ERATOSTHENES: Design and Architecture of an OLAP* System

Nikos Karayannidis, Panos Vassiliadis, Aris Tsois, Timos Sellis

National Technical University of Athens (NTUA)
Department of Electrical and Computer Engineering
Knowledge and Database Systems Laboratory
Zografou 15773, Athens, Greece
{nikos, pvassil, atsois, timos}@dbnet.ece.ntua.gr

Abstract. On-Line Analytical Processing (OLAP) is a trend in database technology, based on the multidimensional view of data. The aim of this paper is twofold: (a) to list general problems and solutions applicable to the design of any OLAP system and (b) to present the specific design decisions that we made for a prototype under development at NTUA, which we call ERATOSTHENES. The paper addresses requirements and design issues for all three models involved in an OLAP system: the presentational, logical and physical model. It also discusses in detail the architecture and the major components of ERATOSTHENES.

1. Introduction

On-Line Analytical Processing (OLAP) is a trend in database technology, based on the multidimensional view of data. The focus of OLAP tools is to provide multidimensional analysis to the underlying information. To achieve this goal, these tools employ multidimensional models for the storage and presentation of data. The goal of this paper is to present requirements, design choices and architecture characteristics for OLAP systems. The perspective that we take is the one of the technology provider: we focus on the internals of an OLAP system, rather than the external behavior of the system. Moreover we present specific design choices for the architecture of an OLAP system that we develop at NTUA. ERATOSTHENES, is an internal project of the database group at NTUA and aims to provide OLAP facilities to the end-user through optimized data structures and query processing techniques.

The structure of this paper is as follows: in Section 2 we present the basic characteristics of an OLAP system. In Section 3 we present requirements and design problems for the construction of such a system. In Section 4 we give some intuition on the ERATOSTHENES system and its architecture. Finally, in Section 5 we conclude our results and provide plans for future work.

* This research has been partially funded by the European Union's Information Society Technologies Programme (IST) under project EDITH (IST-1999-20722).

2. Characteristics of an OLAP System

The core of the multidimensional paradigm is the fact that information is conceptually considered to be defined in terms of a multidimensional space. The axes of the multidimensional space are called *dimensions* and their points functionally determine the value of the points of the multidimensional space, called *measures*. Imagine for example an international publishing company, with traveling salesmen, selling books and CD's to other bookstores all over the world. The dimensions of our example are 'arrival date', 'departure date' (when the salesman arrives/leaves the store), 'product', 'location' and 'salesman'. The functionally dependent measures are 'Sales', 'PercentChange'. The combination of a set of dimensions and a set of measures produces a *cube* (or *hypercube*).

The multidimensional space is also characterized from the fact that each dimension comprises several *levels* of consolidation. The combination of these levels produces a *hierarchy* for each dimension. For example, the 'location' hierarchy can comprise the levels 'city', 'province', 'country' and 'continent'. The values of each level in the hierarchy are related to the values of other levels (e.g. 'Athens', which is a value at the 'city' level, is related to the value 'Europe' at the 'continent' level). The combination of the multidimensional space of a cube with the dimension hierarchies produces a *multi-level multidimensional space*.

Typical OLAP operations include the aggregation or de-aggregation of information (*roll-up* and *drill-down*) along a dimension, the *selection* of specific parts of a cube and the re-orientation of the multidimensional view of the data on the screen (*pivoting)*. A typical session of OLAP queries is essentially a recoil movement along some dimensions. In our motivating example, suppose that a user analyzing sales data of the cube begins the analysis by viewing yearly sales of all products with respect to the continent they were sold. Then he might chose a specific continent for which sales are not so high and decide to drill down to the ('year', 'country') level. If this view is not so enlightening, he might ask to get the five cities with the lowest yearly sales, i.e. drilling down further to ('year', 'city') level, and so on.

The debate on the underlying physical model, supporting OLAP, is centered around two major views. Whereas some vendors, especially vendors of traditional relational database systems (RDBMS), propose the *ROLAP architecture* (Relational On-Line Analytical Processing) [12, 13, 7, 16], others support the *MOLAP architecture* (Multidimensional On-Line Analytical Processing) [1]. In a ROLAP architecture, data are organized in a *star* or *snowflake* schema [4]. A star schema consists of one central *fact* table and several denormalized *dimension* tables. The *measures* of interest for OLAP are stored in the fact table (e.g. 'Dollar Amount', 'Units Sold'). For each dimension of the multidimensional model there exists a dimension table (e.g. 'Geography', 'Product', 'Time', 'Account') with all the levels of aggregation and the extra properties of these levels. The normalized version of a star schema is a snowflake schema, where each level of aggregation has its own dimension table.

Multidimensional database systems (MDBMS) store data in n-dimensional arrays. Each dimension of the array represents the respective dimension of the cube. The contents of the array are the measure(s) of the cube. MDBMS require the precomputation of all possible aggregations: thus they are often more performant than traditional RDBMS [4], but more difficult to update and administer. Another advantage of the MOLAP architecture is, that it provides a direct multidimensional view of the data whereas the ROLAP architecture is just a multidimensional interface to relational data. On the other hand, the ROLAP architecture has two advantages: (a) it can be easily integrated into other existing relational information systems, and (b) relational data can be stored more efficiently than multidimensional data.

In the rest of this paper we will present some general requirements for an OLAP system, independently of its underlying implementation, then we will discuss in more detail the design of ERATOSTHENES system.

3. The Models of an OLAP System

To describe the major requirements for models involved in an OLAP system, we extend the list of requirements provided by [3] and present design choices and reasoning for each one of them. In particular, in order to guarantee the logical and physical data independence, which is necessary in order to achieve maximum flexibility, the database community has devised the ANSI/SPARC three-schema architecture [17]. The corresponding separation of levels that we believe should compose an OLAP system consists also of three levels. The central *logical cube model* defines the concept of a cube and its corresponding operations. The *physical model* deals with how the cubes are stored or indexed for efficient access. The *presentational cube model* is concerned with grouping several logical cubes, defined as parts of one (or more) underlying cube(s), in one presentational entity. The mapping between these levels ensures independence and this is achieved through the use of the intermediate logical model. In the sequel we will try to give a brief description of what we think each of these levels should cover. The following also comprise the models setting the framework of ERATOSTHENES.

3.1 The Presentational Model

Presentational models are not really part of the classical *conceptual-logical-physical* hierarchy of database models; nevertheless there are many reasons for which it is worth trying to explore this layer. First, practice in the field of multidimensional databases is concentrating on models of representation; for example, Microsoft has already issued a commercial standard for multidimensional databases, where the presentational issues are a big part of it [11]. Moreover, data visualization is presently a quickly evolving field, and has proved its power in presenting vast amounts of data to the user [8, 2].

Apart from the industrial proposal of Microsoft, previous proposals already exist in the literature, with the *tape model* [19] being the most prominent one. The most basic characteristic of a presentational model is a declarative query language. The benefits of declarative query languages are best demonstrated by the success of SQL in relational systems. We will not argue more on the subject, but simply note that in the case of OLAP systems a powerful query language enables the possibility of providing the user with complex reports, created from several cubes (or actually subsets of existing cubes). To illustrate the use of a declarative query language, we will use the example of Figure 1, taken from [11].

We use the cube `SalesCube`, mentioned in the `FROM` clause. This cube is defined over the following dimensions: `SalesPerson`, `Geography`, `Quarters`, `Years` (different from the `Quarters` dimension), `Products` and comprises of three measures: `Sales`, `PercentChange` and `BudgetedSales`. In the `WHERE` clause, we also specify that the measure of interest will be `Sales`, restrict the `Year` dimension to `1991` and roll-up the `Products` dimension to the value `ALL`. Then, in the `SELECT` clause we combine the `Salesman` and the `Geography` dimensions in one axis (`COLUMNS`) and the `Quarters` dimension in another axis (`ROWS`). In the `SELECT` clause we also restrict these dimensions through a set of values for each of them; since these values belong to different levels we implicitly roll-up each dimension to different levels of aggregation.

```
SELECT CROSSJOIN({Venk,Netz},{USA_N.Children,USA_S,Japan})ON COLUMNS
{Qtr1.CHILDREN,Qtr2,Qtr3,Qtr4.CHILDREN} ON ROWS
FROM SalesCube
WHERE (Sales,[1991],Products.ALL)
```

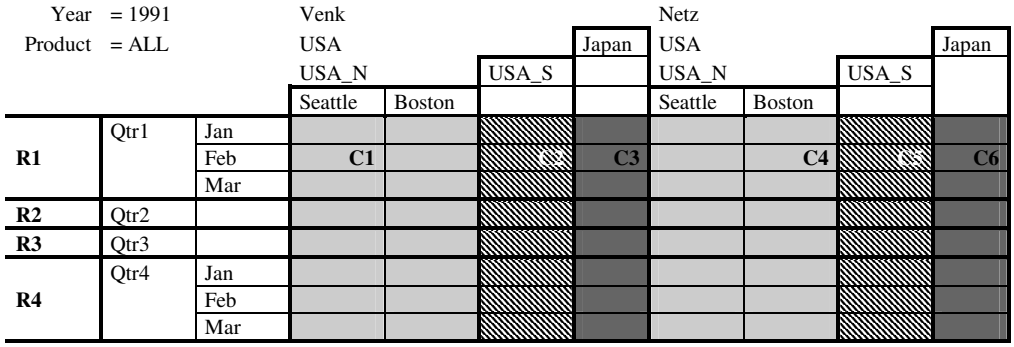

Fig. 1. Motivating example for the presentational model.

In Figure 1, we can see the results of this query depicted in a cross-tab form. For example, across column `C1`, we can see `Sales` values for the `Geography` dimension and specifically for the requested level values: `USA_N` (*region* level) and its children values, `Seattle` and `Boston` (*city* level), for the salesman `Venk`. In column `C2`, `Sales` values for the same salesman but in `USA_S` (*region* level). Note that no children values for `USA_S` appear, since this was not explicitly stated in the `SELECT` clause. Column `C3` corresponds to sales from `Japan` (*country* level), again for the same salesman. Columns `C4`, `C5` and `C6` contain sales values for the same `Geography` levels but this time for salesman `Netz`. Rows `R1`, `R2`, `R3`, `R4` are constructed in a similar manner. Each value appearing in a cell, corresponds to a specific row-column combination and must satisfy the constraints mentioned in the `WHERE` clause.

3.2 The Logical Model

The logical model is the center of any DBMS and an OLAP DBMS could not escape this rule. Apart from the requirement for data independence, which is actually the reason for the existence of a logical model, there are some extra requirements for the logical model of an OLAP DBMS: (a) complex, structured dimensions, (b) complex, structured measures, (c) sequences of operations (since this is actually what a system would perform in practice), (d) completeness of operations (i.e., a set of algebraic operations powerful enough to capture all the usual operations performed from an OLAP system).

In Figure 2, we provide an intuition for the modeling of hierarchies. The multi-level multidimensional space, required by OLAP operations can be supported efficiently through the modeling of hierarchies as lattices. As an example, dimension `arrival date` has the levels `year`, `quarter`, `month`, `week` and `day`. The detailed level of dimension `arrival date` is level `day`. We observe that dimension `arrival date` has two dimension paths, namely `(ALL,year,quarter,month,day)` and `(ALL,year,week,day)`. For further insight into the entities of our multidimensional model, we refer you to [10].

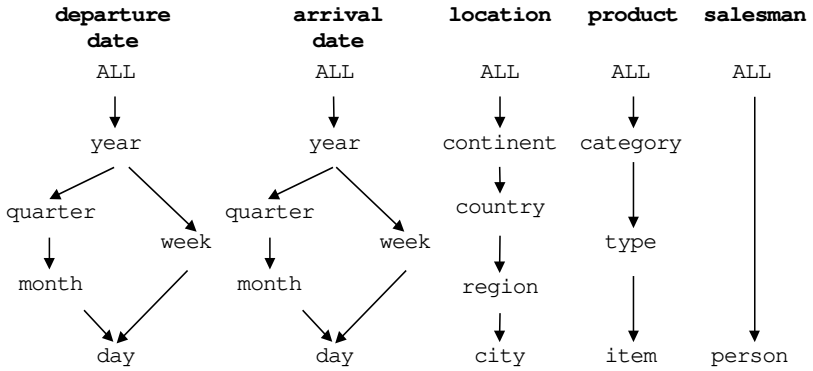

Fig. 2. Examples of dimensions [18].

The operations of the logical model of an OLAP DBMS should be natural and powerful enough to capture the functionality of a true system. Even if not directly covered, sequences of operations such as roll-up, drill-down, select etc., should be enabled through an expressive algebra. Algebraic expressions should be derived directly from the declarative query language of the presentational model.

3.3 The Physical Model

The physical model provides the structures which will be used for the storage of cubes. Conceptually, we can think of cube data as *cells* of a multidimensional array. However, the underlying physical schema could be anything; even something radically different from the array perspective, e.g. conventional relations. The physical data independence mentioned earlier, dictates that changes in the physical schema do not impose any changes to the logical schema.

In the literature there are several proposals for cube storage and indexing structures [15]. Moreover, commercial products rely on their own proprietary techniques for dealing with the same problem [6]. We believe that there is a number of crucial requirements particular to OLAP cubes, that should be the main focus of the physical schema design. In particular, the system should satisfy the following requirements:

Efficient and uniform navigation in the multi-level multidimensional space. We believe that the physical organization of data should be "hierarchy aware" and enable fast access to different hierarchy levels.

Efficient range queries with respect to all dimension levels. The majority of OLAP operations involves some form of a range query. Therefore, there is a need for efficient handling of range queries along any of the levels of a cube.

Exploitation of OLAP access pattern characteristics. As already described in section 2, a typical session of OLAP queries is essentially a recoil movement along some dimension paths. [5] claims that OLAP queries are characterized by a repetitiveness (repeated access of the same data) and a significant predictability (moving along a specific hierarchy path). We believe that this behavior of OLAP access patterns is a major factor for a good physical schema design. An approach in exploiting these characteristics in order to gain in query efficiency, is to impose adequate clustering schemes on the cells of a cube.

Coping with cube sparseness. According to [4], 20% of a typical cube contains real data, i.e. cubes are inherently very sparse. Moreover, empty cells tend to be clustered, rather than randomly distributed in the multidimensional space [14]. Regions of sparseness can be automatically detected during the construction of the cube or derived from the application domain. Provided that we have this knowledge, special care should be taken to avoid occupying space for such empty regions of the multidimensional space. In this case the physical structures used, should provide a flexible pruning capability of the multi-level multidimensional space.

Efficient dimension data processing. Dimension data processing is necessary in order to evaluate dimension restrictions and retrieve the corresponding ranges in the multidimensional space that the target cells reside. A good physical schema design must enable fast retrieval of these ranges through appropriate hierarchical clustering of the dimension data.

Efficient updating. OLAP databases are mostly read-only and batch updates occur at regular intervals. Thus, the physical organization of data must allow such bulk updates to be performed *incrementally*.

Support for many physical cube partitions. A realistic approach in the design of an OLAP system, should consider the possibility of splitting a logical cube into many physical partitions, which might even be geographically distributed. Such a scheme can enable parallel processing of queries, as well as efficient handling of large volume data sets.

4. ERATOSTHENES: Designing an OLAP System

ERATOSTHENES is a specialized DBMS for OLAP cubes. The components of ERATOSTHENES cover the whole spectrum of an OLAP system, starting from the visualization of multidimensional data at the user-end, down to the efficient physical storage of cubes on disk. Basically, ERATOSTHENES architecture consists of three major components, which correspond (more or less) to the three models of an OLAP system discussed earlier.

At the back end lies the *storage engine* (or *storage manager*), which is responsible for the efficient access to the stored data. The cube storage manager used by ERATOSTHENES, is called SISYPHUS [9] and has been implemented on top of the SHORE Storage Manager (SSM)[16]. SISYPHUS includes modules responsible for file management, buffer management, locking management and provides appropriate access methods to the stored data.

The *OLAP engine* component is responsible for all processing tasks. It is responsible for efficiently executing queries submitted by the users, for the parsing and compiling of queries, the optimization of execution plans and the execution of these plans through data retrieval offered by SISYPHUS.

Finally, the *presentational engine* corresponds to the client component of ERATOSTHENES. It is responsible for interacting with the user and presenting the requested results in the specified way. Query results are retrieved from the underlying OLAP engine mostly in an on-demand fashion.

In the sequel, we will discuss in more detail the components of ERATOSTHENES through data flow-like diagrams.

4.1 ERATOSTHENES Software Components

Figure 3(a) illustrates the software components of ERATOSTHENES client (Presentation Engine), while Figure 3(b) those of ERATOSTHENES Server (OLAP and Storage en-

gines). The software components can be roughly distinguished in four categories: (a) system modules, (b) interface objects, (c) cube objects, (d) metadata objects.

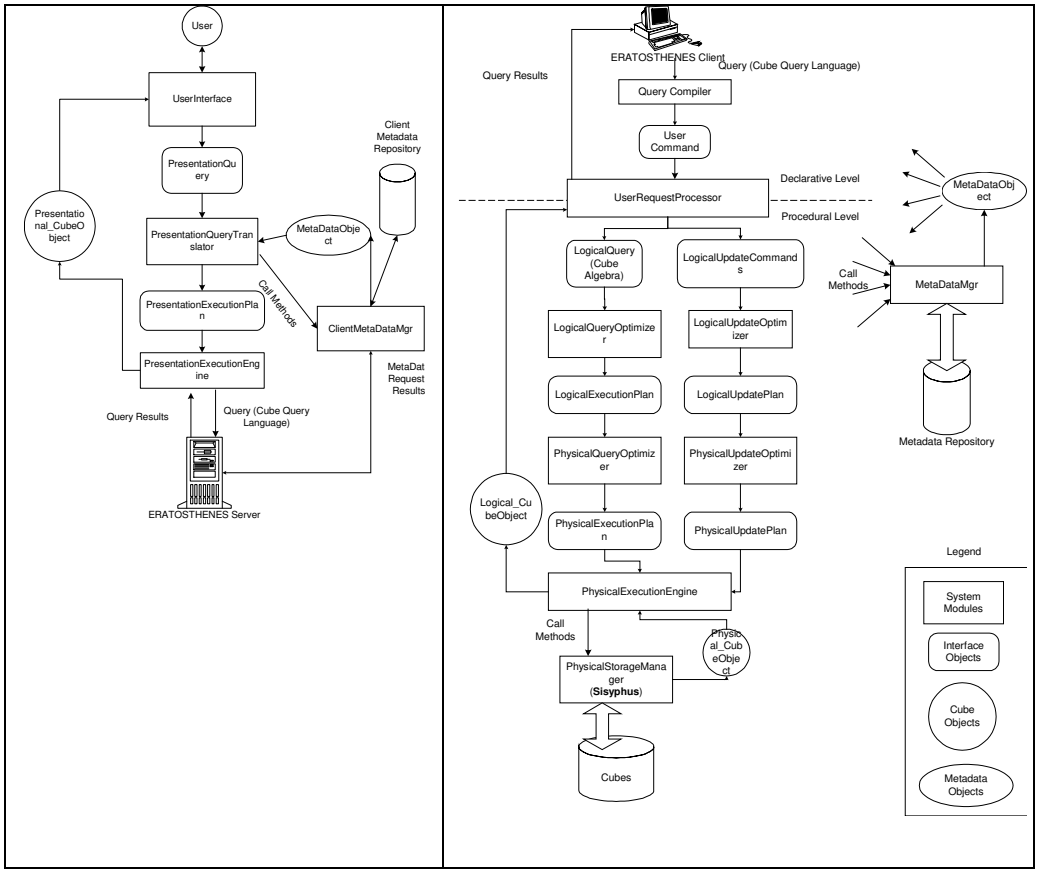

Fig. 3. (a) The software components of ERATOSTHENES Client. (b) The software components of ERATOSTHENES Server.

System modules are typical DBMS components such as storage manager, execution engine etc. These system modules exchange information among them in the form of *interface objects*. For example, an execution plan is encapsulated in a interface object and is handed over from the optimizer module to the execution engine module. Cube objects, serve as main memory placeholders for data coming from disk. In the same sense, metadata objects serve as main memory placeholders for metadata coming from the metadata repository. Each one of these components conceptually belongs, more or less, to one of the three layers of our architecture, namely *presentational*, *logical* and *physical* layer. In Figure 3 this is denoted by the prefix of each module's name.

In Figure 3, we can see the path that a query traverses through ERATOSTHENES until its answer is generated and presented back to the user. Figure 3(a) presents the internal architecture of the client.

The *UserInterface* system module is interacting with the user and typically will have a spreadsheet look and feel. The user's request is wrapped into a *PresentationQuery* interface object and handed over to the next module. A presentational query reflects *what* the user wants to see on the screen and *how* this information is going to be presented.

The *PresentationalQueryTranslator* is responsible for translating a presentation query into queries to the underlying cubes in a declarative form, as well as translating the visualization aspects of the query (e.g., cube orientation, hidden dimensions) into an internal representation. In order to construct these queries, it might request to read metadata information from the *ClientMetaDataMgr.* The result of this module is encapsulated in a *PresentationExecutionPlan* interface object, which integrates the queries to the underlying cubes and the presentation plan.

The *ClientMetaDataMgr* is a module responsible for servicing metadata requests in the client side of ERATOSTHENES. It offers a public interface of methods, which can be called by any other client module (but typically from the *PresentationalQueryTranslator)*, in order to retrieve metadata information, encapsulated in metadata objects. The client keeps a local metadata repository, which is updated whenever needed.

The *PresentationExecutionEngine* sends the declarative queries to the server and waits for the results. When the results arrive, it starts executing the presentational execution plan. In other words it assembles query results into *presentational cube objects*, reflecting the user's visualization requests and hands them over to the *UserInterface* module.

In Figure 3(b) the internal architecture of the server is depicted: the *Query Compiler* receives a list of queries from the client. Each query is parsed and checked for its validity. Then, the *Query Compiler* wraps this query in a *User Command* interface object and hands it over to the *UserRequestProcessor*.

The *UserRequestProcessor* module, translates the query from its declarative form into a procedural form. Naturally, we distinguish an update operation from a read-only query (up to now, we have abused the term 'query' to capture update operations too). The rest of the queries are represented into a cube algebra expression [18]. The *UserRequestProcessor* dispatches the queries to two paths, one for normal queries and one for updates.

The *LogicalQueryOptimizer* module receives as input a query in cube algebra, encapsulated into a *LogicalQuery* interface object. The role of this module is to produce alternative equivalent algebraic expressions, that are estimated to be cheaper to execute. In order to achieve this, it uses (a) a predefined set of algebraic transformation rules, and (b) a set of rules which decide which cubes to use in order to compute another cube (cube usability problem). The output of the module is a set of equivalent algebraic expressions, the *logical execution plans*. Each of these plans is wrapped up in a *LogicalExecutionPlan* interface object.

The *PhysicalQueryOptimizer* receives all these different logical execution plans and tries to evaluate the cost of their execution and chose the cheapest. This module is aware of the physical schema (storage structures and indexes available) and has at its disposal (a) specific algorithms implementing cube operations, (b) a cost model, and (c) size estimation techniques (e.g. histograms). The *PhysicalQueryOptimizer* is the sole responsible to decide on a final execution plan, which is encapsulated into a *PhysicalExecutionPlan* interface object and handed over to the execution engine.

The optimization steps for an update query are similar to those of a normal query. In our motivating example, suppose that at the end of a fiscal quarter the respective data need to be loaded into the OLAP database. The corresponding updates are headed towards four direc-

tions: (a) augmenting the respective levels of the *arrival date* and *departure date* dimensions by a quarter -if this dimension value was not already there- (b) updating the most detailed cube (i.e. the cube whose dimension levels are the most detailed in each hierarchy), (c) updating all affected *derived* cubes (i.e. cubes that can be expressed as the result of OLAP operations on the most detailed cube) and (d) update all affected indexes. For each of these operations several execution plans might exist. For example, it is possible for a derived cube to be updated either incrementally, or by fully recomputing it from scratch. The order which will be followed for the updating of the derived cubes is important, since intermediate results may be reused. Physical restructuring of the most detailed cube might lead to the merging or splitting of data clusters, or even the creation of new ones. Again there is a need for a set of standard update operations, possible transformation rules, implementations of these operations, a cost model and size estimation facilities, in order to produce an optimal update plan. To this end, two modules are involved: the *LogicalUpdateOptimizer*, that produces alternative equivalent logical update plans, expressed as algebraic formulas and the *PhysicalUpdateOptimizer*, that selects the less costly update plan taking into consideration all the other physical parameters (algorithms, indexes, size estimations).

The *PhysicalExecutionEngine* carries out the execution plan. Whenever it needs data, it calls methods of the storage manager and retrieves the data in the form of *PhysicalCubeObjects*. The query answer is wrapped up into a *LogicalCubeObject* handed over to the *UserRequestProcessor*, in order to send it back to the client.

The *PhysicalStorageManager (SISYPHUS)* is a storage engine that supports the creation of persistent cubes [9]. At the storage level, a cube is a set of cells. A cell can contain many measure values. Cube data can be accessed in terms of *cells*, *chunks* and *hyper-rectangles* by specifying the appropriate dimensions in the multi-level multidimensional space. A *chunk* is a cluster of cells that are hierarchically related (e.g. contain sales of the same month), while a *hyper-rectangle* is formed by any range selection query on a cube. Data fetched from the disk are placed into *PhysicalCubeObjects* in memory. Cubes are stored on disk in a compressed form and in an organization that tries to fulfill the requirements of section 3.3.

The *MetaDataMgr* system module offers a public interface of methods for accessing metadata information. These methods can be called by any other system module of the server. This module undertakes the task of building, reading and updating the metadata repository of the DBMS. There is a broad range of information stored in the metadata repository of the server, such as logical schema information (dimensions, cubes, hierarchies, etc.), dimension data, physical schema information (storage structures, indexes, compression schemes, statistics etc.) and system information (SSM files used, SSM identifiers [16], etc.). Metadata fetched from disk are wrapped up into *MetaDataObjects* for in-memory access.

It has to be noted, that the design of the classes of ERATOSTHENES is such that a great degree of extensibility is supported. In other words, it should be easy in the future to experiment with different representations of a cube and different optimization schemes, or use different storage alternatives, etc. In the sequel, we will describe the operational structure of ERATOSTHENES.

5. Conclusions

In this paper, we have presented requirements and design choices for OLAP systems. The aim of the paper has been twofold: on the one hand, we referred to general problems and so-

lutions, applicable to any OLAP system, and on the other hand we presented the specific design decisions made for a prototype under development at NTUA, ERATOSTHENES.

As future work we plan to (a) fully implement a first version of ERATOSTHENES, (b) work on query and update optimization, and (c) experiment with different storage techniques for the cubes.

6. References

[1] Arbor Software Corporation. *Arbor Essbase*.http://www.arborsoft.com/essbase.html, 1996.

[2] P.A. Bernstein, M.L. Brodie, S. Ceri, D.J. DeWitt, M.J. Franklin, H. Garcia-Molina, J. Gray, G. Held, J.M. Hellerstein, H.V. Jagadish, M. Lesk, D. Maier, J.F. Naughton, H. Pirahesh, M. Stonebraker, J.D. Ullman. The Asilomar Report on Database Research. SIGMOD Record 27(4): 74-80 (1998)

[3] M. Blaschka, C. Sapia, G. Hofling, B. Dinter. Finding your way through multidimensional data models. In 9th Intl. DEXA Workshop, Vienna, Austria, August 1998

[4] G.Colliat. Olap relational and multidimensional database systems. SIGMOD Record, 25(3):64-69, Sept 1996.

[5] P. Deshpande, K. Ramasamy, A. Shukla, J. Naughton. *Cashing multidimensional Queries using Chunks*. *Proc. ACM SIGMOD Int. Conf. On Management of data*, 259-270, 1998.

[6] Robert J. Earle. Arbor software corporation u.s. patent #5359724, Oct. 1994, http://www.arborsoft.com.

[7] Informix, Inc.: *The INFORMIX-MetaCube Product Suite.* http://www.informix.com/informix/products/new_plo/metabro/metabro2.htm, 1997.

[8] D.A. Keim. Visual Data Mining. Tutorials of the 23rd International Conference on Very Large Data Bases, Athens, Greece, 1997.

[9] N. Karayannidis, and T. Sellis, "*SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes*", Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW'2001), Interlaken, Switzerland, June 2001.

[10] A. Tsois, N. Karayannidis, and T. Sellis, "MAC: Conceptual data modeling for OLAP", Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses (DMDW'2001), Interlaken, Switzerland, June 2001.

[11] Microsoft Corp. OLEDB for OLAP February 1998. Available at http://www.microsoft.com/data/oledb/olap/

[12] MicroStrategy, Inc. *Relational OLAP: An Enterprise-Wide Data Delivery Architecture*. White Paper, http://www.strategy.com/wp_a_i1.htm, 1995.

[13] MicroStrategy, Inc. *MicroStrategy's 4.0 Product Line*. http://www.strategy.com/launch/4_0_arc1.htm, 1997.

[14] The OLAP Report "Database Explosion", available at http://www.olapreport.com/DatabaseExplosion.htm, 1999.

[15] S. Sarawagi. Indexing OLAP data. *IEEE Data Engineering Bulletin,* March 1997.

[16] *The Shore Storage Manager Programming Interface*, available at: http://www.cs.wisc.edu/shore/doc/ssmapi/ssmapi.html, 1997.

[17] Tsichritzis, D.C., Klug A.: The ANSI/X3/SPARC DBMS framework report of the study group on database management systems, *Information Systems* 3(1978)3, pp.173-191

[18] P. Vassiliadis, S. Skiadopoulos. Modelling and Optimization Issues for Multidimensional Databases. In Proc. 12th Conference on Advanced Information Systems Engineering CAiSE '00), pp. 482-497, Stockholm, Sweden, 5-9 June 2000.

[19] M.Gebhardt et. al. A Toolkit for Negotiation Support Interfaces to Multidimensional Data. In Proc. Of the 1997 ACM SIGMOD Conf., Arizona, USA, 1997.