, the analyzers are interested in the analysis of the shipments depending on the date, the kind of waste, the producer, and the receiver. That is, for them, the **Fact** is `Shipment`; and the `Date`, `Waste`, `Producer`, and `Receiver` are the analysis dimensions. If the same analyst needs to study more than one **Fact**, this schema should contain more than one **Star**. Sometimes, this is called a "Star Constellation" or "Data Warehouse Bus" (in [KRRT98]).

Due to performance reasons, most of these **Star**s are materialized (represented by (7) in figure 3.4), giving rise to "Data Marts" (built with either O3LAP, ROLAP or MOLAP techniques). However, other "External Schemas" used for data mining or solving some sporadic queries would not need to be materialized nor multidimensional.

3.1.3 The whole

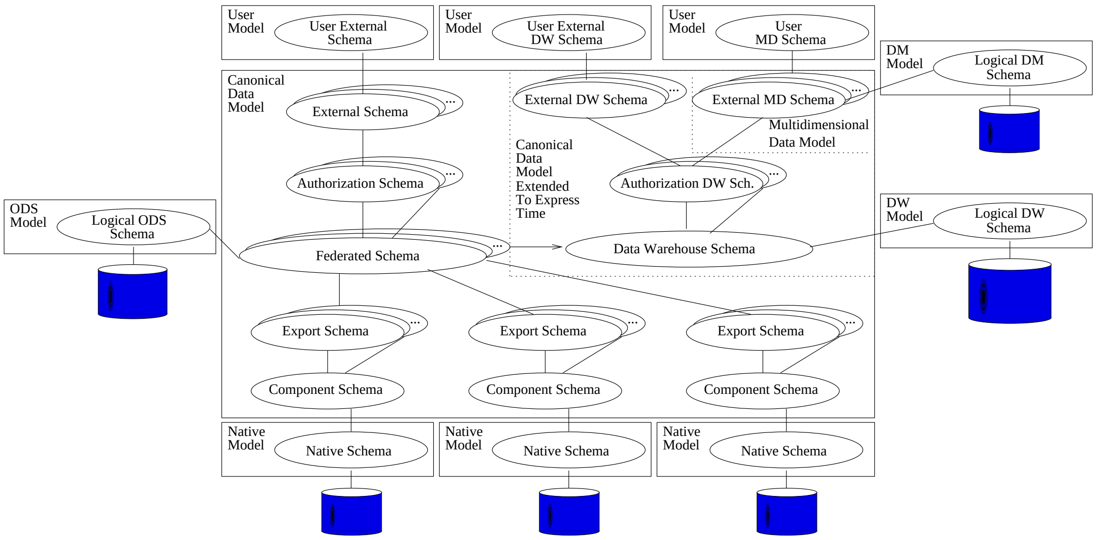


Figure 3.6: Integrated architecture for FIS and DW

In figure 3.6, we can see the result of merging the seven levels architecture for federated databases in figure 3.1, and the schema levels for "Data Warehousing" in figure 3.4.

It is important to notice the location of the **DW Schema**. If we assume that the presence of a processor (performing changes either in data model, in semantics, or in UoD) forces the appearance of a new level, the DW should be placed in between the "Federated Schema" and the "Authorization Schemas". However, the **DW Schema** is at the same level than the "Federated Schema", because they are equally important. If the "Export Schemas" were expressed in a temporal CDM, and we had an integration processor for it, then "Federated Schema" and "DW Schema" would collapse into a single schema.

On the other hand, the double storage system (DW-DM) should not be avoided. The DW is data-driven designed, and will contain data that may not be sure that will some day be useful (which worsen performance). The DM is query-driven designed, oriented to optimize response times. Thus, what we will, likely, have is a temporal, or Relational database supporting time, incrementally designed and populated, as data is generated. From this huge, central DW, we can define and feed smaller DMs, in as-needed basis. Notice that a methodology is not suggested, but just an architecture. Defining a methodology is absolutely out of the scope of this thesis, and the architecture does not impose it.

3.1.4 Operations on schemas

At this point, it is important to mention how the transformations between levels are performed in the architecture. Different kinds of operations in the CDM are necessary to perform the following functions:

1. Conforming operations, like those in [RAO+01], to transform the "Export Schema" of one DB to a form more suitable for integration into a "Federated schema". These operations are also useful in other contexts, in particular to derive external schemas (i.e. views).

2. Generalizing classes from different DBs to a superclass in a "Federated Schema". The schema integration process, which produces a "Federated Schema" from several "Export schemas", can be considered as a two-step process: first, conforming operations change the form of the "Export Schemas" into a common form, and then these are generalized. Discriminated generalization is preferred, because of the reasons explained in [GSC95], in particular the support of "multiple semantics" (as in [SL90]) and no loss of information, because each (virtual) object in a "Federated Schema" is given a tag (i.e. discriminant) showing from which CDB it comes from.

3. "Object Identification Function" (OIF) to assert when an object O_1 in one DB represents the same real world object as an object O_2 in another DB. Different users may use different OIFs, as explained in [SR97].

4. Collapse two objects into one using a particular OIF [GCS95]. If all users share the same OIF for a federated class, or if integrity constraints among the CDBs (interdependencies) must be enforced, then the collapsing operation may take place during the process of schema integration; otherwise, the derivation of each "External Schema" may collapse using a different OIF.

5. Dealing with value discrepancies, preserving all values by having multivalued attributes in "Federated Schemas". "External Schemas" may use different options, such as giving preference to the value coming from a particular DB (shown by its discriminant), or by "aggregation by reduction" operations (sum, average, maximum, etc.), as in [SR97].

6. Protecting security by hiding relationships between abstractions that could reveal confidential information.

7. Transform into a multidimensional data model the structures of the DW data model. O-O models are preferred, as discussed in section 2.4.

3.2 Drilling across semantically related Stars

Quite often, different data cubes in a business model are found closely related, and analysts want to jump from one to another. They use to be interested in generating several reports showing different sets of data organized from the same point of view, so that they are easily comparable. For instance, it could be interesting to analyze evolution of sales along `Time` and `Product`, and compare it with production in the same period of time, for the same product. Probably, that information will be stored in different DMs, and users will need to drill across them. The aim of this section is to dig into the applicability of **Drill-across** by studying how a multidimensional schema could contain several, related **Star**s, even if the data cubes are physically stored in different DMs.

The structure of the section is as follows: section 3.2.1 presents some work on relating different star schemas; section 3.2.2 shows a general approach to the problem and presents a modification to the schemas architecture in previous section to benefit from relationships between **Star**s; section 3.2.3 exemplifies different kinds of relationships found between **Star**s; finally, section 3.2.4 contains a discussion about the relationships found.

3.2.1 Drill-across in the literature

Lately, there has been a lot of work about OLAP tools. We can find literature devoted to specific storage techniques and access mechanisms, as well as to pure multidimensional modeling. Both areas benefit from the duality fact-dimension, and restrict their studies to isolated stars, i.e. how we can store/model one fact and its surrounding analysis dimensions. Nevertheless, some authors have already pointed out the importance of drilling across different data cubes, which means navigating through data in different star shape schemas. Unfortunately, the "Drill-across" operation, in these models, is limited to the case that some analysis dimensions are shared by the data cubes.

[Kim96] proposes a logical model to implement star schemas on Relational databases. Each star schema contains a central "fact table" related by foreign keys to its corresponding "dimension tables". In order to support "Drill-across", Ralph Kimball contends that all constraints on **Dimension** attributes must evaluate to exactly the same set of **Dimension** instances in both schemas. This is clearly satisfied, if both **Dimension**s are exactly the same. However, he also explains how this matching can be satisfied, for instance, if the only difference between the **Dimension**s is their granularity, i.e. the finest detail level they allow.

A later work, [GLK99], presents "multi-star" schemas obtained by normalization of "fact tables", while [PJ99] defines a "multidimensional object family" as multidimensional objects possibly with shared subdimensions (i.e. subsets of levels in the aggregation hierarchy of the analysis dimension). [MK00] goes a little further, and distinguishes three kinds of schemas with more than one star, i.e. "constellation", "galaxy", and "star cluster". A "constellation" schema consists of a set of star shape schemas with hierarchically linked "fact tables". A "galaxy" is a collection of star schemas with shared **Dimension**s. Finally, a "star cluster" is a set of star schemas sharing subdimensions. [Gio00] defines "constellation", as well. In this case, it is a set of stars sharing **Dimension**s. The shared **Dimension**s must be "conforming dimensions", which means their values are consistent among the stars. Even though these models allow several **Star**s in a schema, they do not study relationships between them.

Regarding multi-star architectures, [KRRT98] suggests a "Data Warehouse Bus Architecture", which offers "wire-dimensions" where facts can be plugged. Dimensions tables are conformed in order to be shared by fact tables. This is a simple solution to the problem of integrating DMs, which helps to develop DMs at different times, by different work teams.

3.2.2 Multi-star conceptual schemas

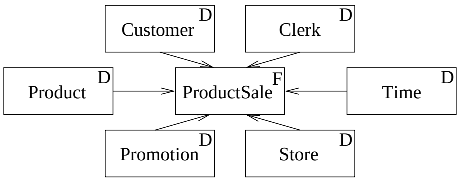


Figure 3.7: Example of multidimensional schema

A **Fact** is a subject of analysis. It could contain different kinds of cells (we will call them **Cell**s), which, in turn, could contain different **Measure**s that we want to analyze. Each data cell is identified by a point in each of its analysis dimensions. These points may correspond to different granularities for every **Cell**. For instance, in the example in figure 3.7, we are interested in the analysis of the **Fact** `ProductSale` with regard to the product which was sold, the time when was sold, the customer whom was sold, the clerk who sold it, the promotion that affects it, and the store where was sold. `ProductSale` would contain different **Cell**s, if some **Measure**s were not available at `Day` granularity. Moreover, other **Measure**s could not be interesting for every `Customer`, but only for `CustomerProfile`s. Thus, there would be different kinds of cells depending on whether **Measure**s are available or meaningful for either `Day` or `Month`, `Customer` or `CustomerProfile`. **Fact**s are deeply studied in section 4.2.

A **Dimension** is a connected, directed graph representing a point of view on analyzing data. Every vertex in the graph corresponds to an aggregation **Level**, and an edge reflects that every instance at target **Level** decomposes into a collection of instances of source **Level** (i.e. edges reflect part-whole relationships between instances of **Level**s). An in depth explanation of **Dimension**s is in section 4.1. Each **Level** corresponds to a granularity in the **Dimension**, and has attributes that allow to select some of its instances. By selecting points in every analysis dimension, we choose the data cells of interest in our analysis. Thus, **Dimension**s contain those data that identify **Cell**s instances.

If users are only interested in a given **Cell**, they just need to access a **Fact**. However, sometimes, they could desire to relate data in different **Star**s, and OLAP tools should allow it. We can understand **Drill-across** as re-using the same condition over the **Dimension**s on querying different **Fact**s. This means that we select a subspace in a given **Cube**, and want to view the corresponding space in a different **Fact**. At a first glance, this can be allowed, if both **Star**s share some **Dimension**s. However, it is also possible, if **Dimension**s are not exactly the same, but exists some semantic relationship between the **Dimension**s and/or **Fact**s in the **Star**s.

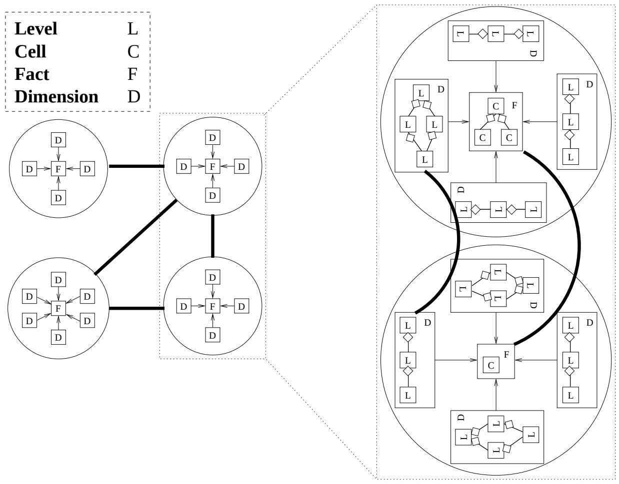


Figure 3.8: Multi-star diagram

Existing multidimensional models do not consider "semantic domains", i.e. domains reflecting the conceptualization of values in the mind of the designer. Actually, the analysis dimensions used on drilling across need not exactly coincide in both **Star**s. They should just be defined on a related semantic domain. Thus, **Drill-across** between different **Fact**s is performed thanks to semantic relationships between **Star**s (like those drawn with thick lines in figure 3.8). Two **Star**s can be related whether their **Fact**s, or their **Dimension**s are. We can see four kinds of semantic relationships in this section, i.e. *Derivation*, *Generalization*, *Association*, and *Flow* (in UML terminology, as defined in [OMG01b]).

Multidimensional data need an integrated access, to be able to drill across through interstellar relationships. In this sense, the architecture presented in section 3.1 needs to be clarified. For the sake of simplicity, let us leave aside the **Authorization DW Schema**. If we are dealing with isolated **Star**s, from the **DW Schema**, data is selected and translated to the multidimensional model, so that we obtain an **External Multidimensional Schema** at conceptual level. Then, this is represented at logical level, and implemented in a DM.

Unfortunately, DMs do not take under consideration multi-star schemas. Moreover, to improve performance or due to management reasons, different **Star**s could be implemented on different DMs. Thus, we should better use a 3-level schema architecture like that shown in figure 3.10 (based on the ANSI/SPARC architecture in figure 3.9). The architecture facilitates the usage of semantic relationships at conceptual level, while allows to store independent **Star**s

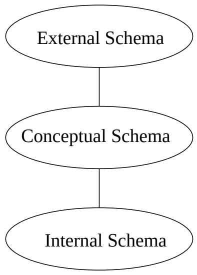


Figure 3.9: ANSI/SPARC database schemas architecture [BFJ+86]

in different DMs. Other levels could also be added to the architecture, as in [SL90], if we were dealing with heterogeneous DMs and multidimensional models.

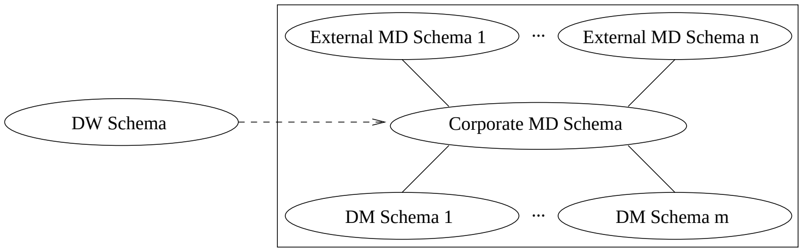


Figure 3.10: 3-levels multidimensional schemas architecture

The only **Corporate Multidimensional Schema** (CMDS) contains all data in the individual DMs, related by inter-stellar semantic relationships (as sketched in figure 3.8), like those offered by **YAM^2** . Thus, the CMDS would be obtained from the **DW Schema** by means of translation operations between the DW model and the multidimensional model. No selection would be necessary, because we can, later on, choose which instances are stored in the DMs, and which views we offer to the users. Problems associated to the obtaining of the **DW Schema** are out of the scope of this section. Just to notice that the possibility of integrating the DMs if they were not defined from a common DW, or, at least, developed based on a common plan, will be much harder. As pointed out in [IIS98], building independent DMs directly from operational applications is a poor idea. In this architecture, DMs are obtained by translating the DW to a multidimensional model, and then, storing the **Cube**s of the different **Star**s found in the DMs.

At the bottom level of the architecture, we have the different **DM Schema**s, which could be stored in either "Relational OLAP" (ROLAP), "Multidimensional OLAP" (MOLAP), "Object-Oriented OLAP" (O3LAP), or any other kind of multidimensional system. These DMs optimize accesses to multidimensional data, and it does not matter whether they were independently defined, or built from a huge, common DW, because we still have a common view of all them at conceptual level, which ensures conformation of data and integrated access. **DM Schema**$_i$ in figure 3.10 represents the logical schemas of the different DMs (in the data model corresponding to the chosen multidimensional system).

At the top level, we can define **External Multidimensional Schemas** (EMDS), to cover the needs of different users or groups of users. These subsets of information would contain different inter-related **Star**s, that users could successively visit. Furthermore, since different people may view things in a different way, these schemas offer the possibility to rename concepts, or even define some derived data that was not physically stored in the DMs. View definition mechanisms should be adapted here to a multidimensional data model.

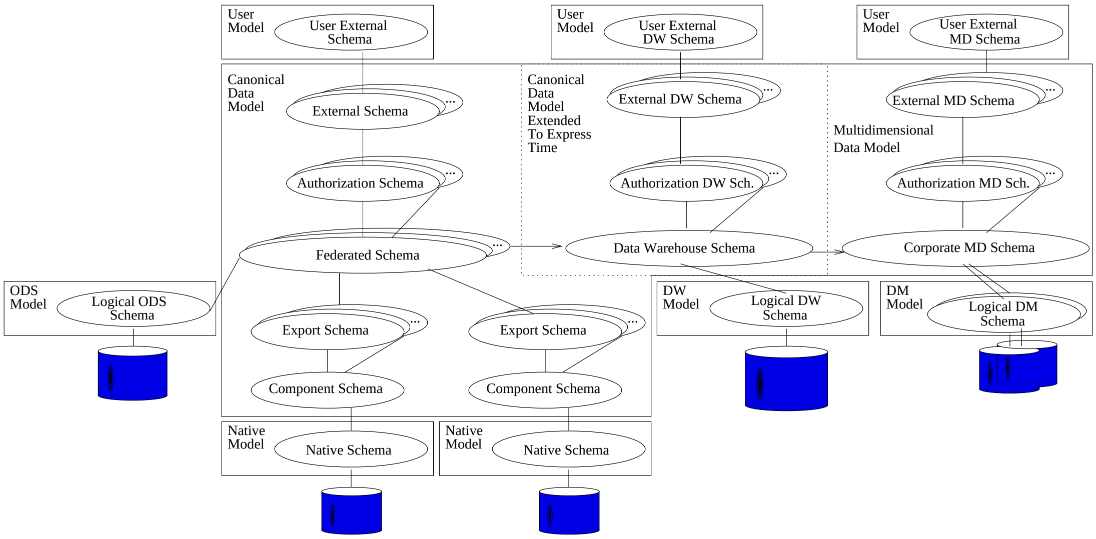


Figure 3.11: Integrated architecture for FIS, DW and multi-star schemas

Thus, architecture in figure 3.6 needs to be modified in order to support the definition of a corporate schema with several **Star**s, so that from it, external views can be defined. By doing so, we obtain that every processor between two schema levels performs only one task. As shown in figure 3.11, firstly, the **Data Warehouse Schema** is translated to a multidimensional model (which should allow multi-star schemas). Later on, the desired multidimensional view would be defined from the **Authorization MD Schema**. On doing this, we could choose the subset of **Star**s and semantic relationships connecting them that are of interest for a set of users.

3.2.3 Inter-stellar semantic relationships

We are interested in providing multidimensional schemas with more than one **Fact**. However, having several **Fact**s in the same schema is absolutely useless, if it contains isolated **Star**s. **Fact**s need to be related in some way to allow **Drill-across**. Some multidimensional models and most OLAP tools allow this operation if two **Star**s share some **Dimension**s. However, purely sharing **Dimension**s does not seem enough at conceptual level, where we could find much more meaningful semantic relationships among data.

In this section, we are going to deep into the capture of semantic relationships between different **Star**s, that allow navigation through them. Firstly, possible relationships between two **Dimension**s are shown; afterwards, relationships between two **Fact**s are exemplified; and finally, how a **Fact** can be related to a **Dimension** in another **Star**, or vice versa is explained.

Dimension-Dimension

[KRRT98], [Gio00], as well as some multidimensional tools stress the importance of having "conforming dimensions". In order to drill across different **Star**s, the corresponding schemas must share analysis dimensions so that their instances exactly coincide. This position unnecessarily restricts the usage of **Drill-across**. Actually, it is just needed that the selected instances of the **Dimension**s of the origin **Star** determine instances in the **Dimension**s of the destination **Star**. Thus, domains used in both **Dimension**s must be related in some way, but **Dimension**s could still be absolutely different. We are going to see four kinds of O-O relationships between **Dimension**s that allow to drill across their **Star**s.

Derivation Firstly, we could find that the same concept has different names depending on the subject. Therefore, the same **Dimension**, with exactly the same instances, will need a different name depending on the context where we are going to use it. Moreover, this **Dimension** could not play the same role for different **Fact**s. `Product`, may be considered `RawMaterial` in a different context. It is not enough to say they are "synonyms" (like in [Kim96]), because they could even have different attributes of interest to the users. For example, keeping or studying the `benefit` of raw material can be meaningless. Moreover, we could find that elements in a **Dimension** are different from those in another one, even though they represent the same concepts. For instance, a given subject implies that `Red`, `Blue`, and `Yellow` are the instances in `Color` domain; while in a different case, we need to distinguish different kinds of `Blue`, like `Dark Blue` or `Light Blue`. It is also possible to find differences in how concepts are codified (ex. letters or numbers).

Sometimes, some **Dimension** instances in a **Star** are only considered grouped in another **Star**, because of lack of interest in the individuals, confidentiality issues, or space problems to keep information at maximum detail. In [Kim96], a **Dimension** whose finer granularity is not of interest is called "Demographic minidimension". Thus, we could use the same **Dimension** in two star schemas at different aggregation levels. For instance, one of them could keep data by hour, while the other does it by day. Clearly, all we have to do to drill across them is roll data at `Hour` up to `Day`. In this way, both **Cell**s will be defined over the same kind of dimensional instances. Hence, they will be comparable.

If two **Dimension**s coincide in one of their aggregation levels, both can be conceived again as derived from a common, more general **Dimension** which contains their hierarchies. Figure 3.12 shows how the `Time` **Dimension**, is included in a more general **Dimension** `CorporateTimeDimension`. `Time` does neither provide `Hour` nor `Week` **Aggregation Levels**, because they are not of interest for the `ProductSale` **Star** where it is used.

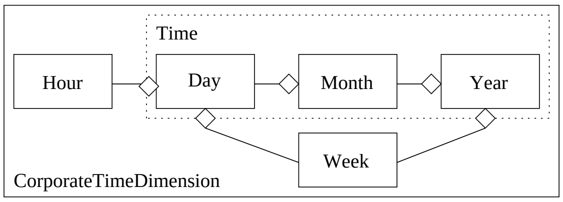


Figure 3.12: Example of containment of **Dimension**s

In terms of O-O, considering "Derivability" or "Point-of-view" would allow us to reflect that two **Dimension**s are derived both from a common concept, in spite of the fact that they look different, like in the mentioned examples. Thus, we find that two **Dimension**s can be related by **Derivation**, and users can drill across from a **Fact** to another one through it. The **Dimension** is not shared, because it appears different in each **Star**. However, by means of this kind of relationship, we can offer users the desired view while they are still able to drill across.

Generalization We can also find relationships between analysis dimensions along "Generalization/Specialization" (also known as "Superclass/Subclass"). **Dimension**s of different **Star**s could be related by *Generalization*, so that **Drill-across** would be allowed. For instance, `Customer` and `Clerk` are both subclasses of `People`. Therefore, we could travel from a **Star** with information about `Customer` to another one with information about `Clerk`, if the sets of instances of both **Dimension**s are not disjoint. Moreover, they will have in common all those attributes in the superclass, and maybe some aggregation levels.

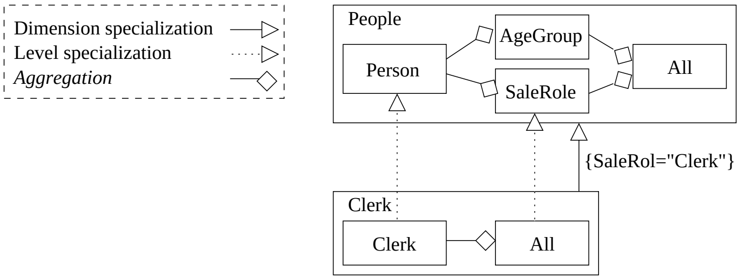


Figure 3.13: Example of *Generalization* between **Dimension**s

As outlined in [Gio00], using superclasses and subclasses in star schemas, we gain better understanding. Consequences of two **Dimension**s being related by specialization are studied in section 4.1. There, we can see that, as exemplified in figure 3.13, we should speak of specializing a **Dimension** at a **Level**, rather that just specializing a **Dimension**. If we specialize `People` **Dimension** at `SaleRole` **Level** (solid arrow) to get `Clerk` **Dimension**, this specialization contains a **Level** (i.e. `Clerk`) with instances corresponding to people acting as clerk, and another one with only one instance representing the set of all clerks (i.e. `All`). Dashed arrows show that a **Level** is specialization of another one. By a similar specialization of `People`, we could obtain `Customer` **Dimension**.

In the example, `AgeGroup` aggregation level is not of interest in `Clerk` **Dimension**. Notice that if it would, it would not be specialization of the homonym **Level** in `People` **Dimension**, since its instances would represent different sets of people. For instance, not everybody between twenty and thirty years is a clerk. Therefore, the instance corresponding to this age group in `People` **Dimension** would represent more people than the instance representing the same age group in `Clerk`.

Sharing **Dimension**s would not be enough in this case, either. `Clerk` has more attributes and much less instances than `Customer`. Therefore, sharing `People` **Dimension** in two **Star**s would generate lots of undesirable null values. Nevertheless, we can still drill across, if instances of `Clerk` can also be instances of `Customer`.

Association It is possible to have associated analysis dimensions, as well. The domain of a **Dimension** could be used as an attribute domain in another **Dimension**. Selected instances in a **Dimension** would allow to identify instances in the other, so that it is possible to drill across the corresponding **Fact**s. Clerks use to be assigned to stores. Thus, `Clerk` would be associated with `Store` **Dimension** (maybe multivalued). This is not what is called "outrigger table" in [Kim96]. That is at **Logical** level, and refers to normalization. In this case, it is not normalizing at all, but showing that two different analysis dimensions are semantically related.

We can also find stronger associations between analysis dimensions, if we join more than one to give rise to another. This is not a simple association, because if we remove one of the aggregated **Dimension**s, we loose the aggregate one. For example, `Color` **Dimension** could be used to define `ColoredProduct`. Dissociating `Colors` from `ColoredProduct` means we do not have colored products any more.

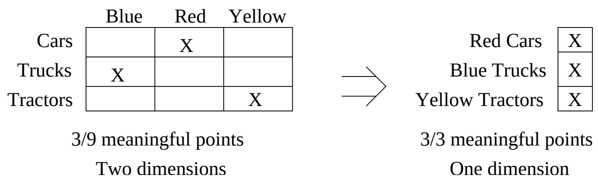


Figure 3.14: Example of correlated **Dimension**s

If `ColoredProduct` would be represented as two separate **Dimension**s (i.e. `Color` and `Product`), all combinations of color-product would be allowed. Sometimes this could be the case, but other times, products are only available for a reduced set of colors (if both analysis dimensions are correlated, as exemplified in figure 3.14). Therefore, it is much better for the designer to reduce the analysis space only to those meaningful values by modeling all of them in just one **Dimension**. In the figure, we have six meaningless values out of nine possibilities. Thus, it should be better to model it as only one **Dimension** where all three values are meaningful.

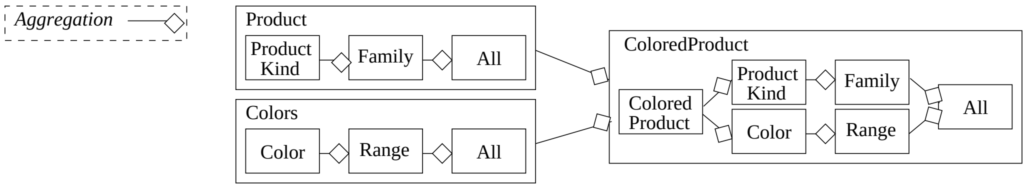


Figure 3.15: Example of *Aggregation* between **Dimensions**

Whether it is modeled as two independent **Dimension**s, or only one, will depend on the distribution of the cells related to the **Dimension** instances, and the user point of view. Thus, we can find `Color` and `Product` in a **Star**, and `ColoredProduct` in another one, and a user should be able to navigate from one to another through *Aggregation* relationships (as shown in figure 3.15). Relating **Dimension**s by *Aggregation* also has consequences in the aggregation hierarchies. Figure 3.15 shows how the hierarchy of the aggregate **Dimension** contains the subgraphs of those hierarchies in the **Dimension**s we are aggregating (as it is explained in section 4.1). However, it is important to notice that the elements at homonym **Level**s in different **Dimension**s, in this case, do not coincide. For instance, `Color` **Level** in `Colors` **Dimension** contains elements representing colors. Nevertheless, `Color` **Level** in `ColoredProduct` **Dimension** contains elements representing sets of colored products grouped by color.

Again, this offers navigation possibilities that just sharing **Dimension**s do not offer. A psychological study could include `Color` **Dimension**. `ColoredProduct` cannot be shared with the **Star** corresponding to that study, because it was about colors, and did not considered products at all. However, analysts will probably be interested on navigating from one **Star** to the other. Thus, navigation should be allowed through the *Association* between `Color` and `ColoredProduct` **Dimension**s.

Flow Because of the long periods of interest in analysis tasks and how fast business change nowadays, it is expected that analysis dimensions in our multidimensional schema evolve. Due to the importance of time, it is not acceptable to throw away old **Dimension**s. Old data would still be stored following the old schema, while we are currently using a new one.

As our business grows, it could become international, so that a new **Level** `Country` will appear in the `Store` **Dimension**. Attributes could also appear or disappear in any **Dimension**, as the information systems and the enterprise evolve. We should not study data with regard to those attributes, because their values at the time data was collected are unknown. Therefore, old **Dimension**s should be kept as they were, but related to the corresponding new **Dimension**s by *Flow* relationships. This will show to the user which dimensional data can be used at each analysis depending on the period of time he/she is interested on. As studied in [EK01], functions can be offered to the users to estimate values of dimension attributes in the periods of time they are unknown. Anyway, the schema should reflect the difference to let users know whether the values are real or estimated.

Sharing is not possible in this case. It would mean studying old data with regard to new attributes or vice versa. In any case, it could generate absolutely wrong results.

Fact-Fact

Points in the multidimensional space are always identified by its analysis dimensions. However, using those **Dimension**s is not always necessary to select a set of points. Having functions from points in a space to points in another one is another possibility. Therefore, if we identify a set of cells in a **Fact**, we will be able to select the corresponding set in a related **Fact**. This means that we can also use relationships between **Fact**s to navigate. The relationships between the structures of two related **Fact**s are studied in section 4.2.

Derivation **Measure**s in one **Fact** could be obtained by applying some operation to **Measure**s in other **Fact**s. For instance, on analyzing efficiency of employees, some **Measure**s could be obtained by operating the benefits of some products sold (the best sales, sales involving relevant products, etc.). Most **Dimension**s will be likely shared by both **Star**s (i.e. `EmployeeEfficiency`, and `ProductSale`). However, we could also travel between them due to the fact that data in some cells are obtained by processing other cells. We could navigate from data in `EmployeeEfficiency` to the data in `ProductSale` used in their calculation. This does not correspond to **Drill-down**, because both **Fact**s represent different subjects, and selection of **Cell** instances is not performed by means of aggregation hierarchies.

Association A **Fact** in a **Star** can be associated with **Fact**s in another **Star**. For instance, a `Deal` is composed by several individual `ProductSale`. Notice that **Measure**s of `Deal` are not necessarily obtained from those of `ProductSale` (for instance, discount in the deal). Thus, if we are studying a set of sales, it can be interesting to see data corresponding to deals in which they were done. Coincidences or differences in **Dimension**s do not matter. We should be able to travel from a **Star** to another one just because the *Association* relationship between the **Fact**s.

Generalization Some **Fact**s do not have exactly the same **Measure**s nor associated **Dimension**s, but still are closely related. For instance, `ProductSale` can be seen as an specialization of `Contract`. Since a sale is a kind of contract, it will have its specific **Measure**s and **Dimension**s. In turn, as it is shown in figure 3.16, `ProductSale` could be specialized into `CashSale` or `CreditSale` depending on how it is paid. We will have different information for each of the specializations (for example, number of credit card). Analysis dimensions are inherited from the superclass, but others could be added, like `Bank`. Users should be allowed to navigate through different **Star**s just because their

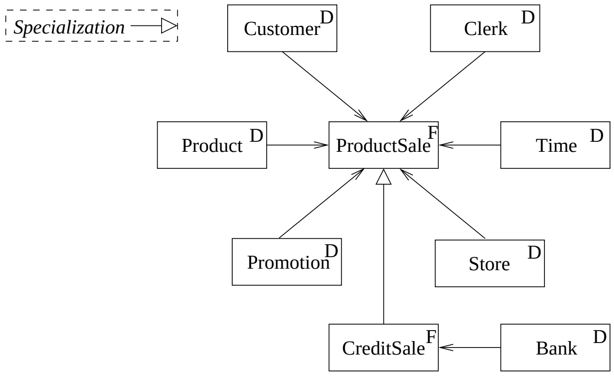


Figure 3.16: Example of *Generalization* between **Fact**s

Cells are specialization one of another. Usually, they will also share most analysis dimensions, but sharing them is not needed in this case to drill across, since **Fact** domains are subset one of the other.

Flow New **Measure**s could appear, possibly replacing others; precision of the measurements could also be improved or even worsened; new interesting analysis dimensions could be found (besides or instead of the already existing ones); and so on. Even if the subject stays, how information is captured can evolve. Data sources, measurement instruments, or calculation algorithms are probably going to change, and these changes should be reflected in our model by means of *Flow* relationships between **Fact**s. All this is not reflected by just relating our **Fact**s to `Time` **Dimension**, since we actually have different **Cell** structures. One day we start recording discount checks in `ProductSale`, hence we need to keep both incomes (i.e. cash, and discount checks). From this day on, we should have different **Star**s containing data about the same kind of facts before and after the acceptance of the checks, because the **Cell** structure changed. An analyst would be able to relate those data by means of *Flow* relationships between **Fact**s. At query time, this relationships would allow to provide conversion functions between different versions of data (like in [EK01]), or just show a warning to the analysts. However, implementation issues are completely out of the scope of this thesis.

None of these relationships can be reflected by sharing **Dimension**s. Instead, they show correspondences among factual information, which is also important to navigate. We can go from a **Star** to another, independently of **Dimension**s, if cells in the first determine cells in the other.

Fact-Dimension

The last possibility to navigate through different **Star**s is that the **Fact** in one of them is used as **Dimension** in the other, or vice versa. This should not be properly regarded as **Drill-across**, since we do not want to analyze data in two **Star**s from the same point of view. Rather, we are using results of querying a **Star** to query a different one. For instance, some people could be interested in the analysis of promotions. Thus, the promotions selected by studying `Promotion` **Fact**, can be used as **Dimension** to study `ProductSale`. A **Fact** is not conceived to be used as **Dimension**, so that it will not exactly coincide in both star schemas.

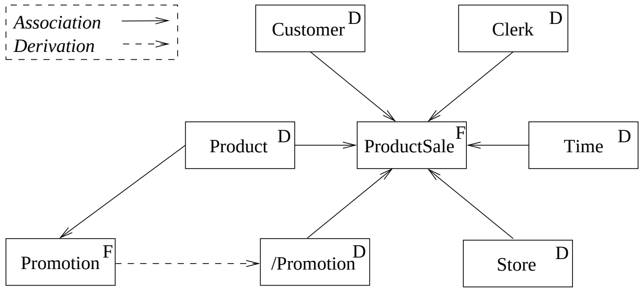


Figure 3.17: Example of *Association/Derivation* between **Fact** and **Dimension**

Derivation A **Dimension** can be obtained by deriving it from a **Fact**. The name can be changed, some attributes added or removed, others recalculated, some instances selected, etc. in order to adapt it to its new usage. **Fact**s use to have much more instances than **Dimension**s. Nevertheless, by grouping them, we could obtain coarser aggregation levels of interest. **Measure**s use to be numerical, while attributes in **Dimension**s use to be descriptive. Therefore, **Derivation** (exemplified in figure 3.17) will probably imply a change in the kind of attributes. `Promotion` **Fact** could have a numerical attribute `benefits`, while `Promotion` **Dimension** would have an enumerated, derived attribute `success` instead. Thus, users can drill across through the relationship between a **Fact** and a **Dimension** even if they do not coincide, but there exists a *Derivation* between them.

Association Instances of a **Fact** could also be associated with those of a **Dimension**, or vice versa. For instance, some products can be affected by promotions. Thus, `Product` **Dimension** could have an attribute defined on domain `Promotion` **Fact** (i.e. association arrow in figure 3.17). Notice the difference between that and relating the `Promotion` to another **Fact** (i.e. deriving a **Dimension** and using it to analyze `ProductSale`). The latter would mean that a sale was performed during a promotion, while the former would show all promotions that have been applied to a kind of product.

3.2.4 Discussion

Table 3.1 sums up the kinds of *Relationship*s found of interest between different **Star**s. What could attract attention is that neither *Generalization*, nor *Flow* relationships between a **Fact** and a **Dimension** were considered. This is because a temporal transformation can convert

Relationships	D-D	F-F	F-D	D-F
Derivation	√	√	√	×
Generalization	√	√	×	×
Association	√	√	√	√
Flow	√	√	×	×

Table 3.1: Summary table of relationships between **Fact**s and **Dimension**s

neither **Fact**s into **Dimension**s, nor vice versa; and they are different enough not to be related by *Generalization*. If we want to obtain one from the other, we must derive it. Nevertheless, factual information cannot be derived from dimensional data. That is because **Fact**s represent measurements, while **Dimension**s show given information.

Drill-across implies the usage of the analysis framework that we are using for a given **Fact**, on analyzing a different one. That is to study different data at the same granularity, and constrained by the same conditions over the analysis dimensions. Other authors restrict that to **Star**s that share **Dimension**s. However, we have seen in previous section, that there are four relationships between **Dimension**s that would also allow it. If two **Dimension**s are related by *Generalization*, or *Flow*, they will be different. However, there will be a one-to-one relationship between their instances so that **Drill-across** can be performed. If the **Dimension**s are related by *Derivation*, or *Association*, instances do not coincide, but an instance of a **Dimension** determines instances in the other.

Actually, it is not necessary the **Dimension**s in the destination **Star** to be related to those in the origin. It could be that selected cells in the latter determine a set of cells in the former. This is the case if both **Facts** are related in some way. Thus, we just need to substitute **Measures** of one cell, by those of its counterpart in the other **Fact**. In this way, we are also able to study data in two different **Star**s whose **Fact**s are related.

Relationships between a **Fact** and a **Dimension** do not allow proper **Drill-across**. However, if selected cells determine a set of points in a **Dimension**, these can be used in the analysis of another **Fact**.

[Kim96] points out that it rarely makes sense to restrict simultaneously two **Dimension**s in the same **Star** by the same condition. Likely, it is senseless to apply the same constraint to two sets of instances over different semantic domains. However, Kimball also explains how an analysis dimension can play different roles in the same **Star** (for instance, `People` acting as `Clerk` or `Customer` in the example). If so, it could be constrained for both roles at once. Furthermore, if we would have two semantically related **Dimension**s in the same **Star**, both could also be constrained at the same time, because they would be "type-compatible" in one way or another. For example, we could study clerks that are our customers. Therefore, the relationships found in last section should not only be considered to **Drill-across**, but also for operations that involve isolated star schemas, i.e. **Dice**. A condition could be simultaneously applied to several **Dimension**s defined over the same semantic domain (like `Clerk` and `Customer`). Moreover, instances of a **Dimension** can be selected by selecting instances of a related one.

3.3 Conclusions

The first part of this chapter paid special attention to "Data Warehousing" schemas architecture, and their conceptual design. The knowledge in the FIS field has been used. Doing this allowed us to consider the integration work as already done. Besides, it invited to consider some problems, regarding data schemas and data protection, from a different point of view.

By locating the different "Data Warehousing" schemas in an architecture for FIS, an integrated architecture that comprises both areas has been obtained. The "Data Warehousing" terminology used by other authors was placed in that architecture. The characteristics of the different schemas, as well as the functions they realize have also been emphasized. The **DW Schema** has been presented as the result of a data-driven design, while the **DM Schemas** result from a query-driven design.

Along the second half of the chapter relationships between star schemas have been described. How different **Star**s can be related by *Derivation*, *Generalization*, *Association*, or even *Flow* relationships (in UML terminology) has been explained. The usage of those relationships, between analysis dimensions and the different kinds of cells, to navigate or **Drill-across** between **Star**s has been shown. Moreover, these relationships could also be used in other multidimensional operations like **Dice** (multidimensional operations are explained in section 5.4).

These relationships are not only useful for analysts, but also for designers. It has been exemplified that semantic relationships between **Dimension**s have nice consequences for aggregation hierarchies inside **Dimension**s, since two related **Dimension**s contain related aggregation hierarchies. The same stands for different **Cell**s in related **Fact**s. Therefore, relationships between **Star**s can also be used to drive designers work, so that they can detect inconsistences and errors. Conformed **Star**s would also drive users to the usage of data in a uniform way.

Chapter 4

Elements of a multidimensional model

> "You mentioned your name as if I should recognize it, but beyond the obvious facts that you are a bachelor, a solicitor, a freemason, and an asthmatic, I know nothing whatever about you."
>
> Sherlock Holmes, "The Norwood Builder"

In this chapter, the different elements of a multidimensional model are studied in order to know things about **Fact**s and **Dimension**s. From some basic definitions and general concepts, their characteristics are deduced.

Multidimensional information can be shown at different aggregation levels (often called granularities) for each analysis dimension. Thus, in the first half of this chapter, i.e. section 4.1, the benefits of understanding the relationships between aggregation levels as part-whole relationships, and how it helps to address some semantic problems are outlined. Moreover, the consequences of the incorporation of other Object-Oriented constructs in the hierarchies of analysis dimensions is analyzed.

In the second half of this chapter, i.e. section 4.2, the meaning of **Fact**s, and the dependencies in multidimensional data is studied. This study is used to find relationships between data cubes (in an Object-Oriented framework).

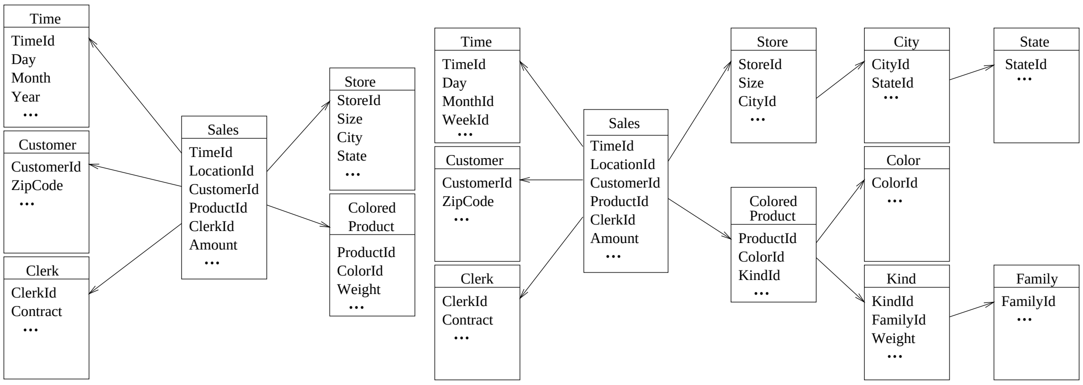
4.1 Analysis dimensions

This section is devoted to investigate problems regarding the representation of analysis dimensions, and their aggregation hierarchies at conceptual level. The stress is on how to solve those problems by showing aggregation semantics and navigation paths along the analysis dimensions. The importance of semantically rich relationships and their usage in conceptual modeling is outlined in [Sto93]. A first approach to how multidimensional modeling could benefit from O-O semantics has already been shown in section 2.4.

Most of those models mentioned in section 2.3 provide some way to represent aggregation hierarchies. Nevertheless, those papers treat the semantics of conceptual modeling constructs rather superficially, often just pointing to a general idea.

Section 4.1.1 discusses benefits of expliciting analysis dimensions. Then, from some well identified semantic problems enumerated in section 4.1.2, the usage of certain modeling abstractions to solve them is studied. These problems are addressed from an O-O point of view in section 4.1.3. Specifically, the usage of *Association*, *Aggregation* and *Generalization* relationships is analyzed.

4.1.1 The importance of aggregation hierarchies



(a) "Star schema" (b) "Snowflake schema"

Figure 4.1: Example of normalization of analysis dimensions

[Kim96], as well as other authors (like [Gio00]), argue that "snowflaking" dimension tables (which implies a higher level of normalization, as shown in figure 4.1) is a serious mistake, except for a reduced set of specific cases. From their point of view, even though it saves some (negligible) storage space, it intimidates users by unnecessarily complicating the schema, and slow down most forms of browsing among dimensional attributes (joins are slower and less intuitive than selections). That normalization would also explicit aggregation hierarchies. These hierarchies would show how **Measure**s can be summarized (known as "roll-up") or decomposed (known as "drill-down"). Nevertheless, they argue that the hierarchies are necessary neither to "roll-up", nor to "drill-down", since these are implicit in attribute values.

However, some people disagree with those ideas (see [PJ99] or [LAW98], for instance), and contend that aggregation hierarchies should be explicit; since they provide basis for defining aggregate data, and show navigation paths in analysis tasks. [HS97] presents a description logics model, which describes aggregation hierarchies as partially ordered sets with part-whole relationship being their strict order. In [TBC99], a multidimensional model, which allows the usage of "specialization", "aggregation", and "membership" relationships, is proposed. The authors claim that **Dimension**s are usually governed by associations of type "membership" forming hierarchies that specify granularities. [TPG00] also used Object-Oriented concepts to model **Dimension**s. Specifically, *Associations* define a directed acyclic graph between aggregation levels, and *Generalization* represents categorization of aggregation levels, allowing to define additional features of the subtypes. There are also some papers specifically related to aggregation hierarchies in analysis dimensions, like [JLS99], and [PR99].

Actually, the context makes the difference. If we are at a logical or physical design phase, as in [Kim96], it is possible to obtain better performance or understandability by denormalizing some tables. However, at a conceptual level, we must represent aggregation paths besides their different semantics. If this puts obstacles in the way of non-expert users understanding schemas, the user interface can hide as much information as necessary to make it understandable to a given user. Performance problems of the system will be addressed at further design phases (i.e. logical and physical).

As already stated in literature, it is important to separate conceptual and physical components. Logical or physical models are semantically poorer than conceptual ones. Conceptual models are very important, because they give to the user much more information about the modeled reality, and are closer to his/her way of thinking. This is specially necessary in analysis tasks, because of the unpredictable nature of user queries in these environments. This kind of users can not be restricted to a small set of predefined queries. Indeed, they need to generate their own queries, most of times based on metadata. Thus, it is essential for a conceptual model to provide means to show aggregation hierarchies, and as much semantics as possible. For instance, showing that two analysis dimensions are specialization of another one, means that their instances (for example, customers and clerks) can be compared.

4.1.2 Semantic problems in present multidimensional modeling

This section outlines some problems found in existing multidimensional models. Some of them were already identified in [SR91], [Leh98] and [PJ99]. Even though [SR91] can be considered as out of place, most of the problems it identifies in statistical modeling are also applicable in the multidimensional context. The problems, related to modeling **Dimension**s, are grouped into five sections.

Aggregation levels graph

At first glance, one could think that aggregation levels graphs are quite simple. Data about stores is aggregated based on the city they belong to, data about cities is aggregated based on the state they belong to, and so on. Although the aggregation hierarchy looks linear and simple, it simply suffices to look at the `ColoredProduct` **Dimension** to find that products can be aggregated either by color or kind. We can see other examples of multiple aggregation paths in [Tho97].

Some OLAP tools impose the constraint that an aggregation graph must be connected and show parent-child relationships between attributes. [LAW98] imposes the existence of a common top aggregation level (called `All`), defining a lattice of aggregation levels for every analysis dimension; and identifies relationships between **Level**s as functional dependencies. [PJ99] also identifies multiple aggregation paths in the same **Dimension**, and presents the different aggregation levels forming a lattice, being related by "greater than" relationships (meaning "logical containment" of the elements at one level into those at the other). It could also be the case that our information sources feeding the DW collect data at `Month`, and `Week` **Level**, but not at `Day` **Level**. Therefore, we could define a common aggregation top, but not a common bottom for both aggregation paths.

There is no justification in literature of the structure of aggregation levels in an analysis dimension and the relationships among them being a lattice, semi-lattice, or just a directed graph. It is necessary to find a wide accepted definition of analysis dimensions. This is the first step to state its structure and properties.

Relationship cardinalities

Almost all the related research argues that aggregation hierarchies are formed by "to-one" relationships. It means that an element at a given aggregation level is related to exactly one element of the next aggregation level in the hierarchy. A store corresponds to exactly one city; a city, in turn, to exactly one state; and so on. As pointed out in [LAW98], this provides nice aggregability properties.

However, we can find examples where hierarchies are not defined by "to-one" relationships in [SR91], [Kim96], and [Tho97]. [PJ99] also presents examples where the dimension hierarchies, besides possibly being "to-many", can be non-covering. In general, the most common (and computationally comfortable) cardinalities are 1..N-1..1 and 1..1-1..1 (meaning minimum..maximum cardinalities at lower-higher aggregation levels).

A difficulty slightly related to this is that of having different path lengths between instances at two aggregation levels in the dimension hierarchy. An instance a at aggregation level L_1 is part of b at aggregation level L_2, which in turn is part of c at aggregation level L_3. However, there can be another instance e at aggregation level L_1 that is directly part of d at aggregation level L_3. This is identified by [PJ99] as non-onto hierarchies.

In general, we could find sixteen different cardinalities for relationships between two aggregation levels (i.e. two - 0 or 1 for minimum, and 1 or N for maximum - raised to the power of four), most of them presenting summarizability problems. Thus, it is needed to clearly identify meaningless cardinalities to avoid misunderstandings on designing, as well as the meaningful ones to strive to solve any problems they generate.

Heterogeneous aggregation levels

[SR91] detects a problem referred as "non-homogeneous statistical objects". This means having objects at the same aggregation level that have different attributes. For example, instances in `People` **Dimension** will have different attributes and will be classified into different categories whether they act in a sale as clerk or customer.

In [Leh98], this is solved by defining the attributes at instance level. However, as pointed out by some authors (see [BSHD98]), explicit separation of cube structure and its contents is a desirable model feature. In this sense, attaching specific attributes to every instance, does not seem a good solution. [LAW98] also tackles the problem, and proposes to solve it by means of attributes with "null" values (showing that a given attribute is non applicable), and restricting the usage of these attributes to selection of instances (forbidding grouping by them). The solution in [BHL00] is much more elegant. It proposes to define different Relations for every set of instances sharing the same attributes.

Still, it is not enough to solve the problem at logical level (by means of Relations). Modeling the concepts so that more semantics are captured is also important. [TBC99] and [TPGS01] propose to specialize the aggregation levels. Nevertheless, they do not study the consequences of such semantic relationship in aggregation hierarchies.

Reuse of analysis dimensions

Multidimensional data cubes are conceived in an isolated manner. However, when we use them, we want to navigate from a kind of fact to another one (known as "drill-across"). This means we are analyzing data in a **Fact** from a given point of view, and want to view data in another **Fact** from the same point of view. Thus, **Fact**s need to have equivalent points of view (i.e. **Dimension**s). Moreover, we can also find the same **Dimension** playing different roles in a **Star**. For instance, in a sale, `People` **Dimension** plays two different roles (i.e. `Clerk` and `Customer`).

Most multidimensional models ignore "drill-across". If it is considered, like in [Kim96], this operation is restricted to the case that both **Star**s have common "dimension tables". As exemplified in [SBHD99], two **Star**s could also use the same analysis dimension at different aggregation levels, still allowing "drill-across".

Multidimensional analysis and research is usually restricted to one **Fact**. Representing inter-dimension relationships would allow more powerful analysis by relating data in different **Star**s. The more semantically rich these relationships are, the better for the analysts.

Correlated analysis dimensions

In general, analysis dimensions use to be independent. Thus, the point of view chosen at one of them does not restrict those possible values available at others. However, we can find some cases where there exist meaningless combinations of dimension values (they are correlated). For instance, it may be that all products are not on sale everywhere. Depending on the product characteristics, it is sold in a store or not. Some other examples of this situation can be found in [Kim96], referring the problem as "many-to-many relationships". If values in two analysis dimensions are correlated, we could choose to keep both in the same "dimension table".

There is no multidimensional conceptual model able to capture this kind of relationship. However, it is needed to capture, at conceptual level, the possibility of combining different **Dimension**s to give rise to a new one. Representing both **Dimension**s together, at logical or physical level, would depend on the number of meaningful combinations with regard to the number of elements of the correlated **Dimension**s.

4.1.3 How to solve them

Relationships between aggregation levels should be interpreted as part-whole (also known as composition) relationships. This allows us to use "Classical Extensional Mereology" (CEM) axioms and other concepts in [GP95] to address problems stated in previous section.

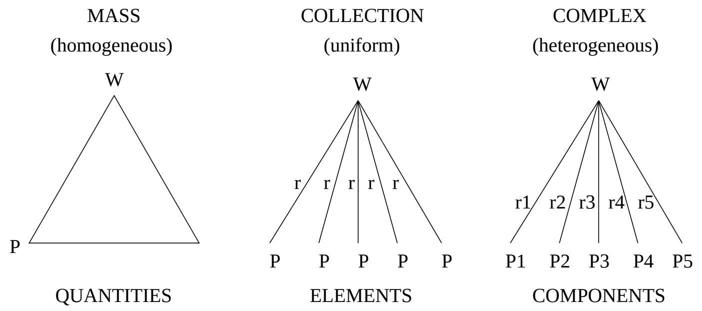


Figure 4.2: Types of wholes

As depicted in figure 4.2, we find three different, domain-independent, kinds of part-whole relations induced by the compositional structure of the whole (i.e. "Mass", "Collection", or "Complex"). If there is no compositional structure, the whole is considered "homogeneous" (for example, an amount of rice). If we take into consideration different elements, it is understood as a collection having a "uniform" compositional structure (for example, a convoy of trucks). If we see different parts playing different roles, we have a complex with an "heterogeneous" compositional structure (for example, the pieces in an engine). "Mass", "Collection", and "Complex" represent extreme cases on a scale leading from a total lack of compositional structure to wholes with complex internal organization. Different people could conceive a composed element at different points of that scale.

The main objective of defining relationships between different instances in an analysis dimension is to show how to apply aggregation functions (i.e. sum, min, max, avg, etc.). Since these functions consider instances as equals (playing the same role in the aggregation), those relationships should be conceived as collections. From here on, part-whole relationships between aggregation levels in an analysis dimension should be understood as forming collections.

In case of having collections, [GP95] considers that the axiomatic system of CEM (as stated in figure 4.3, that is also explained in [AFGP96]) seems to be ideally suited, except for axiom 6. In our case, axiom 6 also perfectly suits, since a user can always be interested in considering a given set of elements as a whole, in order to apply an aggregation function. Semantically, axiom 5 is not true, since the same collection of elements could compose different wholes (i.e. two clubs, at a given point in time, can have the same set of members). However, in order to apply aggregation functions both collections would give the same result. Thus, we would not be talking about clubs, but just sets of members which would be the same individual.

1. EXISTS. If A is part of B, both A and B exist
2. ANTISYMMETRY. If A is part of B, B is not part of A
3. TRANSITIVITY. If A is part of B and B is part of C, then A is part of C
4. SUPPLEMENTATION. If A is a proper-part-of B, then another individual C exists which is the missing part from B
5. EXTENSIONALITY. A and B have the same parts, if and only if A and B are the same individual
6. SUM. There always exists the individual composed by any two individuals of the theory

Figure 4.3: Classical Extensional Mereology axioms

[GP95] also explains that there might be more than one way to decompose the same whole, i.e. some objects could be understood as collection of different kinds of elements (for instance, a year being a collection of either trimesters or four-month periods).

Relationships inside an analysis dimension

Some models, like [CT98a], and [GMR98b], already stated that **Dimension**s contain different **Level**s which represent domains at different granularities. Those granularities show how elements are grouped to apply aggregation functions. Thus, relationships are defined among elements at different aggregation levels standing for composition.

Along Aggregation/Decomposition OO-Dimension, we find different kinds of relationships based on their strength. Those that do not stand for composition or part-whole relationships are *Association*s. In this kind of relationship, an instance is related to another just to show a property of the second one. Every instance in an analysis dimension will be related to some instances because of those being its parts, and to other instances because of those simply showing its properties.

It is essential to distinguish both kinds of relationships in a multidimensional model, since they will allow to understand what was intended on defining a given schema. Part-whole relationships will show how different elements are grouped together in a **Dimension**, while *Association*s will indicate which are the different characteristics available to select instances. Thus, "roll-up" and "drill-down" operations will be performed along part-whole relationships, while selection (known as "slice-dice") will be performed by means of *Association* relationships.

A minimum definition, that everybody could agree, in order to deduce some controversial properties of an analysis dimension using CEM axiomatic system is introduced here. Firstly, on referring to aggregation levels in multidimensional analysis, there is a misuse of language on saying, for instance, "A city decomposes into stores". The real meaning is easily inferred, but it is important having in mind that it should be said "A set of stores in a city decomposes into stores".

An analysis dimension can be defined as follows:

Definition 1 *A* **Dimension** *is a connected, directed graph representing a point of view on analyzing data. Every vertex in the graph corresponds to an aggregation level, and an edge reflects that every instance at target* **Level** *decomposes into a collection of instances of source* **Level** *(i.e. edges reflect part-whole relationships between instances of* **Level***s).*

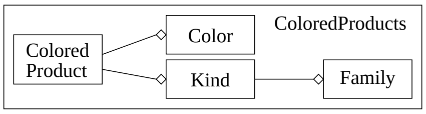


Figure 4.4: Example of analysis dimension

In O-O terminology, **Level**s would be classes, and their instances would be objects. Figure 4.4 shows an example of **Dimension**. It contains a graph with four aggregation levels (i.e. `ColoredProduct`, `Color`, `Kind`, and `Family`), and three edges showing that families of products can be decomposed into different kinds of products, and these into colored products which can be grouped by color. To avoid identification problems pointed out by some authors, we can assume that instances of the **Level**s have unique OIDs.

From definition 1 and CEM axioms, some properties can be deduced with regard to analysis dimensions:

Property 1 *A* **Dimension** *does not contain cycles.*

Proof 1 *Let us suppose that a cycle in the dimension graph exists. By successively considering axiom 3 on any instance A of a* **Level** *forming the cycle, we would obtain that exists another instance B of another* **Level** *forming the cycle so that A is part of B and B is part of A. This contradicts axiom 2, then a cycle can not exist in the graph of a* **Dimension***.*

Property 2 *For every* **Dimension***, there exists a unique aggregation level* `Atomic` *which contains elementary (i.e. that can not be broken down) instances. Notice that elementary instances could be unknown in a given database.*

Proof 2 *By property 1, there is at least a* **Level** *whose instances do not have parts. If there is more than one of those* `Atomic` **Level***s, since a* **Dimension** *is connected and axiom 3, there will exist an instance E conceived as composition of elementary instances at each one of the* `Atomic` **Level***s. By axiom 5, all those collections of elementary instances composing E must be the same collection of elements. Therefore, there exists only one* `Atomic` **Level***.*

Property 3 *For every* **Dimension***, there might exist a level* `All` *containing instances composed by all elementary instances in the* **Dimension***. If this level exists, a) Its instances are not collected by instances at any other aggregation level; b) This aggregation level has exactly one instance; and c) It is unique in the* **Dimension***.*

Proof 3 *By successively considering axiom 6 we can construct an instance E composed by all elementary instances in the* **Dimension***. a) If E would be a proper-part-of an E', by axiom 4 there would be an elementary instance that is not in E. Therefore, E is an instance of a* **Level** *whose instances are not part of any other instance in the* **Dimension***, which contradicts the condition. b) If this* **Level** *would contain two instances, both containing all elementary instances, by axiom 5 they would be the same instance. c) This* **Level** *is unique, since if there were another* **Level** *whose instances collect all elementary instances, they would be the same instance we already have in* `All` *level (by axiom 5).*

Property 4 *Those* **Level***s whose instances are not collected by instances of any* **Level** *(i.e. they are not source of edges in the dimension graph) can be connected with an edge to* **Level** `All`*.*

Proof 4 *The instance of* **Level** `All` *can be decomposed into instances at any* **Level** *covering* `Atomic` **Level***. If there is a* **Level** *not covering* `Atomic` **Level***, a collection can be added to it, by axiom 6, collecting every elementary instance missing.*

Property 5 *Every instance of a* **Level** *that is not* `Atomic` *has at least one part.*

Proof 5 *An instance without parts is elementary, and all elementary instances are at* `Atomic` **Level***, by property 2.*

Property 6 *Every instance of a* **Level** *that is not* `Atomic` *might have more than one part.*

Proof 6 *If the part-of relationship between two instances is a proper-part-of, by axiom 4 the collection will have more than one part.*

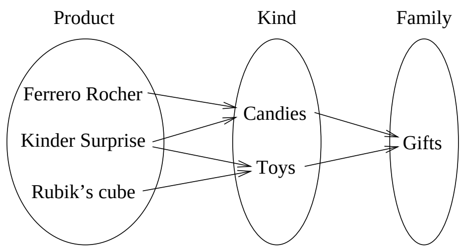


Figure 4.5: Example of overlapping wholes

Property 7 *An element might be part of several collections at the same time.*

Proof 7 *There is no mereological axiom forbidding the sharing of elements among several collections, in spite of it is a necessary condition to ensure summarizability (as shown in [LS97]). Allowing this case is not a conceptual, but a computational problem (addressed as so in [PJ99]). If, as depicted in figure 4.5, a given product (at* **Level** `Product`*) is allowed to belong to two different kinds of products at the same* **Level** `Kind`*, some derived attributes of instances of* **Level** `Family` *(which are composed by elements at* **Level** `Kind`*) must be calculated from elements at* **Level** `Product` *(for example,* $card(\texttt{Gifts}) \neq card(\texttt{Candies}) + card(\texttt{Toys})$*).*

Property 8 *If* **Level** `All` *exists in the* **Dimension***, the graph is a lattice, and collections in each* **Level** *are disjoint; then for every* **Level** S*, every instance in it is part of a collection at each and every other* **Level** T *being target of edges leaving source* **Level** S*.*

Proof 8 *A lattice with* `All` **Level** *at top, by axiom 3, implies that every elementary instance is collected in at least one instance of any other* `Level`*. By imposing that collections in a* **Level** *are disjoint, we obtain that every element in* S *must be collected exactly in one collection in* T*. If elements were not disjoint, there could be an instance of* S *overlapping several collections in* T*, so that it would not be completely contained into any of them.*

With regard to problems stated in section 4.1.2 regarding the graph of aggregation levels, from definition 1 and properties 1 and 2 we ensure that, in general, those aggregation levels in a **Dimension** form a semi-lattice. Moreover, properties 3 and 4 show that `All` **Level** can always be defined in order to obtain a lattice. Those problems about relationships cardinalities are explained by the other properties. Properties 5 and 6 imply that the relationships between two **Level**s will involve 1..N parts for every whole. Property 7 explains that a part could participate in more than one whole or not. Property 8 shows that if we have a lattice with **Level** `All`, and parts do not participate in more than one whole; there is a whole for every part (i.e. we have cardinality 1..N-1..1). If the same part can participate in more than one whole at the same **Level** we can not guarantee that there is a whole for every part (even if `All` exists in the **Dimension**, we have cardinality 1..N-0..N). In any case, axiom 6 shows that the needed instances could be obtained to have 1..N wholes for every part (so that we have 1..N-1..N).

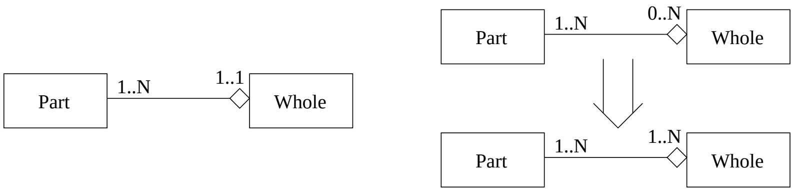


Figure 4.6: Allowed cardinalities between **Level**s

Figure 4.6 summarizes the allowed cardinalities in an aggregation hierarchy. There are two possibilities, both with at least one part for every whole. The most common case is we find exactly one whole for every part. However, it is also possible that a given part belong to several wholes. In this case, if we find parts that do not participate in any whole, wholes can always be built so that every part participate in at least one.

Relationships between analysis dimensions

It is not enough showing relationships inside a **Dimension** or **Level**. It is also important to analyze relationships between elements of analysis dimensions in different star schemas or even in the same one. In this section, we are going to consider two kinds of semantic relationships, i.e. *Generalization*, and *Aggregation*.

Generalization The usage of *Generalization* relationships between aggregation levels is proposed in [TBC99], and [TPG00]. Doubtless, *Generalization* is an essential relationship to be shown in multidimensional schemas. Nevertheless, isolated aggregation levels can not be specialized to show more specific meanings. They must be considered inside a **Dimension**.

Property 9 *In general, a* **Level** *and its specialization can not belong to the same* **Dimension**.

Proof 9 *Let us assume that both a* **Level** *L and its specialization L_S are in the same* **Dimension**. *In order to define a lattice with* **Level** `All`, *since in this case L_S must cover* `Atomic` **Level**, *we could be forced to have some instances in L_S. Those instances we are forced to have in L_S, could not fulfill specialization criterion. Therefore, it is not always possible to have both* **Level***s in the same* **Dimension**.

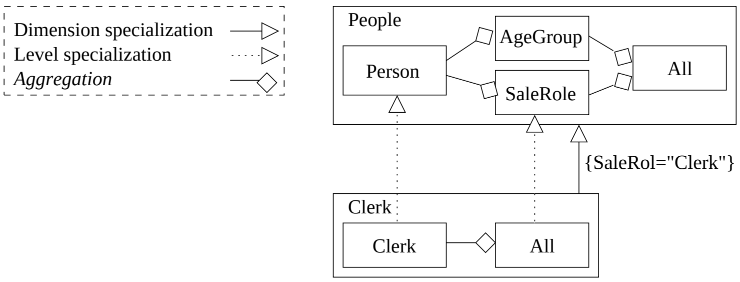


Figure 4.7: Example of dimension specialization

Figure 4.7 shows an example where `People` **Dimension** is specialized at `SaleRole` **Level** (solid arrow) to have a `Clerk` **Dimension**. This specialization contains a **Level** with all people acting as clerk, and another one with only one element (which is also an instance of `SaleRole`) representing the set of all clerks. Dotted arrows show that a **Level** is specialization of another one (the instance of `Clerk::All` is that one of `SaleRole` fulfilling the specialization criterion "SaleRole='Clerk' "). `AgeGroup` **Level** is not of interest in `Clerk` **Dimension**. Notice that if it would, it would not be specialization of the homonym **Level** in `People` **Dimension** since its instances would be different (they would collect less people).

Generalizing the example, if D_S is the specialized **Dimension** of D at **Level** L, D_S contains at least the **Level** L_S (specialization of L), and a specialization of every **Level** in D containing parts of instances of L_S. These specialized **Level**s contain exactly those instances of the corresponding **Level** of D being part of any collection in L_S. Besides those mandatory **Level**s in D_S, it is also possible that D_S contain other **Level**s (that are not specialization of any **Level** in D) with elements not in D.

All instances of a **Level** will have common properties, since it represents a given class of objects able to play the same role in a collection. By specializing a **Dimension**, we will be able to show attributes common only to a subset of instances, besides their specific part-whole relationships, which solves problems presented in section 4.1.2 as heterogeneous aggregation levels. *Association* as well as *Aggregation* relationships are inherited along specializations. Therefore, it also addresses the reuse of analysis dimensions. It is not only possible to "drill-across" from a **Star** S_1 to a **Star** S_2 when both share **Dimension**s, but also when the **Dimension**s of S_1 are specialization of those in S_2.

Semantics are not only useful for users, but they can also improve query performance. In the example, `Clerk` and `Customer` are specialization of the same class, i.e. `People`. On comparing instances in those **Dimension**s, if the specialization is disjoint means they will always be different. Just knowing whether it is covering or not, would allow to obtain thresholds of aggregation results. The specialization being covering and disjoint also suggests parallel computing.

Aggregation Another interesting relationship to be shown is that of elementary instances in a **Dimension** being aggregated in elementary instances in another **Dimension**. This means expressing *Aggregation* relationships between **Dimension**s.

Property 10 *If elementary instances in a* **Dimension** *D are part of elementary instances in* **Dimension** *D_A, the graph of D will be a subgraph of D_A. Notice that instances in D will not be those in D_A, but part of them.*

Proof 10 *Elementary instances in D_A can be grouped so that the same elementary instance in D is part of every element in each collection. By axiom 6, these collections can become instances in D_A. Then, instances in D_A can be grouped by the same criteria used on grouping elements in D.*

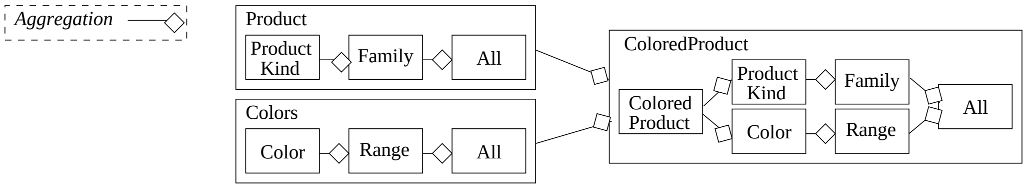


Figure 4.8: Example of dimension aggregation

Besides having `Sales` by `ColoredProduct`, we could obtain data in another star schema by `Color`, or `ProductKind`. Instances in these **Dimension**s would be aggregated to show the kind of product sold, and the color of that product. As depicted in figure 4.8, the composed **Dimension** would contain, at least, the graph of each one of the parts, joining `All` levels, plus a common `Atomic`. However, notice that, for example, instances of `Colors::Color` and `ColoredProducts::Color` do not coincide. While the former represent colors, the latter represent groups of products grouped by colors.

By means of *Aggregation* relationships between **Dimension**s, we address the problem found in section 4.1.2 as correlated analysis dimensions. Two **Dimension**s aggregated to generate a new one mean that there is a relationship between them that should be considered, at design and query time.

4.2 Facts subject of analysis

The aim of this section is to clarify some concepts about facts, and how they should be modeled (this was already done for dimensional data in previous section). "Functional Dependencies" (FDs) were successfully used on developing Relational theory. Thus, how they could also be used to explain multidimensionality is going to be shown here. This does not mean pleading for ROLAP as opposed to MOLAP tools. The discussion is placed at conceptual level, and it is independent of any kind of underlying system.

By better understanding multidimensionality and how it should be modeled, we can obtain several benefits. Firstly, it will help on designing multidimensional schemas, as normal forms do for relational ones. Secondly, users will also benefit from it, since querying will be easier and more understandable. Finally, storage and retrieve systems could also be improved, if knowledge about the real meaning of data is improved.

The meaning of multidimensionality has not been unambiguously stated in the past. This section is not going to re-discover multidimensionality, but just clarify and justify some points. Section 4.2.1 mentions some multidimensional data models that contribute in one way or another to modelize factual data. Then, section 4.2.2 explains multidimensional concepts (placing them at different detail levels), with regard to n-dimensional spaces and FDs between them. It also exemplifies some relationships between **Fact**s and **Cube**s.

4.2.1 Factual data in other models

Lots of work have been devoted to multidimensional modeling. Out of all papers devoted to this subject, some pay more attention than others to modeling facts at conceptual level. [GMR98b] defines a "fact schema" as a set of "measures" related to "dimension attributes". In [SBHD99], "facts" are specialization of "relationships" (in E/R sense). In [CT98a], a "fact" is defined as a function over the cartesian product of domains of its analysis dimensions. [HS97] defines a "cube" as an object which is associated to cells of similar form.

[Kim96] states that the "fact table" has a composite primary key made up of the foreign keys to its "dimension tables". [Gio00] agrees on that, and emphasizes that records in the "fact table" represent points in the multidimensional space. [BPT97] defines aggregation hierarchies in terms of FDs between sets of attributes. [LAW98] contains a proposal of normal forms for multidimensional modeling, based on "weak functional dependencies". It states that there is a functional dependency from analysis dimensions to "summary attributes" (i.e. **Measure**s). A schema in "multidimensional normal form" means analysis dimensions are orthogonal to each other, and "summary attributes" are fully functionally determined by the set of "terminal category attributes" (i.e. atomic aggregation levels).

FDs in the context of multidimensional databases need much more attention. A theoretical, wide study of dependencies is in [Tha91]. For a more application-oriented explanation of dependencies, [EN00] contains two chapters devoted to dependencies and normal forms in Relational databases, and how they help on designing.

4.2.2 Multidimensional elements unleashed

The DW contains lots of **Measure**s analysts want to understand and compare. Studying all together would be almost impossible. In this section, we are going to see how these data can be successively grouped at different detail levels to ease its management. We will have **Measure**s grouped into cells, of different *Class*es (that can be seen as n-dimensional **Cube**s), which will be grouped based on the kind of fact (i.e. **Fact**) they represent.

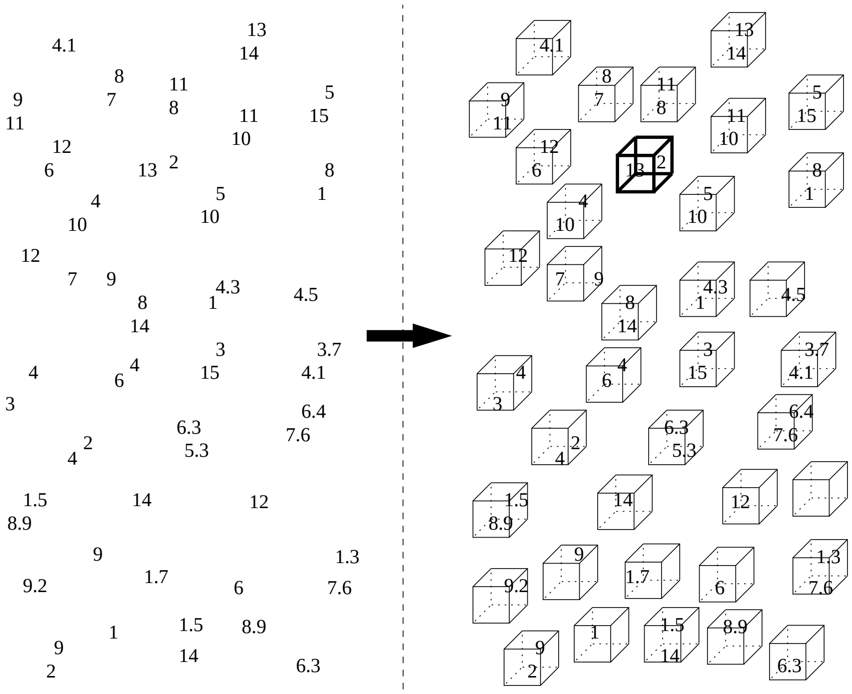


Figure 4.9: Measures grouped into **cell**s corresponding to facts

Measures and cells

Usually, for the same kind of fact subject of analysis, at **Lower** detail level, we have several **Measure**s. For instance, for a `Sale` we could keep `cost`, `revenue`, `amount of product`, etc. Thus, when a cloud of measurements must be faced, those corresponding to the same fact are always grouped in the mind of analysts.

Definition 2 *A* **cell** *contains a (possibly empty) set of measurements, and represents a given fact.*

Figure 4.9 sketches this by drawing several measurements. Those that correspond to the same fact are inside a **cell**, which represents the fact. One of these **cell**s (i.e. an instance of a kind of fact) contains all measurements we have about what was sold to John Doe last Monday in Barcelona (i.e. we sold him 2 items and charged 13€).

Nevertheless, grouping measurements of the same fact is not enough to be able to make decisions. Several facts can be grouped, and it gives rise to more complex facts. Algebraically, the set of **cell**s (C) representing all possible facts in the DW forms a commutative semigroup with union ($x \cup y$ means **cell**s x and y are grouped into a new, complex **cell**). Notice that **Measure**s are not considered in the discussion. This deals with **cell**s (that could have attributes or not), so that summarization functions are not taken into account by now. $< C, \cup >$ fulfills the following properties:

Closed: $\forall x, y \in C, x \cup y \in C$

Commutative: $\forall x, y \in C, x \cup y = y \cup x$

Associative: $\forall x, y, z \in C, (x \cup y) \cup z = x \cup (y \cup z)$

Neutral element: $\forall x \in C, x \cup \emptyset = x$

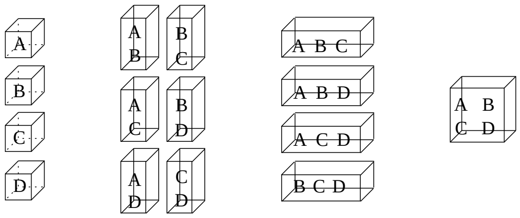


Figure 4.10: $\mathcal{P}(C_A)$ being $C_A = \{A, B, C, D\}$

If we call C_A the set of all **cell**s representing atomic facts (i.e. those that cannot be decomposed) and we allow the union of any kind of **cell**, $\mathcal{P}(C_A)$ should be considered, which contains $2^{Card(C_A)} - 1$ **cells** (figure 4.10 shows a set with four atomic **cell**s).

Fortunately, what analysts really want to study is only a subset of $\mathcal{P}(C_A)$. This subset is defined by the different kinds of facts. We do not need to consider $\mathcal{P}(C_A)$ but, at most, $\bigcup \mathcal{P}(C_i)$, being every C_i the set of all atomic **cell**s of a given kind of fact so that $\bigcup C_i = C_A$.

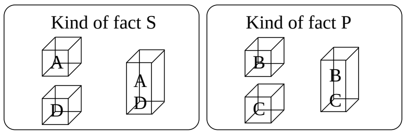


Figure 4.11: $\bigcup_{i \in \{S,P\}} \mathcal{P}(C_i)$ being $C_S = \{A, C\}$ and $C_P = \{B, D\}$

For example, if **cell**s `A` and `D` in figure 4.11 are of the `Sales` kind of fact (S), while **cell**s `C` and `B` are of the `Productions` kind of fact (P), $\bigcup_{i \in \{S,P\}} \mathcal{P}(C_i) = \{A, B, C, D, AD, BC\}$. Thus, in this case, analysts would not be interested on sixteen **cell**s, but only on six.

Analysis dimensions and aggregation levels

The facts only gain meaning when analysis dimensions identify them. If we subtract dimensional information from them, only mute numbers remain. Talking about sales is senseless, if you do not know who sold what, when, whom, etc. Thus, **cell**s are usually grouped to give rise to more complex **cell**s, which contain derived measurements. However, most combinations of **cell**s do not give rise to meaningful more complex **cell**s. It must be done based on analysis dimensions (for example, we should not group data regarding months with those regarding years). In section 4.1 we have already seen semantics and structure of **Dimension**s, which show the different points of view analysts use to study facts. Each **Dimension** contains a graph indicating how the facts can be aggregated along the analysis dimension (see definition 1 at page 74). In this case, those **cell**s that are identified by instances of the same **Level** are grouped to obtain a more complex **cell** identified by an instance of a **Level** above that.

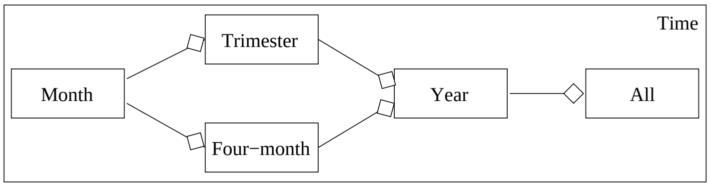


Figure 4.12: Example of **Dimension**

Figure 4.12 shows an example of **Dimension**, which contains five **Level**s, i.e. `Month`, `Trimester`, `Four-month`, `Year`, and `All`. Every instance of `Month` **Level** represents a month, which can be aggregated in two different ways to obtain either trimesters or four-month periods. Both kinds of instances (i.e. `Trimester` or `Four-month`) can be grouped to obtain years. Finally, at top we have `All` **Level** with exactly one instance representing the group of all months in the **Dimension**.

Classes of cells

We can associate every atomic **cell** to an instance of a **Level** in each of its analysis dimensions, showing the meaning of its measurements. If all **cell**s in a complex **cell** are associated with instances in a **Level** l_1, and there exists an instance of l_2 exactly composed by those instances in l_1, we can associate the complex **cell** with the instance of l_2. For example, if all **cell**s composing another one are at level `Month` and correspond to exactly those months in a trimester, the complex **cell** is associated to an instance of `Trimester` **Level**.

Definition 3 *A* Cell *(i.e.* Class *of* **cell***s) contains those* **cell***s representing the same kind of fact and being associated with instances of the same* **Level** *for each of the* **Dimension***s we use to analyze it.*

For example, all **cell**s representing sales during a given month in a given store by a given customer form a *Class*. Instances of this *Class* differ in one or more of the instances of the **Dimensions** they are associated to (i.e. the month it was sold, the store where it was sold, or the customer who bought it). Two **cell**s regarding the same kind of fact, and the same instance in every **Dimension** will be in the same **Cell**. Thus, **Dimension** instances identify **cell**s in a *Class*.

If we allowed to compare or group any set of **cell**s, we would find that there exist $2^{2^{Card(C)}-1}-1$ possible sets of **cell**s in $\mathcal{P}(\mathcal{P}(C))$. Thus, **Cell**s are defined to ease the study of these huge amount of sets of **cell**s. Only **Measure**s in **cell**s of the same *Class* can be compared or treated together, because they represent exactly the same kind of information (i.e. **Fact**) at the same granularity (i.e. **Level**). What's more, analysts are not interested in all **cell**s in $\mathcal{P}(C)$, but only in those corresponding to **Fact**s at a **Level** in each of the **Dimension**s. They are only interested in subsets of every $\mathcal{P}(C_i)$ determined by aggregation hierarchies in analysis dimensions. Aggregation hierarchies in the **Dimension**s restrict the union of **cell**s to those of the same *Class*. For example, a **cell** associated to an instance of `Month` cannot be grouped with another **cell** at `Year` **Level** to give rise to a more complex **cell**.

Facts

Only **cell**s of the same *Class* can be grouped to obtain a coarser **cell**. Thus, instances of a **Cell** are obtained by union of **cell**s in another **Cell**. This is always done following aggregation paths in the analysis dimensions. The **Cell**s generated by grouping **cell**s in another **Cell** always regard the same kind of fact. Thus, we can group **Cell**s into **Fact**s at **Upper** detail level.

Definition 4 *A* **Fact** *is a connected, directed graph representing a subject of analysis. Every vertex in the graph corresponds to a* **Cell***, and an edge reflects that every instance at target* **Cell** *decomposes into a collection of instances of source* **Cell** *(i.e. edges reflect part-whole relationships between instances of* **Cell***s).*

Figure 4.13 shows an example of the structure of a **Fact** with two orthogonal **Dimension**s: `Time` already depicted in figure 4.12, and `Geographic` composed by `City`, `Region`, and `All` **Level**s. We can see that there is a **Cell** in the **Fact** for every combination of **Level**s in the

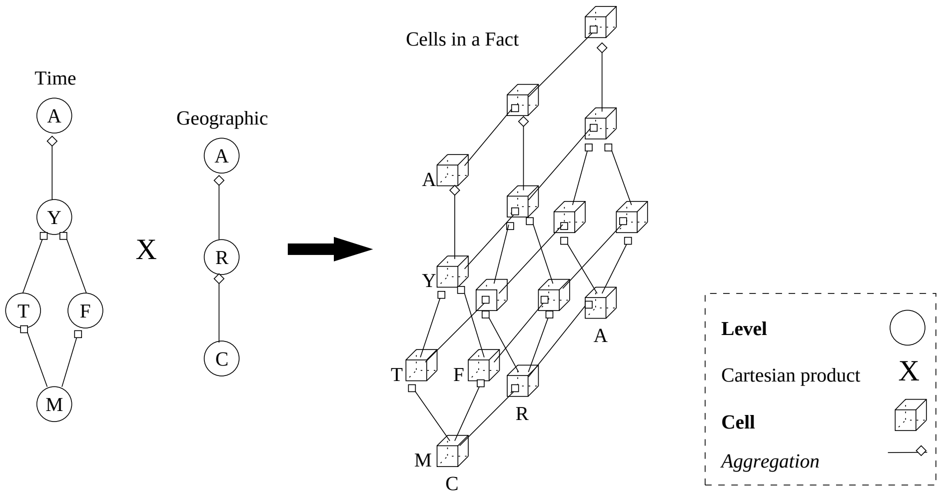


Figure 4.13: Graph of **Cell**s in a **Fact** with two **Dimension**s

Dimensions. Having two orthogonal **Dimension**s with 5 and 3 **Level**s respectively, means that the **Fact** will have 15 **Cell**s. These **Cell**s and the part-whole relationships between them form a lattice. All atomic **cell**s are in the **Cell** at the bottom, while the **Cell** at top contains only one **cell** which is the union of all atomic **cell**s.

A **Cell** may contain any kind of data. It uses to be numerical, because we always know how to summarize numerical data (i.e. "sum", "avg", etc.). However, we just need a set of aggregation operations for a non-numerical data type to be able to keep it in **cell**s. For instance, we could aggregate character strings by set-union. Therefore, we can also have descriptive attributes in **Cell**s. Aggregation operations, for boolean **Measure**s, would be "count", "and", "or", etc. Anyway, if the data types of the **Measure**s have an order, we can always aggregate calculating the median. Thus, **Measure**s in **Cell**s could always be aggregated to obtain the **Measure**s in **Cell**s with more complex instances (except if the summarization function is not transitive or the aggregation level is not a valid source due to any reason, as explained in section 5.3). Different aggregation functions (i.e. "sum", "average", "minimum", etc.) could be used to obtain different **Measure**s in a complex **Cell**.

Some **Cell**s could contain **Measure**s that are not obtained by aggregation of those from other **Cell**s. For example, some data could be collected yearly, so that **cell**s at `Month` **Level** cannot contain it. Moreover, [Gio00] distinguishes between analytical and non-analytical data. Sometimes, we are interested in analyzing data at a given aggregation level, and ignore atomic data. However, in spite of we might not collect **Measure**s at the lowest level of granularity due to either availability, performance or legal reasons (i.e. personal data use to be private), we could be interested in keeping some information about instances at that level (for example, names of people in the census). Thus, we have cases where **Measure**s in a **Cell** are not present for coarser or more detailed **Level**s.

If we know the **Dimension**s that define the **Cell**s in a **Fact**, we know which are those **Cell**s and how they are related. Thus, it could be inferred that it is not necessary to show them in multidimensional modeling. However, this is not true. As stated above, some **Cell**s could have specific **Measure**s, or other could be specially important to be shown to users. As some derived attributes are shown in a conceptual schema for the sake of completeness and clearness, so some **Cell**s with complex **cell**s should also be shown in a multidimensional schema. Most of those **cell**s will be calculated on the fly, but other could be physically stored to improve performance, or just keep specific **Measure**s.

If we only have "to-one" relationships between **Level**s, every one defines a partition of atomic **cell**s (those in the **Cell** at bottom). Each **cell** composes exactly one more complex **cell** in every **Cell** above its. Nevertheless, we cannot assume that, because having "to-many" relationships in the aggregation hierarchies implies we do not obtain partitions of the **Atomic** *Class*. In the worst case analysts could be interested in all $\mathcal{P}(C_i)$. Thus, "to-many" relationships in the aggregation hierarchies generate semantic, as well as computational problems on calculating derived **Measure**s for complex **cell**s, but this does not mean they should be forbidden in a multidimensional model.

Now, we are going to see how different **Fact**s can be conceptually related. In the following paragraphs, it is shown how some Object-Oriented relationships (i.e. *Generalization*, *Aggregation*, and *Derivation*) between **cell**s are represented as relationships between **Cell**s and **Fact**s.

Generalization As it was previously said, **cell**s in a given **Cell** could have **Measure**s that **cell**s in other **Cell**s do not have. For instance, if our company (may be the result of a fusion of preexisting smaller companies) is organized by autonomous regions, it could be that the information systems in one of these regions collect data that those in other regions do not. Thus, we will specialize the corresponding **Cell** depending on the region.

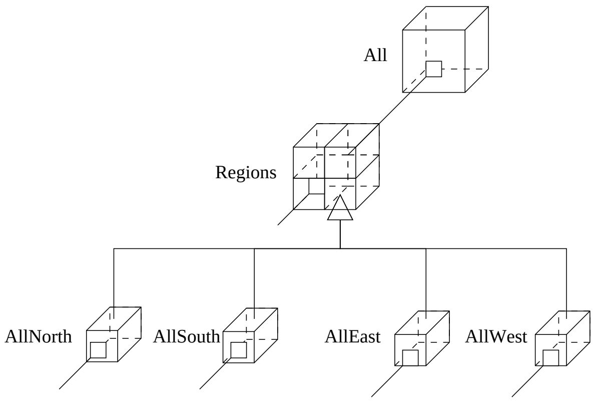


Figure 4.14: Specialization of a **Fact** based on a **Cell**

Specialization of **Cell**s is due to the specialization of the kind of fact they are representing. Specializing means dividing the instances of a superclass into different subclasses. Notice that the sets of atomic **cell**s in each of these "subclasses" (i.e. sets of `north`, `east`, `west`, and `south` **cell**s) are in $\mathcal{P}(C_i)$ (being C_i the **cells** in the superclass). Therefore, if they are meaningful for analysts, as depicted in figure 4.14, there will be a **Cell** in the **Fact** so that each subclass in the specialization corresponds to an instance of the **Cell** (i.e. the **Cell** will have four instances, one per region). Thus, we should rather specialize a **Fact** based on a **Cell**.

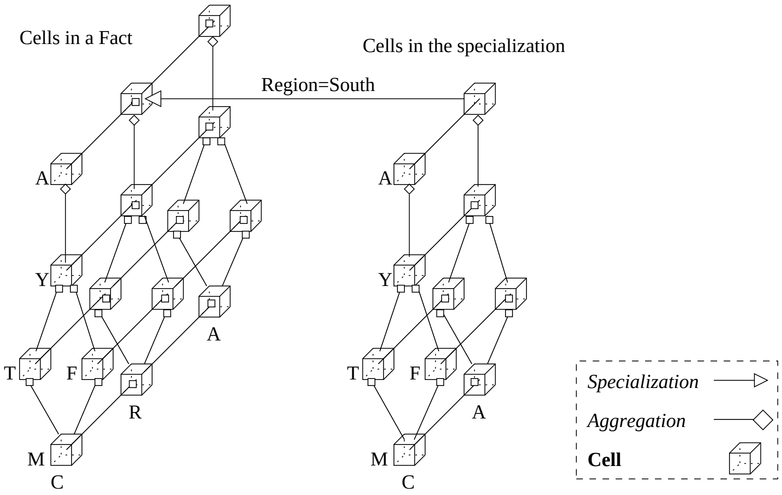


Figure 4.15: Specialization of a **Fact** by region

To specialize a **Fact**, we have to choose the appropriate **Cell** that contains the **cell**s corresponding to the desired "subclasses". Then, it is specialized into **Cell**s, with exactly one instance, that will be the **Cell** at top of the lattice of **Cell**s in the new more specific **Fact**s. The example in figure 4.15 shows the same **Fact** in figure 4.13, that we want to specialize now by region. Thus, we take the **Cell** containing data by regions and specialize it in one **Cell** with one **cell**. This gives rise to a new **Fact** having a **Cell**s sub-graph of that of the superclass, which will be the lattice having the subclass at top. Notice that `Geographic` **Dimension** in the **Fact** specialization is an specialization of the `Geographic` **Dimension** in the original **Fact** (i.e. `All` **Level**s do not coincide).

Aggregation We can also find that different **cell**s are aggregated to obtain a **cell** about another subject (a different kind of fact). For instance, a deal is composed by several individual sales. Notice that **Measure**s of `Deal` are not necessarily obtained from those of `Sale` (for example, discount in the deal). In this case, we do not group **cell**s along any analysis dimension. It does not generate coarser **cell**s in the same **Fact**, but **cell**s in another **Fact**. There could be, or not, coincidences in analysis dimensions. Depending on it, the **Cell**s lattice will have a more or less similar form.

The usefulness of this kind of relationships between **Cell**s is twofold. On one hand, it allows to define complex **Fact**s from simpler ones, which will improve understandability of data. On the other hand, two **Fact**s can be related, so that navigation between them is possible. If we are studying a set of sales, it can be interesting to see data corresponding to deals in which they were done. Coincidences or differences in **Dimension**s do not matter. We should be able to travel from a *Class* to another one just because the *Aggregation* relationship between the **Facts**.

Derivation Another possibility is that **Measure**s in a **cell** are obtained by processing **Measure**s in **cell**s about a different kind of fact. If this is the case, we say that there is a *Derivation* relationship between both **Cell**s (extensively, between both **Facts**). For example, on analyzing efficiency of employees, some **Measure**s could be obtained by processing the benefits of some products sold (the best sales, sales involving relevant products, etc.).

Derivation relationships can also be used to hide information, change names, or units of **Measure**s. Most **Dimension**s will be likely shared by both **Cell**s. However, The **Cell**s are related because of relationships between **cell**s, not because of the **Dimension**s. This does not correspond to part-whole relationships in the lattice of a **Fact**, because both *Class*es represent different subjects, and grouping of **cell**s is not performed by means of aggregation hierarchies, but by conditions over the **Measure**s themselves.

[Gio00] defines a "degenerate fact" as a **Measure** recorded in the intersection "table" of a many-to-many relationship between **Fact**s. It could be seen as data in a **Cell** being related to two different **Cell**s (by one-to-many relationships). Thus, we could also see it as two **Fact**s acting as **Dimension**s of another **Fact**. Therefore, the duality **Fact**-**Dimension**s only exists if we look to an isolated multidimensional schema. Looking to all multidimensional schemas together means that what is considered a **Fact** by an analyst, could be considered a **Dimension** by another one, or vice versa.

The structure of **Cell**s in a **Fact** (a lattice) exactly coincides with that of **Level**s in a **Dimension**. Not only structure, but meaning coincides, as well. In both cases, there is an **Atomic** *Class* at bottom, whose instances are successively aggregated in instances of other *Class*es, until we obtain an instance of the top *Class* which contains all atomic instances. Both, **Fact**s and **Dimension**s, contain a graph of part-whole relationships between **Class**es. The difference is that the aggregation graph of a **Dimension** depends on its proper semantics, while the aggregation graph of a **Fact** depends on the aggregation hierarchies of its analysis dimensions. Thus, we could consider a **Dimension** as a 1-dimensional, self-qualified **Fact**. All we need to obtain a **Dimension** from a **Fact** is to express it in the appropriate base (as explained in next section).

Cubes

[LAW98] explains that analysis dimensions of a "summary attribute" should be orthogonal. This means that there are no dependencies between them. However, having no dependencies between any pair of analysis dimensions of a set of **cell**s could be a really strong constraint. Actually, what really matters is just having no dependencies between the dimensions of the space we are using for a given study. This means dependencies should be forbidden between the dimensions used on visualizing/storing data cubes.

Therefore, it is important to know the valid n-dimensional spaces that can be used on analyzing a given kind of multidimensional data, namely **Cell**. All possible combinations of instances in the **Level**s defining such spaces must be possible. This may be stated as multivalued dependencies with the empty set in the left hand side (**degenerated dependency**) for every pair of **Level**s. Being $\pounds$ the set of **Level**s used to visualize/store a **Cell**,

$$\forall L_i, L_j \in \pounds \ and \ i \neq j, \ \emptyset \rightarrow\rightarrow L_i \mid L_j$$

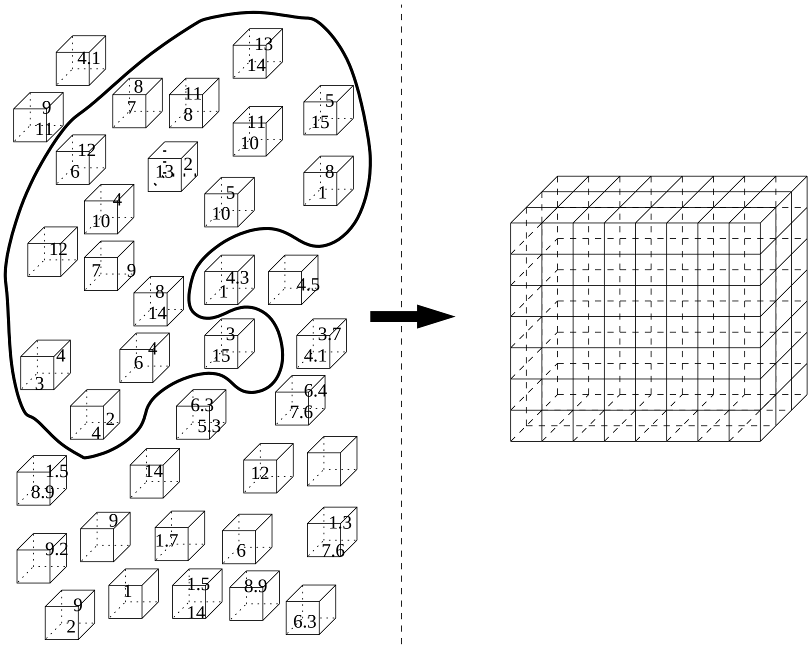


Figure 4.16: Diagram of a **Cell** with three independent analysis dimensions

Degenerated dependencies for every pair of **Level**s means we are talking about the cartesian product of all of them. Since we also have that **Level**s identify the **cells** in a *Class*, that cartesian product fully functionally determines the **cells**, i.e. $L_1 \times .. \times L_n \rightarrow C_{class}$. Thus, a **Cell** (either atomic or complex) determined by analysis dimensions could be drawn, as those in figure 4.16, forming an n-dimensional data cube. Notice that we could have "alternative keys", i.e. a **Cell** could be organized in different n-dimensional data cubes.

Definition 5 *A* **Cube** *is an injective function from an n-dimensional finite space (defined by the cartesian product of n functionally independent* **Level**s $\{L_1, .., L_n\}$*), to the set of instances of a* **Cell** *(C_c).*

$$c : L_1 \times .. \times L_n \rightarrow C_c, \ injective$$

Being a function means a **Cube** is not allowed to have "holes". Any combination of **Dimension** instances must be valid (i.e. related to a **cell**). However, missing **cells** should be allowed, if they mean that the fact is "unknown" or that it could have happened, but it did not. To avoid these "holes", this can be represented as a boolean **Measure** per **cell** meaning whether the corresponding fact happened or not (a "null" value in this boolean **Measure** means we do not know if it happened). What must be forbidden is an sparse **Cube** because of "inapplicable" combinations in the cartesian product, since it means we have dependencies between **Dimension**s, which is a bad conceptual design.

On the other hand, a **Cube** needs to be injective in order to allow spaces that do not contain all instances of a **Cell**. This means we will be able to visualize/store only a subset of the *Class*.

In general, different **Cell**s are determined by cartesian products of different **Level**s. However, it could also be that the same set of **Level**s determine two different **Cube**s for different kinds of facts (for example, `Sales` and `Purchases` in our business, both being analyzed by `Month`, `Region`, and `Product`). That is, **Dimension**s can be freely reused for different **Cube**s.

Base changes Steinitz's theorem regarding vectorial spaces states that if $\{e_1, .., e_n\}$ are a base for a space, and $\{v_1, .., v_m\}$ are linearly independent, we can change m elements in the base by v_i, and it still be a base. Since **Cube**s are nothing else that finite spaces, we can also find that two **Cube**s are related by a base change (**Dimension**s change in our case). Both **Cubes** contain the same **cell**s, but just place them in a space defined by different analysis dimensions. Thus, **Dimension**s in one of them must functionally determine the ones in the other.

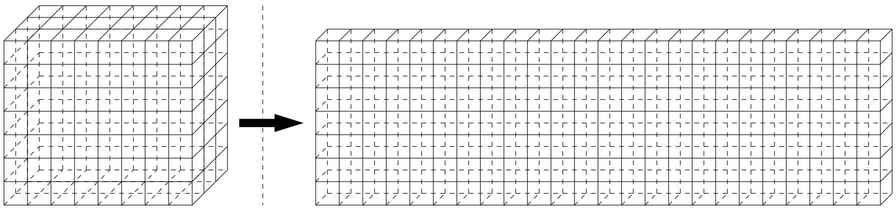


Figure 4.17: Reduction of a 3-dimensional **Cube** to a 2-dimensional **Cube**

If **Level**s $\{L_1, .., L_n\}$ determine a **Cube**, and there exists a set of functionally independent **Level**s $\{L'_1, .., L'_m\}$ so that $L'_1 \times .. \times L'_m \rightarrow L_1 \times .. \times L_n$, we can change the **Level**s that define the space of the **Cube**. If $L'_i \rightarrow L_j \times L_k$, dimensionality can be reduced by replacing L_j and L_k by L'_i, as sketched in figure 4.17. Some authors propose to join two correlated analysis dimensions in order to avoid meaningless combinations. This is not the case. The number of **cell**s is exactly the same. They are just placed in another way. As dimensionality of the **Cube** can be decreased, it can also be increased, if $L'_i \times L'_j \rightarrow L_k$. All these base changes between **Cubes** can be seen as an application of the transitive property of FDs between **Level**s.

As a special case, a surrogate generated by a sequence is always a base for the (1-dimensional) space. However, it can be considered a degenerate case, since it is meaningless for analysts and implies the loss of all benefits in multidimensionality. Nevertheless, as mentioned in previous section, it is important to convert a **Fact** in a **Dimension**. In

[Gio00], the **Dimension** of a 1-dimensional space is called a "shadow dimension", which has a one-to-one relationship with the "fact table". Since there are not two **cell**s associated to the same instance in the **Dimension**, it is not going to be used neither to restrict, nor to group. However, the information could be attached to a given report by way of awareness concept.

Another problem, already discused by some authors, is how **Measure**s can be transformed into analysis dimensions for its own **Fact**. For example, [LAW98] stated that using a **Measure** as **Dimension** means a change in the schema. In this framework, we have just a base change in the space. Whether numerical or descriptive, if a set of attributes fully functionally determines **cell**s in a **Cube**, they can be used as analysis dimensions. Thus, **Measure**s could also be used as analysis dimensions, if they allow to identify **cell**s.

4.3 Conclusions

There is some controversy about whether aggregation hierarchies must be implicit or explicit. This chapter, shows that, at conceptual level, it is essential to explicit aggregation hierarchies, and as much information as possible about analysis dimensions. That information will ease the user to understand data, and pose ad-hoc queries. Users will be able to classify and group data sets in an appropriate manner.

Some problems on explicitly modeling aggregation hierarchies have been identified, and addressed by providing part-whole semantics to relationships between aggregation levels, and considering mereology axioms. Thus, an analysis dimension is defined as a connected, directed graph of aggregation levels, and for each one of the problems, some mereological properties were inferred to solve it. This is the first work deducing properties of analysis dimensions instead of just imposing them. As a result of this study, we can see that "non-onto" and "non-covering" hierarchies as presented in [Ped00] should not be allowed. This is mainly due to considering that all instances of a *Class* must have the same structure. If we find that it is absolutely necessary having different structures for different instances of the same *Class*, we can obtain it by specializing the *Class*.

Not only part-whole, but other kinds of relationships were found interesting for analysis dimensions (i.e. *Generalization*, and *Association*). It was also shown how different **Dimension**s can be related and the consequences that relationships have in aggregation hierarchies.

Detail level	Subject of analysis	Analysis dimensions
Lower	**Measure**s	**Descriptor**s
Intermediate	**Cell**s (representing a *Class* of **cell**s)	**Level**s
Upper	**Fact**s (representing a kind of facts)	**Dimension**s

Table 4.1: Summary table of the different elements in a multidimensional model

Once dimensional data has been analyzed, the second half of the chapter aimed to help on clarifying what multidimensionality means. N-dimensional spaces, and functional dependencies were used to explain what "measures", "cells", "cubes", and "facts" exactly are, which will help on designing as well as querying multidimensional data.

As summarized in table 4.1, we can distinguish three different detail levels. At **Lower** detail level, we have **Measure**s that are the *Attributes* of the **cell**s. Then, we can group **cell**s into different *Class*es that can be drawn as n-dimensional **Cube**s (at **Intermediate** detail level), thanks to that the different analysis dimensions defining a **Cube** are functionally independent. Finally, at **Upper** level, several **Cell**s representing the same kind of fact at different aggregation levels are grouped into a **Fact**. Parallelism between the structure of analysis dimensions and factual data has been outlined.

Chapter 5

YAM^2 (Yet Another Multidimensional Model)

Blooming yam

"Don't hurry, don't worry. You are only here for a short visit. So be sure to stop and smell the flowers."

In the New York Times 1997

Several papers appeared in the last years regarding multidimensional modeling. However, few of them place the discussion at a conceptual level. Moreover, most of them focus on the representation of isolated star schemas, i.e. the representation of only one kind of facts surrounded by its analysis dimensions. In spite of the fact that the dominant trend in data modeling is the "Object-Oriented" (O-O) paradigm, only a couple of proposals on O-O multidimensional modeling exist: [TP98] and [BTW00]. These proposals use "Unified Modeling Language" (UML) standard (defined in [OMG01b]) in some way, but none of them proposes an extension of it to include multidimensionality. Only the "Common Warehouse Metamodel" (CWM) standard (defined in [OMG01a]) extends UML metaclasses to represent some multidimensional concepts. However, it is too general, and not conceived as a conceptual model.

Next section explains the main contributions of this multidimensional model. Then, sections 5.2, 5.3, and 5.4 present its structures, inherent integrity constraints, and operations, respectively. Section 5.5 shows the metaclasses of the model and their relationships with UML metaclasses. Finally, section 5.6 compares **YAM**^{2} with other multidimensional models against several items (most of them already introduced by other authors).

5.1 YAM^2 is not JAM^2 (Just Another Multidimensional Model)

As stated in [AHV95], a "database model" provides the means for specifying particular data structures, for constraining the data sets associated with these structures, and for manipulating the data. It is also explained there that, as Relations are the data structures of the Relational model, so graphs are the structures of O-O models. A precise, easily understandable semantics for graphs in this O-O model is provided by defining **YAM**^{2} structures as an extension of a wide accepted modeling language, i.e. UML (each and every **YAM**^{2} metaclass is a subclass of a UML metaclass). There are some multidimensional models that use UML notation, but no one extends its concepts for multidimensional purposes. By using UML as a base for the definition of structures of **YAM**^{2}, it is built on solid, well accepted foundations, and avoids the definition and exemplification of basic concepts. It makes unnecessary to explain what *Class*es, *Attribute*s, etc. are.

The main goal of multidimensionality is to help non-expert users to query data. Therefore, the data structures of a multidimensional model should show how data can be accessed, driving users in their understanding. They should keep as much information as possible, but the resulting schema must be easily understandable by final users. Thus, the different modeling elements in **YAM**^{2} have been defined at three levels (i.e. **Upper**, **Intermediate**, and **Lower**), so that they are successively decomposed to give the desired detail.

"Expressiveness" or "Semantic Power", as it is defined in [SCG91], is the degree to which a model can express or represent a conception of the real world. It measures the power of the elements of the model to represent conceptual structures, and to be interpreted as such conceptual structures. The most expressive a model is, the better it represents the real world, and the more information about the data gives to the user. As outlined in [FBSV00], due to the presence of multidimensional aggregation, data warehouse - and specially OLAP - applications ask for the vital extension of the expressive power and functionality of traditional conceptual modeling formalisms. Therefore, this is crucial for conceptual multidimensional models like **YAM**^{2}, since they are used to represent user ideas. Different kinds of nodes and arcs in the graphs will be defined to improve the "Expressiveness" of the model. The applicability of the different kinds of relationships supported by UML has been systematically studied.

Another important point for a data model is its "Semantic Relativism". It is defined in [SCG91] as the degree to which the model can accommodate not only one, but many different conceptions. Since different persons perceive and conceive the world in different ways, the semantic relativism of a data model is really important to be able to capture all those conceptions. The information kept in the DW should be shown to users in the form they expect to see it, independently of how it was previously conceived or is actually stored. Therefore, **YAM**^{2} also provides mechanisms (derivation relationships at different detail levels) to model the same data from different points of view.

YAM^2 also pays special attention to show how data can be classified and grouped in a manner appropriate for subsequent summarization. Summarized data can be reflected in the schema, as well as the ways to obtain it. For instance, this information can be used at later design phases to decide materialization.

Therefore, main advantages of **YAM**^{2} are its expressiveness and semantic relativism, besides the flexibility offered in the definition of summarization constraints (it generalizes the work in [LS97]). Moreover, from the separate study of characteristics of analysis dimensions and factual data in section 4.1 and 4.2, it is ensured that it is defined on solid foundations. That study mainly impacts in the definitions in 5.2.1. In section 5.6, **YAM**^{2} is compared with other models to show its advantages and disadvantages. There, its contributions can be clearly seen, regarding specific items.

5.2 Structures

In this section, the structures in the model (i.e. nodes and arcs) are defined.

5.2.1 Nodes

Multidimensional models are based on the duality fact-dimensions. Intuitively, a "fact" represents data subject of analysis, and "dimensions" show different points of view we can use in analysis tasks. The "facts" represent measurements (in a general sense), while "dimensions" represent given information we already have before taking the measurements (on the understanding that they can always be modified). As previous work for the definition of this model, "dimensions" and "facts" were separatedly studied in chapter 4. The reader is referred to it for an specific, deeper explanation of each of both kinds of data. Now, the definition of the different nodes found in a multidimensional O-O schema is given.

Definition 6 *A* **Level** *represents the set of instances of the same granularity in an analysis dimension. It is an specialization of* Class *UML metaclass.*

Definition 7 *A* **Descriptor** *is an attribute of a* **Level**, *used to select its instances. It is an specialization of* Attribute *UML metaclass.*

Definition 8 *A* Dimension *is a connected, directed graph representing a point of view on analyzing data. Every vertex in the graph corresponds to a* **Level**, *and an edge reflects that every instance of target* **Level** *decomposes into a collection of instances of source* **Level** *(i.e. edges reflect part-whole relationships between instances of* **Level***s). It is an specialization of* Classifier *UML metaclass.*

Notice that the acyclicity of **Dimension** hierarchies does not need to be part of their definition. It can be proved from mereology axioms.

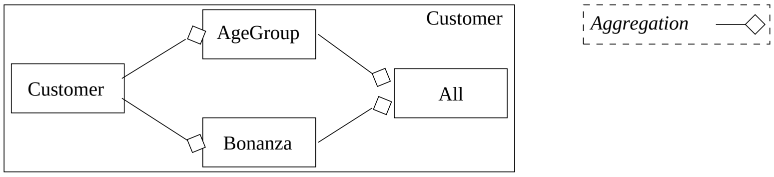


Figure 5.1: Example of **Dimension**

Figure 5.1 shows an example of **Dimension**. It contains four **Level**s: `Customer`, `AgeGroup`, `Bonanza`, and `All`. Every instance of `Customer` **Level** represents a customer, which can be aggregated in two different ways to obtain either age or goodness groups of customers. At top we have `All` level with exactly one instance representing the group of all customers in the **Dimension**. The structure and properties of the graphs of the **Dimension**s were carefully explained in section 4.1. Just to note here that it forms a lattice, and due to the transitive property of part-whole relationships, some arcs are redundant, so that they do not need to be explicited (for instance, `Customer` being aggregated into `All`).

Definition 9 *A* **Cell** *represents the set of instances of a given kind of fact measured at the same granularity for each of its analysis dimensions. It is an specialization of* Class *UML metaclass.*

Definition 10 *A* **Measure** *is an attribute of a* **Cell** *representing measured data to be analyzed. Thus, each instance of* **Cell** *contains a (possibly empty) set of measurements. It is an specialization of* Attribute *UML metaclass.*

Definition 11 *A* **Fact** *is a connected, directed graph representing a subject of analysis. Every vertex in the graph corresponds to a* **Cell**, *and an edge reflects that every instance of target* **Cell** *decomposes into a collection of instances of source* **Cell** *(i.e. edges reflect part-whole relationships between instances of* **Cell***s). It is an specialization of* Classifier *UML metaclass.*

Figure 5.2 shows an example of the structure of a **Fact** with two orthogonal **Dimension**s: `Customer`, already depicted in figure 5.1; and `Clerk`, composed by `Clerk`, `Team`, and `All` **Level**s. We can see that there is a **Cell** in the **Fact** for every combination of **Level**s in the **Dimension**s. Thus, a **Fact** contains all data regarding the same subject at any granularity. Having two independent **Dimension**s with 4 and 3 **Level**s respectively, means that the **Fact** will have 12 different **Cell**s. These **Cell**s and the part-whole relationships between them form a lattice, as was already explained in section 4.2. It is not necessary to represent all those **Cell**s in the schema. **Cell**s just containing derived data are optional, and should only be explicited to emphasize the importance of summarized data at a given aggregation level.

These six kinds of nodes (i.e. **Fact**, **Dimension**, **Cell**, **Level**, **Measure**, and **Descriptor**) are grouped in three pairs. At **Intermediate** level, there are **Cell**s and **Level**s. Looking at **Lower** detail we see **Measure**s and **Descriptor**s. Moreover, at this level, we also define **KindOfMeasure** to show that several **Measure**s in different **Cell**s correspond to the same

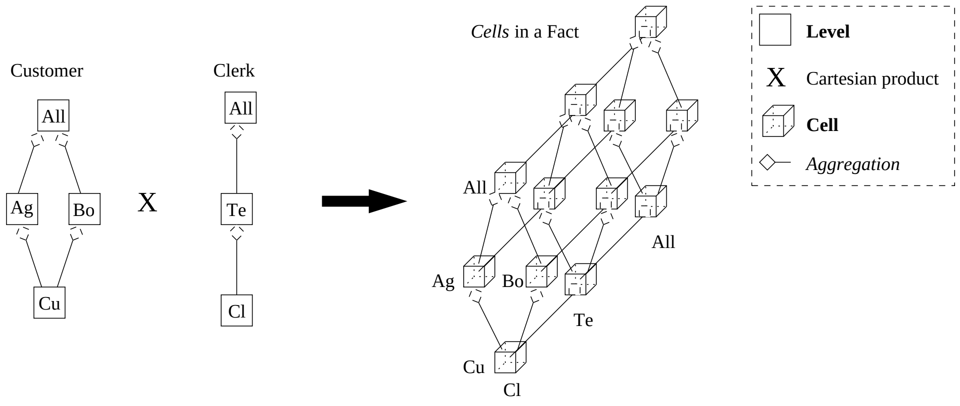


Figure 5.2: Graph of **Cell**s in a **Fact** with two **Dimension**s

measured concept at different aggregation levels. Moreover, at **Upper** detail level, we have **Fact**s and **Dimension**s (one **Fact** and the **Dimension**s associated to it compose a **Star**).

Definition 12 *A* **Star** *is a modeling element composed by one* **Fact***, and several* **Dimension***s that can be used to analyze it. It is an specialization of* Package *UML metaclass.*

5.2.2 Arcs

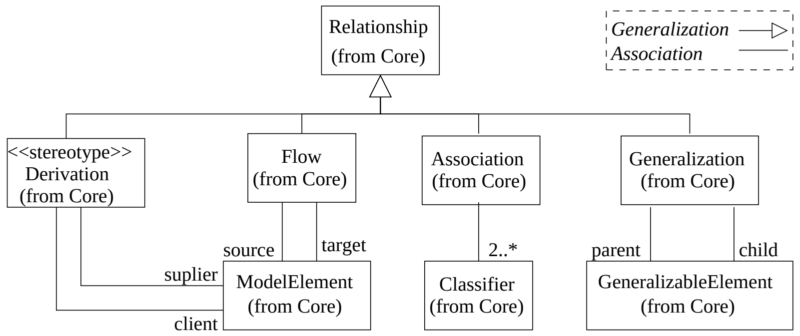


Figure 5.3: UML Relationships between model elements

Once the nodes have been defined, in this section, we are going to see the different kinds of arcs we could find between them. UML provides four different kinds of relationships: *Generalization*, *Flow*, *Association*, and *Dependency*. As depicted in figure 5.3, *Generalization* relationships relate two *GeneralizableElements*, one with a more specific meaning than the other. *Classifiers* and *Associations* are *GeneralizableElements*. *Flow* relationships relate two elements in the model, so that both represent different versions of the same thing. *Association*, as defined in UML specification, defines a semantic relationship between *Classifiers*. By means of a stereotype of *AssociationEnd*, UML allows to use a stronger type of *Association* (i.e. *Aggregation*), where one classifier represents parts of the other (i.e. it shows part-whole relationships). Finally, UML allows to represent different kinds of *Dependency* relationships between *ModelElements* like *Binding*, *Usage*, *Permission*, or *Abstraction*. We are not going to consider the three first, because they are rather used on application modeling, and **YAM**^{2} is just a data model. Moreover, due to the same reason, out of the different stereotypes of *Abstraction* we are only going to use *Derivation*. Derivability, also known as "Point of View", helps to represent the relationships between model elements in different conceptions of the UoD.

The usability of these relationships between concepts was briefly explained and exemplified in section 3.2. Here we are systematically going to see how they can be used to relate multidimensional constructs at every detail level. For every pair of constructs at each detail level it will be shown whether they can be related by a given kind of *Relationship* or not. Moreover, if two constructs can be related, it will also be shown whether they must belong to the same construct at the level above, or not (i.e. inter or intra relationships, respectively).

Upper detail level

	Fact-Fact	**Fact-Dimension**	**Dimension-Fact**	**Dimension-Dimension**
Generalization	Inter	-	-	Intra/Inter
Association	Intra/Inter	Intra/Inter	Intra/Inter	Intra/Inter
Aggregation	Intra/Inter	-	-	Intra/Inter
Flow	Inter	-	-	Intra/Inter
Derivation	Inter	Intra/Inter	-	Intra/Inter

Table 5.1: Relationships between elements at **Upper** detail level

Table 5.1 shows the different relationships we can find at this detail level. Since a **Star** only contains one **Fact**, in order to have two related **Fact**s, they must belong to different **Star**s. Therefore, relationships between **Fact**s will always be inter-stellar, but for reflexive *Associations* and *Aggregations*. However, we can have inter-stellar as well as intra-stellar relationships between two **Dimension**s, because a **Star** contains several **Dimension**s, which can be related.

Figure 5.4 shows examples of most relationships at this level. Firstly, corresponding to the upper-left corner of the table, we see that two **Fact**s can be related by *Generalization* (i.e., `ProductSale` and `CreditSale`). We will have different information for the more specific **Fact** (for example, number of credit card). Thus, analysis dimensions are inherited from the more general **Fact**, but others could be added, like `Bank`. `ProductSale` and `Production` are related by *Association* to show the correspondences between produced and sold items. We can also find *Aggregation* relationships between **Fact**s. A **Fact** in a **Star** can be composed by **Fact**s in another **Star**. For instance, a `Deal` is composed by several individual `ProductSale`. Notice that it is not always possible to calculate all measurements of `Deal` from those of `ProductSale` (for instance, discount in the deal). Data sources, measure instruments, or calculation algorithms

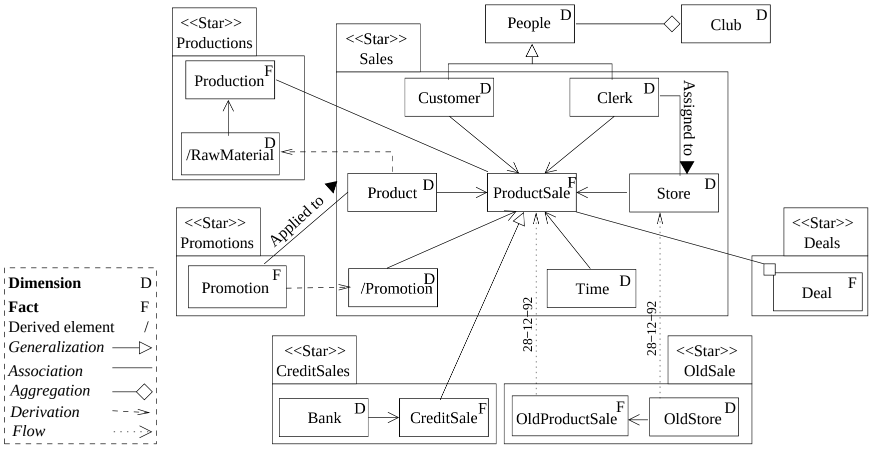


Figure 5.4: Example of **YAM**^{2} schema at **Upper** detail level

are probably going to change, and these changes should be reflected in our model by means of *Flow* relationships between **Fact**s. All these changes are not reflected by just relating our **Fact**s to `Time` **Dimension**, since we actually have different **Cell**s. On `December 28th of 1992`, we started recording discount checks in `ProductSale`, so that we kept both incomes (i.e. cash, and discount checks). From that day on, we have different **Fact**s containing the same kind of data before and after the acceptance of the checks (i.e. `OldProductSale`, and `ProductSale`). Finally, two **Fact**s could also be related by *Derivation* relationships to show that they are the same concept from different points of view.

In the upper-right corner of table 5.1, we can see that there exist *Generalization* relationships between **Dimension**s. For instance, `People` **Dimension** generalizes `Clerk` and `Customer` ones. Notice that if we suppose that all people are customers, both related **Dimension**s would belong to the same **Star**. It is also possible to have analysis dimensions related by *Association*. Thus, `Clerk` is associated with `Store` **Dimension** to show that clerks are assigned to stores. We can also find stronger associations between analysis dimensions, if we join more than one to give rise to another. For example, `People` **Dimension** is used to define `Club` by means of an *Aggregation* relationship. Every instance of `Club` is composed by a set of people. Several years ago, when our local business grew, `Store` **Dimension** was changed to reflect the new **Level** `Region`. At conceptual level, those changes are represented by a *Flow* relationship between `OldStore` and `Store`. *Derivation*s allow to state that there are different views of the same **Dimension**. We could find that the same concept has different names depending on the subject we are. Thus, a **Dimension** could be used in different **Star**s. For example, `Product` is considered `RawMaterial` in a different context. Therefore, the same **Dimension**, with exactly the same instances, needs a different name depending on the context. These **Dimension**s could even have different aggregation hierarchies or attributes of interest to the users. For example, studying the raw material grouped by profit margin can be meaningless.

The middle columns in table 5.1 show how a **Fact** can be related to a **Dimension** and vice versa. Firstly, we see that a **Fact** is related to its analysis dimensions by means of *Association* relationships. Moreover, they can also be associated to **Fact**s in another **Star** as shown in the example, where `Promotion` **Fact** is associated to `Product` **Dimension** in the `Sales` **Star**. A **Dimension** can be obtained by deriving it from a **Fact**. The name can be changed, some aggregation levels added or removed, others modified, some instances selected, etc. in order to adapt it to its new usage. In our example, some people is interested in the analysis of promotions. Thus, the promotions selected by studying `Promotion` **Fact**, can be used as **Dimension** to study `ProductSale`. Notice the difference between deriving a **Dimension** and associating it to a **Fact** in another **Star**. The former allows to study the sales performed during a promotion, while the latter shows all promotions that have been applied to a kind of product. That *Derivation* between a **Fact** and a **Dimension** uses to be an inter-stellar relationship (i.e. from a **Fact**, we derive a **Dimension** to analyze another **Fact**). However, we could also use information derived from a **Fact** to analyze the same **Fact**. It is also important to say that a **Fact** cannot be derived from a **Dimension**, because **Fact**s represent measurements, so that they cannot be found a priori in the form of **Dimension**. The rest of relationships (i.e. *Generalization*, *Aggregation*, and *Flow*) cannot be found between a **Fact** and a **Dimension**, nor vice versa. All three imply obtaining a new element based on a preexisting one, and the difference between **Fact** and **Dimension** is so important that the obtaining of one from the other should be restricted to derivation mechanisms. For instance, a **Fact** cannot eventually become a **Dimension**.

Intermediate detail level

	Cell-Cell	**Cell-Level**	**Level-Cell**	**Level-Level**
Generalization	Inter	-	-	Inter
Association	Intra/Inter	Inter	Inter	Intra/Inter
Aggregation	Intra/Inter	-	-	Intra/Inter
Flow	Inter	-	-	Inter
Derivation	Inter	Inter	-	Inter

Table 5.2: Relationships between elements at **Intermediate** detail level

Table 5.2 shows the relationships we can find at this level. Most of them are exemplified in figure 5.5. Our company (resulting from the fusion of preexisting smaller companies) is organized in autonomous regions. Thus, the information systems in one of these regions collect data that those in other regions do not, so we specialize our **Cell**s (i.e. `AtomicSale`) depending on the region. This specialization is due to the specialization of the kind of fact they are representing. Therefore, we can see in the upper-left corner of the table that two **Cell**s can be related by *Generalization*, but they must belong to different **Fact**s (i.e. it is an inter-factual relationship). **Cell**s in different **Fact**s can be associated (for instance, each **Cell** representing a sale with its corresponding **Cell** representing the production of what was sold). Moreover, we can also have *Association* relationships between **Cell**s in the same **Fact** (for instance, computers are associated to those other products that are plugged to them). In general, we only have intra-factual *Aggregation* relationships, which correspond to those relationships between **Level**s, and are not necessary in the schema. However, we could also find that different **Cell**s are aggregated to obtain a **Cell** about a different kind of fact (when both **Fact**s are also related like `ProductSale` and `Deal`). In this case, we do not group **Cell**s along any analysis dimension, i.e. it does not generate coarser **Cell**s in the same **Fact**, but **Cell**s in another **Fact** (i.e. `AtomicDeal`). If a new **Measure** would appear for a kind of fact, we would obtain a new **Cell** related to the old one by means of a *Flow*. Both would represent the same concept. However, they would belong to different versions of the same **Fact** (it is an inter-factual relationship). *Derivation* relationships can be used to hide information, change names, or **Measure**s in the **Cell**s, giving rise to new **Fact**s.

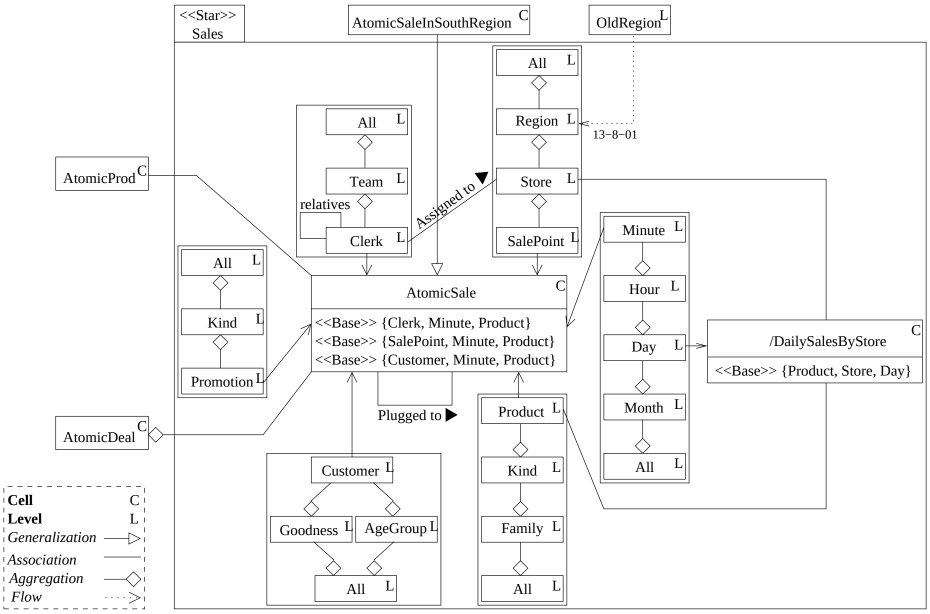


Figure 5.5: Example of **YAM**^{2} schema at **Intermediate** detail level

The rightmost column shows that we could also find *Generalization* relationships between two **Level**s. As in the case of **Cell**s, it must be an inter-dimensional relationship, because both **Level**s cannot be related, at the same time, by *Generalization* and part-whole relationships. *Associations* between **Level**s can be intra- as well as inter-dimensional. The **Level** representing clerks is associated with other clerks (his/her relatives) in the same **Dimension**, and with stores in another **Dimension**. Intra-dimensional *Aggregation*s define the graph of the **Dimension**. However, we could also find inter-dimensional *Aggregation*s between **Level**s, if two **Dimension**s are so related. When the company was restructured and the regional division changed, the aggregation level showing it also changed. Both, new and old **Level**s are related by means of a *Flow* (although they represent the same concept, they belong to different versions of the same

Dimension). Finally, as for any other concept, a **Level** could be derived from another one to show it from a different point of view.

All relationships in the central columns must be inter-structure, because **Cell**s and **Level**s always belong to different structures (i.e. **Fact**s and **Dimension**s, respectively). As for relationships at **upper** detail level, a **Cell** cannot be converted into a **Level** nor vice versa by means of *Generalization*, *Aggregation*, or *Flow*. It must always be done using derivation mechanisms. Moreover, because of the same reason that a **Fact** cannot be derived from a **Dimension**, a **Cell** cannot be derived from a **Level**. Nevertheless, if a **Dimension** is derived from a **Fact**, its **Level**s are also derived from the **Cell**s of the **Fact**. *Association*s exist between **Cell**s and **Level**s, or vice versa (showing the granularity of the **Cell**s).

Lower detail level

	Measure-Meas.	**Measure-Descr.**	**Descriptor-Meas.**	**Descriptor-Descr.**
Flow	Inter	-	-	Inter
Derivation	Intra/Inter	Inter	Inter	Intra/Inter

Table 5.3: Relationships between elements at **Lower** detail level

Elements at this level are neither *Classifier*s nor *GeneralizableElement*s, but just *Attribute*s. Therefore, as it is shown in table 5.3, they can only be related by those relationships between *ModelElement* (i.e. *Derivation*, and *Flow*).

If a change affects a **Measure** or **Descriptor**, they will belong to new versions of their **Cell** and **Level**, respectively. Thus, *Flow* relationships are in both cases inter-structure. Moreover, simply evolution cannot convert a **Measure** into a **Descriptor**, nor vice versa.

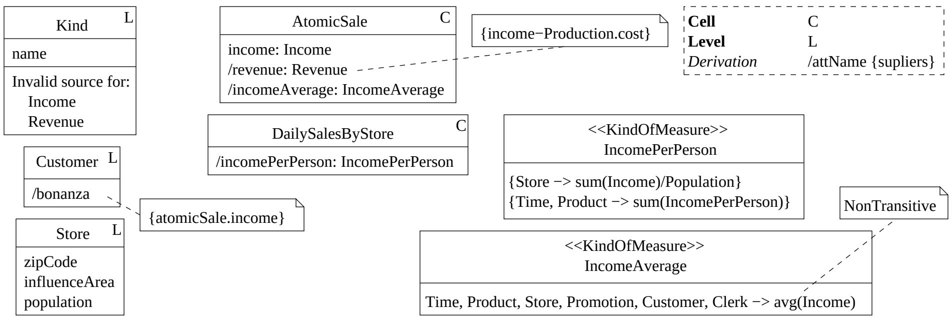


Figure 5.6: Example of **YAM**^{2} schema at **Lower** detail level

It is always possible to define derived **Measure**s from other **Measure**s in the same **Cell**, as well as **Descriptor**s from other **Descriptor**s in the same **Level**. Moreover, in both cases, supplier *Attribute*s could also be in other *Class*es. **Measure**s in a **Cell** could be obtained by applying some operation to **Measure**s in other **Cell**s. For instance, looking to **Lower** detail level elements in figure 5.6, we see that measurements of `revenue` in `AtomicSale` are obtained from subtracting `cost` in `Production`. What is more, a **Descriptor** can be obtained from some **Measure**s (for example, the goodness of a customer from the income of his/her purchases), or vice versa (for example, impact of sales obtained by dividing incomes by the population of the influence area of the store). This figure does not show arcs between **Cell**s and **Level**s, because they are at **Intermediate** detail level.

5.3 Inherent integrity constraints

The metaclasses of the model define constraints on multidimensional schemas, but constraints should also be defined on their instances. In this section, that kind of constraints are going to be addressed, paying special attention to two important aspects in multidimensional modeling, namely placement of data in an n-dimensional space, and summarizability of data.

The main contribution of multidimensionality is the placement of data in an n-dimensional space. This improves the understanding of those data and allows the implementation of specific storage techniques. It is important that the n dimensions of the space (i.e. **Cube**) are orthogonal. If not, i.e. if a **Dimension** determines others, the visualization of data will be unnecessarily complicated (we are showing more information than it is needed and it will be more difficult for users to understand it); moreover, storage mechanisms are affected, as well, because they are not considering that several combinations of dimension values are impossible, maybe resulting in a waste of space. This does not mean that all **Dimension**s in a **Star** must be orthogonal. Nevertheless, those defining **Cube**s (which are used for visualization as well as storage purposes) should be, or at least the user should know whether they are.

A **Cell** instance is related to one object or set of objects (if it is an *Association* with upper-bound multiplicity greater than one) at each associated analysis dimension, and those objects or sets of objects completely identify it. Thus, regarding placement of data in n-dimensional spaces, we could say that the set of **Level**s a **Cell** is associated with form a "superkey" (in Relational terms) of that **Cell**. In **YAM**^{2} , every minimal set of **Level**s being "superkey" (i.e. "key" in the Relational model) of a **Cell** is called a **Base**. When one of these **Base**s (that define spaces of orthogonal **Dimension**s) is associated to a **Cell**, we obtain a **Cube**. For instance, `AtomicSale` (in figure 5.5) can be associated with points in the 3-dimensional space defined by **Level**s `Clerk`, `Minute`, and `Product`, so that `AtomicSale` is fully functionally determined by those three **Level**s (a **Base** of the space).

Definition 13 *A* **Cube** *is an injective function from an n-dimensional finite space (defined by the cartesian product of n functionally independent* **Level***s $\{L_1, .., L_n\}$), to the set of instances of a* **Cell** *(C_c).*

$$c : L_1 \times .. \times L_n \to C_c, \ \textit{injective}$$

If the **Level**s were not functionally independent (i.e. they did not form a **Base**), we would use more **Dimension**s than strictly needed to represent the data, and would generate empty meaningless zones in the space.

Another interesting group of constraints to deal with is that related to summarization anomalies and how to solve (or prevent) them. In multidimensional modeling, it is essential to know how a given kind of measure must be aggregated to obtain it at a coarser granularity. [LS97] identifies three necessary (intuitively also sufficient) conditions for summarizability:

1. Disjointness: the subsets of objects to be aggregated must be disjoint.
2. Completeness: the union of subsets must constitute the entire set.
3. Compatibility: category attribute (i.e. **Level**), summary attribute (i.e. **KindOfMeasure**), and statistical function (i.e. **Summarization**) must be compatible.

The first two conditions are absolutely dependent on constraints over cardinalities in the part-whole relationships in the **Dimension**s, because these define the grouping categories. Therefore, let us briefly talk also about this third group of integrity constraints of the model.

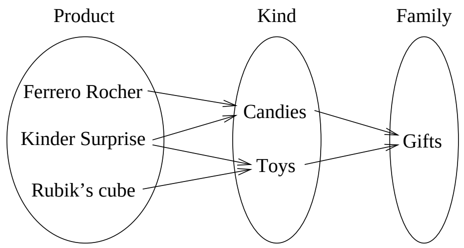


Figure 5.7: Example of sharing of parts between several instances

To avoid those anomalies on summarizing data, some models forbid "to-many" relationships in the aggregation hierarchies. This means that instances of a **Part Level** can only belong to one **Whole**. Nevertheless, there is no mereological axiom forbidding the sharing of parts among several wholes. As exemplified in figure 5.7, a given product `Kinder Surprise` (at **Level** `Product`) belongs to two different kinds of products at the same **Level** `Kind` (i.e. `Candies`, and `Toys`). We argue that this case should not be ignored by a multidimensional model. Therefore, non-strict hierarchies are allowed in the **Dimension**s, and they need to be taken into account to decide summarizability of **Measure**s.

The other problem on cardinalities is that of "non-onto" and "non-covering" hierarchies (as presented is [Ped00]). That is, having different part-whole structures for instances at the same **Level** is allowed. For example, if we would have a state-city (like `Monaco` in a `Geographic` linear **Dimension** with **Level**s `City`, `State`, and `All`), we could generate both situations. If we consider that `Monaco` is a city, we have a "non-covering" hierarchy (we are skipping `State` **Level**). On the other hand, if it is considered a state, we obtain a "non-onto" hierarchy (we have different path lengths from the root to the leaves depending on the instances). In this case, **YAM**^{2} proposes the usage of what some authors call "Dummy Values" to guarantee the existence of at least one part for every whole in the hierarchy. These values are not dummy at all. `Monaco` being a state-city does not mean it is either a state or a city, but a state and a city at the same time. Thus, both instances will represent city and state facets of the same entity. Therefore, in **YAM^2** , cardinalities in aggregation hierarchies are "1..*" parts for every whole, and "*" wholes for every part, on the understanding that **Dimension** instances can always be defined so that there are "1..*" wholes for every part. The interested reader can refer to section 4.1 for a deeper explanation of these cardinalities.

Going back to the group of constraints regarding summarizability, in the model, there are three different elements to deal with that problem (all exemplified in figure 5.6). These elements allow to represent summarizability conditions in a more flexible way than just distinguishing "additive", "semi-additive", and "non-additive" **Measure**s. Firstly, we have that some **Level**s are an **InvalidSource** for the calculation of a given **KindOfMeasure** (for example, `Kind` is an invalid source for `Income` and `Revenue`). This means that measurements at an aggregation level cannot be used to obtain data at higher aggregation levels, and we must go to finner granularities (maybe to the **Atomic Level**) to obtain the source data for the calculation. This can be due to the fact that the instances of that **Level** are not disjoint or not complete (i.e. summarizability conditions 1 and 2 mentioned above). A **Level** being invalid or not cannot be deduced just from the cardinalities of its associations, but also depends on the **KindOfMeasure**. For instance, if a **Measure** is obtained as the minimum of a set of measurements, it does not matter whether the source sets of instances are disjoint or not. For example, in some cases, double counting could even be desirable.

Moreover, **Induce** *Association* shows the summarization that must be performed on aggregating a given **KindOfMeasure** along a **Dimension**. This constraint regards the third condition mentioned above. Along a given analysis dimension we can use a summarization operation, while along a different analysis dimension we use a different function. For instance, we aggregate `IncomePerPerson` along `Time` and `Product` by means of sum, while along `Store` it needs to be recalculated from `Incomes`. Incompatibilities are not always associated to `Time` **Dimension**. Furthermore, instances of **Induction** could be partially ordered, if necessary, to show that operations are not commutative, and must be performed in a given order, as pointed out in [Tho97]. For example, sums along a **Dimension** must be performed before averages along another one, so that, we aggregate up to the desired **Level** in a **Dimension**, and then we aggregate along the other.

Finally, another point to take into account, usually overlooked in other models, is that of transitivity. If a summarization operation is not transitive, we cannot use precalculated aggregates at a given **Level** to obtain those at higher levels. Going to the atomic source is mandatory (for instance, we should not perform the average of averages, if we want to obtain the average of raw data).

5.4 Operations

The multidimensional model is just a query model, i.e. it does not need operations for update, since this is not directly performed by final users. **YAM^2** operations focus on identifying and uniformly manipulating sets of data, namely **Cube**s. In a **Cube**, data are identified by their properties. Thus, these operations are separated from the physical storage of the data.

Moreover, they are not presentation oriented like those in [Tes01].

Detail level	Subject of analysis	Point of view
Upper	Drill-across	ChangeBase
Intermediate	-	Roll-up
Lower	Projection	Dice

Table 5.4: **YAM**^{2} operations

As everything in a multidimensional model, operations are also marked by the duality fact-dimensions. Table 5.4 shows the operations in two columns. The first one contains those operations having effect on the subject of analysis (i.e. **Fact**, **Cell**, and **Measure**). They select the part of the schema we want to see. In the other column, there are those operations affecting the point of view we will use in the analysis (i.e. **Dimension**, **Level**, and **Descriptor**). They allow to reorganize the data, modify their granularity, and focus on a specific subset, by selecting the instances we want to see.

$$
\begin{tikzcd}
L^i_1 \times .. \times L^i_k \times .. \times L^i_m \arrow[r, "c_i"] \arrow[d, "h"] & C^i_c \arrow[d, bend left, "g"] \\
L^o_1 \times .. \times L^o_j \times .. \times L^o_n \arrow[r, "c_o"] \arrow[u, bend left, "f"] & C^o_c
\end{tikzcd}
$$

Figure 5.8: Multidimensional operations as composition of functions

In the sense of [AHV95], these operations are conceptually a "procedural language", because queries are specified by a sequence of operations that construct the answer. We generally say that a query is from (or over) its input schema to its output schema. Thus, there exists an input m-dimensional **Cube** (c_i), and we want to obtain an output n-dimensional **Cube** (c_o). Since, we defined a **Cube** as a function (see definition 13), operations must transform a function into another function. Operations in the first column work on the image of the function (i.e. **Cell**), while operations in the second column change its domain (i.e. **Base**). As depicted in figure 5.8, we have three families of functions (i.e. f, g, and h), that can be used to transform a **Cube**.

Obtaining c_o from c_i, can be seen as mathematical composition of functions ($c_o = \psi \circ c_i \circ \phi$, with ψ and ϕ belonging to the families of functions g and f, respectively). Firstly, we can see how **ChangeBase**, given c_i and ϕ (a function belonging to a family of functions f between the finite spaces defined by cartesian product of **Level**s of each **Cube**), we obtain a new **Cube** ($c_o = c_i \circ \phi$). Nevertheless, **Drill-across** does change the **Cell**. Thus, it works in the opposite way, in the sense that it needs a **Cube** c_i and the function ψ (belonging to a family of functions g from a **Cell** to another **Cell**) to obtain the new **Cube** ($c_o = \psi \circ c_i$).

Unfortunately, it is not possible to define all operations in such a way. **Roll-up** changes the space as well as the **Cell**. Thus, obtaining it as a composition of functions is not possible, because a coordinate in the space of c_o corresponds to several points in c_i. Therefore, there is no ζ, so that c_o is a composition of ζ and c_i. It can neither be defined as an homomorphism (like those in [FM95]), because the problem is not the conversion of a set of instances into one instance (which is always performed by union), but deciding which is the set of instances to be converted (defined by a function of family h).

Drill-across: This operation changes the image set of the **Cube** by means of an injective function ψ of the family g (relationships in section 5.2.2 can be used for this purpose). The space remains exactly the same, only the **cell**s placed in it change. This function relates instances of a **Fact** to instances of another one.

$$\psi : C_c^i \rightarrow C_c^o, \ injective$$

$$c_o(x) = \delta_\psi(c_i) = \psi(c_i(x))$$

Projection: This just selects a subset of **Measure**s from those available in the selected **Cell**. Since it works at the attribute level, it is absolutely equivalent to the homonym operation in Relational algebra.

$$c_o(x) = \pi_{m_1,..,m_k}(c_i) = c_i(x)[m_1, .., m_k]$$

ChangeBase: This operations reallocates exactly the same **Cell** in a new space. It changes the domain set of the **Cube** by means of an injective function ϕ of the family f (i.e. ϕ relates points in an n-dimensional finite space to points in an m-dimensional finite space). Thus, it actually modifies the analysis dimensions used.

$$\phi : L_1^o \times .. \times L_n^o \rightarrow L_1^i \times .. \times L_m^i, \ injective$$

$$c_o(x) = \gamma_\phi(c_i) = c_i(\phi(x))$$

Roll-up: It groups **cell**s in the **Cube** based on an aggregation hierarchy. This operation modifies the granularity of data, by means of an exhaustive function φ of the family h (i.e. φ relates instances of two **Level**s in the same **Dimension**, corresponding to a part-whole relationship). It reduces the number of **cell**s, but not the number of **Dimension**s.

$$c_o(x) = \rho_\varphi(c_i) = \bigcup_{\varphi(y)=x} c_i(y)$$

Dice: By means of a predicate P over **Descriptor**s, this operation allows to choose the subset of points of interest out of the whole n-dimensional space. Like **Projection**, it is absolutely equivalent to an operation of Relational algebra. In this case, the operation is "Selection".

$$c_o(x) = \sigma_P(c_i) = \begin{cases} c_i(x) & if \ P(x) \\ undef & if \ \neg P(x) \end{cases}$$

With this set of operations, we can derive **Slice**, which reduces the dimensionality of the original **Cube** by fixing a point in a **Dimension**. This is obtained by means of **Dice** and **ChangeBase** operations.

$$c_o(x) = slice_{L_i=k}(c_i) = \delta_{L_1\times..\times L_{i-1}\times L_{i+1}\times..\times L_n}(\sigma_{L_i=k}(c_i))$$

Looking to the empty cell in table 5.4, it is clear that there is another operation missing, which would allow to select the **Cell** we want to query in the same way we choose **Measure**s or **Fact**s. However, the specific **Cell** we analyze cannot be selected by itself, but it is absolutely determined by the selected aggregation levels in every **Dimension**. Moreover, **Drill-down** (i.e. the inverse of **Roll-up**) is neither defined, because as argued in [HS97], we can only apply it, if we previously performed a **Roll-up** and did not lose the correspondences between **cell**s. This can be expressed as an "undo" of **Roll-up**, or if we do not want to keep track of results, by means of views over the atomic data as in [Vas98]. **Drill-down** would be really useful to **Dice** at a higher level of aggregation than the result data.

If we want to know the production cost of every product sold under a given promotion, by month and plant, we should perform the following operations over our `AtomicSale` schema: 1) **Dice** to select promotion "A", 2) **Drill-across** to "Production" **Fact**, 3) **Projection** to see just the desired **Measure** "cost", 4) **Roll-up** to obtain data at "Month" **Level** (notice that summarization operation is not explicited, because a **YAM**^{2} schema shows how a given **KindOfMeasure** must be summarized along each **Dimension**), and finally 5) **ChangeBase** to choose the appropriate n-dimensional space to place data.

$$\gamma_{Month\times Plant\times Product}(\rho_{Month}(\pi_{cost}(\delta_{Production}(\sigma_{Promotion="A"}(AtomicSale)))))$$

Property 11 *The* **cube** *algebra composed by these operations is closed (i.e. they operate on* **Cube***s and the result of all operations is always a* **Cube***).*

Proof 11 *Being closed seems clear, for* **ChangeBase** *and* **Drill-across** *(since composition of functions is always a function, and all functions in these operations are injective) as well as for* **Projection** *and* **Selection** *(since the former only removes attributes from the image, and the latter removes points from the domain of the function). Therefore, all this operations result in an injective function from a cartesian product of* **Level***s to a* **Cell***. In the case of* **Roll-up***, φ being exhaustive implies that multidimensional operation defines a function over a* **Cell** *(if there is at least one y for every x so that $\phi(y) = x$, then c_o will be defined for every x). Moreover, φ being a function means the result is injective (if $\phi(y)$ has only one image, then c_i will only belong to one $c_o(x)$ resulting in different images for every x).*

Property 12 *The* **cube** *algebra composed by these operations is complete (i.e. any valid* **Cube** *can be computed as the combination of a finite set of operations).*

Proof 12 *Being complete is also true since if there is an FD between two* **Cube***s in the closure of FDs, there is a sequence of operations that allows to obtain one from the other. We can change the left hand side of the function defining a* **Cube** *(i.e. the cartesian product of* **Level***s)* *in two ways: the domain (by means of* **ChangeBase***) and its elements (by means of* **Dice***). As can be seen in definition 9, the right hand side of that function, a* **Cell***, is defined by two characteristics: a subject (that can be changed by* **Drill-across***), and an* **aggregation level** *(that can be changed by* **Roll-up***).* Attribute*s inside the* Class *can be selected by means of* **Projection***.*

Property 13 *The* **cube** *algebra composed by these operations is minimal (i.e. none can be expressed in terms of others, nor can any be dropped without affecting their functionality) and the operations are atomic (i.e. each operation performs exactly one task).*

Proof 13 *This can be easily inferred from the explanation of each operation above, and table 5.4. Each operation works inside only one detail level. Moreover, they work either on factual or dimensional data.* **Roll-up** *could be thought as working on both sides, however, it really operates only on factual data based on dimensional data.*

We could also compare this set of operations with those three in [Vas00a] (i.e. "Navigate", "Selection", and "Split measure"). "Selection" and "Split measure" are absolutely equivalent to **Dice** and **Projection** respectively. Regarding "Navigate", it could be obtained by means of **Roll-up** and its corresponding "undo" ("Navigate" always operates on a "base cube", so that atomic data can be used to **Drill-down**). **Drill-across** and **ChangeBase** have no counterpart. They work on semantic relationships between different **Star**s and functional dependencies between **Level**s in different **Dimension**s associated to a **Fact**, and were not treated as first class citizens in any other multidimensional model before.

These operations allow to build **Cube**s on solid mathematical foundations. Semantic relationships in the multidimensional schema define functions between *Class*es. By composing those functions appropriately, we can obtain the desired vision of data. If we want to analyze instances of a given *Class* in the space defined by the cartesian product of a set of *Class*es, all we have to do is find the appropriate composition of functions. If that "chain" of functions exists, we can analyze data in the desired way.

Thus, properties of mathematical functions can be applied. For instance:

- Similar to operations between functions ($f\ op\ g = f(x)\ op\ g(x)$), we can also define operations between **Cube**s, if both are defined over the same domain (n-dimensional space).

  $$c_1\ op\ c_2 = c_1(x)\ op\ c_2(x)$$

  If the operation is defined over the image of the **Cube**s, it is defined over **Cube**s. Thus, **Union**, and **Intersection** of **Cube**s can be easily defined, as it is defined for **cell**s.

- Two functions over different domains can define a function over the union of the domains. This means that **Cube**s defined over subclasses can give rise to a broader **Cube** over a superclass. For instance, if predicate P defines an specialization,

  $$c_o(x) = \begin{cases} c_1(x) & if\ P(x) = 1 \\ .. & \\ c_n(x) & if\ P(x) = n \end{cases}$$

This invites to parallelize the calculation of the cube c_o.

$$f\ op\ c_o(x) = \begin{cases} f\ op\ c_1(x) & if\ P(x) = 1 \\ .. & \\ f\ op\ c_n(x) & if\ P(x) = n \end{cases}$$

- If the result of operations is not a function, it is not a **Cube**. Therefore, by validating whether a sequence of relationships is a function, we can validate the existence of **Cube**s.

5.5 Metaclasses

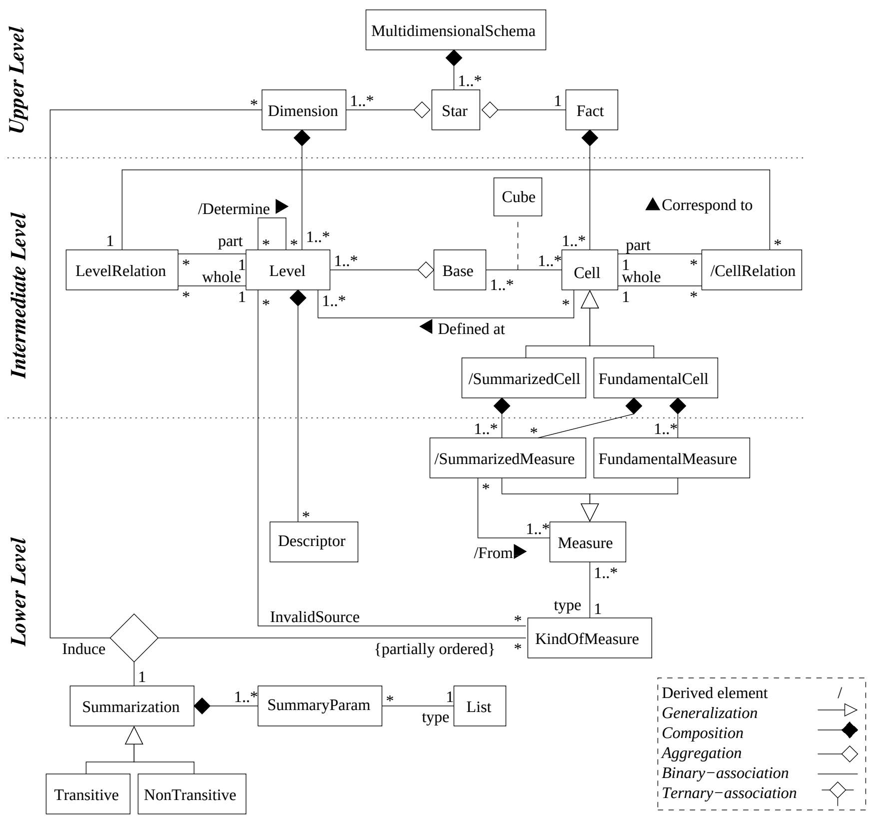


Figure 5.9: **YAM**^{2} metaclasses in UML notation (as in [OMG01b])

As it was defined in [Inm96], a DW is a subject-oriented set of data. When analysts want to study a given subject, they want to see together all data regarding it. Thus, a subject-oriented model, where all *Class*es related to a subject are shown together in the multidimensional schema is proposed. For this purpose **YAM**^{2} uses the **Upper** detail level which, as depicted in figure

5.9, shows that a **Star** is composed by one **Fact** and several **Dimension**s. Subject-oriented does not imply subject-isolated. Therefore, relationships between different **Star**s will exist, as it was shown in section 5.2.2.

At the **Intermediate** detail level, we can see that **Dimension**s are composed by **Level**s related by **LevelRelation**s, representing part-whole relationships. Hence, a **Dimension** is a lattice stating how measured data can be aggregated. On the other hand, we see that a **Fact** is composed by a set of **Cell**s. Each of those **Cell**s is defined at an aggregation level for each of the analysis dimensions of its **Fact**. If there is a **Level** (l_2) whose elements are obtained by grouping those of another **Level** (l_1) at which a **Cell** (c_1) is defined, then we have another **Cell** (c_2) related to l_2 whose instances are composed by those of c_1. **Cell**s c_1 and c_2 are related by a **CellRelation**, which corresponds to the **LevelRelation** between l_1 and l_2. A set of functionally independent **Level**s form a **Base**, and the pair **Base**-**Cell** (where the **Base** fully determines instances of the **Cell**) is a **Cube**.

Some data must be physically stored while other will or could be derived. In the same way, some model elements must be explicited in the schema, while other (for instance, **CellRelation**) can be derived. In this sense, those **Cell**s that need to be explicited (i.e. **FundamentalCell**s) are distinguished from those that do not (i.e. **SummarizedCell**s), because all data they contain can be derived.

At **Lower** detail level, we can see information regarding the attributes of the concepts we are representing. The **Level**s contain **Descriptor**s, and the **Cell**s contain **Measure**s. **SummarizedCell**s only contain data that can be derived (i.e. **SummarizedMeasure**s). They are shown in the schema to outline the importance of the **Cell** (they are first class candidates to be precalculated). On the other hand, **FundamentalCell**s can contain derived and not derived data (they must be physically stored). **SummarizedMeasure**s are obtained from other **Measure**s, while **FundamentalMeasure**s are not. Notice that it is possible to obtain one **Measure** from more than one supplier (for instance, to be able to weigh an average).

Every **Dimension** induces a **Summarization** over a given **KindOfMeasure**. In general, **SummarizedMeasure**s are obtained by sum of other. However, this is not always the case, product, minimum, maximum, average, or any other operation could be used. It depends on the **KindOfMeasure** and the **Dimension** along which we are summarizing ([LS97] studies the influence of the temporal dimension on three different kinds of attributes). Thus, when we want to obtain a **SummarizedMeasure** in a **Cell** (c_1), from a **Measure** in another **Cell** (c_2), the **Summarization** performed is that induced by the **Dimension** that contains the **LevelRelation** to which the **CellRelation** between c_1 and c_2 corresponds.

Summarizations over a **KindOfMeasure** are partially ordered to state that some must be performed before others. Moreover, some data at an aggregation level could be an invalid source to summarize some **KindOfMeasure**s, which is also captured in a **YAM**^{2} schema. A summarization operation being non-transitive, implies that any summarization that uses it must be done from the atomic data.

Figure 5.10 shows how all these multidimensional concepts perfectly fit into UML. A **Star** is a *Package* that contains a subject of analysis. **Fact**s and **Dimension**s are *Classifier*s containing *Class*es (i.e. **Cell**s, and **Level**s respectively). Finally, **Measure** and **Descriptor** are just

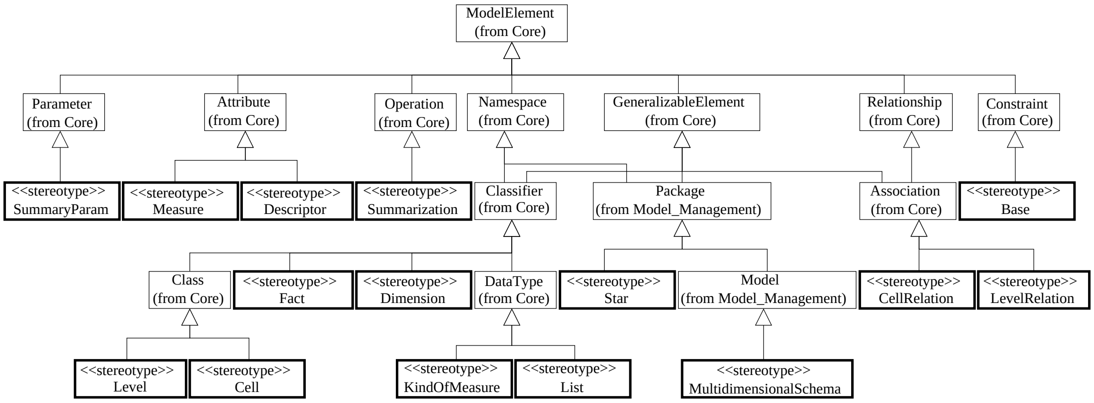


Figure 5.10: Extension of UML with **YAM**^{2} stereotypes

*Attribute*s of the *Class*es. All other elements in **YAM**^{2} have also been placed as *Stereotype*s of some UML concept (the formal definition of these *Stereotype*s is in appendix A). Maybe, the most relevant ones are **CellRelation** and **LevelRelation** that are *Aggregation*s. Moreover, a **Base** is just a *Constraint* stating that a set of functionally independent **Level**s fully determine instances of a **Cell**.

This proves that multidimensional modeling is just an specialization of general data modeling. We could roughly say that all we are doing is splitting elements in the model based on whether they refer to factual or dimensional data. It can be seen that some specific concepts are defined, besides properties and constraints of the new structures. [LST99] claims that E/R provides the complete functionality and support necessary for OLAP applications. Here, we can see that UML also provides such support. However, it is well known that the more specific the *Class*es in a schema are, the better they represent reality. In the same way, the more specific our data model is, the better it will represent reality. Therefore, in multidimensional modeling, it is important to show **Fact**s, **Dimension**s, **Cell**s, **Level**s, **Measure**s, and **Descriptor**s instead of just *Class*es, *Classifier*s, and *Attribute*s.

5.6 Comparison with other multidimensional models

Some O-O multidimensional models have already been defined, and some of them used UML syntax to do it. However, this is the first extension of UML for multidimensional modeling. As previously said, CWM does extend UML. Nevertheless, it is not a multidimensional data model, but a metadata standard for data warehousing.

In [BSHD98], a list of requirements for a multidimensional model in order to be suitable for OLAP were derived from general design principles, and from characteristics of OLAP applications. [Ped00] also presents eleven requirements (found in clinical data warehousing) for multidimensional data models. [Vas00a] gave yet another classification of logical cube models, which are not considered here, because **YAM**^{2} is at conceptual level. These comparisons are

Reference	Metamodel 1	2	Structures 3	4	5	6	7	8	9	10	11	12	13	14	15	Constraints 16	17	18	19	20	21	Operations 22	23	24	25	26
[Kim96]	NL	Rel.	✓	-	✓	✓	✓	-	-	p	-	-	p	-	-	-	-	-	-	p	-	-	Headers	✓	-	p
[LW96]	Maths	Rel.	✓	p	✓	✓	-	-	-	p	-	-	-	-	-	-	-	-	-	-	-	A	Relations	✓	✓	-
[AGS97]	Maths	-	-	p	✓	-	p	-	✓	-	-	-	-	-	-	-	-	p	-	-	-	A	Cubes	✓	✓	-
[HS97]	Maths	DL	✓	✓	✓	-	✓	-	-	-	-	-	-	-	-	p	p	p	-	-	-	A	Cubes	-	✓	-
[GL97]	Maths	Rel.	✓	-	✓	✓	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	A & C	Cubes	✓	✓	-
[DT97]	Maths	-	✓	-	✓	✓	✓	-	✓	-	-	-	-	-	-	-	-	p	-	-	-	A	Cubes	✓	✓	-
[CT98a]	Maths	-	✓	✓	✓	✓	-	-	-	✓	-	-	-	-	-	-	-	-	-	-	-	A & C	Cubes	✓	✓	-
[Leh98]	NL	-	-	✓	-	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	-	A	Cubes	-	-	-
[GMR98b]	NL	-	✓	✓	✓	✓	✓	-	-	✓	-	-	-	-	-	-	-	-	-	p	-	QL	Cubes	-	-	p
[TPGS01]	NL	O-O	✓	✓	✓	✓	✓	-	-	✓	p	p	-	p	-	✓	-	✓	✓	p	p	-	Cubes	-	-	-
[SBHD99]	E/R	E/R	✓	✓	✓	✓	✓	-	-	✓	p	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
[TBC99]	NL	E/R	✓	✓	✓	✓	✓	-	p	-	p	p	-	-	-	-	-	✓	✓	p	-	-	-	-	-	-
[BTW00]	UML	-	-	✓	✓	✓	✓	-	-	✓	-	-	-	-	-	-	-	✓	-	-	-	-	Cubes	-	-	-
[Vas00a]	Maths	-	-	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	A	Cubes	-	-	-
[Ped00]	Maths	-	✓	✓	✓	-	✓	p	✓	-	-	-	✓	-	✓	✓	✓	✓	✓	p	-	A	Cubes	-	-	-
[TKS01]	NL	-	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	✓	✓	✓	-	-	-	-	-	✓	-	-
YAM^2	UML	UML	✓	✓	✓	✓	✓	✓	p	✓	✓	✓	✓	✓	-	-	-	✓	✓	✓	✓	A	Cubes	p	✓	✓

Tick: Supported in the model.
p: Partially supported.
Rel.: Relational.
E/R: Entity-Relationship.
O-O: Object-Oriented paradigm.
A: Algebra.
Hyphen: Not supported or not explained how to support it.
NL: Natural Language.
DL: Description Logics.
UML: Unified Modeling Language.
QL: Query Language.
C: Calculus.

Table 5.5: Comparison between **YAM**^{2} and other multidimensional models

reviewed in section 2.2.1. In next pages, the items (most of them taken from those papers) used in the comparison of models, summarized in table 5.5, are briefly explained.

1. **Language used to define the model.** This column shows the language mainly used by every multidimensional model to express its metaschema.

2. **Extended framework.** Some models redefine or extend concepts in other, more general models or design frameworks, which is reflected in this column. In spite of [TP98] uses UML notation, it is not extending UML, because neither stereotypes, properties nor constraints (i.e. the extension mechanisms of UML) are used on defining the multidimensional model.

3. **Explicit separation of structure and contents** (from [BSHD98]). The data structure should be represented in the schema, while the contents should correspond to instances.

4. **Explicit aggregation hierarchies** (from [BSHD98] and [Ped00]). The model should show how data can be successively aggregated along analysis dimensions.

5. **Multiple hierarchies in each Dimension** (from [Ped00]). Although, aggregation hierarchies can be linear, most dimensions show multiple aggregation paths, so this should also be allowed.

6. **Dimension attributes** (from [BSHD98]). Showing other characteristics of the analysis dimensions that do not define hierarchies should also be possible.

7. **Measures sets** (from [BSHD98]). This refers to the possibility of defining complex **Cell** structures (grouping more that one **Measure**) related to the same **Fact**. Support provided by [AGS97] is considered partial, because in spite of it allows to manage tuples of measurements, they do not have any extra meaning as a whole.

8. **Measures at different levels of granularity**. Measurements could be taken at different aggregation levels. If so, **Measure**s belonging to the same **Fact**, or even showing the same kind of measure should be related in some way. [Ped00] proposes a comparison item slightly similar to this. However, it is stated as having exactly the same kind of measure being measured at different aggregation levels, so that sometimes it should be stored in a **Cell**, and others in a different one. It would be solved in **YAM**^{2} by specializing the **Cell**s depending on whether the **Measure** is derived or not.

9. **Descriptions and measurements are treated symmetrically** (from [BSHD98] and [Ped00]). The data model should allow **Fact**s to be treated as **Dimension**s and vice versa. **YAM**^{2} allows the usage of measurements as descriptors for other measurements by means of derivation mechanisms.

10. **Multi-star schemas**. Users should not be restricted to an isolated subject. They need to see several **Fact**s in one schema. It is not enough sharing **Dimension**s, as in [Kim96], since richer semantic relationships can be used.

11. ***Generalization* relationships**. *Generalization*s should be shown between **Dimension**s and **Fact**s (either in the same **Star** or in different **Star**s), as well as between **Level**s and **Cell**s.

12. ***Association* relationships**. Representing *Association*s should be allowed between **Dimension**s and **Fact**s (either in the same **Star** or in different **Star**s), as well as between **Level**s and **Cell**s.

13. **Change and time** (from [Ped00]). Although the business being reflected in the schema change, it should be possible to compare data over time.

14. **Derived elements** (from [BSHD98]). The definition of concepts by means of other concepts should be part of the schema.

15. **Imprecision** (from [Ped00]). The problem of representing and querying imprecise data has not been tackled in **YAM**^{2} .

16. **Non-onto hierarchies** (from [Ped00]). That is, hierarchies with paths of different lengths from the root to the leaves should be represented. **YAM**^{2} does not fulfill this point because, every object in an aggregation level must have the same structure, i.e. the *Class* structure. Thus, it is not possible that some instances of a *Class* can be divided into parts, while others can not (if so, it should be specialized in some way).

17. **Non-covering hierarchies** (from [Ped00]). That is, hierarchies where there exist relationships between instances of **Level**s that are not directly related. It is not necessary to be supported in this model, because if those relationships really exist, they should be explicitly represented in the schema by a part-whole relationship between the corresponding **Level**s.

18. **Many-to-many relationships between two Levels** (from [Ped00]). Some models just mention the possibility of having this kind of relationships (i.e. [AGS97], [HS97], and [DT97]).

19. **Many-to-many relationships between Fact and Dimension** (from [Ped00]). There is no constraint forbidding this in **YAM**^{2} . Like these relationships are allowed in UML, so they are in **YAM**^{2} . However, we can always see it as the fact being related to one set of elements in the **Dimension** so that we obtain a "to-one" relationship with a new **Dimension** of sets of elements. [SRME01] analyzes different implementations of these relationships.

20. **Additivity semantics** (from [BSHD98] and [Ped00]). Multidimensional models should show how a concept is obtained (if it can) at coarser granularities, and which aggregation functions can be applied to a given **Measure** in order to obtain the same **KindOfMeasure** at higher aggregation levels. Some multidimensional models, like [GMR98b] or [TP98], show possible functions that can be applied to a **Measure**. However, they do not show the specific operation that keeps the meaning of the measurement at coarser aggregation levels.

21. **Identification of facts**. The model should show how the different data subject of analysis can be identified by means of dimensional data. Most models just show the aggregation levels at which data are taken, but they do not show the functional dependency that fully determine the measurements. [Vas00a] mentions that the data set in a cube is a set of tuples such that contains a primary key. However, it is not reflected by his model in any way.

22. **Mathematical construct used for the operations** (from [BSHD98]). This column shows the mathematical formalism used in the models to define the operations over data.

23. **Elements over which operations are defined**.

24. **Queries using ad-hoc hierarchies not included in the schema** (from [BSHD98]). In order to roll data up, it is necessary a function showing the correspondence between **Level**s. If that function is not in the schema, where is it? **YAM**^{2} allows to define specific star schemas for every user profile. Thus, ad-hoc hierarchies for ad-hoc queries can be defined there.

25. **User defined aggregation functions** (from [BSHD98]). As any operation can be defined in a UML schema, so **YAM**^{2} supports it.

26. **Drill-across**. Some models allow to **drill-across** if the **Star**s share analysis dimensions. However, we can find semantic relationships that also allow it.

5.7 Conclusions

In the last years, lots of work have been devoted to OLAP technology in general, and multidimensional modeling in particular. However, there is no well accepted model, yet. Moreover, in spite of the acceptance of the O-O paradigm, only a couple of efforts take it into account for conceptual modeling.

In this chapter, **YAM**^{2} , a multidimensional conceptual model, which allows the usage of semantic O-O relationships between different **Star**s has been presented. This model has been defined as an extension of UML to make it much more understandable, and avoid its definition from scratch. As a side effect, this shows that multidimensional modeling is just an special case of data modeling.

Structures in the model have been defined by means of metaclasses, which are specialization of UML metaclasses. Thus, possible relationships among multidimensional elements have been systematically studied in terms of UML relationships among its elements, so that they allow to show semantically rich multi-star schemas. The inherent integrity constraints of the model pay special attention to identification of data, and summarizability (providing much more flexibility than those of previous multidimensional models).

It could be argued that all those semantic relationships and integrity constraints make **YAM**^{2} too complex or even cumbersome. However, we could find CASE tools to ease designer's work. As shown in appendix A, standard extension mechanisms of UML have been used to define **YAM**^{2} constructs. Therefore, any CASE tool following UML standard could easily be adapted for multidimensional design.

Finally, a set of well-known multidimensional operations has been explained in this framework by means of functions. Understanding operations over **Cube**s as operations over mathematical functions would allow to apply work done in that field to multidimensional query processing. This set has been shown as a closed and complete algebra for **Cube**s. Specifically, an operation to change the **Dimension**s of a **Cube** (i.e. **ChangeBase**) has been defined. Thus, by having candidate **Base**s and this operation, the most appropriate representation of data can be selected in every situation.

Chapter 6

Conclusions

> "... before I could come to any conclusion it occurred to me that my speech or my silence, indeed any action of mine, would be a mere futility. What did it matter what anyone knew or ignored? What did it matter who was manager? One gets sometimes such a flash of insight. The essentials of this affair lay deep under the surface, beyond my reach, and beyond my power of meddling."
>
> Joseph Conrad

The aim of this last chapter is to outline the main contributions of this thesis. Moreover, some research lines continuing this work are also indicated in section 6.2.

6.1 Survey of results

The main contribution of this thesis is **YAM**^{2} . However, there are other results that were obtained in the way to it, that should also be stressed here.

Firstly, the different "Data Warehousing" schemas were placed in the framework of "Federated Information Systems" (FIS). This allowed to study them from a different point of view, and understand their usefulness much better. Leaving aside the importance of performance, and paying special attention to conceptual design, a seven layers architecture of conceptual schemas was proposed to integrate the DW in a FIS. From that architecture of schemas, and the benefits of O-O data models as canonical models for federations bloomed the idea of using O-O concepts on designing the DW. Out of the different problems that raised in that context, it was decided to deep into the improvement of multidimensional data models.

Multidimensionality as such was not born in the research community, but as a response of tools vendors to the demand of analysts. Thus, there was not a strong mathematical foundation for it, like that of the Relational databases. Concepts were not clearly stated, and most efforts were devoted to improve performance and presentation. In the last years, it captured the attention of researchers, and data models have appeared without a standard, not even well accepted nomenclature. Therefore, another important milestone in this work was to define a framework that allowed to classify and compare all previous work. Six different elements were identified in multidimensional modeling literature. A few models used all of them, others only a subset. However, all constructs in these models perfectly fit into the classification grid of six holes presented in this thesis.

The six elements in the multidimensional models were clearly divided by the duality fact-dimension. Thus, they have been carefully studied separatedly. Firstly, some authors already pointed out that dimensional relationships between aggregation levels were part-whole relationships. However, most papers just referred to them as "roll-up" relationships. Anyway, properties of aggregation hierarchies were always imposed, never deduced. Thus, this thesis shows how they can be demonstrated from mereology axioms. Regarding factual data, they used to be considered as pure numbers that can be operated in some way. However, it is also shown in this thesis that, we can consider it as a commutative semigroup with union, that allows to define classes based on subjects and aggregation levels, and that can be placed in n-dimensional spaces defined by functional dependencies from dimensional data.

All that study was done with the O-O paradigm in mind. The ultimate goal was to benefit from its semantic relationships to improve multidimensional modeling. Thus, the first problem was that most OLAP tools consider semantically poor, isolated star shape schemas. To solve this, an architecture of schemas was proposed so that we could have semantically related stars, while they can still be easily implemented.

Finally, semantic relationships between multidimensional elements were studied. For this purpose, the relationship constructs offered by UML standard were analyzed. The usefulness of each and every relationship was exemplified and their consequences in the data structures showed.

Data structures, integrity constraints, and operations had to be defined in order to have a true data model. For the structures, it was chosen to extend UML by means of the mechanisms it offers (i.e. *Stereotypes*). Regarding integrity constraints, due to the importance of aggregation, they pay special attention to it. Moreover, another forgotten point was also considered here: identification of multidimensional data. Lastly, since data cubes were found to be functions, a closed and complete algebra of multidimensional operations was defined in terms of mathematical functions.

6.2 Future work

This thesis work can be continued following several different research lines. It can be related to other areas like database security, temporal issues, query optimization, and translation to logical/physical level methodologies, or just keep on studying modeling problems at conceptual level.

[Oli01] has studied the problems of integrating the security layers of different component databases in a federation. The architecture of seven levels of schemas shows that this work can also be used for "Data Warehousing". Nevertheless, its application is not automatic. New problems arise, mainly due to materialization of data (for instance, polyinstantiation).

As already said, the schema of the DW has characteristics of a bitemporal database. Thus, it should be studied how this area can benefit from those advances in temporal databases. Moreover, schema as well as data evolution in the analysis dimensions should also be studied in detail.

A semantically rich schema is really useful to help users on understanding data. However, it would also be half used, if not taken into account for query optimization. Semantic optimization should be considered, specially for drilling-across, as a future subject to be studied. Moreover, mathematical theory of functions could also be used on query optimization, because $\mathbf{YAM}^2$ operations work on functions.

An essential issue, not tackled in this thesis, is the study of a methodology for schema definition. Patterns should be detected in the data-oriented DW schema, in order to be translated in some way to the query-oriented DM schema. Multidimensional structures should be identified and captured from a non-dimensional schema. Moreover, the definition of multidimensional views should also be studied in order to support symmetric usage of factual and dimensional data, as well as ad-hoc hierarchies.

Once the multidimensional conceptual schema has been defined, it is necessary to implement it. It is well known that several options are available at this point. The implementation can be done on a Relational DBMS, an O-O DBMS, or a pure multidimensional DBMS. Thus, either the implementation of $\mathbf{YAM}^2$ on a MOLAP tool, or its translation to Relational or ODMG standards should be taken under consideration. Performance issues could be considered at this point (for instance, the sparsity of cubes).

These two tasks more closely related to modeling issues (i.e. defining a methodology and implementing multidimensional schemas on a given kind of DBMS) would be closely related to the implementation of a CASE tool. Two possibilities can be considered at this point. Firstly, any CASE tool following UML standard could be extended to support $\mathbf{YAM}^2$ constructs. The other option would be build a new tool from scratch, which would allow the implementation of more specific features. Both strategies look promising.

Finally, several conceptual multidimensional modeling problems are still open. As can be read in [AFGP96], *Aggregation* and *Association* relationships are closely related. Namely, there are some properties that the whole inherits from its parts (ex. being defective), others that the parts inherit from the whole they are part of (ex. location), and some properties in the parts which are systematically related to properties of the whole (ex. weight of parts being less than weight of the whole). The implications on aggregability, as well as the inheritance of properties between parts and wholes is another interesting research line to follow.

Bibliography

[AFGP96] Alessandro Artale, Enrico Franconi, Nicola Guarino, and Luca Pazzi. Part-Whole relations in Object-centered systems: an overview. *Data and Knowledge Engineering (DKE)*, 20, 1996.

[AGS97] Rakesh Agrawal, Ashish Gupta, and Sunita Sarawagi. Modeling Multidimensional Databases. In *Proceedings of 13th International Conference on Data Engineering (ICDE'97)*, pages 232–243. IEEE Computer Society, 1997.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AORS99] Alberto Abelló, Marta Oliva, Elena Rodríguez, and Fèlix Saltor. The syntax of BLOOM99 schemas. Technical Report LSI-99-34-R, Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 1999.

[AR00] Alberto Abelló and Elena Rodríguez. Describing BLOOM99 with regard to UML Semantics. In *Proceedings of the V Jornadas de Ingeniería del Software y Bases de Datos (JISBD'2000)*, pages 307–319. Gráficas Andrés Martín, 2000.

[BCN92] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design-an Entity-Relationship Approach*. Benjamin/Cummings, 1992.

[BFG97] Elisa Bertino, Elena Ferrari, and Giovanna Guerrini. T_Chimera: A Temporal Object-Oriented Data Model. *Theory and Practice of Object Sytems*, 3(2):103–125, 1997.

[BFJ+86] Thomas Burns, Elizabeth N. Fong, David Jefferson, Richard Knox, Leo Mark, Christopher Reedy, Louis Reich, Nick Roussopoulos, and Walter Truszkowski. Reference Model for DBMS Standardization, Database Architecture Framework Task Group (DAFTG) of the ANSI/X3/SPARC Database System Study Group. *SIGMOD Record*, 15(1):19–58, 1986.

[BHL00] Andreas Bauer, Wolfgang Hümmer, and Wolfgang Lehner. An Alternative Relational OLAP Modeling Approach. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000)*, volume 1944 of *LNCS*, pages 189–198. Springer, 2000.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 156–165. Morgan Kaufmann, 1997.

[BSH98] Jan W. Buzydlowski, Il-Yeol Song, and Lewis Hassell. A Framework for Object-Oriented On-line Analytical Processing. In *Proceedings of the 1st International Workshop on Data Warehousing and OLAP (DOLAP 98)*, pages 10–15. ACM, 1998.

[BSHD98] Markus Blaschka, Carsten Sapia, Gabriele Höfling, and Barbara Dinter. Finding your way through multidimensional data models. In *Proceedings of 9th International Workshop on Database and Expert Systems Applications (DEXA'98)*, pages 198–203. IEEE Computer Society, 1998.

[BTW00] Nguyen Thanh Binh, A Mint Tjoa, and Roland R. Wagner. An Object Oriented Multidimensional Data Model for OLAP. In *Proceedings of the 1st International Conference on Web-Age Information Management (WAIM'2000)*, volume 1846 of *LNCS*, pages 69–82. Springer, 2000.

[CCS93] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP to user-analysts: An IT mandate. Technical report, E. F. Codd & Associates, 1993.

[CD97] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[Cod79] E. F. Codd. Extending the relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[CSG94] Malú Castellanos, Fèlix Saltor, and Manolo García-Solaco. A Canonical Model for the Interoperability among Object-Oriented and Relational Databases. In *Distributed Object Management (Proceedings, International Workshop on Distributed Object Management)*, pages 309–314. Morgan Kaufmann, 1994.

[CT97] Luca Cabibbo and Ricardo Torlone. Querying Multidimensional Databases. In *Proceedings of the 6th International Workshop on Database Programming Languages (DBPL6)*, pages 319–335. Springer, 1997.

[CT98a] Luca Cabibbo and Ricardo Torlone. A Logical Approach to Multidimensional Databases. In *Advances in Database Technology - EDBT'98*, volume 1377 of *LNCS*, pages 183–197. Springer, 1998.

[CT98b] Luca Cabibbo and Ricardo Torlone. From a Procedural to a Visual Query Language for OLAP. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM 1998)*, pages 74–83. IEEE Computer Society, 1998.

[DGK+94] Curtis E. Dyreson, Fabio Grandi, Wolfgang Käfer, Nick Kline, Nikos Lorentzos, Yannis Mitsopoulos, Angelo Montanari, Daniel Nonen, Elisa Peressi, Barbara Pernici, John F. Roddick, Nandlal L. Sarda, Maria Rita Scalas, Arie Segev, Richard T. Snodgrass, Mike D. Soo, Abdullah Tansel, Paolo Tiberio, and Gio Wiederhold. A consensus glossary of temporal database concepts. *SIGMOD Record*, 23(1):52–64, 1994.

[DSHB99] Barbara Dinter, Carsten Sapia, Gabriele Höfling, and Markus Blaschka. The OLAP Market: State of the Art and Research Issues. *Journal of Computer Science and Information Management*, 2(3), 1999.

[DT97] Anindya Datta and Helen Thomas. A Conceptual Model and an algebra for On-Line Analytical Processing in Data Warehouses. In *Proceedings of the 7th Workshop on Information Technologies and Systems (WITS'97)*, pages 91–100, 1997.

[Dyr96] Curtis E. Dyreson. Information retrieval from an incomplete data cube. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB96)*, pages 532–543. Morgan Kaufmann, 1996.

[EK01] Johann Eder and Christian Koncilia. Changes of Dimension Data in Temporal Data Warehouses. In *Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2001)*, volume 2114 of *LNCS*, pages 284–293. Springer, 2001.

[EN00] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems.* Benjamin Cummings, third edition, 2000.

[FBSV00] Enrico Franconi, Franz Baader, Ulrike Sattler, and Panos Vassiliadis. *Fundamentals of Data Warehousing*, chapter Multidimensional Data Models and Aggregation, pages 87–105. Springer-Verlag, 2000. Matthias Jarke, Maurizio Lenzerini, Yannis Vassilious and Panos Vassiliadis editors.

[Fir97] Joseph M. Firestone. Object–Oriented Data Warehousing. Technical report, Executive Information Systems, Inc., 1997. White Paper No. Five.

[FM95] Leonidas Fegaras and David Maier. Towards an Effective Calculus for Object Query Languages. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD'95)*, pages 47–58. ACM Press, 1995.

[FS99] Enrico Franconi and Ulrike Sattler. A data warehouse conceptual data model for multidimensional aggregation. In *Proceedings of the 1st International Workshop on Design and Management of Data Warehouses (DMDW'99)*. CEUR-WS (http://www.ceur-ws.org), 1999.

[Gar98] Stephen R. Gardner. Building the data warehouse. *Communications of the ACM*, 41(9):52–60, 1998.

[GBLP96] Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, pages 152–159. IEEE Computer Society, 1996.

[GCS95] Manolo García-Solaco, Malú Castellanos, and Fèlix Saltor. A Semantic-Discriminated Approach to Integration of Federated Databases. In *Proceedings of the 3rd International Conference on Cooperative Information Systems (CoopIS-95)*, pages 19–31. University of Toronto, 1995.

[Gio00] William A. Giovinazzo. *Object-Oriented Data Warehouse Design.* Prentice Hall, 2000.

[GJJ97] Michael Gebhardt, Matthias Jarke, and Stephan Jacobs. A Toolkit for Negotiation Support Interfaces to Multi-Dimensional Data. *SIGMOD Record*, 26(2):348–356, 1997.

[GL97] Marc Gyssens and Laks V. S. Lakshmanan. A Foundation for Multi-dimensional Databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 1997)*, pages 106–115. Morgan Kaufmann Publishers, 1997.

[GL98] Frédéric Gingras and Laks V. S. Lakshmanan. nD-SQL: A multi-dimensional language for interoperability and OLAP. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 134–145, 1998.

[GLK99] Vivekanand Gopalkrishnan, Qing Li, and Kamalakar Karlapalem. Star/Snow-Flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies. In *Proceedings of the 1st International Workshop on Data Warehousing and Knowledge Discovery (DaWaK 1999)*, volume 1676 of *LNCS*, pages 11–22. Springer, 1999.

[GMR98a] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. Conceptual Design of Data Warehousing from E/R Schemes. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, pages 334–343. IEEE Computer Society, 1998.

[GMR98b] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The Dimensional Fact Model: a Conceptual Model for Data Warehouses. *International Journal of Cooperative Information Systems*, 7(2&3):215–247, 1998.

[GP95] Peter Gerstl and Simone Pribbenow. Midwinters, end games, and body parts: A classification of part-whole relations. *International Journal of Human-Computer Studies*, 43(5,6):865–889, 1995.

[GR98] Matteo Golfarelli and Stefano Rizzi. A Methodological Framework for Data Warehouse Design. In *Proceedigns of the 1st International Workshop on Data Warehousing and OLAP (DOLAP 98)*, pages 3–9. ACM, 1998.

[GR99] Matteo Golfarelli and Stefano Rizzi. Designing the data warehouse: key steps and crucial issues. *Journal of Computer Science and Information Management*, 2(1), 1999.

[GSC95] Manolo García-Solaco, Fèlix Saltor, and Malú Castellanos. A Structure Based Schema Integration Methodology. In *Proceedings of the 11th International Conference on Data Engineering (ICDE'95)*, pages 505–512. IEEE Computer Society, 1995.

[HLV00] Bodo Hüsemann, Jens Lechtenbörger, and Gottfried Vossen. Conceptual Data Warehouse Design. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'2000)*. CEUR-WS (http://www.ceur-ws.org), 2000.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. *SIGMOD Record*, 25(2):205–216, 1996.

[HS97] Mohand-Saïd Hacid and Ulrike Sattler. An Object-Centered Multi-dimensional Data Model with Hierarchically Structured Dimensions. In *Proceedings of the IEEE Knowledge and Data Engineering Exchange Workshop (KDEX 1997)*, pages 65–72. IEEE Computer Society, 1997.

[IIB96] William H. Inmon, Claudia Imhoff, and Greg Battas. *Building the operational data store*. John Wiley & Sons, 1996.

[IIS98] William H. Inmon, Claudia Imhoff, and Ryan Sousa. *Corporate Information Factory*. John Wiley & Sons, 1998.

[Inm96] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, second edition, 1996.

[ISO99] ISO. *ISO/IEC 9075:1999: Information technology — Database languages — SQL*. International Organization for Standardization, 1999.

[JLS99] Hosagrahar V. Jagadish, Laks V. S. Lakshmanan, and Divesh Srivastava. What can Hierarchies do for Data Warehouses? In *Proceedings of 25th International Conference on Very Large Data Bases (VLDB 1999)*, pages 530–541. Morgan Kaufmann, 1999.

[JLVV00] Matthias Jarke, Maurizio Lenzerini, Yannis Vassilious, and Panos Vassiliadis, editors. *Fundamentals of Data Warehousing*. Springer-Verlag, 2000.

[Kim96] Ralph Kimball. *The Data Warehouse toolkit*. John Wiley & Sons, 1996.

[KRRT98] Ralph Kimball, Laura Reeves, Margy Ross, and Warren Thornthwaite. *The Data Warehouse lifecycle toolkit*. John Wiley & Sons, 1998.

[LAW98] Wolfgang Lehner, Jens Albrecht, and Hartmut Wedekind. Normal Forms for Multidimensional Databases. In *Proceedings of 10th International Conference on Statistical and Scientific Database Management (SSDBM 1998)*, pages 63–72. IEEE Computer Society, 1998.

[Leh98] Wolfgang Lehner. Modeling Large Scale OLAP Scenarios. In *Advances in Database Technology - EDBT'98*, volume 1377 of *LNCS*, pages 153–167. Springer, 1998.

[LS97] Hans-J. Lenz and Arie Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of the 9th International Conference on Scientific and Statistical Database Management (SSDBM 1997)*, pages 132–143. IEEE Computer Society, 1997.

[LST99] Jana Lewerenz, Klaus-Dieter Schewe, and Bernhard Thalheim. Modelling Data Warehouses and OLAP Applications by Means of Dialoge Objects. In *Proceedings of the 18th International Conference on Conceptual Modeling (ER1999)*, volume 1728 of *LNCS*, pages 354–368. Springer, 1999.

[LW96] Chang Li and X. Sean Wang. A data model for supporting on-line analytical processing. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM'96)*, pages 81–88, 1996.

[MK00] Daniel L. Moody and Mark A. R. Kortink. From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'2000)*. CEUR-WS (http://www.ceur-ws.org), 2000.

[MO96] James Martin and James Odell. *Object-Oriented Methods: Pragmatic Considerations*. Prentice-Hall, 1996.

[MTW99] Oscar Mangisengi, A Min Tjoa, and Roland R. Wagner. Multidimensional Modeling Approaches for OLAP Based on Extended Relational Concepts. In *Proceedings of the 9th International Database Conference on Heterogeneous and Internet Databases (IDC 1999)*, 1999.

[OLA97] OLAP Council. OLAP and OLAP Server Definitions. Available at the URL http://www.olapcouncil.org/research/glossaryly.htm, 1997.

[Oli01] Marta Oliva. *Integració dels Criteris de Seguretat per Realitzar el Control d'Accés en un Sistema Federat de Bases de Dades Heterogènies*. PhD thesis, Department de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2001.

[OMG01a] OMG. *Common Warehouse Metamodel*, February 2001. Version 1.0.

[OMG01b] OMG. *Unified Modeling Language Specification*, September 2001. Version 1.4.

[OS01] Marta Oliva and Fèlix Saltor. Integrating Multilevel Security Policies in Multilevel Federated Database Systems. In *Data and Applications Security: Developments and Directions (Proceedings of the 14th IFIP 11.3 Working Conference in Database and Applications Security -DBSec-)*, pages 135–147. Kluwer Academic Publishers, 2001.

[Ped00] Torben B. Pedersen. *Aspects of Data Modeling and Query Processing for Complex Multidimensional Data.* PhD thesis, Faculty of Engineering & Science (Aalborg University), 2000.

[Pen01] Nigel Pendse. The OLAP Report - What is OLAP? Available at the URL http://www.olapreport.com/fasmi.html, 2001. Business Intelligence Ltd.

[PJ98] Torben B. Pedersen and Christian S. Jensen. Research Issues in Clinical Data Warehousing. In *Proceedings of the 10th International Conference on Statistical and Scientific Database Management (SSDBM 1998)*, pages 43–52. IEEE Computer Society, 1998.

[PJ99] Torben B. Pedersen and Chistian S. Jensen. Multidimensional Data Modeling for Complex Data. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, pages 336–345. IEEE Computer Society, 1999.

[PR99] Elaheh Pourabbas and Maurizio Rafanelli. Characterization of Hierarchies and Some Operators in OLAP Environment. In *Proceedings of the 2nd International Workshop on Data Warehousing and OLAP (DOLAP 99)*, pages 54–59. ACM, 1999.

[RAO+01] Elena Rodríguez, Alberto Abelló, Marta Oliva, Fèlix Saltor, Cecilia Delgado, Eladio Garví, and José Samos. On Operations along the Generalization/Specialization Dimension. In *Proceedings of the 4th International Workshop on Engineering Federated Information Systems (EFIS 2001)*, pages 70–83. IOS Press, 2001.

[ROSC97] Elena Rodríguez, Marta Oliva, Fèlix Saltor, and Benet Campderrich. On Schema and Functional Architectures for Multilevel Secure and Multiuser Model Federated DB Systems. In *Proceedings of the International Workshop on Engineering Federated Database Systems (EFDBS), held in conjuntion with CAISE'97*, pages 93–104, 1997.

[SA86] Richard Snodgrass and Ilsoo Ahn. Temporal Databases. *IEEE Computer*, 19(9):35–42, 1986.

[Sal96] Fèlix Saltor. *Panorama Informático*, chapter Semántica de datos, pages 39–64. Federación Española de Sociedades de Informática (FESI), 1996.

[SBHD99] Carsten Sapia, Markus Blaschka, Gabriele Höfling, and Barbara Dinter. Extending the E/R Model for the Multidimensional Paradigm. In *Proceedings of the 1st International Workshop on Data Warehouse and Data Mining (DWDM), in conjunction with ER'98*, volume 1552 of *LNCS*, pages 105–116. Springer, 1999.

[SCdMM99] Adolfo Sánchez, Jose María Cavero, Adoración de Miguel, and Paloma Martínez. IDEA: A Conceptual Multidimensional Data Model and Some Methodological Implications. In *Proceedings of the VI Congreso Internacional de Investigación en Ciencias Computacionales (CIICC'99)*, pages 307–318. Instituto Tecnológico de Cancún, 1999.

[SCG91] Fèlix Saltor, Malú Castellanos, and Manolo García-Solaco. Suitability of Data Models as Canonical Models for Federated DBs. *SIGMOD Record*, 20(4):44–48, 1991.

[Sho97] Arie Shoshani. OLAP and Statistical Databases: Similarities and Differences. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97)*, pages 185–196. ACM Press, 1997.

[SL90] Amit P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[SR91] Arie Shoshani and Maurizio Rafanelli. A Model for Representing Statistical Objects. In *Proceedings of the 3rd International Conference on Information Systems and Management of Data (COMAD'91)*. McGraw-Hill, 1991.

[SR97] Fèlix Saltor and Elena Rodríguez. On Intelligent Access to Heterogeneous Information. In *Proceedings of the 4th International Workshop on Knowledge Representation meets DataBases (KRDBi 1997)*. CEUR-WS, 1997.

[SRME01] Il-Yeol Song, William Rowen, Carl Medsker, and Edward Ewen. An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling. In *Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW'2001)*. CEUR-WS (http://www.ceur-ws.org), 2001.

[SSSB98] José Samos, Fèlix Saltor, Jaume Sistac, and Agustí Bardés. Database Architecture for Data Warehousing: An Evolutionary Approach. In *Proceedings of 9th International Conference on Database and Expert Systems Applications (DEXA'98)*, volume 1460 of *LNCS*, pages 746–756. Springer, 1998.

[Sto93] Veda C. Storey. Understanding semantic relationships. *VLDB Journal: Very Large Data Bases*, 2(4):455–488, October 1993.

[TBC99] Nectaria Tryfona, Frank Busborg, and Jens G. Borch Christiansen. starER: A conceptual model for data warehouse design. In *Proceedings of the 2nd International Workshop on Data Warehousing and OLAP (DOLAP 99)*, pages 3–8. ACM, 1999.

[Tes01] Olivier Teste. Towards Conceptual Multidimensional Design in Decision Support Systems. In *Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems (ADBIS 2001)*, pages 77–88, 2001.

[Tha91] Bernhard Thalheim. *Dependencies in Relational Databases.* B.G. Teubner, 1991.

[Tho97] Erik Thomsen. *OLAP Solutions.* John Wiley & Sons, 1997.

[TKS01] Aris Tsois, Nikos Karayannidis, and Timos Sellis. MAC: Conceptual Data Modeling for OLAP. In *Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW'2001).* CEUR-WS (http://www.ceur-ws.org), 2001.

[TP98] Juan Carlos Trujillo and Manuel Palomar. An Object-Oriented Approach to Multidimensional Database Conceptual Modeling. In *Proceedings of the 1st International Workshop on Data Warehousing and OLAP (DOLAP 98)*, pages 16–21. ACM, 1998.

[TPG00] Juan Carlos Trujillo, Manuel Palomar, and Jaime Gómez. Applying Object-Oriented Conceptual Modeling Techniques to the Design of Multidimensional Databases and OLAP applications. In *Proceedings of the 1st International Conference on Web-Age Information Management (WAIM'2000)*, volume 1846 of *LNCS*, pages 83–94. Springer, 2000.

[TPGS01] Juan Carlos Trujillo, Manuel Palomar, Jaime Gómez, and Il-Yeol Song. Designing Data Warehouses with OO Conceptual Models. *IEEE Computer*, 34(12):66–75, 2001.

[Tru01] Juan Carlos Trujillo. *El modelo GOLD: un modelo conceptual orientado a objetos para el diseño de aplicaciones.* PhD thesis, Departamento de Lenguajes y Sistemas Informáticos (Universidad de Alicante), 2001.

[TS97] Dimitri Theodoratos and Timos K. Sellis. Data Warehouse Configuration. In *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 126–135. Morgan Kaufmann, 1997.

[UW97] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems.* Prentice-Hall, 1997.

[Vas98] Panos Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube operations. In *Proceedings of the 10th International Conference on Scientific and Statistical Database Management (SSDBM 1998)*, pages 53–62. IEEE Computer Society, 1998.

[Vas00a] Panos Vassiliadis. *Data Warehouse Modeling and Quality Issues.* PhD thesis, Department of Electrical and Computer Engineering (National Technical University of Athens), 2000.

[Vas00b] Panos Vassiliadis. Gulliver in the land of data warehousing: practical experiences and observations of a researcher. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses (DMDW'2000).* CEUR-WS (http://www.ceur-ws.org), 2000.

[VS99] Panos Vassiliadis and Timos Sellis. A Survey of Logical Models for OLAP Databases. *SIGMOD Record*, 28(4):64–69, 1999.

[WB97] Ming-Chuan Wu and Alejandro P. Buchmann. Research issues in data warehousing. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW)*, Informatik Aktuell, pages 61–82. Springer, 1997.

[Wid95] Jennifer Widom. Research problems in data warehousing. In *Proceedigns of the 4th International Conference on Information and Knowledge Management (CIKM'95)*, pages 25–30. ACM, 1995.

Appendix A

UML Profile for Multidimensional Modeling

> "The nice thing about standards is that there are so many of them to choose from."
>
> Andrew S. Tanenbaum

This appendix contains a UML *Profile* for multidimensional modeling. It is the formalization, using UML standard notation in [OMG01b], of **YAM**^{2} . Thus, UML is customized for a multidimensional domain.

Firstly, section A.1 introduces the profile and A.2 contains a table of the stereotypes defined in it. Section A.3 shows the definition of each *Stereotype* following UML notation. Finally, section A.4 lists the integrity constraints in OCL.

A.1 Introduction

This *Profile* aims to facilitate the modeling of multidimensional data. Basically, it defines two *Stereotype*s for some metaclasses in the UML core. Thus, it allows to represent factual and dimensional data.

Moreover, besides those *Stereotype*s, others are also defined to facilitate the representation of specific multidimensional constraints regarding summarizability and identification of data. Some *Stereotypes* are also specialized to establish the difference between basic and derived elements in the model.

A.2 Summary of Profile

The *Stereotype*s that are defined by this *Profile* are summarized in the following table:

Stereotype	Base Class
MultidimensionalSchema	*Model*
Star	*Package*
Fact	*Classifier*
Dimension	*Classifier*
Cell	*Class*
SummarizedCell	*Class*
FundamentalCell	*Class*
Level	*Class*
Measure	*Attribute*
SummarizedMeasure	*Attribute*
FundamentalMeasure	*Attribute*
Descriptor	*Attribute*
Base	*Constraint*
Summarization	*Operation*
Transitive	*Operation*
NonTransitive	*Operation*
SummaryParam	*Parameter*
CellRelation	*Association*
LevelRelation	*Association*
KindOfMeasure	*DataType*
List	*DataType*
Induction	*ModelElement*

A.3 Stereotypes and Notation

Data multidimensionally modeled consist of several interrelated **Star**s. Each **Star** is composed by one **Fact** and several **Dimension**s. On the one hand, **Fact**s are composed by **Cell**s, and these are composed by **Measure**s. On the other hand, **Dimension**s are composed by **Level**s, and these are composed by **Descriptor**s. In addition, there are *Stereotype*s to show how data can be aggregated and identified.

A.3.1 MultidimensionalSchema

Stereotype	Base Class	Parent	Tags	Constraints	Description
Multidimensional-Schema	*Model*	N/A	stars	N/A	Specifies a schema showing multi-dimensional data, composed by different **Star**s.

A *Model* stereotyped as ≪MultidimensionalSchema≫ is the notation used for a **MultidimensionalSchema**.

Tag	Stereotype	Type	Multiplicity	Description
stars	**Multidimensional-Schema**	≪stereotype≫ **Star**	1..*	Shows the different star shape schemas in the multidimensional schema.

A.3.2 Star

Stereotype	Base Class	Parent	Tags	Constraints	Description
Star	*Package*	N/A	fact, dimensions	The **Fact** is associated to every **Dimension**.	Specifies a star shape schema. It represents data regarding only one subject of analysis.

The notation used for a **Star** is a *Package* stereotyped as ≪Star≫.

Tag	Stereotype	Type	Multiplicity	Description
fact	**Star**	≪stereotype≫ Fact	1	Shows the subject of analysis.
dimensions	**Star**	≪stereotype≫ Dimension	1..*	Shows the different points of view analysts could use on the study of the subject of analysis.

A.3.3 Fact

Stereotype	Base Class	Parent	Tags	Constraints	Description
Fact	*Classifier*	N/A	cells	The **Cell**s are related by **CellRelation**s. A **Fact** can neither be specialization of a **Dimension**, nor be obtained by evolution of a **Dimension**, nor be aggregate into a **Dimension**. A **Fact** cannot be derived from a **Dimension**. Moreover, its **Cell**s cannot be related by specialization, evolution nor derivation.	Specifies a subject of analysis.

The notation used for a **Fact** is a box with an "F" in an upper corner.

Tag	Stereotype	Type	Multiplicity	Description
cells	**Fact**	≪stereotype≫ Cell	1..*	Shows the different granularities of the data subject of analysis.

A.3.4 Dimension

Stereotype	Base Class	Parent	Tags	Constraints	Description
Dimension	*Classifier*	N/A	levels	The **Level**s are related by means of **LevelRelation**s. A **Dimension** can neither be specialization of a **Fact**, nor be obtained by evolution of a **Fact**, nor be aggregate into a **Fact**. Moreover, its **Level**s cannot be related by specialization, evolution nor derivation.	Specifies a point of view on analyzing a given subject of analysis.

The notation used for a **Dimension** is a box with a "D" in an upper corner.

Tag	Stereotype	Type	Multiplicity	Description
levels	**Dimension**	≪stereotype≫ Level	1..*	Shows the different granularities that can be used in the analysis, along the analysis dimension

A.3.5 Cell

Stereotype	Base Class	Parent	Tags	Constraints	Description
Cell	*Class*	N/A	bases	Can only be associated to **Measure** *Attributes*. Every **Cell** belongs to exactly one **Fact**. A **Cell** can neither be specialization of a **Level**, nor evolution of a **Level**, nor be aggregated into a **Level**, nor derived from a **Level**. Moreover, its **Measures** cannot be related by evolution.	Specifies a subject of analysis at a given granularity.

The notation used for a **Cell** is a *Class* with a "C" in an upper corner.

Tag	Stereotype	Type	Multiplicity	Description
bases	**Cell**	≪stereotype≫ Base	1..*	Shows the different spaces that can be used to place the instances of the **Cell**.

A.3.6 SummarizedCell

Stereotype	Base Class	Parent	Tags	Constraints	Description
SummarizedCell	*Class*	**Cell**	N/A	It must be possible to derive all its **Measures**.	Specifies **Cell**s whose data can be derived.

The notation used for a **SummarizedCell** is a **Cell** with a derivation bar in front of its name.

A.3.7 FundamentalCell

Stereotype	Base Class	Parent	Tags	Constraints	Description
FundamentalCell	*Class*	**Cell**	N/A	It must be associated to some **Measures** that cannot be derived.	Specifies **Cell**s whose data cannot be derived.

The notation used for a **FundamentalCell** is that of a **Cell**. It is not necessary any special notation since the specialization into fundamental and summarized **Cell**s is alternative. Thus, a given **Cell** is marked if it is **SummarizedCell** and not market otherwise.

A.3.8 Level

Stereotype	Base Class	Parent	Tags	Constraints	Description
Level	*Class*	N/A	N/A	Can only be associated to **Descriptor** *Attributes*. Every **Level** belongs to exactly one **Dimension**. A **Level** can neither be specialization of a **Cell**, nor evolution of a **Cell**, nor be aggregated into a **Cell**. Moreover, its **Descriptors** cannot be related by evolution.	Specifies a point of view at a given granularity.

The notation used for a **Level** is a *Class* with an "L" in an upper corner.

A.3.9 Measure

Stereotype	Base Class	Parent	Tags	Constraints	Description
Measure	*Attribute*	N/A	type	It must be associated to a **Cell**, and cannot be obtained by evolution of a **Descriptor**.	Specifies *Attributes* of the subject of analysis.

The notation used for a **Measure** is that of an *Attribute*. It is not necessary to use any special notation since this kind of elements will always be associated to **Cell**s and are the only *Attribute*s that can be associated to **Cell**s.

Tag	Stereotype	Type	Multiplicity	Description
type	**Measure**	≪stereotype≫ KindOfMeasure	1	Specifies the kind of measure the **Measure** is.

A.3.10 SummarizedMeasure

Stereotype	Base Class	Parent	Tags	Constraints	Description
SummarizedMeasure	*Attribute*	Measure	from	N/A	Specifies a **Measure** that can be derived.

The notation used for a **SummarizedMeasure** is that of a **Measure** with a bar in front of its name. Moreover, a *Comment* can be attached to it showing the formula used in the calculation.

Tag	Stereotype	Type	Multiplicity	Description
from	**SummarizedMeasure**	≪stereotype≫ Measure	1..*	Shows the **Measures** used in the calculation of a **SummarizedMeasure**.

A.3.11 FundamentalMeasure

Stereotype	Base Class	Parent	Tags	Constraints	Description
FundamentalMeasure	*Attribute*	Measure	N/A	It can only be associated to **FundamentalCells**	Specifies a **Measure** that cannot be derived.

The notation used for a **FundamentalMeasure** is that of a **Measure**. It is not necessary any special notation since the specialization into fundamental and summarized **Measures** is alternative. Thus, a given **Measure** is marked if it is **SummarizedMeasure** and not market otherwise.

A.3.12 Descriptor

Stereotype	Base Class	Parent	Tags	Constraints	Description
Descriptor	*Attribute*	N/A	N/A	It must be associated to a **Level**, and cannot be obtained by evolution of a **Measure**.	Specifies *Attributes* of the different points of view.

The notation used for a **Descriptor** is that of an *Attribute*. It is not necessary to use any special notation since this kind of elements will always be associated to **Levels** and are the only *Attributes* that can be associated to **Levels**.

A.3.13 Base

Stereotype	Base Class	Parent	Tags	Constraints	Description
Base	*Constraint*	N/A	components	The different components must be functionally independent.	Specifies a finite space defined by the cartesian product of a given set of **Levels**.

Similar to constraints of *Stereotypes*, this kind of **Constraint** is also drawn in the box of the corresponding **Cell**, stereotyped as ≪Base≫.

Tag	Stereotype	Type	Multiplicity	Description
components	**Base**	≪stereotype≫ Level	1..*	Shows the set of **Levels** that compose the **Base**.

A.3.14 Summarization

Stereotype	Base Class	Parent	Tags	Constraints	Description
Summarization	*Operation*	N/A	N/A	Its *parameters* are **SummaryParam**.	Specifies the operation applied to a given kind of measure on aggregating along a given analysis dimension.

A summarization is represented by means of a *String*.

A.3.15 Transitive

Stereotype	Base Class	Parent	Tags	Constraints	Description
Transitive	*Operation*	**Summarization**	N/A	N/A	Specifies that a given **Summarization** is transitive.

The notation used for a **Transitive** is that of a **Summarization**. It is not necessary any special notation since the specialization into transitive and not transitive is alternative. Thus, a given **Summarization** is marked if it is **NonTransitive** and not market otherwise.

A.3.16 NonTransitive

Stereotype	Base Class	Parent	Tags	Constraints	Description
NonTransitive	*Operation*	**Summarization**	N/A	N/A	Specifies that a given **Summarization** is not transitive.

A *Comment* can be attached to the **Summarization** showing it is not transitive.

A.3.17 SummaryParam

Stereotype	Base Class	Parent	Tags	Constraints	Description
SummaryParam	*Parameter*	N/A	N/A	Are always part of a **Summarization**. Their type is a **List**.	Specifies the parameters of a **Summarization**.

The notation used for a **SummaryParam** is that of a *Parameter*. It is not necessary any special notation, since these are always part of a **Summarization**, and only **SummaryParam** can be part of it. Since their type is always a **List**, it can just be noted by the type of the **List**.

A.3.18 CellRelation

Stereotype	Base Class	Parent	Tags	Constraints	Description
CellRelation	*Association*	N/A	correspond	It is associated to two **Cells**. Both **Cells** must belong to the same **Fact**. One of its *AssociationEnd* must be an *Aggregation*.	Specifies relationships between **Cells** in a **Fact**.

The notation used for a **CellRelation** is that of an *Aggregation*. It is not necessary any special notation since it is the only kind of aggregation allowed between **Cell**s inside a **Fact**.

Tag	Stereotype	Type	Multiplicity	Description
correspond	**CellRelation**	≪stereotype≫ **LevelRelation**	1	Specifies the relationship between **Levels** that generates the relationship between **Cells**.

A.3.19 LevelRelation

Stereotype	Base Class	Parent	Tags	Constraints	Description
LevelRelation	*Association*	N/A	N/A	It is associated to two **Levels**. Both **Levels** must belong to the same **Dimension**. One of its *AssociationEnd* must be an *Aggregation*.	Specifies relationships between **Levels** in a **Dimension**.

The notation used for a **LevelRelation** is that of an *Aggregation*. It is not necessary any special notation since it is the only kind of aggregation allowed between **Level**s inside a **Dimension**.

A.3.20 KindOfMeasure

Stereotype	Base Class	Parent	Tags	Constraints	Description
KindOfMeasure	*DataType*	N/A	invalidSource	N/A	Specifies a kind of **Measure**.

A *DataType* stereotyped as ≪KindOfMeasure≫ is the notation used for a **KindOfMeasure**.

Tag	Stereotype	Type	Multiplicity	Description
invalidSource	**KindOfMeasure**	≪stereotype≫ **Level**	*	Specifies the different aggregation levels that cannot be used as source for the calculation of a given kind of **Measure**.

A.3.21 List

Stereotype	Base Class	Parent	Tags	Constraints	Description
List	*Classifier*	N/A	type	N/A	Specifies a list of elements of a type.

The notation used for a **List** is a *Classifier* stereotyped as ≪List≫.

Tag	Stereotype	Type	Multiplicity	Description
type	**List**	*Classifier*	1	Specifies the type of the elements in the list.

A.3.22 Induction

Stereotype	Base Class	Parent	Tags	Constraints	Description
Induction	*ModelElement*	N/A	inductor, subject, induced	Each **Dimension** only induces one **Summarization** on a **KindOfMeasure**.	Specifies that a given summarization function is induced by aggregations along an analysis dimension, over all **Measures** of a given kind.

The notation used for a **Induction** is a *String* added to the box of the **KindOfMeasure** stating the list of **Dimension**s and the **Summarization** that these induce.

Tag	Stereotype	Type	Multiplicity	Description
inductor	**Induction**	≪ stereotype ≫ **Dimension**	1..*	Specifies a set of analysis dimensions that induce a given **Summarization**.
subject	**Induction**	≪ stereotype ≫ **KindOfMeasure**	1	Specifies a **KindOfMeasure** subject of summarization.
induced	**Induction**	≪ stereotype ≫ **Summarization**	1	Specifies the **Summarization** induced on a kind of **Measure** when aggregating along some analysis dimensions.

A.4 Well-Formedness Rules

Similar to those rules, expressed in OCL, specified for every UML element, this section contains such rules for the multidimensional stereotypes.

A.4.1 Star

[1] The **Fact** is associated to each and every **Dimension**.

context Star **inv**:
self.dimensions–>forall(d | d.oppositeAssociationEnds.association–>includes(self.fact))

A.4.2 Fact

[1] The **Cell**s that form a **Fact** are connected by means of **CellRelation**s.

context Fact **inv**:
self.cells–>forall(c:Cell | c.connectedSubsetOfCells = self.cells)

[2] A **Fact** cannot be specialization of a **Dimension**.

context Fact **inv**:
self.generalization.parent–>forall(p | not p.oclIsKindOf(Dimension))

[3] A **Fact** cannot be obtained by evolution of a **Dimension**.

context Fact **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Dimension))

[4] A **Fact** cannot be aggregate into a **Dimension**.

context Fact **inv**:
self.allOppositeAssociaitonEnds–>select(aggregation=♯aggregate).participant–>
forall(p | not p.oclIsKindOf(Dimension))

[5] A **Fact** cannot be derived from a **Dimension**.

context Fact **inv**:
self.clientDependency–>select(oclIsKindOf(derive)).supplier–>
forall(p | not p.oclIsKindOf(Dimension))

[6] **Cell**s in a **Fact** cannot be related by specialization.

context Fact **inv**:
self.cells–>forall(c | c.specialization.child–>intersection(self.cells)–>isEmpty())

[7] **Cell**s in a **Fact** cannot be related by evolution.

context Fact **inv**:
self.cells–>forall(c | c.targetFlow.source–>intersection(self.cells)–>isEmpty())

[8] **Cell**s in a **Fact** cannot be related by derivation.

context Fact **inv**:
self.cells–>forall(c | c.clientDependency–>select(oclIsKindOf(derive)).supplier–>
intersection(self.cells)–>isEmpty())

A.4.3 Dimension

[1] The **Level**s that form a **Dimension** are connected by means of **LevelRelation**s.

context Dimension **inv**:
self.levels–>forall(l:Level | l.connectedSubsetOfLevels = self.levels)

[2] A **Dimension** cannot be specialization of a **Fact**.

context Dimension **inv**:
self.generalization.parent–>forall(p | not p.oclIsKindOf(Fact))

[3] A **Dimension** cannot be obtained by evolution of a **Fact**.

context Dimension **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Fact))

[4] A **Dimension** cannot be aggregate into a **Fact**.

context Dimension **inv**:
self.allOppositeAssociaitonEnds–>select(aggregation=♯aggregate).participant–>
forall(p | not p.oclIsKindOf(Fact))

[5] **Level**s in a **Dimension** cannot be related by specialization.

context Dimension **inv**:
self.levels–>forall(l | l.specialization.child–>intersection(self.levels)–>isEmpty())

[6] **Level**s in a **Dimension** cannot be related by evolution.

context Dimension **inv**:
self.levels–>forall(l | l.targetFlow.source–>intersection(self.levels)–>isEmpty())

[7] **Level**s in a **Dimension** cannot be related by derivation.

context Dimension **inv**:
self.levels–>forall(l | l.clientDependency–>select(oclIsKindOf(derive)).supplier–>
intersection(self.levels)–>isEmpty())

A.4.4 Cell

[1] A **Cell** can only be associated to **Measure** *Attribute*s.

context Cell **inv**:
self.feature–>forall(s | s.oclIsKindOf(Attribute) **implies**
s.oclIsKindOf(Measure))

[2] A **Cell** belongs to exactly one **Fact**.

context Cell **inv**:
Fact.allInstances–>select(f | f.cells–>exists(self))–>size = 1

[3] A **Cell** cannot be specialization of a **Level**.

context Cell **inv**:
self.generalization.parent–>forall(p | not p.oclIsKindOf(Level))

[4] A **Cell** cannot be obtained by evolution of a **Level**.

context Cell **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Level))

[5] A **Cell** cannot be aggregate into a **Level**.

context Cell **inv**:
self.allOppositeAssociaitonEnds–>select(aggregation=♯aggregate).participant–>
forall(p | not p.oclIsKindOf(Level))

[6] A **Cell** cannot be derived from a **Level**.

context Cell **inv**:
self.clientDependency–>select(oclIsKindOf(derive)).supplier–>
forall(p | not p.oclIsKindOf(Level))

[7] **Measure**s in a **Cell** cannot be related by evolution.

context Cell **inv**:
self.feature–>select(oclIsKindOf(Measure))–>forall(c | c.targetFlow.source–>
intersection(self.feature–>select(oclIsKindOf(Measure)))–>isEmpty())

Additional operations

[1] The operation connectedSubsetOfCells results in the set of all **Cell**s connected to a given one by means of **CellRelation**s.

connectedSubsetOfCells : Set(Cell)

connectedSubsetOfCells = self–>union(self.oppositeAssociationEnds–>select (association.oclIsTypeOf(CellRelation)).participant.connectedSubsetOfCells)

A.4.5 SummarizedCell

[1] All its **Measure**s must be summarizable.

context SummarizedCell **inv**:
self.feature–>forall(s | s.oclIsKindOf(Attribute) **implies**
s.oclIsKindOf(SummarizedMeasure))

A.4.6 FundamentalCell

[1] It must be associated to some **Measure**s that are not derived.

context FundamentalCell **inv**:
self.feature–>exists(s | s.oclIsKindOf(FundamentalMeasure))

A.4.7 Level

[1] A **Level** can only be associated to **Descriptor** *Attribute*s.

context Level **inv**:
self.feature–>forall(s | s.oclIsKindOf(Attribute) **implies**
s.oclIsKindOf(Descriptor))

[2] A **Level** belongs to exactly one **Dimension**.

context Level **inv**:
Dimension.allInstances–>select(d | d.levels–>exists(self))–>size = 1

[3] A **Level** cannot be specialization of a **Cell**.

context Level **inv**:
self.generalization.parent–>forall(p | not p.oclIsKindOf(Cell))

[4] A **Level** cannot be obtained by evolution of a **Cell**.

context Level **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Cell))

[5] A **Level** cannot be aggregate into a **Cell**.

context Level **inv**:
self.allOppositeAssociaitonEnds–>select(aggregation=♯aggregate).participant–>
forall(p | not p.oclIsKindOf(Cell))

[6] **Descriptor**s in a **Level** cannot be related by evolution.

context Level **inv**:
self.feature–>select(oclIsKindOf(Descriptor))–>forall(c | c.targetFlow.source–>
intersection(self.feature–>select(oclIsKindOf(Descriptor)))–>isEmpty())

Additional operations

[1] The operation connectedSubsetOfLevels results in the set of all **Level**s connected to a given one by means of **LevelRelation**s.

connectedSubsetOfLevels : Set(Cell)

connectedSubsetOfLevels = self–>union(self.oppositeAssociationEnds–>select
(association.oclIsTypeOf(LevelRelation)).participant.connectedSubsetOfLevels)

A.4.8 Measure

[1] A **Measure** can only be associated to a **Cell**.

context Measure **inv**:
self.owner.oclIsKindOf(Cell))

[2] A **Measure** cannot be obtained by evolution of a **Descriptor**.

context Measure **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Descriptor))

A.4.9 FundamentalMeasure

[1] A **FundamentalMeasure** can only be associated to a **FundamentalCell**.

context FundamentalMeasure **inv**:
self.owner.oclIsKindOf(FundamentalCell))

A.4.10 Descriptor

[1] A **Descriptor** can only be associated to a **Level**.

context Descriptor **inv**:
self.owner.oclIsKindOf(Level))

[2] A **Descriptor** cannot be obtained by evolution of a **Measure**.

context Descriptor **inv**:
self.targetFlow.source–>forall(p | not p.oclIsKindOf(Measure))

A.4.11 Base

[1] The different components of a **Base** must be functionally independent.

context Base **inv**:
self.components–>forall(l_1 | self.components–>forall(l_2 | l_1 <> l_2 implies $\emptyset \rightarrow\rightarrow l_1 \mid l_2$))
being $\emptyset \rightarrow\rightarrow l_1 \mid l_2$ a degenerated dependency as explained in page 88.

A.4.12 Summarization

[1] The *parameter*s of a **Summarization** are **SummaryParam**.

context Summarization **inv**:
self.parameter–>forall(oclIsKindOf(SummaryParam))

A.4.13 SummaryParam

[1] A **SummaryParam** is always part of a **Summarization**.

context SummaryParam **inv**:
self.BehaviouralFeature.oclIsKindOf(Summarization))

[2] The type of a **SummaryParam** is a **List**.

context SummaryParam **inv**:
self.type.oclIsKindOf(List))

A.4.14 CellRelation

[1] A **CellRelation** associates two **Cell**s.

context CellRelation **inv**:
self.connection–>forall(participant.oclIsKindOf(Cell))

[2] Both **Cell**s must belong to the same **Fact**.

context CellRelation **inv**:
Fact.allInstances–>forall(f | f.cells–>intersection(self.allConnections.participant)–>
notEmpty() implies f.cells–>includes(self.allConnections.participant))

[3] One of its *AssociationEnd* must be an *Aggregation*.

context CellRelation **inv**:
self.allConnections–>select(aggregation = ♯aggregate)–>size = 1

A.4.15 LevelRelation

[1] A **LevelRelation** associates two **Level**s.

context LevelRelation **inv**:
self.connection–>forall(participant.oclIsKindOf(Level))

[2] Both **Level**s must belong to the same **Dimension**.

context LevelRelation **inv**:
Dimension.allInstances–>forall(d | d.levels–>intersection(self.allConnections.participant)
–>notEmpty() implies d.levels–>includes(self.allConnections.participant))

[3] One of its *AssociationEnd* must be an *Aggregation*.

context LevelRelation **inv**:
self.allConnections–>select(aggregation = ♯aggregate)–>size = 1

A.4.16 Induction

[1] Every **Dimension** can only induce one **Summarization** on a **KindOfMeasure**.

context Induction **inv**:
Induction.allInstances–>forall(i | i.inductor–>intersection(self.inductor)–>notEmpty()
and i.subject=self.subject implies i.induced=self.induced)

Appendix B

Design examples with YAM^2

"Few things are harder to put up with than the annoyance of a good example."

Mark Twain, "Pudd'nhead Wilson's Calendar"

This appendix exemplifies the usage of **YAM^2** on modeling a database. Cases of study used by other authors are modeled here with **YAM^2** . The aim of this appendix is to exemplify the usage of **YAM^2** , at the same time that it is compared with other contributions in the area.

B.1 Sales of products in a grocery chain

The grocery chain example has been used by several authors (like [Kim96], or [GMR98a]) to exemplify their work. Different nuances are introduced by each author. In this case, a chain of supermarkets is modeled, so that each supermarket is divided into departments that offer different kinds of products. The supermarkets are spread over different states. We want to analyze what, where, and which day the products are sold.

B.1.1 Kimball's schema

Figure B.1 represents the case of study at logical level as in [Kim96]. It shows a central "fact table" related to its "dimension tables" by foreign keys. The "dimension tables" do not explicit aggregation hierarchies, but contain a list of attributes. Aggregability constraints are not present in the schema, either.

In this case, the `Promotion` **Dimension** is of special interest to analyze the impact of different offers. Moreover, the finner granularity chosen has been the items sold by promotion by store by day.

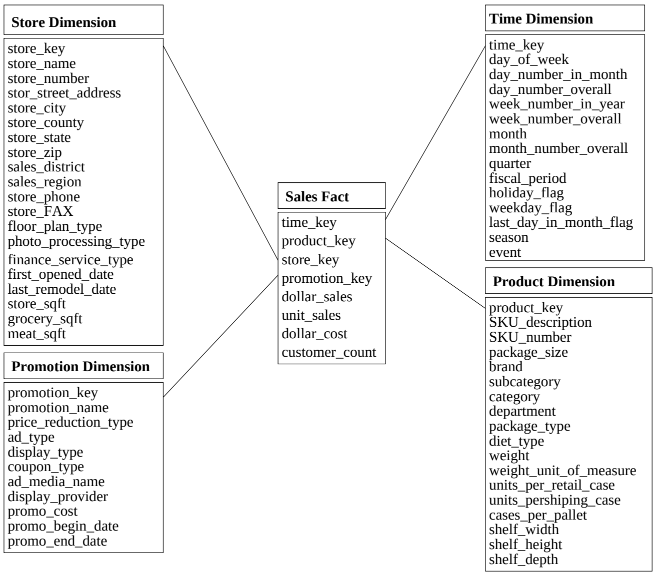


Figure B.1: Schema of the grocery chain case study [Kim96]

B.1.2 Golfarelli's version of Kimball's schema

In figure B.2, the same schema is represented at conceptual level by M. Golfarelli. Circles represent "dimension attributes" (i.e. **Levels**), while "non-dimension attributes" (i.e. **Descriptors**) are represented by lines. Arcs represent "to-one" relationships. The non-additivity of the number of customers along `Product` **Dimension** is shown by a dashed line. Moreover, a dash crossing an arc indicates optionality in the relationship.

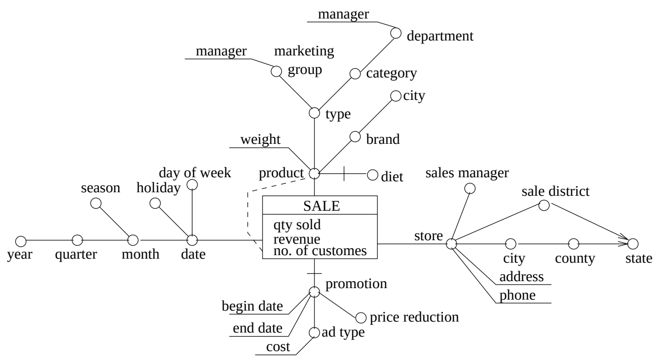


Figure B.2: Schema of the grocery chain case study [GMR98a]

B.1.3 YAM^2 schema

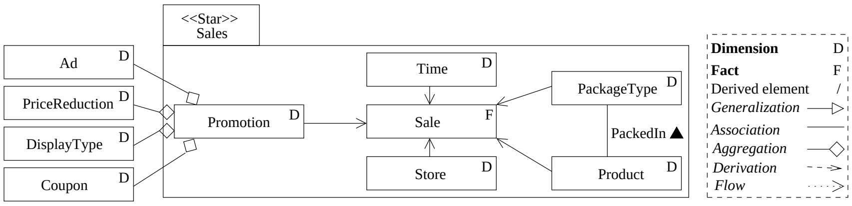


Figure B.3: **Upper** level schema of the grocery chain case study modeled with **YAM**^{2}

Firstly, we can see in figure B.3 the grocery chain schema, at **Upper** detail level, modeled with **YAM**^{2} . In contrast with figure B.1, this schema contains five **Dimension**s. This fact reflects the independence between product and the package type. Any product might be packed in any kind of package. Therefore, we could aggregate independently along both hierarchies.

Another interesting point rises when looking at the `Promotion` **Dimension**. We can see that a promotion is the combination of four different concepts `Advertisement`, `PriceReduction`, `DisplayType`, and `Coupon`. In this case, we are interested in the study of combinations of those promotion mechanisms. However, other analysts could be interested in the influence of those concepts separately. Therefore, the corresponding analysis dimensions could be being used in other **Star**s.

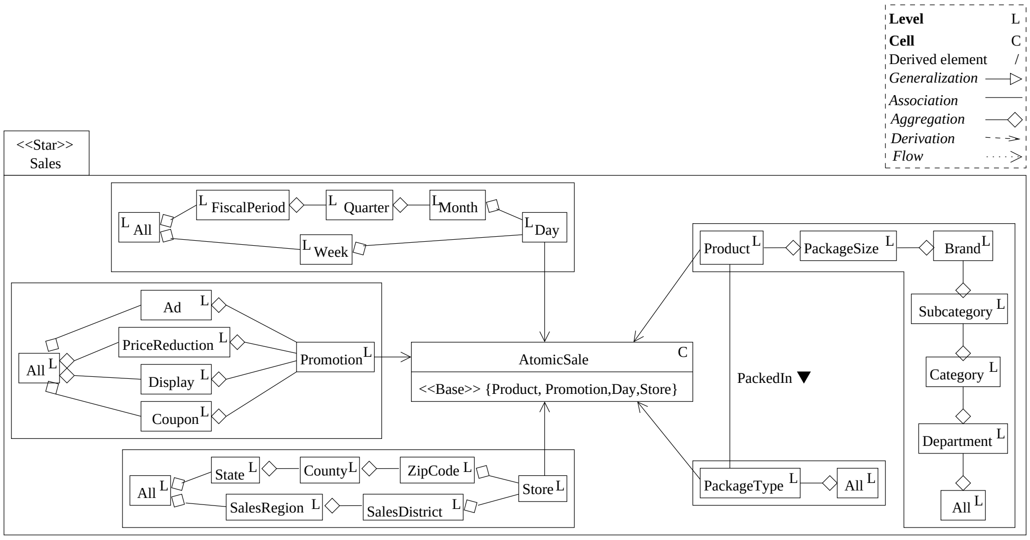


Figure B.4: **Intermediate** level schema of the grocery chain case study modeled with **YAM**^{2}

In figure B.4, the details of the **Dimension**s are shown. Since `Promotion` is composed by four **Dimension**s the aggregation hierarchies of those **Dimension**s are alternative aggregation paths in `Promotion`. For the sake of simplicity those **Dimension**s have not been depicted at **Intermediate** detail level.

At this level, we also show that even though there are five **Dimension**s, four of them are enough to identify a sale. A product determines a package type.

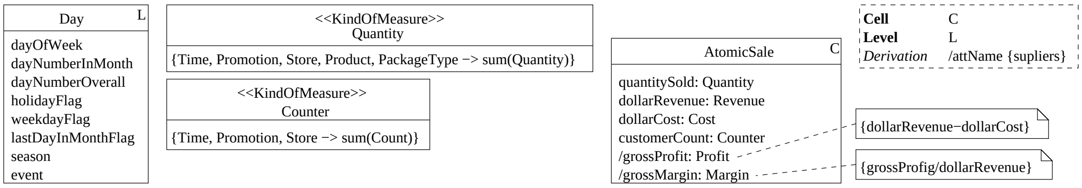


Figure B.5: **Lower** level schema of the grocery chain case study modeled with $\mathbf{YAM^2}$

Finally, at **Lower** detail level, we can see the attributes of the different *Class*es. For the sake of simplicity, only the attributes of `Day` **Level** have been depicted. Other dimensional attributes in figure B.1 would be translated in a similar way. Notice that **Dimension** keys do not appear in figure B.5. Since we are dealing with an O-O data model, OIDs are assumed. Thus, in order to represent a foreign key, drawing the corresponding *Association* between *Class*es is enough.

Much more interesting is the information regarding factual attributes. Firstly, we are able to show which attributes are basic, and which are derived (besides the corresponding derivation formula). Moreover, summarizability can be explicited, as well. We can see that `Quantity` **Measure**s will be summarizable along any analysis dimension. Nevertheless, `Counter` **Measure**s will not be summarizable along `Product`, nor `PackageType` **Dimension**s.

B.1.4 Discussion

Maybe, at logical level, depending on the DBMS and applications we use for the implementation, expliciting this information would be a serious mistake. However, at conceptual level, representing users aggregation intentions is critical, as can be seen in Golfarelli's version as well as in $\mathbf{YAM^2}$. Moreover, depicting these hierarchies, we are able to outline the relationships between hierarchies of related **Dimension**s (for instance, `Promotion` **Dimension** in our case).

Additivity is not reflected in Kimball's schemas as it is in Golfarelli's version. This can easily be reflected in $\mathbf{YAM^2}$, as a particular case of aggregability where we apply "sum" function. Golfarelli's schemas offer the possibility of depicting optionality of attributes or dimensional relationships, as well, which can also be easily shown in $\mathbf{YAM^2}$ by means of standard UML mechanisms (i.e. cardinalities).

B.2 Warehouse

Another interesting example in [Kim96] is this one about warehouse inventories. It is originally used to exemplify semi-additivity of **Measure**s, i.e. those that cannot be summarized by means of the "sum" function. The stocks are not additive along the temporal dimension because they represent snapshots of a level.

Three different schemas are explored. The first one, every day, measures the inventory levels and places them in separate records. The second schema contains one record for each delivery to the warehouse, which registers the disposition of all the items until they have left the warehouse. The third and last data schema records every change of the status of delivery products.

B.2.1 Original schema

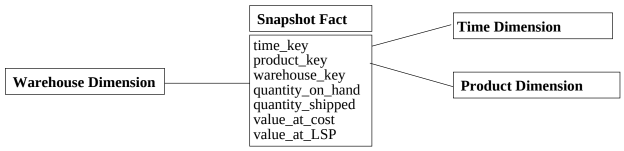


Figure B.6: Schema of the warehouse snapshot case study [Kim96]

Figure B.6 only shows three "dimension tables", namely `Warehouse`, `Time`, and `Product`. The most interesting **Measure** is `quantity_on_hand`. It records the stock of a given product in the warehouse. The other **Measure**s allow to obtain more elaborated derived **Measure**s like "number of turns", "days supply", or "gross margin return on inventory", which are not reflected in the schema.

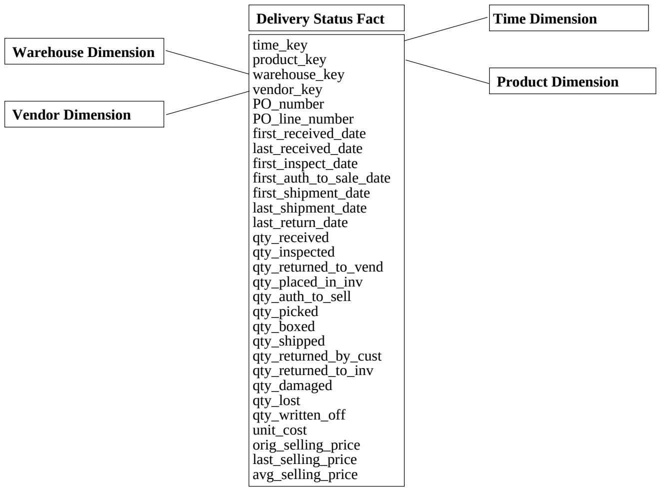


Figure B.7: Schema of the warehouse delivery status case study [Kim96]

The second possibility is reflected in figure B.7. It assumes we are able to distinguish the different items, so that we know which was supplied from each vendor. Thus, every time we obtain a new shipment, it is recorded and tracked until we sell it. Dates and quantities are registered for each step.

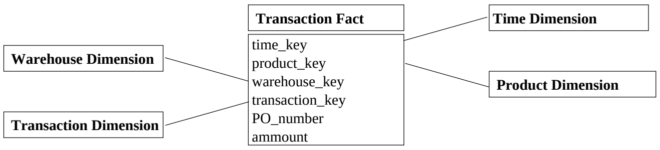


Figure B.8: Schema of the warehouse transaction case study [Kim96]

The last inventory schema is drawn in figure B.8. It records every transaction in the warehouse. Four analysis dimensions are proposed, i.e. `Warehouse`, `Time`, `Product`, and `Transaction`. `Transaction` **Dimension** has one instance for every kind of transaction. The only **Measure** in the "fact table" is `amount`.

B.2.2 YAM^2 schema

All three schemas regard warehouse inventory, the same subject. Therefore, the analysts will probably want to see them together or navigate from one to another. With **YAM**^{2} , this is easy, because they belong to the same **Fact**, hence, the same **Star**.

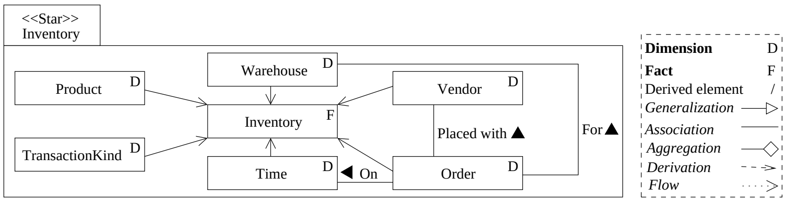


Figure B.9: **Upper** level schema of the warehouse case study modeled with **YAM**^{2}

At **Upper** detail level, in figure B.9, we have the only **Star**. It contains `Inventory` **Fact** and six **Dimension**s, namely `Warehouse`, `Vendor`, `Order`, `Time`, `TransactionKind`, and `Product`. Moreover, we observe that an order is placed with a vendor, for a given warehouse, on a given moment.

In figure B.10, we have the same schema at **Intermediate** detail level. To avoid complicating unnecessarily the figure, aggregation hierarchies and *Association*s between **Level**s have not been depicted. Thus, we can appreciate that there are three **Cell**s of interest (corresponding to the three different schemas in [Kim96]). The more detailed one is `Transaction`, which can be analyzed based on **Level**s `Minute`, `TransactionKind`, `Warehouse`, `Order` and `Product` that identify it (it is assumed that at any time we can distinguish the order by means of which the

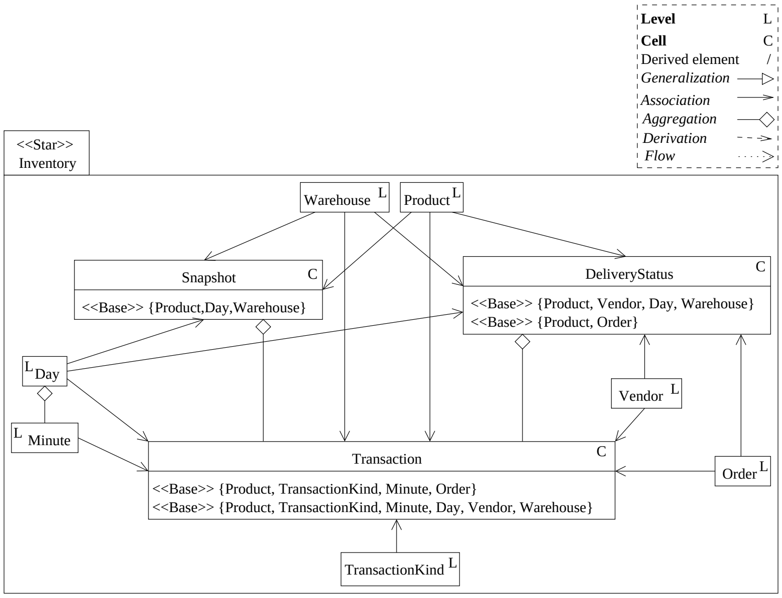


Figure B.10: **Intermediate** level schema of the warehouse case study modeled with **YAM**^{2}

different items were obtained). `Vendor`, `Warehouse`, and `Day` fully determine `Order`, so that can substitute it in the **Base**.

If we aggregate appropriately (adding or subtracting depending on the kind of transaction) instances of that **Cell** along `Time` and `Warehouse` **Dimension**s, we obtain snapshots of inventory at the desired granularity (`Day` and `Warehouse` in this case). Therefore, if it would not contain **Measure**s regarding costs, it could be considered a derived **Cell**. It can be studied along `Warehouse`, `Product`, and `Time` **Dimension**s.

Finally, if we would aggregate `Transaction` instances based on the order, we would obtain instances of `DeliveryStatus` **Cell**. Its instances are identified by `Order` and `Product`, or by `Product`, `Vendor`, `Day`, and `Warehouse`. In this case, this **Cell** does contain some **Measure**s that cannot be obtained from those in the **Atomic Cell**. Therefore, it cannot be considered as derived. Storing its instances is not optional but mandatory.

At **Lower** detail level, we can see the attributes of every *Class*. Derivation formulas of the different *Attribute*s can be explicited, as well as aggregability of the different **KindOfMeasure**s. We can see that stocks can be added along `Product` and `Warehouse` **Dimension**s. However, to obtain stocks at coarser granularities along `Time`, "avg" should be performed. Another possibility is to obtain the stock of any unit of time as the stock at the upper bound of the period. This is used on the derivation of some **Measure**s.

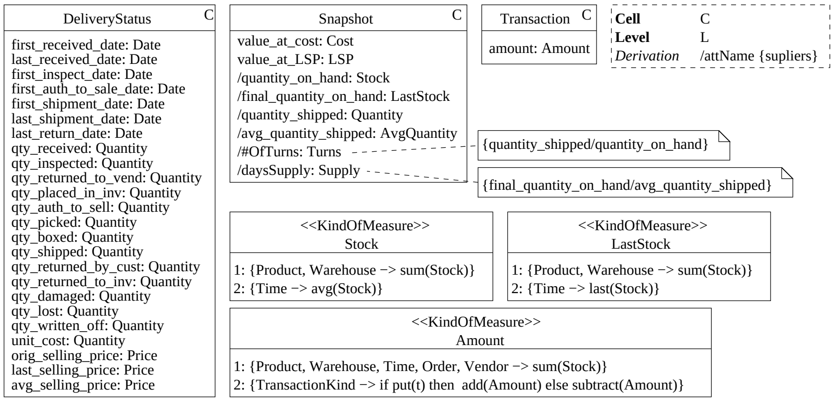


Figure B.11: **Lower** level schema of the warehouse case study modeled with **YAM**^{2}

B.2.3 Discussion

In this case of study, we can appreciate the advantages of a semantically rich multidimensional model. If we have Kimball's schemas, we could say that they share some "dimension tables", so that we can drill across. However, by means of **YAM**^{2} we can place all data in the same schema and represent the different relationships we find among them. `Transaction` and `Snapshot` not only share **Dimension**s. `Snapshot` is the aggregation of `Transaction` and can be computed from it.

Moreover, the specific "AVG_TIME_SUM" operation proposed by Kimball to aggregate stocks is not necessary any more. The problem can be solved in a more general way, showing how data is aggregated along each **Dimension**. Thus, the aggregation operation will aggregate any kind of data based on its specification.

`PO_number` is referred in [Kim96] as a "degenerate dimension" (i.e. a "dimension key" without a corresponding "dimension table"), because it does not contain any attribute. In this case, with **YAM**^{2} , it is just another **Class** (i.e. `Order`) associated to the **Fact**, acting as **Dimension**. It could have *Attribute*s or not, and give rise to a Relational table in a ROLAP system or not. We are at conceptual level, yet.

Finally, just to mention that we could imagine yet another possibility, besides those three schemas in figures B.6, B.7, and B.8. We could specialize `Transaction` depending on the kind of transaction. This would give rise to a different **Cell** for every kind of transaction. These **Cell**s would generate new **Star**s, and could contain specific **Measure**s or use other **Dimension**s in the analysis. Moreover, once we have defined the more general schema, we could define different **Star**s offering the appropriate views (i.e. the same three independent schemas we found in [Kim96]).

B.3 Tickets in supermarkets

[Tru01] proposes, as case study, a modification of the grocery chain proposed in [Kim96]. The main difference is that the study focuses on tickets and ticket lines, which adds some difficulty to the schema.

The `Client` analysis dimension substitutes `Promotion`, especialization hierarchies of products are considered, and derivation and summarization information is also modeled into the schema. Moreover, some "many-to-many" relationships appear in this case, like that between products and tickets (the same product can be found in several ticket lines), or between sales districts and communities.

B.3.1 Original schema

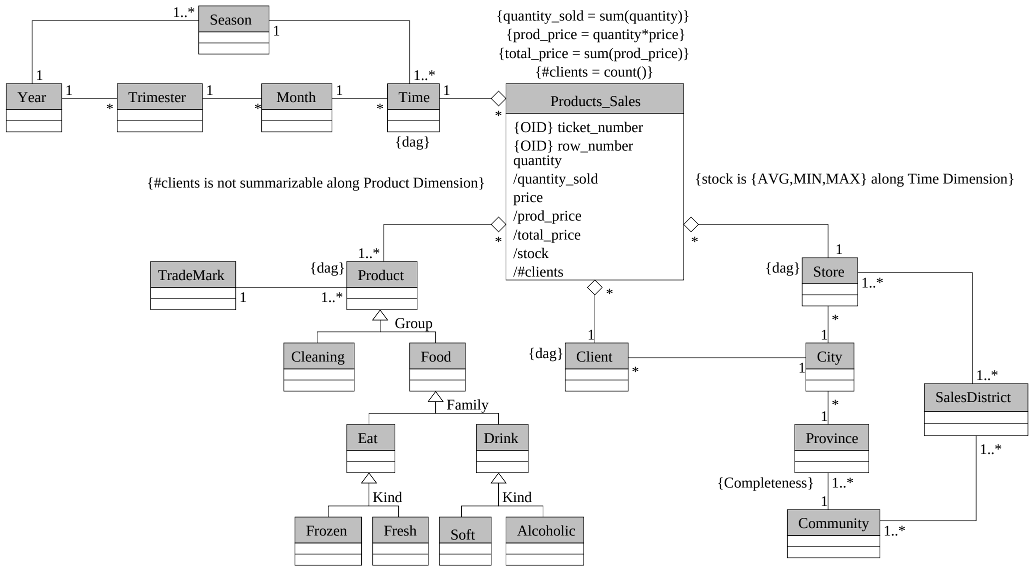


Figure B.12: Schema of the tickets case study modeled with GOLD

Figure B.12 shows the case study expressed in GOLD (for the sake of simplicity, the attributes of every class have not been depicted in the figure). Of special interest in this schema are the specialization hierarchy of `Product`, and its "many-to-many" relationship with the `Product_Sales` "facts class". It is also important to notice that `Client` and `Store` share part of their aggregation hierarchies. Moreover, several constraints and derivation formulas are depicted around the "facts class".

Moreover, besides the information in the data schema, user requirements are also represented in the GOLD model. For instance, in figure B.13 we can see four of such requirements, which (as shown in [TPGS01]) could be directly translated to standard "Object Query Language" (OQL) syntax:

CC_1		CC_2		CC_3		CC_4	
Measures	SUM(quantity)	Measures	SUM(quantity)	Measures	SUM(quantity)	Measures	SUM(quantity)
Slice		Slice		Slice		Slice	
Time.Year = "1999"		Store.community = "Comunidad Valenciana" Product.Group = "Food"		Time.Year = "1999"			
Dice		Dice		Dice		Dice	
Store.Community		Store.Province, Store.City, Product.Family, Product.Type		Store.Community Client		Store.Community	
OLAP operations		OLAP operations		OLAP operations		OLAP operations	

Figure B.13: User requirements for the case study modeled with GOLD graphical notation

CC_1 Quantity sold per product during the year 1999, grouped by community where they were sold.

CC_2 Quantity of food sold in the "Comunidad Valenciana" aggregated by family and kind of product, and by the province and city where the store is placed.

CC_3 Quantity of product sold during 1999 grouped by clients and community.

CC_4 Quantity of product sold grouped by community.

B.3.2 YAM^2 schema

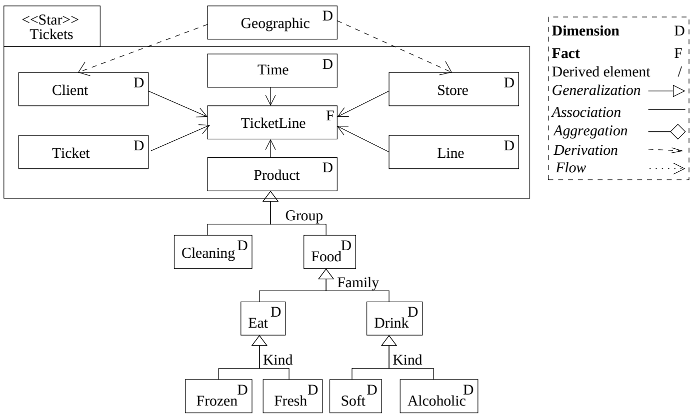


Figure B.14: **Upper** level schema of the tickets case study modeled with **YAM^2**

In figure B.14, we can see information about this case of study, modeled with **YAM^2** , at **Upper** detail level. Three points are of interest here. Firstly, we can see that geographic information has been used on defining both `Client` and `Store` **Dimension**s. Moreover, the specialization of `Product` is also shown here. Finally, we can see that two **Dimension**s have been added. If information is stored per line per ticket, it is supposed that these will be analysis dimensions of our data. If not, why is the information stored in the **Fact** at that granularity? Thus, both **Dimension**s are shown at this detail level.

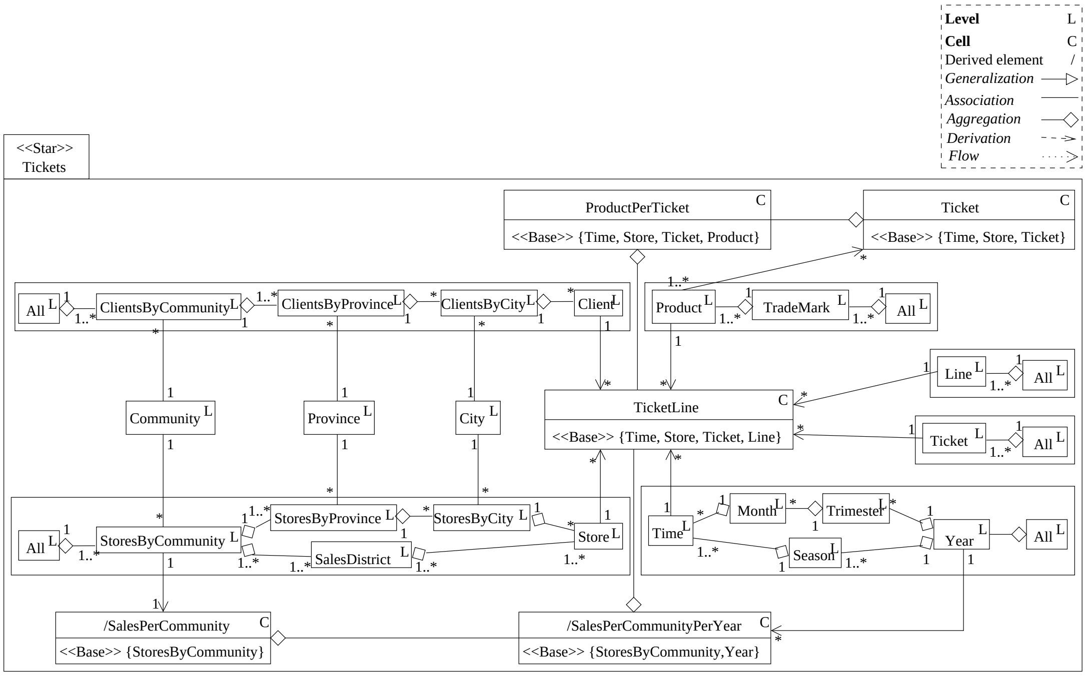


Figure B.15: **Intermediate** level schema of the tickets case study modeled with **YAM^2**

At **Intermediate** detail level, in figure B.15, things get much more complicated. Firstly, we can see that the aggregation hierarchy of every **Dimension** is depicted with the corresponding cardinalities. Notice that in this case, `Client` and `Store` do not share their hierarchies. In spite of that, **Level**s in both **Dimension**s are related to the same *class*es that would belong to a `Geographic` **Dimension**. Instances of `Client` **Dimension** represent clients or sets of clients. Instances of `Store` represent stores or sets of stores. Instances of `Geographic` **Dimension** represent geographic areas. Therefore, **Level**s in either `Client` or `Store` can be associated to the same geographic areas, but they cannot be geographic areas.

Another difference with figure B.12 is that the cardinality of the *Association* between `Product` and `TicketLine` is "one-to-many". The cardinality between `Ticket` and `Product` is "many-to-many". Notice that this information is represented in **YAM^2** with different **Cell**s related by *Composition*. It shows that several `TicketLine`s compose a `ProductPerTicket`, and several `ProductPerTicket` compose a `Ticket`. The **Association**s between these secondary **Cell**s and the corresponding aggregation levels have not been depicted to keep the schema as clear as possible.

Finally, user requirements can be outlined here by depicting the desired **SummarizedCell**s.

In this case, `SalesPerCommunityPerYear` and `SalesPerCommunity` are shown to exemplify it. Besides showing these derived **Class**es in the schema, it is also possible to represent in more detail the user requirements by means of **YAM**^{2} query algebra.

CC_1 From the class `SalesPerCommunityPerYear`, we select year 1999, and project only `quantity` attribute. Then, we change the base to see it in a unidimensional space of communities.

$$\gamma_{StoresByCommunity}(\pi_{quantity}(\sigma_{Year='1999'}(SalesPerCommunityPerYear)))$$

CC_2 For each kind of product, we would have its own **Star**. Thus, from the corresponding **Cell**, we select the tickets sold in "Comunidad Valenciana". Afterwards, we roll them up to `StoresByCity` and `All` **Level** in the corresponding subclass of `Product`. Then, the desired attribute is chosen, and finally, data are placed in a 2-dimensional space defined by cities and kinds of products.

$$\begin{gathered}\gamma_{StoresByCity\times\{Frozen,Fresh,Soft,Alcoholic\}}(\pi_{quantity}(\\ \rho_{StoresByCity}(\sigma_{StoresByCommunity="ComunidadValenciana"}(FrozenPerTicket))\cup\\ \rho_{StoresByCity}(\sigma_{StoresByCommunity="ComunidadValenciana"}(FreshPerTicket))\cup\\ \rho_{StoresByCity}(\sigma_{StoresByCommunity="ComunidadValenciana"}(SoftPerTicket))\cup\\ \rho_{StoresByCity}(\sigma_{StoresByCommunity="ComunidadValenciana"}(AlcoholicPerTicket))))\end{gathered}$$

CC_3 Selected tickets of 1999 are rolled up to `Clients` and `StoresByCommunity`. Then, `quantity` **Measure** is projected, and placed in a 2-dimensional space defined by `Client` and `StoresByCommunity`.

$$\gamma_{Client\times StoresByCommunity}(\pi_{quantity}(\rho_{Clients,StoresByCommunity}(\sigma_{Year='1999'}(Ticket))))$$

CC_4 Instances of `SalesPerCommunity` **Cell** are placed in a unidimensional space defined by `StoresByCommunity`.

$$\gamma_{StoresByCommunity}(\pi_{quantity}(SalesPerCommunity))$$

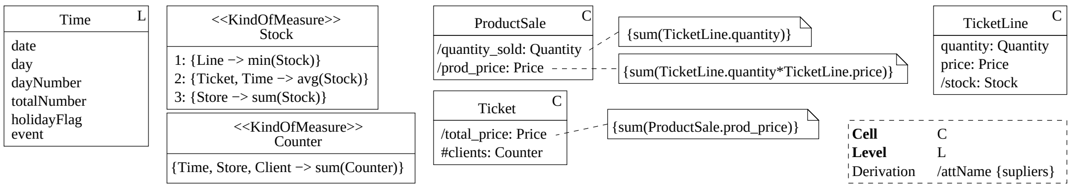


Figure B.16: **Lower** level schema of the tickets case study modeled with **YAM**^{2}

Finally, figure B.16 shows the more detailed elements in the schema. `Time` **Level** shows its **Descriptors**, and we see that a `Counter` can only be added along `Time`, `Store`, and `Client`. Moreover, it is shown that `Stock` measurements must be summarized by means of "min" along `Line`, "avg" along `Ticket` and `Time`, and "sum" along `Store`, in that order. The **Measure**s in the different **Cell**s are also shown in the figure, besides the corresponding formula for the derived ones.

B.3.3 Discussion

Both, GOLD and **YAM**^{2} , use UML notation. Thus, it is quite simple to appreciate some differences. Firstly, we can see that relationships between **Level**s are simply considered *Associations* in GOLD, and *Compositions* in **YAM**^{2} . This means that instances of **Level**s in **YAM**^{2} show sets of elements, while in GOLD they represent elements that identify grouping characteristics.

From that difference in the conception of the structure of **Dimension**s, comes the difference in the sharing of hierarchies. If elements in the **Dimension**s represent sets of instances, it is not possible that two different **Dimension**s share a **Level**, because they represent different concepts (even if the grouping characteristic is the same). Therefore, with **YAM**^{2} , it must be represented as a derivation from a common **Dimension** used in the definition of both hierarchies. However, since GOLD classes represent grouping characteristics, they can be freely shared between **Dimension**s.

Another point regarding aggregation hierarchies is that of specialization hierarchy. GOLD understands specializations as aggregation paths, so that aggregation is also allowed by subclasses. In **YAM**^{2} , aggregation is strictly represented by aggregation hierarchies. Nevertheless, to facilitate it, a **Level** could be defined so that their instances correspond to the subclasses. This **Level** can be used in the definition of the sub**Dimension**s.

There are two concepts explicit in GOLD, and implicit in **YAM**^{2} . Firstly, aggregation hierarchies being a DAG comes from the definition of **Dimension** and mereological axioms. Moreover, completeness is also assumed in **YAM**^{2} , if level `All` exists in the hierarchy. Also comes from mereological axioms that aggregation hierarchies can always be defined so that they are complete.

OIDs are not explicited in **YAM**^{2} . They are considered as meaningless identifiers. The fact that several attributes identify instances is only of interest for **Cell**s, and it is shown in the form of several **Level**s forming a **Base**.

On the other hand, in **YAM**^{2} , relationships between **Level**s and **Cell**s are *Associations*, while they are *Aggregations* in GOLD. Thus, GOLD considers that a "fact class" is composed by its analysis dimensions. However, in **YAM**^{2} , **Dimension**s are simply used for the identification of **Cell** instances. Instances of **Level**s are not necessarily part of the **cell**s.

An important advantage of **YAM**^{2} , comes from the possibility of defining several **Cell**s inside the same **Star**. It allows to normalize the **Fact**s, so that attributes are fully functionally determined by the analysis dimensions. For instance, we can see in figure B.12 that `row_number` would not fully functionally determine `total_price`.

Finally, another important difference between both models is how aggregability is understood. While GOLD shows possible ways of aggregating **Measure**s, **YAM**^{2} shows how a **KindOfMeasure** must be aggregated along the **Dimension**s to obtain exactly the same **KindOfMeasure** at coarser granularity.

B.4 Clinical Data Warehousing

A clinical case of study is used in [Ped00] to motivate and exemplify his work. In this context, clinical data about patients is used to address quality management, and medical research issues. More specifically, a diabetes treatment domain is modeled.

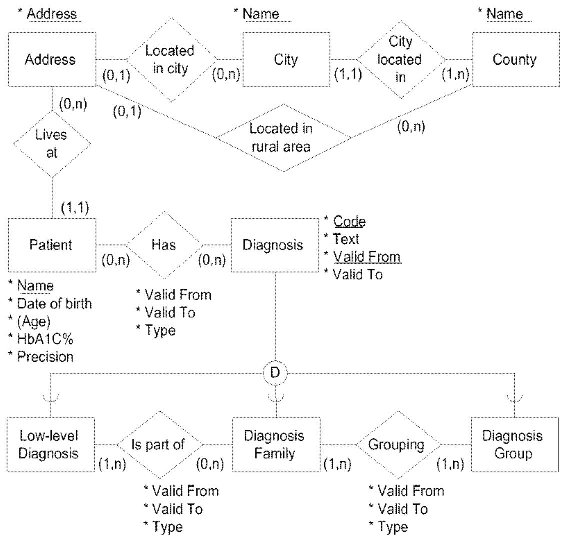


Figure B.17: Patient diagnosis case study [Ped00]

As shown in figure B.17, the system registers, for each patient, the blood sugar level (shown by `HbA1C%`), diagnosis, and place of residence. The variation of blood sugar levels among diagnoses, and the frequency of diagnoses per areas want to be studied. `Age` is a derived attribute (indicated by means of parenthesis), and `Precision` shows how precise the value of `HbA1C%` is. It admits three different values, i.e. `precise`, `imprecise`, and `inapplicable`.

`Diagnosis` represents a condition that a physician identifies in a patient. Every patient could have one or more diagnosis, and the time interval of validity of the diagnosis is also stored. The `Type` of a diagnosis indicates whether it is considered `primary` or `secondary`. There is only one `primary` diagnosis per patient.

The different kinds of diagnosis (i.e. `Low-level`, `Family`, and `Group`) show the different precision in diagnosing. The most precise is a `Low-level` diagnosis, and the least precise is the `Group` diagnosis. The diagnosis hierarchy is "non-strict", i.e. an element can be member of several collections at higher levels. The hierarchy evolves over time (new diseases are added, and old ones are reclassified), and is not "onto" (i.e. some families are not divided into `Low-level` diagnosis). Moreover, regarding addresses hierarchy, not every `Address` is located in a `City`. It is "non-covering".

A typical query in this domain is the average `HbA1C%` grouped by `Low-level` diagnosis.

B.4.1 Original schema

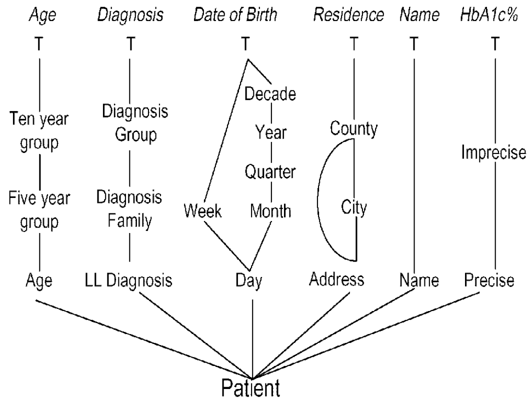


Figure B.18: Schema of the clinical case study [Ped00]

In [Ped00], everything that characterizes the fact type is considered to be "dimensional". Therefore, if `patient` is considered the fact, we obtain an schema like that in figure B.18. Notice that even **Measures** (i.e. `HbA1C%`) are considered "dimensional".

B.4.2 YAM^2 schema

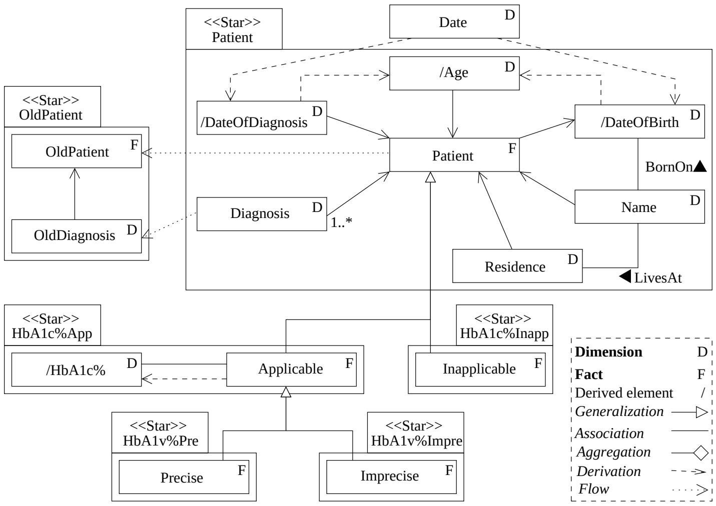


Figure B.19: **Upper** level schema of the clinical case study modeled with **YAM**^{2}

Figure B.19 shows the same schema in figure B.18, modeled with **YAM**^{2} , at **Upper** detail level. Here we can see the **Fact**s and **Dimension**s of interest in the domain. Notice that `Patient` has been specialized to show whether the long term blood sugar level (i.e. `HbA1C%`) has been measured or not, and if measured, whether a precise or imprecise method was used.

Possible changes in the diagnosis hierarchy have been represented by a *Flow* relationship between the old and new version of the **Dimension** and **Fact**. *Association*s between `Name`, and `DateOfBirth` and `Residence` are also shown.

Regarding derived information, both temporal **Dimension**s come from a more general temporal **Dimension**. Moreover, `HbA1C%` **Dimension** derives from the measurements of sugar level (if done). As in the original schema, `Age` is also derived from the `DateOfBirth` **Dimension**.

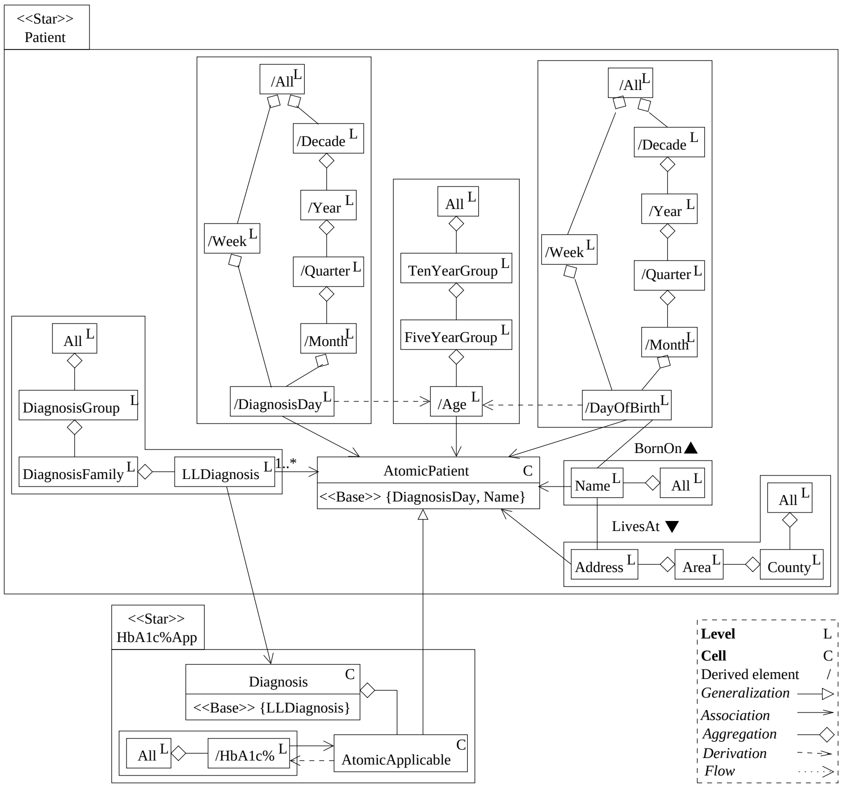


Figure B.20: **Intermediate** level schema of the clinical case study modeled with **YAM**^{2}

In figure B.20 the same schema at **Intermediate** detail level is shown. The same information at **Upper** level is now depicted with regard to **Level**s and **Cell**s. Moreover, it is also stated the **Base** of the **Cell**s. `AtomicPatient` is identified by `DiagnosisDay`, and a `Patient`; while `Diagnosis` is identified by `LLDiagnosis`. `Diagnosis` **Dimension** is defined as "non-strict".

For the sake of simplicity, some information has not been reflected at this level. For instance, there should be in the figure the common, more general **Dimension** for `DiagnosisDay`, and `DayOfBirth`; as well as the temporal evolution of `Diagnosis`. Moreover, the schema has been centered in `Patient` **Star**. Thus, `HbA1C%App` and other **Star**s have been neglected. For example, how **Cell**s in `HbA1C%App` are related to **Level**s in the different **Dimension**s have not been depicted. `Diagnosis` **Cell** has been explicited in the diagram to outline the importance of measuring the average `HbA1C%` at that **Level**.

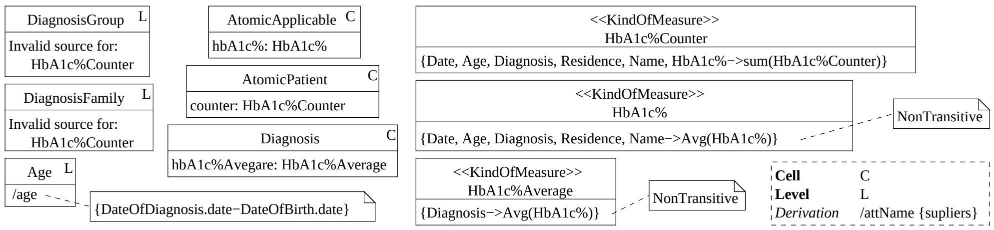


Figure B.21: **Lower** level schema of the clinical case study modeled with **YAM**^{2}

Finally, figure B.21 depicts the most detailed schema. Basically, it shows the *Attribute*s of the most representative **Class**es. We can see the *Attribute*s of `AtomicPatient`, how `age` is derived; `AtomicApplicable`, and `Diagnosis`; and the way to summarize the different kinds of measures. Non transitivity of summarizations has also been explicited.

B.4.3 Discussion

The first thing that attracts attention is that figure B.18 shows six analysis dimensions, while figure B.19 shows seven. This is due to the temporal mechanisms offered by Pedersen's model. These mechanisms allow to have the temporal dimension implicit in the schema. Nevertheless, **YAM**^{2} does not offer such mechanisms, so that `Date` **Dimension** needs to be explicited. If we consider that the temporal dimension is always present in analysis tasks, and it is well known, we can omit it. However, if we want to reflect the importance of that analysis dimension, and be able to define specific elements for it in every schema or view, it is much better to explicit it. Evolution of `Diagnosis` hierarchy can also be explicited in a **YAM**^{2} schema.

Another important point is that `Residence` **Dimension** becomes linear when modeled with **YAM**^{2} . This is because **YAM**^{2} does not allow "non-covering" hierarchies. Having instances skipping aggregation levels is not necessary. If some addresses do not belong to any city, all we need to do is define rural areas together with the urban ones, so that they cover the set of addresses.

Regarding `HbA1C%` **Dimension**, and the `Precision` *Attribute*, figure B.18 does not reflect reality. Grouping instances in different steps depending on the precision of measurements is not enough. Analysts should be able to study precise and imprecise measurements separately or not at will; and be able to know whether a given measurement is precise or not. Moreover, by defining subclasses of `Patient`, we show how an attribute is present in some instances and absent from others.

Derivation is a first class concept when modeling with **YAM**^{2} . While figure B.18 not even shows that `age` comes from a derived *Attribute*, with **YAM**^{2} we can explicit how it is obtained from other *Attributes*. Moreover, two **Dimension**s deriving from a common one can also be shown if appropriate.

[Ped00] just distinguishes three types of aggregate functions. One of those types is associated to every **Level** in the **Dimensions**. However, this information is not included in the schema. **YAM**^{2} allows to show the specific aggregation properties of every aggregation (for instance, transitivity, aggregability along a given analysis dimension, or the proper source level for the aggregation).

Another interesting point of **YAM**^{2} is that it provides mechanisms to reflect the importance of measures at different aggregation levels. For instance, in this case, figure B.18 does not reflect the importance of the average of `HbA1C%` at `LLDiagnosis`. Nevertheless, figure B.20 shows that there exists a **Cell** identified by `LLDiagnosis`, and B.21 zooms into that **Cell** to show that it contains an `HbA1C%Average` *Attribute*. More details about this aggregation and other is also reflected at **Lower** detail level.

To finish, just to say that a diagnosis being `primary` or `secondary` is neither shown in figure B.18, nor when using **YAM**^{2} . However, it can be easily modeled by means of Generalization/Specialization.

B.5 Vehicle repairs

Another interesting case study is that presented in [BSHD98], and [SBHD99]. It is a real world project with an industrial partner, where a car manufacturer wants to analyze the repairs of his vehicles to improve the products, define new warranty policies, and to assess the quality of the garages. Thus, he is interested in analyzing vehicle repairs based on the specific vehicle, garage where it is repaired, the day of the repair and the customer. Several measures are of interest, namely wages, part costs, total costs, duration of the repair, and number of persons that are involved.

B.5.1 Original schema

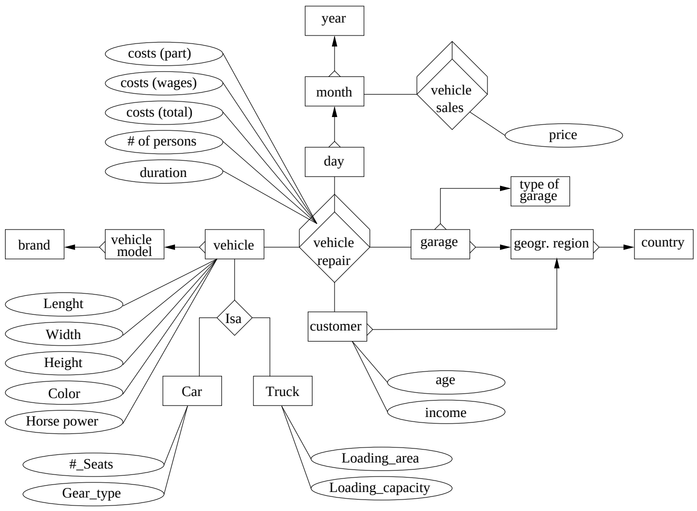


Figure B.22: Schema of the repairs case study [SBHD99]

Figure B.22 presents the data schema as in [SBHD99]. It is presented in "Multidimensional E/R" (ME/R) an extension of the E/R model. We can see that several **Fact**s (an specialization of Relationship) are allowed in the same schema (i.e. `vehicle repair` and `vehicle sales` in this case). Moreover, aggregation hierarchies are explicited, and they can be shared by different **Dimension**s (like `customer` and `garage`). Attributes that describe instances of **Dimension** (an specialization of Entity) but do not define aggregation hierarchies are also allowed. Specialization of concepts (like `vehicle` into `Car` and `Truck`) is also exemplified and justified by the existence of specific attributes.

Specific queries of interest to be possed on this schema are:

1. Give me the average total repair costs of a vehicle per month for garages in Bavaria by type of garage during the year 1997.

2. Give me the five vehicle types that had the highest average part cost per repair in the year 1997.

B.5.2 YAM^2 schema

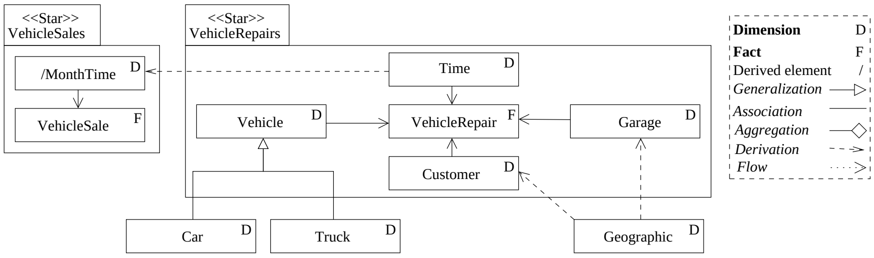


Figure B.23: **Upper** level schema of the repairs case study modeled with **YAM^2**

Figure B.23 shows the same schema at **Upper** detail level with **YAM^2** . Here, the two different subjects of analysis are separated into two **Star**s (i.e. `VehicleSales` and `VehicleRepairs`). The fact that both use the `Time` **Dimension** is show by means of a *Derivation* relationship (from the more general we derive the more specific). Moreover, at this level we can also observe that in the definition of `Garage` and `Customer` **Dimension**s geographic information was used in some way. Finally, specialization of `Vehicle` is also shown.

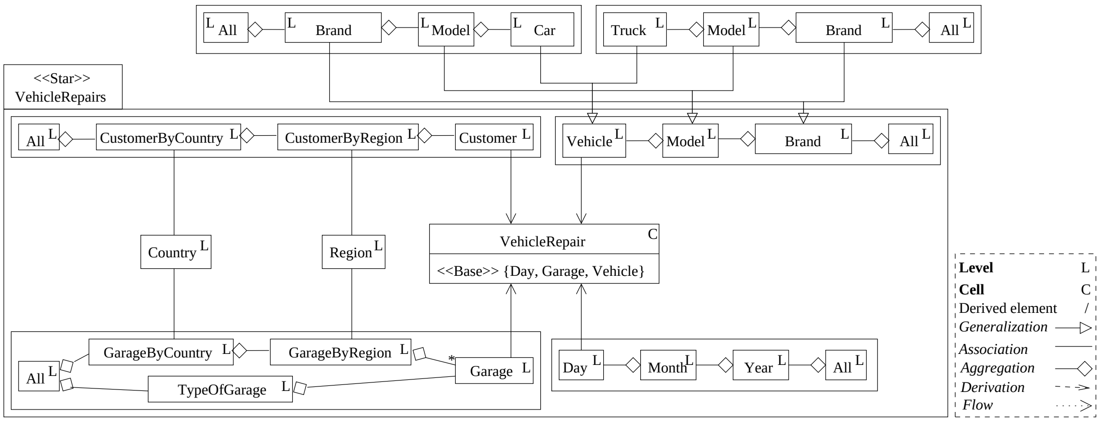


Figure B.24: **Intermediate** level schema of the repairs case study modeled with **YAM^2**

At **Intermediate** level, as shown in figure B.24, we can see the details of **Dimension**s and **Fact**s. Firstly, it is shown that **Level**s of specialized **Dimension**s (i.e. `Vehicle`) are also specialized into **Level**s of the corresponding **Dimension**s (i.e. `Car` and `Truck`). Moreover, geographic **Level**s in `Customer` and `Garage` are associated to **Level**s that should belong to a `Geographic` **Dimension**. Finally, it is also important to outline that the **Base** of the only **Cell** is composed just by three **Level**s (i.e. `Day`, `Garage`, and `Vehicle`). **Customer** could be used in analysis tasks, but it is determined by the other three **Level**s. Therefore, it would be a waste of space to use a 4-dimensional space to store instances of `VehicleRepair`.

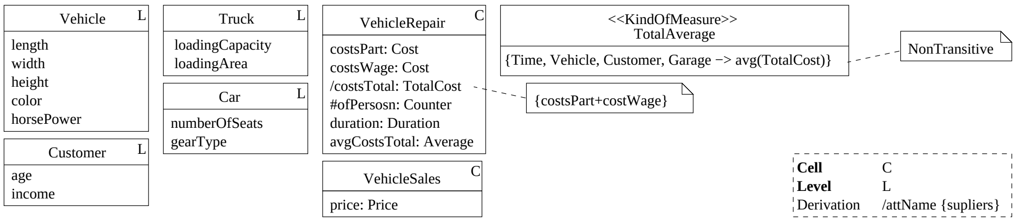


Figure B.25: **Lower** level schema of the repairs case study modeled with **YAM^2**

At the more detailed level, depicted in figure B.25, we see the *Attribute*s. Just to notice here that derivation of *Attribute*s is explicited.

Regarding the example queries, the first one would be solved by selecting repairs in 1997 in Bavaria, rolling them up to `Month` and `TypeOfGarage`, projecting `costsTotal` **Measure**, and placing data in a 2-dimensional space defined by `Month` and `TypeOfGarage`.

$$\gamma_{Month \times TypeOfGarage}(\pi_{avgCostsTotal}(\rho_{Month,TypeOfGarage}(\sigma_{year=1997\ AND\ region="Bavaria"}(VehicleRepair))))$$

The second query asks for "vehicle types", which does not correspond to the factual data in this schema. To be able to solve it, **Vehicle** should be considered a **Fact**, and average part repair costs should be a (derived) *attribute* of its associated **Dimension**.

B.5.3 Discussion

First of all, we see that, like in section B.3, several **Level**s can be shared between **Dimension**s. In [SBHD99], it is pointed out that in spite of that they are modeled together, the schema still contains two different **Dimension**s. Thus, at conceptual level, no redundant modeling of the shared **Level**s is necessary, and at later phases of design this can be used to avoid redundancies storing both **Dimension**s only once.

[BSHD98] explicitly explains the importance of having the definition of derived **Measure**s as part of the schema. However, in [SBHD99], it is specified that this cannot be included in this model, because like E/R, it is only able to reflect static structure of the application domain. There is no problem in **YAM^2** to show such information by means of UML notation in *Static Structure Diagrams*.

It is also said in [BSHD98] that the computation of aggregation functions might not be semantically meaningful for all **Measure**s, and it should be expressible in the conceptual model.

Nevertheless, it is not shown in figure B.22. It is shown at **Lower** detail level when modeling with $\mathbf{YAM}^2$.

Finally, another interesting issue, not reflected with ME/R, is the dependencies between **Dimension**s. $\mathbf{YAM}^2$ allows to show that, in this example, only three **Dimension**s are necessary to identify the facts.

Appendix C

List of publications

> "Every paper published in a respectable journal should have a preface by the author stating why he is publishing the article, and what value he sees in it. I have no hope that this practice will ever be adopted."
>
> Morris Kline

This appendix contains the publications that generated this thesis work, besides those that in one way or another also influenced it. Those that are closer to the subject are classified based on the chapters. Section C.5 contains those papers written during the elaboration of this thesis, that cannot be regarded as proper thesis work.

C.1 Related to chapter 2

- Alberto Abelló, José Samos, and Fèlix Saltor. A Framework for the Classification and Description of Multidimensional Data Models (©Springer-Verlag). In Proceedings of the *12th International Conference on Database and Expert Systems Applications (DEXA 2001)*. Munich (Germany), September 2001. Pages 668-677, Lecture Notes in Computer Science volume 2113. Springer, 2001. ISSN 0302-9743, ISBN 3-540-42527-6.

  > The words On-Line Analytical Processing bring together a set of tools, that use multidimensional modeling in the management of information to improve the decision making process. Lately, a lot of work has been devoted to modeling the multidimensional space. The aim of this paper is twofold. On one hand, it compiles and classifies some of that work with regard to the design phase they are used in. On the other hand, it allows to compare the different terminology used by each author, by placing all the terms in a common framework.

- Alberto Abelló, José Samos, and Fèlix Saltor. A Data Warehouse Multidimensional Data Models Classification. Technical Report LSI-2000-6. Departamento de Lenguajes y Sistemas Informáticos (Universidad de Granada), December 2000.

Extended version of the previous paper.

- Alberto Abelló, José Samos, and Fèlix Saltor. Benefits of an Object-Oriented Multidimensional Data Model (© Springer-Verlag). In Proceedings of the *Objects and Databases - International Symposium-* in *14th European Conference on Object-Oriented Programming (ECOOP 2000)*. Sophia Antipolis and Cannes (France), June 2000. Pages 141-152, Lecture Notes in Computer Science volume 1944. Springer, 2000. ISSN 0302-9743, ISBN 3-540-41664-1.

  In this paper, we try to outline the goodness of using an O-O model on designing multidimensional Data Marts. We argue that multidimensional modeling is lacking in semantics, which can be obtained by using the O-O paradigm. Some benefits that could be obtained by doing this are classified in six O-O Dimensions (i.e. Classification/Instantiation, Generalization/Specialization, Aggregation/Decomposition, Behavioural, Derivability, and Dynamicity), and exemplified with specific cases.

C.2 Related to chapter 3

- José Samos, Alberto Abelló, Marta Oliva, Elena Rodríguez, Fèlix Saltor, Jaume Sistac, Francisco Araque, Cecilia Delgado, Eladio Garví and Emilia Ruiz. Sistema Cooperativo para la Integración de Fuentes Heterogéneas de Información y Almacenes de Datos. In *Novatica*, 142 (Nov-Dec 1999), pages 44-49. Asociación de Técnicos de Informática (ATI), 1999. (In Spanish)

  This work presents our proposal for the creation of a prototype of cooperative systems for the integration of heterogeneous information sources and data warehouses, which is at the core of our research. The general goal is to provide a software layer that allow the cooperation among several information sources interconnected by means of a communication network. Each source owns its answer services to queries that regarding its data perform its users, and additionally, wants to offer to some users the opportunity of accessing the whole set of data in a uniform manner (integrated access), either in real time, or by means of the data warehouse.

- Alberto Abelló, Marta Oliva, José Samos, and Fèlix Saltor. Information System Architecture for Data Warehousing from a Federation. In Proceedings of the *International Workshop on Engineering Federated Information Systems (EFIS 2000)*. Dublin (Ireland), June 2000. Pages 33-40, IOS Press. ISBN 1-58603-075-2

  This paper is devoted to Data Warehousing schemas architecture and its data schemas. We relate a federated databases architecture to Data Warehouse schemas, which allows us to provide better understanding to the characteristics of every schema, as well as the way they should be defined. Because of the confidentiality of data used to make decisions, and the federated architecture used, we also pay attention to data protection.

- Alberto Abelló, Marta Oliva, José Samos, and Fèlix Saltor. Information System Architecture for Secure Data Warehousing. Technical Report LSI-00-26-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), April 2000.

  Extended version of the previous paper.

- Fèlix Saltor, Marta Oliva, Alberto Abelló, and José Samos. Building Secure Data Warehouse Schemas from Federated Information Systems. In Proceedings of the *International CODATA Conference on Data and Information for the Coming Knowledge Milenium (CODATA2000)*, Baveno (Italy), October 2000. (Extended abstract)

  There are similarities between architectures for Federated Information Systems and architectures for Data Warehousing. In the context of an integrated architecture for both Federated Information Systems and Data Warehousing, we discuss how additional schema levels provide security, and operations to convert from one level to the next.

- Alberto Abelló, José Samos, and Fèlix Saltor. Multi-star conceptual schemas for OLAP systems. Technical Report LSI-01-45-R of the Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2001.

  OLAP tools divide concepts based on whether they are used as analysis dimensions, or are the fact subject of analysis, which gives rise to star shape schemas. Operations are always provided to navigate inside such star schemas. However, the navigation among different stars uses to be forgotten. This paper studies different kinds of conceptual relationships between stars (i.e. Derivability, Generalization/Specialization, Aggregation/Decomposition, and Temporal), and proposes a 3-level schemas architecture to ease the implementation and usage of multi-star schemas.

C.3 Related to chapter 4

- Alberto Abelló, José Samos, and Fèlix Saltor. Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model. In Proceedings of the *3rd International Workshop on Design and Management of Data Warehouses (DMDW'2001)*. Interlaken (Switzerland), June 2001. SwissLife, ISSN 1424-4691.

  OLAP defines a set of data warehousing query tools characterized by providing a multidimensional view of data. Information can be shown at different aggregation levels (often called granularities) for each dimension. In this paper, we try to outline the benefits of understanding the relationships between those aggregation levels as Part-Whole relationships, and how it helps to address some semantic problems. Moreover, we propose the usage of other Object-Oriented constructs to keep as much semantics as possible in analysis dimensions.

- Alberto Abelló, José Samos, and Fèlix Saltor. Understanding Facts in a Multidimensional Object-Oriented Model (©ACM). In Proceedings of the *4th International Workshop on Data Warehousing and OLAP (DOLAP 2001)*. Atlanta (USA), November 2001. Pages 32-39. ACM Press, 2001. ISBN 1-58113-437-1.

  "On-Line Analytical Processing" tools are used to extract information from the "Data Warehouse" in order to help in the decision making process. These tools are based on multidimensional concepts, i.e. facts and dimensions. In this paper we study the meaning of facts, and the dependencies in multidimensional data. This study is used to find relationships between cubes (in an Object-Oriented framework) and explain navigation operations.

C.4 Related to chapter 5

- Alberto Abelló, José Samos, and Fèlix Saltor. **YAM**^{2} (Yet Another Multidimensional Model): An extension of UML. Technical Report LSI-01-43-R of the Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), 2001.

  This paper presents a multidimensional conceptual Object-Oriented model, its structures, integrity constraints and query operations. It has been developed as an extension of UML core metaclasses to facilitate its usage, as well as to avoid the introduction of completely new concepts. **YAM**^{2} allows the representation of several semantically related stars, as well as summarizability and identification constraints.

C.5 Other publications

- Alberto Abelló, Francisco Araque, José Samos, and Fèlix Saltor. Bases de Datos Federadas, Almacenes de Datos y Análisis Multidimensional. In *Taller de Almacenes de Datos y Tecnología OLAP* dentro de las *VI Jornadas de Ingeniería del Software y Bases de Datos (JISBD2001)*. Almagro (Spain), November 2001. (In Spanish)

  This pages present out work, in the BLOOM project of federated databases, regarding data warehousing and multidimensional analysis.

- Elena Rodríguez, Alberto Abelló, Marta Oliva, Fèlix Saltor, Cecilia Delgado, Eladio Garví and José Samos. On Operations along the Generalization/Specialization Dimension. In Proceedings of the *International 4th Workshop on Engineering Federated Information Systems (EFIS 2001)*. Berlin (Germany), October 2001. Pages 70-83, Infix. ISBN 3-89838-027-0

The need to derive a database schema from one or more existing schemas arises in Federated Database Systems as well as in other contexts. Operations used for this purpose include conforming operations, which change the form of a schema. In this paper we present a systematic approach to establish a set of primitive conforming operations that operate along the Generalization/Specialization dimension in the context of Object-Oriented schemas.

- Elena Rodríguez, Alberto Abelló, and Marta Oliva. Resumen del Simposium en Objetos y Bases de Datos del ECOOP'2000. In *Taller de Bases de Datos Orientadas a Objetos* dentro de las *V Jornadas de Ingeniería del Software y Bases de Datos (JISBD2000)*. Valladolid (Spain), November 2000. (In Spanish)

  The aim of this contribution is just to popularize the results of the *Symposium on Objects and Databases* held on June 13^{th} in Sophia-Antipolis (France) in conjunction with the 14^{th} *European Conference on Object-Oriented Programming* (ECOOP'2000). This event continued the (short) tradition established the year before in Lisbon (Portugal), where was held the first *Workshop on Object-Oriented Databases*.

- Alberto Abelló, and Elena Rodríguez. Describing BLOOM99 with regard to UML Semantics. In Proceedings of the *V Jornadas de Ingeniería del Software y Bases de Datos (JISBD2000)*. Valladolid (Spain), November 2000. Pages 307-319. Gráficas Andrés Martín, 2000. ISBN 84-8448-065-8.

  In this paper, we describe the BLOOM metaclasses with regard to the Unified Modeling Language (UML) semantics. We concentrate essentially on the Generalization/Specialization and Aggregation/Decomposition dimensions, because they are used to guide the integration process BLOOM was intended for. Here we focus on conceptual data modeling constructs that UML offers. In spite of UML provides much more abstractions than BLOOM, we will show that BLOOM still has some abstractions that UML does not. For some of these abstractions, we will sketch how UML can be extended to deal with this semantics that BLOOM adds.

- Alberto Abelló, Marta Oliva, Elena Rodríguez, and Fèlix Saltor. The syntax of BLOOM99 schemas. Technical Report LSI-99-34-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), July 1999.

  The BLOOM (BarceLona Object Oriented Model) data model was developed to be the Canonical Data Model (CDM) of a Federated Database Management System prototype. Its design satisfies the features that a data model should have to be suitable as a CDM. The initial version of the model (BLOOM91) has evolved into the present version, BLOOM99. This report specifies the syntax of the schema definition language of BLOOM99. In our model, a schema is a set of classes, related through two dimensions: the generalization/specialization dimension, and the aggregation/decomposition dimension. BLOOM supports several features in each of these dimensions, through their corresponding metaclasses.

Even if users are supposed to define and modify schemas in an interactive way, using a Graphical User Interface, a linear schema definition language is clearly needed. Syntax diagrams are used in this report to specify the language; an alternative using grammar productions appears as Appendix A. A possible graphical notation is given in Appendix B. A comprehensive running example illustrates the model, the language and its syntax, and the graphical notation.

- Alberto Abelló, Marta Oliva, Elena Rodríguez, and Fèlix Saltor. The BLOOM model revisited: An evolution proposal (poster session). In Workshop Reader of the *13th European Conference on Object-Oriented Programming (ECOOP'99)*. Lisbon, June 1999. Pages 376-378, Springer-Verlag, Lecture Notes in Computer Science volume 1743, Springer, 2000. ISBN 3-540-66954-X

  The growing need to share information among several autonomous and heterogeneous data sources has became an active research area. A possible solution is providing integrated access through a *Federated Information System* (FIS). In order to provide integrated access, it is necessary to overcome semantic heterogeneities, and represent related concepts. This is accomplished through an integration process in which a *Canonical Data Model* (CDM) plays a central role.

  Once argued the desirable characteristics of a suitable CDM, the BLOOM model (BarceLona Object Oriented Model) was progressively defined. Recently, we have revised the BLOOM model giving rise to BLOOM99. We discuss the change reasons and the main innovations that BLOOM99 includes.

- Alberto Abelló. CORBA: A middleware for an heterogeneous cooperative system. Technical Report LSI-99-21-R. Departament de Llenguatges i Sistemes Informàtics (Universitat Politècnica de Catalunya), May 1999.

  Two kinds of heterogeneities interfere with the integration of different information sources, those in systems and those in semantics. They generate different problems and require different solutions. This paper tries to separate them by proposing the usage of a distinct tool for each one (i.e. CORBA and BLOOM respectively), and analizing how they could collaborate. CORBA offers lots of ways to deal with distributed objects and their potential needs, while BLOOM takes care of the semantic heterogeneities. Therefore, it seems promising to handle the system heterogeneities by wrapping the components of the BLOOM execution architecture into CORBA objects.

- Alberto Abelló, and Fèlix Saltor. Implementation of the BLOOM data model on ObjectStore. Technical Report LSI-99-7-T. Departament de Llenguatges i Sistemes Informàtics

(Universitat Politècnica de Catalunya), May 1999.

BLOOM is a semantically enriched Object-Oriented data model. It offers extra semantic abstractions to better represent the real world. Those abstractions are not implemented in any commercial product. This paper explains how all them could be simulated with a software layer on an Object-Oriented database management system. Concretely, it proved to work on ObjectStore.

Glossary

Association As explained in [OMG01b], defines a semantic relationship between *Classifiers* (see figure 5.3, in page 97). The instances of an *Association* are a set of tuples relating instances of the *Classifiers*.

Aggregation As explained in [OMG01b], a kind of *Association* relationship so that one end is part of the other (see figure 5.3, in page 97).

Attribute As explained in [OMG01b], a named slot within a *Classifier* that describes a range of values that instances of the *Classifier* may hold.

BLOOM BarceLona Object-Oriented Model. It was conceived as a semantically rich O-O model to be used to overcome semantic heterogeneities in the integration process of a FIS.

Canonical Data Model Common model used to overcome the heterogeneities in the different data models of the CDBs in a federation.

CASE Computer Aided Software Engineering.

CDB See "Component Database".

CDM See "Canonical Data Model".

Cell (i.e. class of **cell**s) contains those **cell**s representing the same kind of fact and being associated with instances of the same **Level** for each of the **Dimension**s we use to analyze it (see pages 81 and 83).

CIF See "Corporate Information Factory".

Class As defined in [OMG01b], a description of a set of objects that share some *Attribute*s, *Operation*s, *Method*s, *Relationships*, and semantics.

Classifier As defined in [OMG01b], an element that describes behavioral and structural features; it comes in several specific forms, including *Class*, *DataType*, *Interface*, *Component*, and others (see figure 5.10, in page 112).

CMDS See "Corporate Multidimensional Schema".

Common Warehouse Metamodel As explained in [OMG01a], a metadata standard, which purpose is to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms, and warehouse metadata repositories in distributed heterogeneous environments.

Component Database Each database participating in a FIS.

Conceptual model Data model close to the way users perceive data, and independent of the implementation (see page 14).

Corporate Information Factory Data Warehousing architecture defined in [IIS98] (see figure 1.1 in page 2).

Corporate Multidimensional Schema Intermediate level of a 3-levels architecture for the management of multidimensional data (see figure 3.10, in page 55).

Cube An injective function from an n-dimensional finite space (defined by the cartesian product of n functionally independent **Level**s), to the set of instances of a **Cell** (see page 88).

CWM See "Common Warehouse Metamodel".

Data Mart As defined in [IIS98], a collection of data tailored to the "Decision Support Systems" processing needs of a particular department (see page 2).

Data Warehouse As it was defined in [Inm96], an integrated, subject-oriented, historic, and non-volatile set of data in support for the decision making process (see page 1).

Data Warehousing As it was defined in [Gar98], a process, not a product, for assembling and managing data from various sources for the purpose of gaining a single, detailed view of part or all of a business (see page 1).

Data Cube A metaphor that represents how analysts conceive data (see page 10).

DB Database.

DBMS Database Management System.

Derivation As defined in [OMG01b], a kind of relationship which specifies that the client may be computed from the supplier (see figure 5.3, in page 97).

Descriptor An attribute of a **Level**, used to select its instances (see page 95).

Dimension A connected, directed graph representing a point of view on analyzing data. Every vertex in the graph corresponds to an aggregation level, and an edge reflects that every instance at target **Level** decomposes into a collection of instances of source **Level** (i.e. edges reflect part-whole relationships between instances of **Level**s) (see page 74).

DM See "Data Mart".

DW See "Data Warehouse".

E/R Entity-Relationship.

Expressiveness As it is defined in [SCG91], the degree to which a model can express or represent a conception of the real world (see page 34).

Fact A a connected, directed graph representing a subject of analysis. Every vertex in the graph corresponds to a **Cell**, and an edge reflects that every instance at target **Cell** decomposes into a collection of instances of source **Cell** (i.e. edges reflect part-whole relationships between instances of **Cell**s) (see page 83).

FD Functional Dependency.

FIS See "Federated Information System".

Federated Information System A collection of cooperating but autonomous component systems.

Flow As explained in [OMG01b], a relationship between two versions of an object (see figure 5.3, in page 97).

Generalization As explained in [OMG01b], a taxonomic relationship between a more general element and a more specific element (see figure 5.3, in page 97).

Hypercube See "Data Cube".

Intermediate Detail level that contains *Class*es, i.e. **Cell**s and **Level**s (see page 15).

Key As defined in [AHV95], a minimal "superkey".

Level Represents the set of instances of the same granularity in an analysis dimension (see page 95).

Logical model A data model providing concepts that can be understood by end users, but that are not too far removed from the way data is organized within the computer (see page 14).

Lower Detail level that contains *Attribute*s, i.e. **Measure**s and **Descriptor**s (see page 15).

Measure An attribute of a **Cell** representing measured data to be analyzed (see page 96).

Measurement Act of measuring. Each instance of **Measure**.

Mereology The science that studies part-whole relationships.

MOLAP See "Multidimensional OLAP".

Multidimensional OLAP Pure multidimensional DBMS.

Nexus Any kind of semantic relationship between two objects (see page 35).

O3LAP See "Object-Oriented OLAP".

Object Constraint Language A formal language to express side-effect-free constraints, defined in [OMG01b].

Object-Oriented OLAP Object-Oriented DBMS adapted for OLAP.

Object Query Language Query language of the ODMG (Object Data Management Group) data model.

OCL See Object Constraint Language.

ODS See "Operational Data Store".

OLTP On-Line Transactional Processing.

OLAP See "On-Line Analytical Processing".

On-Line Analytical Processing As defined in [OLA97], a category of software technology that enables analysts, managers and executives to gain insight into data through fast, consistent, interactive access to a wide variety of possible views of information that has been transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user (see page 2).

O-O Object-Oriented.

OO-Dimension Each one of the six dimensions of the O-O paradigm identified in [Sal96] (see section 2.4).

Operational Data Store As defined in [IIB96], an architectural construct that is subject-oriented, integrated, volatile, current-valued and contains only corporate detailed data (see page 2).

OQL See "Object Query Language".

Physical model A data model tightly coupled to the specific DBMS used, and conceived to describe how data is actually stored (see page 14).

ROLAP See "Relational OLAP".

Relational OLAP Relational DBMS adapted for OLAP.

Semantic Domain Domain reflecting the conceptualization of values in the mind of the designer.

Semantic Power See "Expressiveness".

Semantic Relativism As is defined in [SCG91], the degree to which a model can accommodate not only one, but many different conceptions (see page 40).

Slice and Dice As defined in [OLA97], the user-initiated process of navigating by calling for page displays interactively, through the specification of slices via rotations and drill down/up (see page 11).

SQL Structured Query Language.

Star A modeling element composed by a **Fact**, and several **Dimension**s that can be used to analyze it (see page 97).

Superkey As defined in [AHV95], a subset of the attributes of a Relation that functionally determines it.

Transaction Time As defined in [DGK+94], is the time when a fact is current in a database and may be retrieved.

UML See "Unified Modeling Language".

Unified Modeling Language As said in [OMG01b], provides a consistent language for specifying, visualizing, constructing and documenting the artifacts of software systems.

UoD Universe of Discourse.

Upper Detail level that contains *Classifier*s, i.e. **Fact**s and **Dimension**s (see page 15).

Valid Time As it is defined in [DGK+94], the time when a fact is true in the modeled reality.