International Journal of Cooperative Information Systems
© World Scientific Publishing Company

AN ARCHITECTURE FOR DATA WAREHOUSING SUPPORTING DATA INDEPENDENCE AND INTEROPERABILITY[*]

LUCA CABIBBO
*DIA, Università di Roma Tre, Via della Vasca Navale 79
00146 Roma, Italy. Email: cabibbo@dia.uniroma3.it*

RICCARDO TORLONE
*DIA, Università di Roma Tre, Via della Vasca Navale 79
00146 Roma, Italy. Email: torlone@dia.uniroma3.it*

Received (to be inserted
Revised by Publisher)

We report on the design of a novel architecture for data warehousing based on the introduction of an explicit "logical" layer to the traditional data warehousing framework. This layer serves to guarantee a complete independence of OLAP applications from the physical storage structure of the data warehouse and thus allows users and applications to manipulate multidimensional data ignoring implementation details. For example, it makes possible the modification of the data warehouse organization (e.g., MOLAP or ROLAP implementation, star scheme or snowflake scheme structure) without influencing the high level description of multidimensional data and programs that use the data. Also, it supports the integration of multidimensional data stored in heterogeneous OLAP servers. We propose $\mathcal{MD}$, a simple data model for multidimensional databases, as the reference for the logical layer. $\mathcal{MD}$ provides an abstract formalism to describe the basic concepts that can be found in any OLAP system (fact, dimension, level of aggregation, and measure). We show that $\mathcal{MD}$ databases can be implemented in both relational and multidimensional storage systems. We also show that $\mathcal{MD}$ can be profitably used in OLAP applications as front-end. We finally describe the design of a practical system that supports the above logical architecture; this system is used to show in practice how the architecture we propose can hide implementation details and provides a support for interoperability between different and possibly heterogeneous data warehouse applications.

Keywords: Data warehousing; Data analysis; Multidimensional data model; Data independence; Logical layer; Interoperability

1. Introduction

During the past decade, the separation of the enterprise information architecture into two separate environments has quickly become popular. Alongside the traditional On Line Transaction Processing (OLTP) environment, a new On Line Analytical Processing (OLAP) component has been introduced, dedicated to decision-oriented analysis of historical data. The central element of the new environment is

[*]This work was partially supported by *IASI-CNR* and by *MURST*.

the *data warehouse*, a read-only archive of historical snapshots of operational data, rearranged into a multidimensional format more suitable for decision support.[18,21] In general, the term *data warehousing* indicates the various activities involved in the construction, maintenance, and use of this information architecture component, which we will therefore call *data warehousing system*.[6]

Nowadays, current technology provides a lot of software tools supporting data warehousing.[27] Apart a number of facilities implementing specific activities (extraction and filtering of source data, refreshing of the data warehouse, etc.), the main components of a data warehousing system are the *data warehouse server* and the front-end clients, often called *OLAP tools*. Data warehouse servers can be relational (ROLAP), multidimensional (MOLAP), or hybrid systems. ROLAP systems store a data warehouse in relational tables and maps multidimensional operations to extended SQL statements. A quite standard organization of a data warehouse in a ROLAP system is the *star scheme* (or variant thereof), where a central table represents the fact on which the analysis is focused, and a number of tables, only partially normalized, represent the dimensions of analysis (e.g., time, location, type).[21] Conversely, MOLAP systems are special servers that directly store and manipulate data in multidimensional proprietary formats. In hybrid OLAP systems data can be stored in both relational and multidimensional form. Front-end tools offer powerful querying and reporting capabilities, usually based on interactive graphical user interfaces similar to spreadsheets. Standard Application Program Interfaces (API) for communication between data warehouse server and clients are also emerging (e.g., OLE DB for OLAP and MD-API).[24,27]

Although the available tools are powerful and very useful, we believe that there is a limitation in the traditional organization of a data warehousing system. The problem is that, very often, the way in which information is viewed and manipulated in an OLAP tool strictly depends on how data is stored in the server. For instance, in a ROLAP implementation, the user often needs to refer to a large collection of tables that hardly capture the "logical" multidimensional aspects of a data warehousing application. Several OLAP tools somehow filter the underlying technology, by providing a high-level graphical interface, but they are usually customized to work with a specific data warehouse server or class of servers (relational or multidimensional). Also and more important, if the organization of the data warehouse changes, there is often the need to rebuild the front-end views of the data warehouse. For example, if an OLAP tool provides a multidimensional view of a relational star scheme in the form of a spreadsheet and the scheme is just normalized (or denormalized) for efficiency reasons, the view is no more valid and needs to be rebuilt. Another consequence of the dependency of OLAP applications from their implementation is that the integration of data coming from different data warehousing applications is difficult to achieve, since no standardization exists in this context. This need arises in large companies where various data warehouses are developed independently or when different companies merge or get involved in a federated project.

In this paper we illustrate an approach aimed at solving this problem. The basic idea is to add an explicit level of abstraction to the traditional data warehousing framework. The new level, which we call "logical," serves to describe in abstract terms the multidimensional aspects of an OLAP application, and to guarantee an independence of the application from the physical storage structure of the data warehouse. This is similar to what happens with relational technology, in which the property of data independence allows users and applications to manipulate tables and views ignoring implementation details.

We also present $\mathcal{MD}$, a logical data model for multidimensional databases, which can be profitably used as the reference data model for the logical level. The $\mathcal{MD}$ model provides a number of constructs used to describe, in an abstract but natural way, the basic notions that can be found in almost every OLAP system (fact, dimension, level of aggregation, and measure). We show how an $\mathcal{MD}$ database can be implemented using a relational database or a multidimensional structure. We also show how an $\mathcal{MD}$ database can be easily queried, according to different user needs. This serves to demonstrate how the use of a logical model simplifies the development of OLAP applications.

We finally describe the design of a system, called MDS, that supports the above logical architecture. MDS is able to provide an $\mathcal{MD}$ view of one or more existing data warehouses (which can be stored in a variety of systems) and to map queries and data between these environments. The system relies on a collection of *Data Warehouse Interfaces* (DWI) which actually implement the correspondence between the data warehouses and their $\mathcal{MD}$ view. When a data reorganization occurs in a server, we just need to adapt the corresponding DWI to the new situation, without changing query and view definitions at the user level. It turns out that DWI *templates* can be defined for supporting the construction of new DWIs.

The rest of the paper is organized as follows. In Section 2 we give a general overview of the approach and in Section 3 we present the $\mathcal{MD}$ model. In Section 4 we show how $\mathcal{MD}$ databases can be practically implemented. Different database languages for the manipulation of $\mathcal{MD}$ databases are illustrated in Section 5. Section 6 is devoted to the presentation of the system we have designed. In Section 7 we relate our work to the relevant literature and finally, in Section 8, we draw some conclusions.

2. A Logical Approach to Data Warehousing

The traditional architecture for data warehousing is shown in Figure 1. According to Inmon,[18] four different levels can be identified.

- *Operational layer.* At the bottom level we have the operational sources data that are often heterogeneous and distributed. The data warehouse is populated from these data sources, through cleaning, filtering, and integration operations.

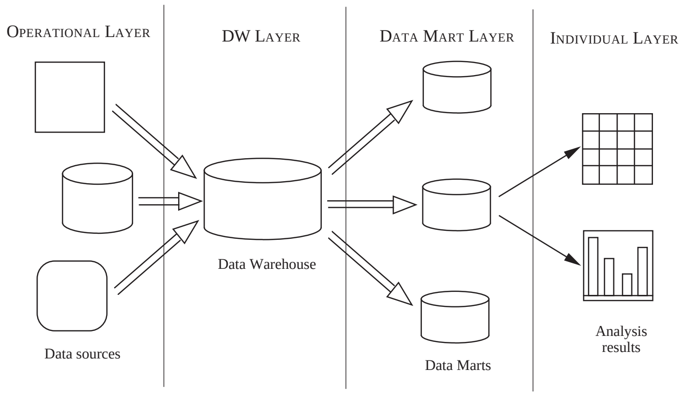


Fig. 1. The traditional architecture for data warehousing.

- *Data Warehouse layer.* The data warehouse lies at this level and is kept separate from the operational sources. Data is represented here either in relational form (ROLAP systems) or in multidimensional proprietary form (MOLAP systems).
- *Data mart layer.* A data mart is generally built from a selected portion of the data warehouse for studying a specific, well-bounded business problem.
- *Individual layer.* This level corresponds to the final form in which data is presented to analysts for interpretation. Various types of graphical representations are used here.

Variants of this organization are possible. In particular, either the data warehouse or the data mart layer can be absent. However, there is always an intermediate level of persistent data between the operational and the individual layers and, for simplicity, we will refer to this level as to the *data warehouse level* hereinafter.

As we have said in the Introduction, a major problem in the traditional architecture is that, in many cases, data is viewed and manipulated at the individual level in a way that strictly depends on the data warehouse level. As a consequence, changes to the organization of the data warehouse are propagated to the views defined on it, and thus to the external level. Moreover, the interoperability between different data warehouses must be solved case by case.

A possible solution to alleviate this problem is the introduction of an explicit "logical" level in the data warehousing architecture. The new level serves to hide details of the actual implementation of the data warehouse and to concentrate on describing the basic, multidimensional aspects of data that are employed in analytical processing. Specifically, as shown in Figure 2, the data warehouse level is split into a *logical layer*, which describes the content of the data warehouse in multidimensional but abstract terms, and an *internal layer*, which describes how

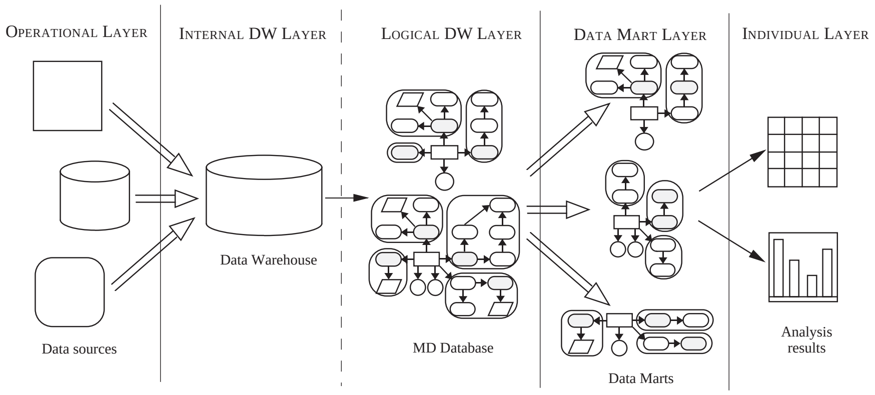


Fig. 2. A new architecture for data warehousing.

the data warehouse is implemented (relational or multidimensional structures and their organization). Queries and views are defined with respect to the logical data warehouse layer. According to this organization, data warehouse restructuring operations occur at the internal level and just require the modification of the mapping between the internal and the logical layer; we show that this task can be performed automatically. In this way we guarantee that the logical representation of the data warehouse remains always the same and so queries and views defined on it.

We believe that this architecture has a number of further advantages:

- analysts can manipulate data easier since they can refer to a high-level representation of data;
- interoperability between different data warehousing systems is facilitated, since the new level provides an abstract, and thus unifying, description of data;
- the system can scale-up easily, for instance in case of addition of a new source of data, since this requires, in most cases, just a change of the data warehouse description at the internal level.

3. The $\mathcal{MD}$ Data Model

We now present a data model for multidimensional databases, which can be profitably used as the reference data model for the logical layer in the architecture we propose.

The MultiDimensional data model ($\mathcal{MD}$ for short) is based on a few constructs modeling the basic concepts that can be found in any OLAP system.[2] This model has two main constructs: dimension and f-table. *Dimensions* are syntactical categories that allow the designer to specify multiple "ways" to look at the information, according to natural business perspectives under which data analysis can be performed. For instance, a retail company may organize its business data along dimensions like **time**, **product**, and **location**. Each dimension is organized in a hierarchy of *levels*, corresponding to data domains at different granularity. For example, the **time** dimension may be organized in levels day, month, quarter, and year. A level can have *descriptions* associated with it, which provide further information about it. For instance, a description of the level store in the **location** dimension, can be its *address*. Within a dimension, values of different levels are related through a family of *roll-up functions*. A roll-up function can state that the element *Feb 19, 1999* of the level day is associated with the element *Feb-1999* in the level month. Finally, *F-tables* are used to represent factual data and associate a *measure* with a combination of values over a fixed set of levels, called *symbolic coordinate*. For example, an f-table SALES can be used to store the number of sales of our company for each item (level of the product dimension), store (level of the location dimension), and day (level of the time dimension).

Actually, the "f" in the term f-table has a double meaning. On one hand, it stands for "function," because each f-table is indeed a function, from coordinates to measures. On the other hand, it stands for "fact," since it represents what, in this context, is usually called *fact table*.

We now formally define the $\mathcal{MD}$ model. We fix two disjoint countable sets of *names* and *values*, and denote by $\mathcal{L}$ a set of names called *levels*. Each level $\ell \in \mathcal{L}$ is associated with a countable set of values, called the *domain of* ℓ and denoted by $\textsc{dom}(\ell)$. The various domains are pairwise disjoint.

Definition 1 (Dimension). An $\mathcal{MD}$ *dimension* consists of:

- a finite set of *levels* $L \subseteq \mathcal{L}$;
- a partial order $\preceq$ on the levels in L — whenever $\ell_1 \preceq \ell_2$ we say that ℓ_1 *rolls up to* ℓ_2;
- a family of *roll-up functions*, including a function $\textsc{r-up}_{\ell_1}^{\ell_2}$ from $\textsc{dom}(\ell_1)$ to $\textsc{dom}(\ell_2)$ for each pair of levels $\ell_1 \preceq \ell_2$ — whenever $\textsc{r-up}_{\ell_1}^{\ell_2}(o_1) = o_2$ we say that o_1 *rolls up to* o_2.

A dimension with just one level is called *atomic*. For the sake of simplicity, we will not make any distinction between an atomic dimension and its unique level.

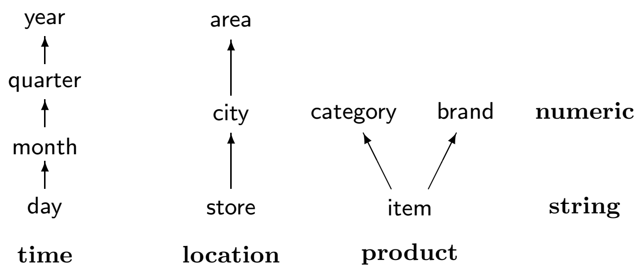
Definition 2 (Scheme). An $\mathcal{MD}$ *scheme* consists of:

- a finite set D of *dimensions*;
- a finite set F of *f-table schemes* of the form

$$f[A_1 : \ell_1, \ldots, A_n : \ell_n] \to [M_1 : \ell'_1, \ldots, M_m : \ell'_m],$$

  where f is a name, the A_is are names called *attributes of* f, the M_js are names called *measures of* f, and the ℓ_is and the ℓ'_js are levels in D.

Dimensions



F-tables

$\textsc{Sales}[\mathit{Period} : \mathsf{day}, \mathit{Product} : \mathsf{item}, \mathit{Location} : \mathsf{store}] \rightarrow [\mathit{NSales} : \mathsf{numeric}, \mathit{Income} : \mathsf{numeric}]$

$\textsc{CostOfItem}[\mathit{Product} : \mathsf{item}, \mathit{Month} : \mathsf{month}] \rightarrow [\mathit{Cost} : \mathsf{numeric}]$

Level descriptions

$\mathit{Address}\ (\mathsf{store}) : \mathsf{string}, \quad \mathit{Name}\ (\mathsf{item}) : \mathsf{string}$

Fig. 3. The sample $\mathcal{MD}$ scheme `Retail`.

- a finite set Δ of *level descriptions* of the form $\delta(\ell) : \ell'$, where ℓ and ℓ' are levels of a dimension in D and δ is a name called *description of* ℓ.

Example 1. Consider the $\mathcal{MD}$ scheme `Retail`, shown in Figure 3. This database refers to a data warehousing application for a chain of stores and is organized along dimensions **time**, **product**, and **location** (shown on top of the figure). The **time** dimension is organized in a hierarchy of levels involving day, month, quarter, and year. The domain associated with the level day contains, for instance, values *Jan 5, 1999*, *Feb 19, 1999*, and *Mar 10, 1999*, all of which roll up to the element *1Q-1999* of the level quarter. Similarly, the **location** dimension is based on a hierarchy of levels involving store, city, and area. The level store contains, for instance, values *Colosseum* and *Navona*, both of them rolling up to *Rome* (in level city) and *Italy* (in level area). A description of the level store, in the **location** dimension, can be its *address*. Finally, the **product** dimension contains levels item, category, and brand. According to the corresponding hierarchy, any element of the level item rolls up to both a brand and a category. Note that there are two further atomic dimensions that are used to represent **numeric** values and **string**s.

In this framework, two f-tables have been defined: Sales and CostOfItem. The former describes summary data for the sales of the chain, organized along dimensions **time** (at day level), **product** (at item level), and **location** (at store level). The measures for this f-table are *NSales* (the number of items sold) and *Income* (the gross income), both of type numeric. The f-table CostOfItem is used to represents the costs of the various items, assuming that costs may vary from month-to-month.

SALES

Period	*Product*	*Location*	*NSales*	*Income*
Jan 5, 1999	*Scrabble*	*Navona*	32	543.68
Jan 5, 1999	*Risiko*	*Navona*	27	512.73
Jan 5, 1999	*Lego*	*Sun City*	42	713.58
Jan 5, 1999	*Risiko*	*Sun City*	22	439.78
Feb 19, 1999	*Scrabble*	*Navona*	32	479.68
Feb 19, 1999	*Scrabble*	*Atomium*	26	422.90
Feb 19, 1999	*Lego*	*Navona*	25	299.75
Feb 19, 1999	*Lego*	*Colosseum*	11	142.89
Mar 10, 1999	*Risiko*	*Navona*	5	69.95
Mar 10, 1999	*Lego*	*Sun City*	6	71.94

COSTOFITEM

Cost	*Jan-1999*	*Feb-1999*	*Mar-1999*
Lego	12.99	9.99	9.99
Risiko	14.99	12.99	12.99
Scrabble	12.99	12.99	12.49
Trivia		18.99	17.99

Description of store

store	*Address*
Navona	*Piazza Navona, 8*
Colosseum	*Via dell'Impero, 55*
Sun City	*Via Condotti, 67*

Fig. 4. An $\mathcal{MD}$ instance for the `Retail` database.

Instances can be defined over f-tables as follows.

Definition 3 (Symbolic coordinate and instance). A *(symbolic) coordinate* over an f-table scheme $f[A_1 : \ell_1, \ldots, A_n : \ell_n] \rightarrow [M_1 : \ell'_1, \ldots, M_m : \ell'_m]$ is a function mapping each attribute name A_i to an element in $\textsc{dom}(\ell_i)$. An *instance over* f is a function that maps coordinates over f to tuples over $[M_1 : \ell'_1, \ldots, M_m : \ell'_m]$. An *instance* over a level description $\delta(\ell) : \ell'$ in Δ is a function from $\textsc{dom}(\ell)$ to $\textsc{dom}(\ell')$. We call *entry* of an f-table instance f a coordinate over which f is defined.

Example 2. A possible instance for the `Retail` scheme is shown in Figure 4. Note that different (graphical) representations can be used for an f-table instance: a table and an array. This depends on the fact that an f-table is an abstract entity (specifically, a function). A symbolic coordinate over the f-table SALES is [day : *Jan 5, 1999*, item : *Scrabble*, store : *Navona*]. The actual instance associates with this entry the measure 32 for *NSales* and the measure 543.68 for *Income*. The description *Address* associates the string *Via Condotti, 67* with the value *Sun City* of level store.

It is apparent that our notion of "symbolic coordinate" is related with that of "tuple" in the relational model. It can also be noted that the notation we use for symbolic coordinates resembles subscripting into a multi-dimensional array (although in a non-positional way). There is however an important difference between f-tables and multi-dimensional arrays. Specifically, in arrays, "physical" coordinates vary over intervals within linearly-ordered domains, whereas we do not pose any restrictive hypothesis on the domains over which coordinates range. In this sense, our notion of coordinate is "symbolic."

Roll-up functions are a distinctive feature of our model: they describe *intensionally* how values of different levels are related. Such a description is indeed independent of any effective implementation, which can be based on stored relations, built-in functions, or external procedures. As we will show shortly, roll-up functions also provide a powerful tool for querying multidimensional data, since they allow us to specify how data must be aggregated, and how f-tables involving data at different levels of granularity can be joined.

4. Implementation of $\mathcal{MD}$ Databases

In this section we show how an $\mathcal{MD}$ database can be practically implemented, using a relational database (as in ROLAP systems) or a set of multidimensional arrays (as in MOLAP systems).

4.1. Relational databases

The natural representation of a multidimensional database in the relational model consists of a collection of "fact" and "dimension" tables. The former are normalized, whereas the latter can be denormalized. Since there exist several different definitions of star schemes, we refer in the following to a basic formulation.[18,21] The approach can however be easily adapted to variants of this model (e.g., the snowflake scheme).

A mapping between an $\mathcal{MD}$ database and a relational star scheme can be defined as follows. We have: (i) a relation scheme R_d for each non-atomic dimension d, and (ii) a relation scheme R_f for each f-table f. The atomic dimensions do not need to be represented since they generally correspond to basic domains.

- R_d contains an attribute A_ℓ for each level ℓ occurring in d, an attribute A_δ for each description δ of a level in d, and an attribute A_d denoting a (generated) key for R_d.
- R_f contains an attribute A_i for each attribute A_i of f over a level ℓ of a dimension d, whose domain coincides with the domain of the key A_d of the relation R_d; R_f also contains an attribute M_j for each measure M_j of f. The concatenation of the A_i's is a key for R_f.

The corresponding instances are defined as follows.

- The relation R_d contains a tuple t_v for each value v of each level ℓ occurring in d. The tuple t_v is defined as follows:
  - $t_v.A_d$ is a unique identifier k_v for the value v and $t_v.A_\ell = v$;
  - for each description δ of ℓ, $t_v.A_\delta = \delta(v)$;
  - for each level ℓ' to which ℓ rolls up, $t_v.A_{\ell'} = \textsc{r-up}_\ell^{\ell'}(v)$, and
  - for each description δ' of ℓ', $t_v.A_{\delta'} = \delta'(\textsc{r-up}_\ell^{\ell'}(v))$.

  The other attributes carry nulls.

Dimension relations:
Location(*l-id*, *store*, *address*, *city*, *area*)
Product(*p-id*, *item*, *category*, *brand*)
Time(*t-id*, *date*, *month*, *quarter*, *year*)

Fact relations:
Sales(*p-id*, *t-id*, *l-id*, *nsales*, *income*)
CostOfItem(*p-id*, *t-id*, *cost*)

Location

l-id	*area*	*city*	*store*	*address*
...	...	...	...	...
l_1	North	Venice	La Gondola	Rialto
l_2	North	Milan	Boys 'R Us	P. Cordusio
l_3	Center	Rome	Sun City	P. Navona
...	...	...	...	...
l_{100}	North	Venice	*null*	*null*
l_{101}	North	Milan	*null*	*null*
...	...	...	...	...
l_{1000}	North	*null*	*null*	*null*
...	...	...	...	...

Product

p-id	*item*	*category*	*brand*
...	...	...	...
p_{43}	Trivia	Toy	Micro
...	...	...	...

Sales

p-id	*t-id*	*l-id*	*nsales*	*income*
...	...	...	...	...
p_{43}	t_{99}	l_2	25	245.95
...	...	...	...	...

CostOfItem

p-id	*t-id*	*cost*
...	...	...
p_{43}	t_{504}	6.50
...	...	...

Time

t-id	*date*	*month*	*quarter*	*year*
...	...	...	...	...
t_{99}	Apr 1, 1999	Apr-1999	2Q-1999	1999
...	...	...	...	...
t_{504}	*null*	Apr-1999	2Q-1999	1999
...	...	...	...	...

Fig. 5. Star scheme representation of the `Retail` database.

- The relation R_f contains a tuple t_e for each entry e of f. If e equals $[A_1 : v_1, \ldots, A_n : v_n]$ and $[M_1 : v'_1, \ldots, M_m : v'_m]$ are the corresponding measures, then:
  - for each attribute A_i, $t_e.A_i = k_{v_i}$, and
  - for each measure M_j, $t_e.M_j = v'_j$.

  Note that a value v in the entry is represented by k_v (which is a key for the dimension relation identifying v), rather than by v itself.

As an example, the star scheme representation of the `Retail` database is shown in Figure 5. Note that the instance is just outlined. The relation Location is more detailed to show the structure of a dimension table.

The scheme so obtained can be optimized in several ways. For instance, relation schemes corresponding to different f-tables over the same levels can be merged, suitable indexes can be defined, and some views involving aggregation can be materialized.[6]

4.2. Multidimensional arrays

We now briefly outline how an $\mathcal{MD}$ database can be represented by means of multidimensional arrays. Since there is no agreed model for MOLAP systems, we assume that factual data are represented by means of matrices whose indexes range over contiguous, initial segments of the natural numbers.

First of all, for each dimension d of our $\mathcal{MD}$ scheme we define a bijection β_d assigning a unique integer to each value of each level in d. More specifically, if m is the number of those values, β_d associates with each of them an integer varying from 0 to $m - 1$ (and vice versa). In this way, we obtain a one-to-one correspondence between symbolic and numeric coordinates. Then, an f-table $f[A_1 : \ell_1, \ldots, A_n : \ell_n] \rightarrow [M_1 : \ell'_1, \ldots, M_m : \ell'_m]$ is represented by a n-dimensional matrix, storing each tuple of measures, corresponding to the symbolic entry $[A_1 : v_1, \ldots, A_n : v_n]$, in the cell having physical coordinate $[\beta_{d_1}(v_1), \ldots, \beta_{d_n}(v_n)]$.

An $\mathcal{MD}$ dimension can be represented by means of a special data structure, with a hierarchical organization according to the partial order between the levels. This data structure is used to store both the roll-up functions and the assignment between values of levels and integers. We can then use this structure as an index to access the multidimensional arrays. The resulting scheme can be tuned by using the tools provided by the specific storage system chosen.[11]

5. Paradigms for Data Analysis

A flexible OLAP system should provide a family of query languages, possibly using different paradigms. On one hand, the non-expert user should be enabled to perform point-and-click operations by means of graphical metaphors. On the other hand, the sophisticated user that needs to express more complex queries should be allowed to use a declarative textual language, such as an extension of SQL, since graphical languages have often limited expressive power. Finally, query optimization can be effectively performed by using a procedural, algebraic language, involving a number of atomic operators that can be implemented efficiently.

In this section, we discuss the use of these categories of languages in the context of the $\mathcal{MD}$ model, making use of the `Retail` database, introduced in Section 3, as an example. This serves mainly to show how $\mathcal{MD}$ can be profitably used as front-end to multidimensional data. More details on query languages for $\mathcal{MD}$ can be found in previous works of the authors.[2,4]

5.1. Graphical languages

Typical ways of manipulating a multidimensional data collection are: *roll up* (summarize data), *drill down* (go to more detailed data), *slice-and-dice* (select and project on a bidimensional view), and *pivot* (reorient a data cube representation, projecting on some dimension).[6] These simple operations can be easily expressed in a graphical manner with reference to an $\mathcal{MD}$ scheme.[4]

Figure 6 shows an example of a graphical query over the `Retail` database. The query specifies an aggregation over the f-table SALES, which stores the number of sales for each item, store, and day. Initially, we assume that the dimensions relevant to this f-table, and the granularity of data are shown to the user (gray

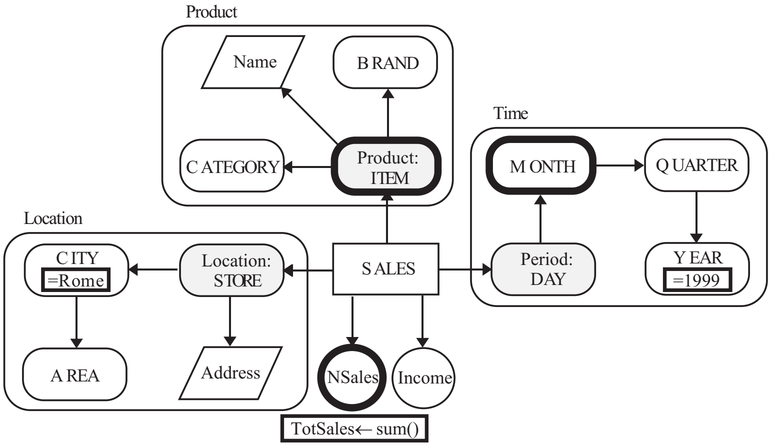


Fig. 6. A graphical query over the `Retail` database.

boxes). Then, the user can select different levels within these dimensions (boxes with bold margins) to specify how data must be aggregated (roll up operation). Specific measures are selected in the same way. Constant values can be used to denote selections (*Rome* for city and *1999* for year). Note that, since no level is selected for the **location** dimension, this dimension does not belong to the result of the query (this corresponds to a slice-and-dice operation). Finally, a label on the selected measure specify the aggregation function that we intend to apply. We obtain in this way a bidimensional f-table representing the number of items sold in Rome during 1999, for each item and month.

A richer set of metaphors can enable the user to express more complex operations. For example, the juxtaposition of boxes can be used to join data in different f-tables. Also, views can be defined and queries (possibly involving aggregations) on views are allowed.

5.2. Declarative languages

A more powerful language is needed to express complex queries, for instance those involving multiple f-tables. To this end, we have introduced a calculus-based query language for $\mathcal{MD}$.[2] This language enables the user to express cross-dimensional analytical equations, based on logical expressions over f-tables, in a simple and declarative way. Roll-up functions are directly used within the language to specify selections, to perform aggregations, and to join data at different levels of granularity.

As an example, consider the following query over the `Retail` database. The query computes the f-table PROFIT storing, for each store and month, the net revenue as the difference between total income and cost of the items sold.

$$\Big\{store, month : nr \mid nr = \mathtt{sum}\Big(day, item : dr \mid \exists n, i, c(\ [n, i] = \textsc{Sales}[day, item, store] \ \wedge \\ month = \textsc{r-up}_{\mathsf{day}}^{\mathsf{month}}(day) \wedge \ [c] = \textsc{CostOfItem}[month, item] \ \wedge \ dr = i - n * c\Big)\Big\}$$

The identifiers *store, month, day*, and *item* denote variables ranging over the corresponding levels; the others denote numeric variables. Intuitively, the expression specifies that, for each *store* and *month*, the total net revenue *nr* is given by the sum of the daily net revenues *dr*, for each *item* and *day* in the *month*. The inner expression specifies that the quantity *dr* is given (for a given *day*, *store* and *item*) by the difference between the total income i and the product of the cost c by the number n of items sold. A roll-up function is used to state the relationship between a month and the corresponding days.

The $\mathcal{MD}$ calculus is a query language having a formal syntax and semantics, but is clearly unsuited for a direct use. However, we can easily derive a practical language from it, more suitable for the final user. For example, the above query could be expressed, in a textual, SQL-like query language, as follows.

```
SELECT store, month : netRev
WHERE  netRev = SUM OF ( SELECT day, item : dailyRev
                         WHERE  i = Sales[day,item,store].income
                         AND    n = Sales[day,item,store].nsales
                         AND    c = CostOfItem[month,item].cost
                         AND    day ROLLS UP TO month
                         AND    dailyRev = i - n*c )
```

5.3. Procedural languages

As it is customary in relational database systems, it is useful to translate user queries into an internal procedural language, suitable to perform optimization and directly implementable. We have therefore defined an algebra for f-tables, which is similar in spirit to other proposals.[1,15] The operators of this algebra can be directly implemented or can be mapped to statements of a language (e.g., SQL or API) for an existing database server (a relational, a ROLAP, or a MOLAP system).

Table 1 briefly describes the main operators of the $\mathcal{MD}$ algebra.[4] Apart from a number of operators that are variants of relational algebra operators, there are three new operators. Two of them allow the application of scalar (φ) and aggregative functions (ψ) on attributes of an f-table. The other operator (ϱ) is specific for the $\mathcal{MD}$ model: it allows the application of a roll-up function $\textsc{r-up}_{\ell}^{\ell'}$ to an f-table f having an attribute A defined on the level ℓ. This operator extends the scheme of f with a new attribute A on ℓ' holding, for each entry t of f, the result of the application of $\textsc{r-up}_{\ell}^{\ell'}$ to the value of t on A.

The query computing the f-table $\textsc{Profit}$ introduced in Section 5.2 can be specified by means of the following $\mathcal{MD}$ algebra expression.

$$\psi_{Product,Month}^{netRev=sum(dr)}(\varphi^{dr=income-nsales*cost}(\\ \varrho_{period:day}^{month}(\textsc{Sales})\bowtie_{Product,Month}\textsc{CostOfItem}))$$

Table 1. Operators of the $\mathcal{MD}$ algebra.

Operator	**Description**
σ_C	Selection: it selects f-table entries on the basis of the condition C
π_X	Simple projection: it projects an f-table over a subset X of its attributes and measures
$\bowtie_X$	Join: it combines entries of two f-tables that are equal over the set X of common attributes
$\varphi^{M=f(\dots)}$	Scalar operator: it computes the scalar function f and puts the result in the new measure M
$\psi_X^{N=g(\dots)}$	Projection with aggregation: it computes the aggregation g while projecting over the set X of attributes, and put the result in the new measure N
$\varrho_\ell^{\ell'}$	Roll up operator: it extends the coordinates with a new level

The operator ϱ in the expression denotes the application of a roll-up function, to extend the coordinates of the f-table Sales with a new attribute *month* corresponding to the *day* of the sale. The natural join operator $\bowtie$ is used to combine the f-table so obtained and the f-table CostOfItem, according to the values over the attributes *Product* and *Month*. The operator φ introduces to the result f-table the measure *dr* (the daily revenue) computed from the other measures according to the specified scalar function. Finally, the aggregate operator ψ computes the sum of the measure *dr* of the f-table we have obtained, grouping the result by the attributes *Product* and *Month*.

We have shown that the $\mathcal{MD}$ algebra is equivalent to the $\mathcal{MD}$ calculus and more powerful than the $\mathcal{MD}$ graphical language. We have also shown that it is possible to automatically translate graphical and calculus queries into algebraic expressions.[5]

6. Design of the System Architecture

On the basis of the issues discussed in previous sections, we have designed an environment for data warehousing called MDS (MultiDimensional System). The general organization of MDS is reported in Figure 7. The system provides a uniform view of multidimensional data (according to the $\mathcal{MD}$ model) coming from multiple and possibly heterogeneous data warehouses managed by systems that we call generically *Data Warehouse Management Systems*. This task is charged to a collection of *Data Warehouse Interfaces* (DWI), which perform the needed data transformation.

Integrated access to several warehouse systems is important in many practical situations. For instance, in large companies where various data warehouses are developed independently or when different companies merge or get involved in a federated project.

6.1. MDS *users*

Before examining the main components of MDS, it is useful to discuss the classes of users involved with the various activities (see Figure 7).

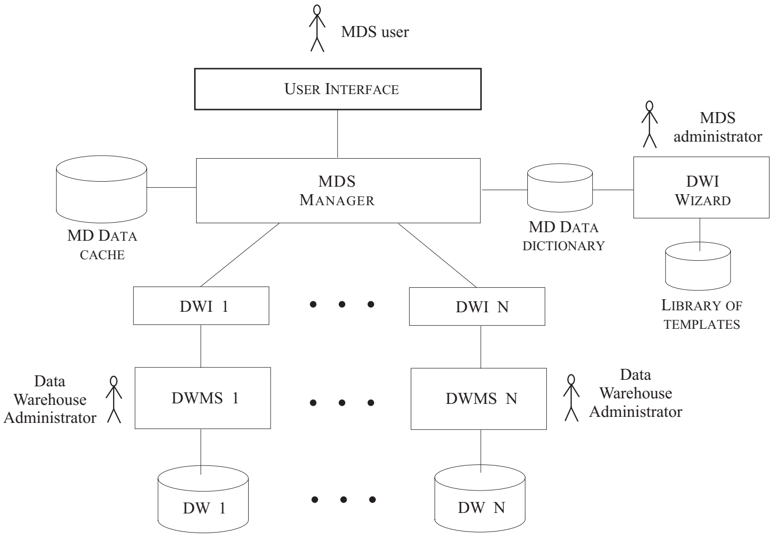


Fig. 7. The MDS system environment.

- The main user of MDS is a traditional analyst (or *end user*), interested in specifying analytical queries on data extracted from one or more data warehouses (or data marts) ignoring implementation details. As we have said, this task is implemented in MDS by special software components called Data Warehouse Interfaces (DWI).

- The construction and maintenance of a DWI is performed by a special user, called MDS *administrator*, who has the responsibility of setting and possibly extending the environment. As we will clarify shortly, his or her work is supported by an external facility (called DWI *wizard*) that allows the creation and modification of a DWI.

- Finally, we envision the role of a *data warehouse administrator*, responsible for building and maintaining a data warehouse accessed by MDS. He or she could ignore the representation of the data warehouse provided by MDS, but must communicate any change in the data warehouse organization to the MDS administrator.

Clearly, the various tasks (and thus the various user classes) can overlap.

6.2. The MDS *structure*

Let us now give a look at the structure of MDS. The overall framework is reported in Figure 7 and consists of the components described below. In the figure, a box represents a software module, a cylinder represents a repository of data, and a line denotes a flow of information.

- The whole environment is loosely coupled with one or more data warehouses managed by external Data Warehouse Management Systems. These systems can be, in principle, OLAP servers (relational or multidimensional) or, more in general, generic data storage systems. The only hypothesis we make is that the multidimensional aspects of data analysis can be identified in the representation of the data warehouse provided by each system.

- The *Data Warehouse Interfaces* (DWI) perform two important functions: (i) they realize the mapping between an $\mathcal{MD}$ database scheme and the actual organization of a data warehouse, and (ii) they convert $\mathcal{MD}$ queries into operations understandable by the underlying data warehouse management systems.

- The *DWI Wizard* supports the MDS Manager with the construction of a DWI. This tool is separated from the MDS Manager (see below) but integrated with it. The DWI Wizard makes use of a library of DWI *templates* that refer to a specific data model used in a data warehouse management system, but to a generic database for this model (this will be clarified shortly). The DWI Wizard generates a DWI from a DWI template on the basis of the actual data warehouse scheme, as specified by the MDS Manager.

- The MDS *Manager* is the core of the system and performs several tasks: (i) it imports the scheme of a data warehouse from a data storage system through the corresponding DWI and stores it into a local $\mathcal{MD}$ Data Dictionary; (ii) it receives from a client $\mathcal{MD}$ queries (in graphical or textual form) against a stored $\mathcal{MD}$ scheme, translates them into an algebraic format, and sends the outcome to the corresponding DWIs for execution; (iii) it receives the result of a query from one or more DWIs, possibly combines them, and then gives the final result to the client for its visualization; (iv) it receives the request for a creation of a view and stores its definition in the $\mathcal{MD}$ Data Dictionary. Note that data integration is in charge of the MDS Manager which acts as a mediator over the DWIs, which in turn act as wrappers over underlying DWMSs. Efficient query evaluation is supported by the management of a local cache of $\mathcal{MD}$ data. Caching is based on a subsumption relationship between f-tables. This relationship can be easily defined according to the hierarchy of levels defined in an $\mathcal{MD}$ scheme. The system stores in the cache computed aggregations of basic f-tables. When a new query involves an aggregation of a basic f-table, the system uses the subsumption relationship to check whether there is an f-table in the cache that can be used to answer it efficiently.

- The *User Interface* allows the interaction with the system on the client side through both textual and graphical tools. The work of the user is supported by a number of menus and forms.

From the viewpoint of the end user, MDS offers the following operations.

- Visualization of the scheme of an external data warehouse in terms of the $\mathcal{MD}$ model; this scheme can be used as reference for querying.
- Specification of textual and graphical queries against available $\mathcal{MD}$ schemes and views.
- Definition of views, which can be later used in specifying queries.
- Visualization of the result of a query in tabular form.[a]

6.3. Creation and management of DWIs

In order to access the first time an existing data warehouse d having scheme S of a model M and stored in a system $DWMS$, a new DWI I_S for S needs to be created.

Basically, such DWI has three components:

1. a mapping Θ, which relates concepts of S and concepts of a corresponding $\mathcal{MD}$ scheme $\widehat{S}$;
2. a query translation algorithm, which converts a query over $\widehat{S}$ into a query over S that $DWMS$ can understand; and
3. a data translation algorithm, which transforms the results of queries and views over the scheme S into $\mathcal{MD}$ instances.

The mapping Θ describes how each $\mathcal{MD}$ concept in $\widehat{S}$ is represented by concepts in S in the underlying system. For instance, if the data model M supported by the underlying system $DWMS$ is the relational model, then concepts in $\widehat{S}$ are related to relations, attributes, and constraints in S. More specifically, a relational mapping over a star schema with one fact table and several dimensional tables associates: (i) an $\mathcal{MD}$ level with an attribute of a dimensional table, (ii) a roll-up function with one or more foreign keys (which indicate a path over the relational tables to compute the roll-up function), (iii) an $\mathcal{MD}$ f-table with a fact table, (iv) coordinates of an $\mathcal{MD}$ f-table with attributes and foreign keys from a fact table to a dimensional table, and (v) measures of an $\mathcal{MD}$ f-table with attributes of a fact table.

Both the algorithms used in a DWI make use of the mapping Θ and depend only on the features of $DWMS$. Therefore, once $DWMS$, and thus M, has been fixed, the only component that needs to be changed in the evolution of the data warehouse is the mapping Θ. The idea of DWI template is actually based on this consideration: we can fix the algorithms and obtain a new DWI for a scheme of M by just modifying Θ, as long as the organization of the data warehouse changes.

[a] We point out that visualization, although important in this context, is not central in our research.

The construction of the DWI I_S is in charge of the MDS administrator that is supported in this work by the DWI wizard. As described above, this operation is based on the use of the DWI template for the model M, if one exists in a special library. Indeed, even if there is no standardization in this framework, it often happens that several systems adopt the same multidimensional model. In this case, the same DWI template can be used to access several data warehouse servers. We follow here an "extensible" approach, according to which DWI templates are added gradually to the library as long as new systems need to be accessed.

Once the DWI I_S has been built, the scheme S can be imported in MDS, translated into the scheme $\widehat{S}$ of the $\mathcal{MD}$ model, and stored locally in the $\mathcal{MD}$ Data Dictionary. The system is now ready to access d and allows the end user to specify queries and views over $\widehat{S}$. If the scheme S is modified (because of a reorganization or an extension of d), we just need to change I_S, without any need to change $\widehat{S}$ and the views defined on it. The integration of several data sources, possibly stored in different systems, can be achieved by simply defining views on their $\mathcal{MD}$ representation.

7. Related Work

A lot of research has been recently done on the subject of OLAP and data warehousing. The term OLAP has been introduced by Codd et al. to characterize the category of analytical processing over a large, historical database (the data warehouse) oriented to decision making.[7] A comprehensive discussion on OLAP, multidimensional analysis, and data warehousing has been provided by Chaudury and Dayal.[6] There are also several Web sites dedicated to this subject.[27] An up-to-date on-line research-oriented bibliography is maintained by Mendelzon.[23]

The general reference architecture for data warehousing described in Section 2 has been defined in the seminal book by Inmon.[18] The traditional organization of a data warehouse is based on the notion of star scheme or variants thereof (snowflake, star constellation, and so on).[18,21] More recent works on data warehousing architecture investigate minor variants of the above architecture and are more focused on supporting special techniques for improving the performance of data warehousing systems.[9] For example, the selection of materialized views and the construction of special indexes to make query evaluation more efficient. Samos et al. propose a data warehousing architecture defined as an extension of a federated database architecture.[28] In our approach we introduce an explicit logical layer, which allows analysts to view and manipulate multidimensional data independently of the data warehouse organization and structure. The addition of a logical level to achieve data independence has been advocated also within the DWQ project,[19,20] where a conceptual level is also introduced, and by other authors.[14,29]

The $\mathcal{MD}$ data model illustrated in this paper has been used in other works as a basis to the investigation of several issues related to multidimensional database analysis.[2,3,4,5,8,16] In earlier works of ours, we have introduced a number of query languages and studied their expressiveness,[2,4,5] and we have proposed a system-independent design methodology for multidimensional databases.[3] The present paper is focused on a comprehensive coverage of the $\mathcal{MD}$ model and on the design of an extensible data warehousing system, independent of the organization of the data warehouse, based on such model. We have used $\mathcal{MD}$ here just because it is a simple and abstract model. Clearly other data models with the same characteristics could have been used.

Several data models for multidimensional databases have been proposed in the literature.[31] Many of them are focused on the development of querying and restructuring languages rather than data modeling. Agrawal et al. have proposed a framework for studying multidimensional databases, consisting of a pragmatic data model based on a notion of multidimensional cube, and an algebraic query language.[1] Gyssens and Lakshmanan have defined a logical model for multidimensional databases, in which the contents are clearly separated from structural aspects.[15] Gebhardt et al. proposed a tape-based model of multidimensional data, together with an operational framework for visualization and querying that makes use of operations on tapes (e.g., scrolling and intersection) as graphical metaphors.[13] This framework yields a way to build worksheets from operational sources and define their visualization. Lehner proposes a nested multi-dimensional data model, based on two orthogonal structuring mechanisms for dimensions: classification hierarchies and features, and an algebra for data manipulation.[22] Vassiliadis provides a rather complex data model which has some similarities with ours, and a number of basic operations that form an algebra.[30] Golfarelli et al. have proposed a graphical conceptual model for DWs, called Dimensional Fact Model (DFM), based on a notion of fact scheme, whose basic elements are facts, dimensions and hierarchies.[14] Extensions of the Entity Relational model (ME/R) including multidimensional aspects have been also proposed.[10,29] We believe that our approach differs from the above proposals in two major aspects. First, our model is based on a purely abstract representation of multidimensional data, which does not refer to any specific concept (cubes, tapes, entities and so on). Second, we provide within the data model an explicit and abstract representation of the way in which data can be aggregated. This is usually implemented in other approaches through operations on data.

The aspect of manipulating multidimensional databases is covered by commercial OLAP systems in a pragmatic way, by providing powerful OLAP tools that communicate, in many cases, with traditional database engines. Notable examples are MetaCube and Oracle Express, which use as back-ends major database management systems.[17,26] Standard API specifications between OLAP clients and data warehouse servers, such as OLE BD for OLAP and MD-API, are now emerging.[24,25] They allow the communication between heterogeneous systems and provide a mean for making the individual layer (at the client site) independent of the internal layer (at the server site). However, they still have several disadvantages. First, the maintenance of the mapping between the internal to the external layer is often in charge to the programmer. We have provided a technique that automatically accomplishes this task. Moreover, OLE DB for OLAP lacks a solid theoretical background and blurs presentational and computational issues.[31] On the other hand, MD-API is not equipped with a declarative query facility.

We finally note that the proposed architecture for data warehousing somehow resembles the mediator based approach to the integration of heterogeneous information sources.[32] Specifically, the DWIs can be viewed as wrappers and the MDS Manager acts as a mediator. Our context is however much more specific than that of general mediation systems, in which a light-weight model is usually adopted at the integrated level.[12] For this reason we can use a richer data model in which the multidimensional aspects of data sources can be directly described. This makes the task of integration easier to achieve.

8. Conclusion

We have described a new architecture for data warehousing that introduces a new level of abstraction to the traditional data warehousing framework. The new level serves to guarantee an independence of OLAP applications from the physical storage structure of the data warehouse. This property allows users and applications to manipulate multidimensional data ignoring implementation details. Moreover, it facilitates interoperability between different data warehousing systems. We have proposed $\mathcal{MD}$, a simple data model for multidimensional databases, as the reference for the logical layer. We have then designed a system, called MDS, that supports the above logical architecture. MDS is able to view an existing data warehouse, which can be stored in a variety of systems, in terms of the $\mathcal{MD}$ model and to map queries and data between the two frameworks.

A first prototype of MDS has been designed in Java and is able to communicate with an Oracle data warehouse server. Since the corresponding DWI template produces standard SQL statements, it can be used to generate DWIs for other relational database management systems. We have also designed further DWI templates; among them, a template for the OLE DB for OLAP API, which can be used for coupling MDS with several available data warehouse servers.

References

1. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Thirteenth Int. Conf. on Data Engineering*, pages 232–243, 1997.
2. L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Sixth Int. Workshop on Database Programming Languages (DBPL'97), Springer-Verlag*, 1997.
3. L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. In *Sixth Int. Conference on Extending Database Technology (EDBT'98), Springer-Verlag*, pag. 183–197, 1998.
4. L. Cabibbo and R. Torlone. From a procedural to a visual query language for OLAP. In *Tenth Int. Conference on Scientific and Statistical Database Management (SS-DBM'98)*, IEEE Computer Society Press, pag. 74–83, 1998.

5. L. Cabibbo and R. Torlone. A logical framework for querying multidimensional data. *Manuscript*, 2000.
6. S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.
7. E.F. Codd, S.B. Codd, and C.T. Salley. Providing OLAP (On Line Analytical Processing) to user-analysts: An IT mandate. Arbor Software White Paper, *http://www.arborsoft.com*.
8. S. Dekeyser, B. Kuijpers, J. Paredaens, and J. Wijsen. Nested data cubes for OLAP. In *Int. Workshop on Data Warehousing & Data Mining, Singapore, Springer-Verlag*, pag. 129–140, 1998.
9. J. M. Firestone. Architectural evolution in datawarehousing and distributed knowledge management architecture. White Paper, Executive Information Systems, Inc. *http://www.dkms.com/ARCHEV.htm*. 1998
10. E. Franconi and U. Sattler. A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. In *Int. Workshop on Design and Management of Data Warehouses*, 1999.
11. P. Furtado and P. Baumann. Storage of multidimensional arrays based on arbitrary tiling. In *Fifteenth Int. Conf. on Data Engineering*, pages 480–489, 1999.
12. H. Garcia-Molina et al. The TSIMMIS approach to mediation: data models and languages. *Journal of Intellingent Information Systems*, 8:117–132, 1997.
13. M. Gebhardt, M. Jarke, and S. Jacobs. A toolkit for negotiation support interfaces to multi-dimensional data. In *ACM SIGMOD Int. Conf. on Manag. of Data*, pag. 348–356, 1997.
14. M. Golfarelli, D. Maio, and S. Rizzi. The Dimensional Fact Model: a conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7(2-3):215–247, 1998
15. M. Gyssens and L.V.S. Lakshmanan. A foundation for multi-dimensional databases. In *Twenty-third Int. Conf. on Very Large Data Bases, Athens*, pages 106–115, 1997.
16. C. Hurtado, A. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. In *Fifteenth Int. Conf. on Data Engineering*, pages 346–355, 1999.
17. Informix Software Inc. The Informix-MetaCube product suite. *http://www.informix.com*.
18. W. H. Inmon. *Building the data warehouse*. John Wiley & Sons, second edition, 1996.
19. M. Jarke et. al. Concept Based Design of Data Warehouses: The DWQ Demonstrators. *ACM SIGMOD Int. Conf. on Manag. of Data*, pag. 591, 2000.
20. M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis (eds.). *Fundamentals of data warehousing*. Springer-Verlag, 2000.
21. R. Kimball. *The data warehouse toolkit*. John Wiley & Sons, 1996.
22. W. Lehner. Modeling Large Scale OLAP Scenarios. In *Sixth Int. Conference on Extending Database Technology (EDBT'98), Springer-Verlag*, pag. 153–167, 1998.
23. A. O. Mendelzon. Data warehousing and OLAP: a research-oriented bibliography. *http://www.cs.toronto.edu/∼mendel/dwbib.html*.
24. Microsoft Corporation. OLE DB for OLAP. *http://www.microsoft.com/*, 1999.
25. OLAP Council. MDAPI: the OLAP Application Program Interface. *http://www.olapcouncil.org/*, 1998.
26. Oracle Corporation. Oracle OLAP products: adding value to a data warehouse. White Paper, *http://www.oracle.com*.
27. N. Pendse and R. Creeth. The OLAP report. *http://www.olapreport.com*.
28. J. Samos, F. Saltor, J. Sistac, and A. Bardés. Database architecture for data ware-

housing: an evolutionary approach. In *Database and Expert Systems Applications (DEXA'98)*, pages 746–756, 1998.

29. C. Sapia, M. Blaschka, G. Höfling, and B. Dinter: Extending the E/R Model for the Multidimensional Paradigm, In *International Workshop on Data Warehouse and Data Mining, Springer-Verlag*, pag. 105–116, 1998.
30. P. Vassiliadis. Modeling multidimensional databases, cubes and cube operations. In *Tenth Int. Conference on Scientific and Statistical Database Management (SS-DBM'98)*, IEEE Computer Society Press, pag. 53–62, 1998.
31. P. Vassiliadis and T. Sellis. A survey of logical models for OLAP databases. *ACM SIGMOD Record*, 28(4):64–69, December 1999.
32. G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.