Multidimensional Database Representation of Real-time, Volatile, and Multi-behavioral Environments

David Levit

Thomas Jefferson High School for Science and Technology
Alexandria, Virginia

Abstract

Traditional relational and multidimensional databases are poorly equipped to deal with data collected from dynamic and volatile environments, where characteristics and conditions are erratic or goals of the system could alter rapidly. These databases' data models of real-world objects require human involvement for creation and modification, since neither relational nor multidimensional databases have adequate structures that can employ autonomous self-modification while adjusting to changes in environment. The goal of this research is to design a database architecture that does not depart far from the foundations in relational and multidimensional databases, but has a sufficiently flexible structure to allow the database to adequately self-manage its data model. Motivated applications presented range from intelligent agents to combat management and search and rescue operations.

Multidimensional Database Representation of Real-time, Volatile, and Multi-behavioral Environments

David Levit
Thomas Jefferson High School for Science and Technology

1 Introduction

The goal of this research is to create a database architecture that utilizes best features of relational and multidimensional database models, and employs a flexible structure to adequately self-manage its data model in real-time, volatile, and multi-behavioral environments. Relational and multidimensional systems primarily address data management needs of environments in which characteristics are well known at database design and modeling stage. Steady nature of such environments tolerates need of human involvement to make modifications of the database structure. However, highly dynamic environments would require an ability to autonomously change database structure in real time by the database itself.

A multidimensional organization is fundamental for the proposed database design. Dimensional structure is a robust and adaptable mechanism to express objects, complex real-world characteristics, and relationships. The ability to simplify mapping between real-world objects and database structure substantially reduces complexity of Structured Query Language (SQL) statements, used in communication with relational databases.

This project addresses foundations of the proposed database organization, explains the database architecture, discusses methods of manipulation of the data model, and demonstrates an implementation. Motivated applications for the database architecture also are presented.

2 Background

The field of database systems has been an important development area in software engineering for the past 30 years. Relational Database Management Systems (RDBMS) became predominant for many business management applications. Relational databases use relations (tables with columns and rows), attributes (named columns of relations), and tuples (rows of relations) to organize data. The databases have a relatively easy structure to understand and use. Relational databases are capable of handling enormous volumes of data, while providing high performance for processing of data updating. However, organization properties of relational database become disadvantages in dynamic environments. Relational data structures could not directly model after real-world characteristics, because the structures are oriented for dealing with data semantics, consistency, and redundancy problems. Therefore, relational databases require human involvement for conceptual, logical, and physical design phases before real world objects could be mapped to their respective data models. Another considerable obstacle for applying relational databases in dynamic environments is the need for client software to adjust SQL statements to fit changing database structure on the fly.

Multidimensional databases, a core component of On-Line Analytical Processing (OLAP) systems, address some of the weaknesses of relational databases. These database systems implement cube database structures that associate variables based on dimensional coordinates. Dimensions represent intuitive and direct form of characteristics of real-world objects. The databases hide as much complex syntax as possible from users and provide consistent response times for all queries. Multidimensional databases still have difficultly adapting to real-time processes, because they do not have a flexible mechanism to dynamically change dimensionality of the database. A new multidimensional structure (cube) must be constructed to support addition and removal of dimensions whenever characteristics change in the environment.

Considering the advantages and disadvantages of relational and multidimensional databases, this project’s goal is to find a database architecture that would include best of both database models and serve dynamic environments effectively.

3 Proposed Design

The proposed design includes database architecture and methods of interaction between client’s software and the database. The design extends a multidimensional database model to include new elements and capabilities that would serve well in real-time, volatile environments. A multidimensional model was chosen as fundamental for the proposed model because of its ability to describe complex entities by subdividing them into basic constituents, embodied within the database as a collection of dimensions. Dimensions consist of a collection of members, representing components of those constituents. For instance, a dimension representing geographical locations would consist of geographical sites, such as cities, as its members. This fundamental property of multidimensional databases can be used to effectively describe real-world characteristics using a set of dimensions.

Consider an environment where numerous agents exist and are capable of interacting with the database. Agents can have different purposes and data with which they operate can constitute different sets of dimensions. The database receives data about the environment it serves only through agents. To make data exchange between a database and an agent possible, an agent must be connected to the database. Therefore, upon database initialization, when no agents are connected, the database is either empty or contains a predefined set of dimensions (Figure 1). Interaction between agents and the database will lead to database self-modification in order to accommodate input data and store in multidimensional space (Figure 2). Client software incorporated on the agent side organizes data by dimensions and cell values. Obtaining dimensions and cell values through communication with agents allows the database to construct its multidimensional structure.

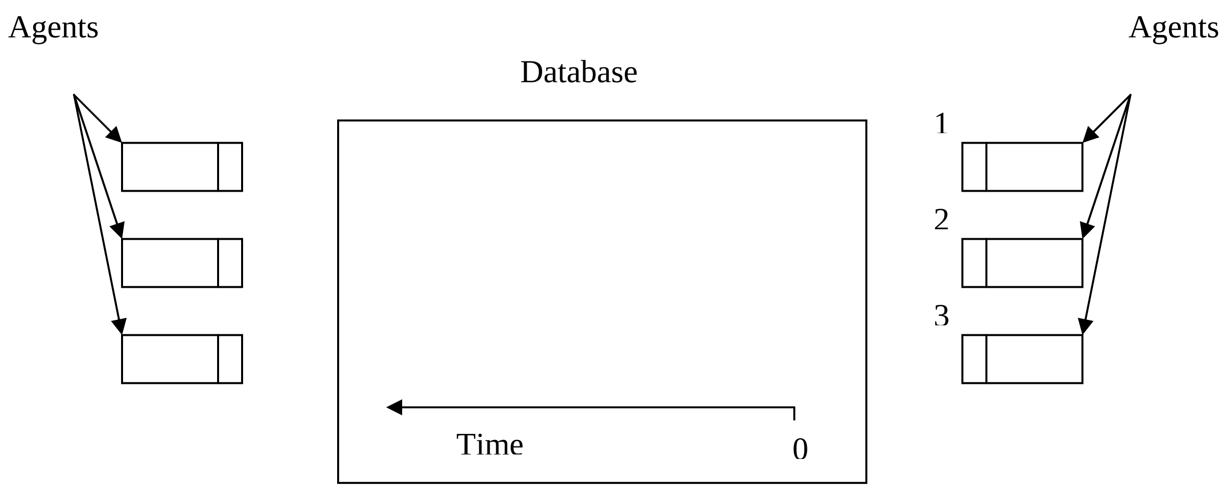

Figure 1 - No agents connected, no dimensions created

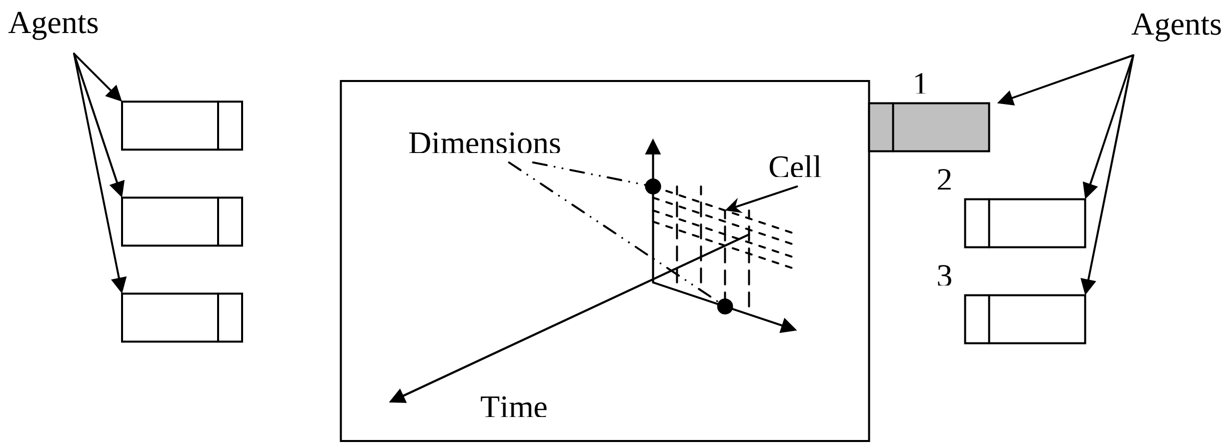

Figure 2 – Agent connected, dimensions and cube created

Apart from dimensions that the database can import from agents, a permanent set of dimensions will be present in the database. For example, Time measurement is present in the database as a separate dimension, allowing the cells in the database to be described by time and other dimensions (Figure 3).

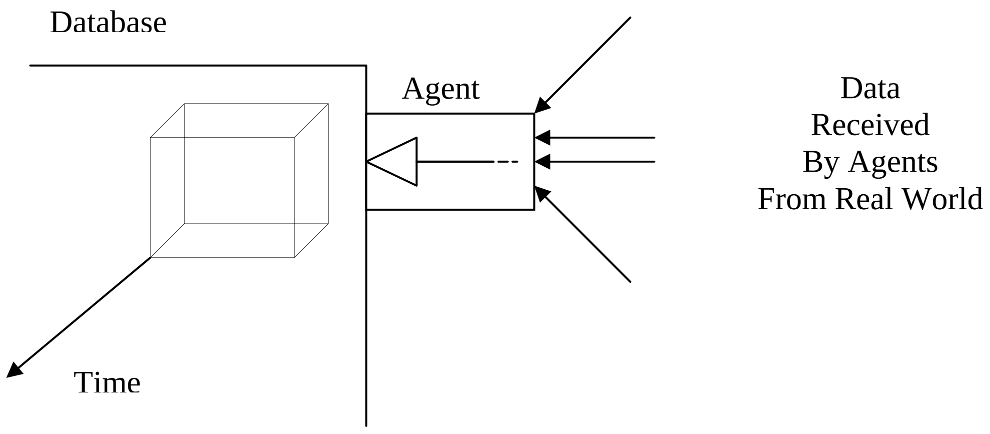

Figure 3 – Multidimensional representation of data

In order for the database to distinguish dimensions and members, each is assigned a unique identification across all the agents that may participate in communication with the database. Subsequent agents connected to the database either share existing dimensions and cells or create new dimensions and cells (Figure 4).

Volatility of the environment also implies the possibility of a disconnection of an agent from the database. Disconnecting the last agent that solely uses a particular set of dimensions triggers removal of those dimensions from the multidimensional model. However, to preserve the already collected data, the current cube would be saved in an array of historic cubes. The rationale behind transformation of the multidimensional model of the database is that the database does not have a way to conclude about future agent departure and arrival (connection and disconnection). Maintaining unused dimensions is costly and inefficient for the database. At the same time, dimensional and cell values produced during agent connection session could be utilized when dimensions reappear.

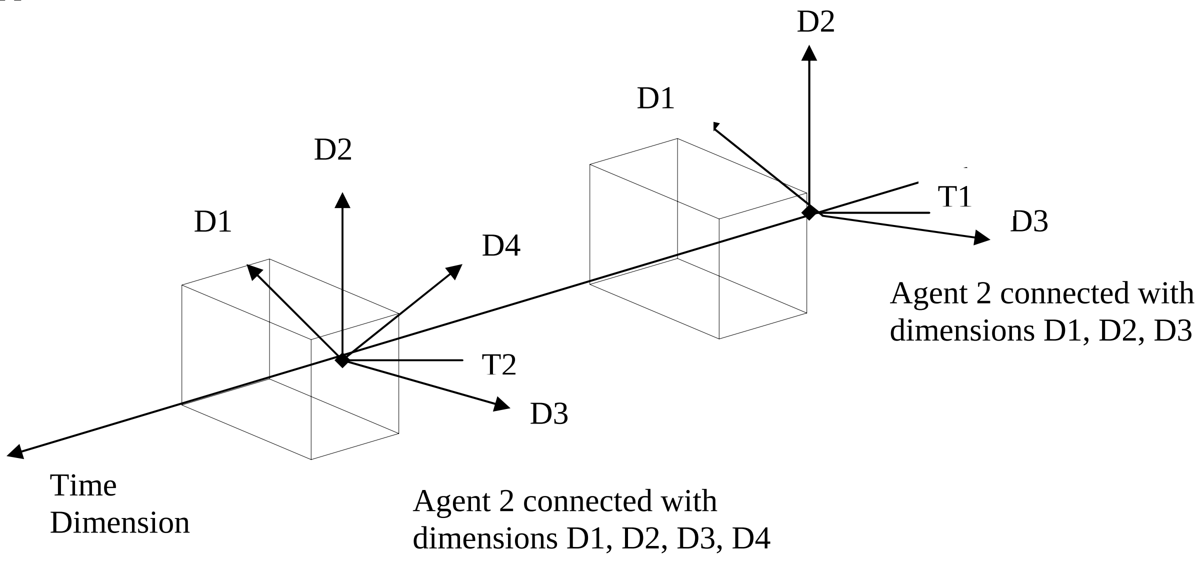
D2
D1
T1
D3
Agent 2 connected with dimensions D1, D2, D3
D2
D1
D4
T2
D3
Time Dimension
Agent 2 connected with dimensions D1, D2, D3, D4

Figure 4
Dimensionality of the cube changes when an agents connects to the database

For example, in Figure 5, an agent that collects data represented by D1, D2, D3, D4 dimensions reconnects with the database whose dimensional structure does not include D4 at time period T3. That event triggers a database search across historic cubes for one that matches the current dimensional model. If such a cube is found, in this case at T1, then the current cube incorporates cell data from time period T2 and T1.

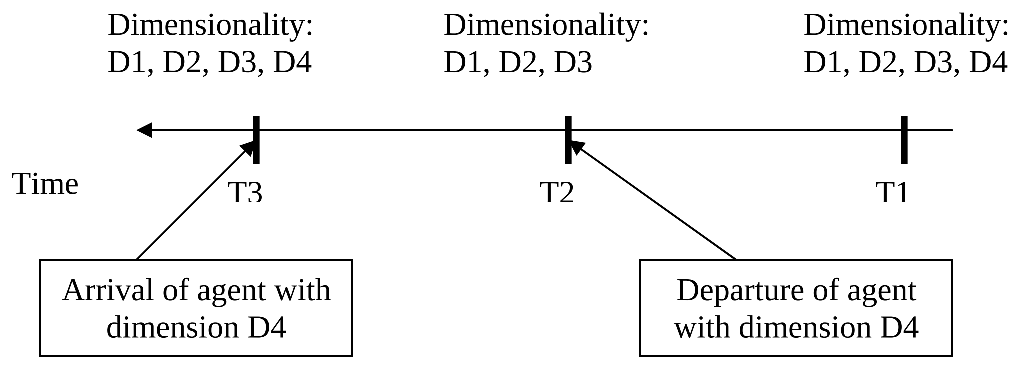

Figure 5 - Recovery of dimension from historic cube

Standard multidimensional databases institute a hierarchical organization of members within dimensions in order to reflect most common business organization and relations. However, solely hierarchical structure is substantially insufficient to represent the complexity of real world relationships. The proposed design embraces a graph structure as a method to describe the relationship between members of dimensions. A graph is constructed using member as vertices and relationships as edges. Each edge is associated with a weighted value that represents different attributes of a relation, such as cost or distance (Figure 8).

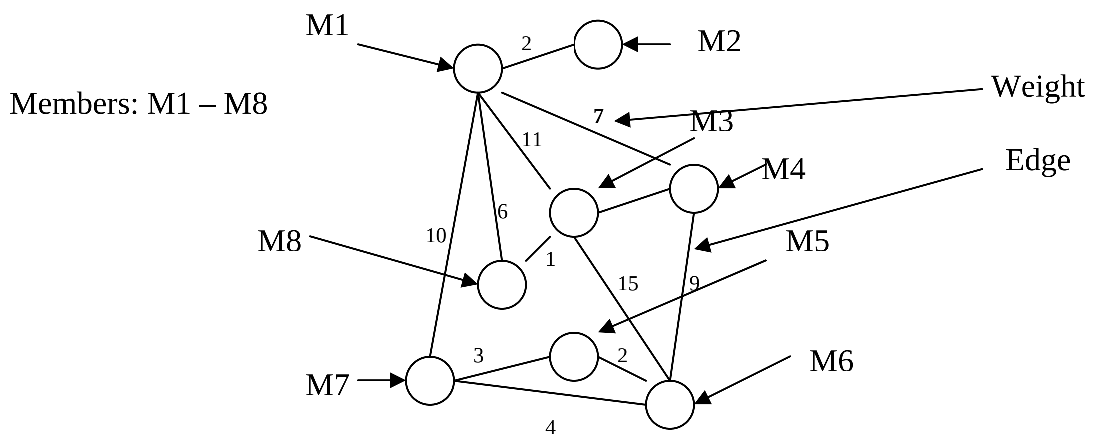

Figure 6 – Dimension organization

Multiple attributes are defined for the same weighted edge, providing enough flexibility to express complex relationships (Figure 7). For example, an edge can represent distance and travel time. Unique names for each edge attribute make it possible to distinguish separate attributes.

Edge	*Weight – distance*	*Weight – time*
M7 – M6	4	20
M7 – M5	3	25
M5 – M6	2	30

Figure 7 – Example of multiple edge weights

A substantial advantage of relational databases over other types of databases is the ability to establish relations between tables. The proposed design implements such an ability to create relationships between not only members of the same dimension but also those of other dimensions (Figure 8). For instance, one dimension represents locations while another represents agents in the system. Relationships associate particular agents with particular locations.

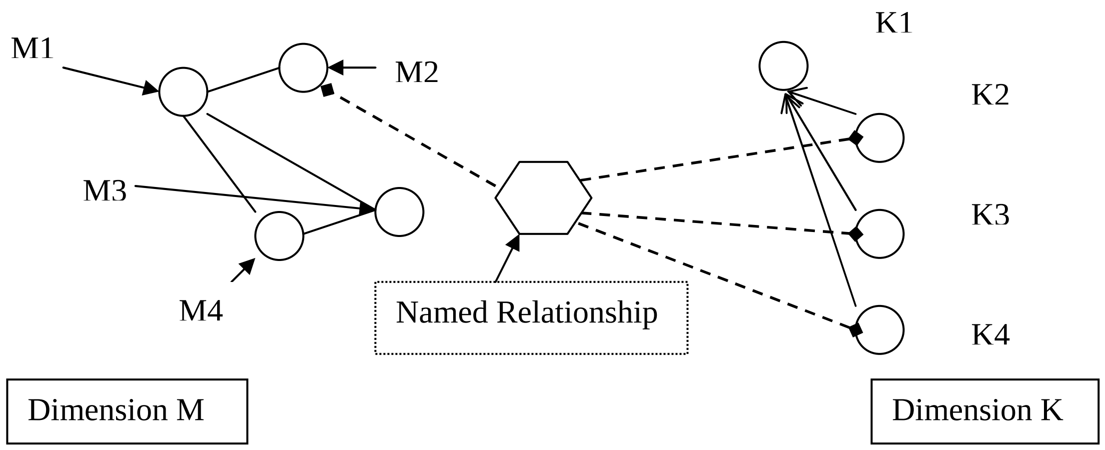

Figure 8 – Cross-dimensional relationships

Whenever a new dimension is created, two default members within it are constructed. The first member, named “ALL”, is the highest level of aggregation for cell data in that dimension. For example, a multidimensional structure has three dimensions Time, X-coordinate, Y-coordinate. The three dimensions form a cube that stores data transferred by the agent during an exploration of a 2-D surface. As more locations on the surface with X- and Y- coordinates are surveyed, more data is populated in the 3-D cube. While time category is important for particular measurements, a representation of the entire pool of data on X- and Y- coordinates surface is needed with no regard to the time data was collected. Therefore, aggregation of this data over the Time dimension to the “ALL” member consists of geometrically projected values on the surface formed by X- and Y- coordinates (Figure 9).

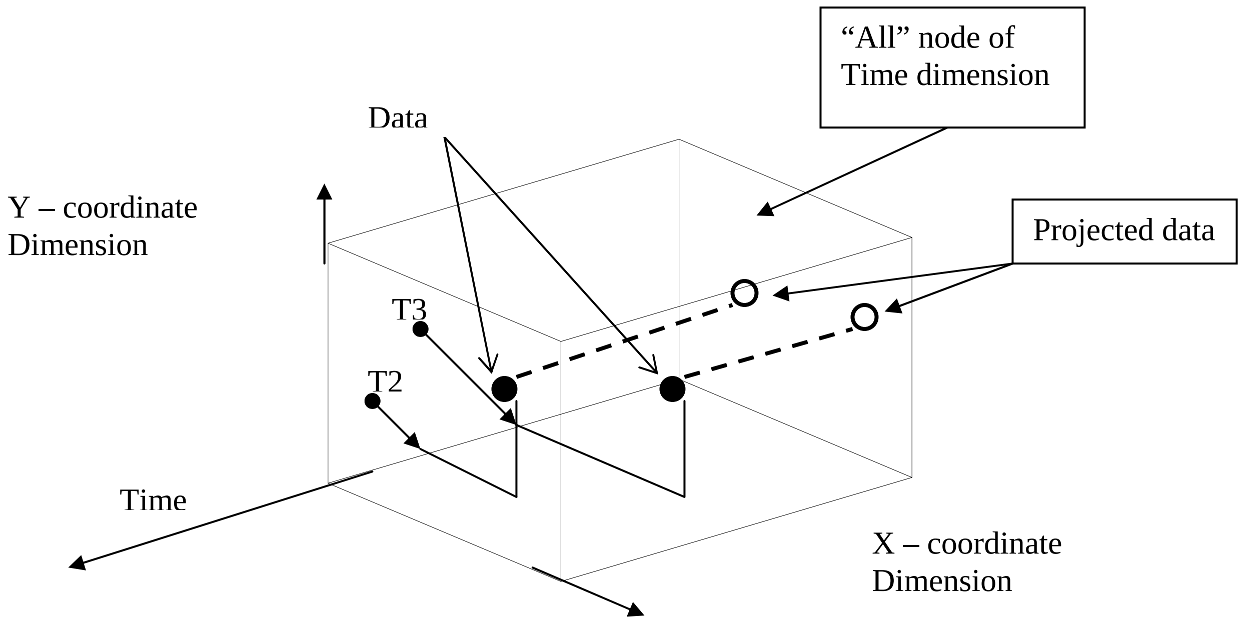

Figure 9 – Projectile of data based on X- Y- surface

The second default is the base member. It links cells that have no associations with some dimensions in the cube. Returning to the previous example, the 2D-surface is represented by three dimensions in the cube. An additional agent is attached with the capability to explore the space formed by and the height above the 2D-surface (Z-coordinate). Since both agents share the same cell values, the database alters dimensional structure to incorporate the Z-dimension, becoming a four dimensional structure (X, Y, Z, and Time dimensions). The first agent still operates in a two dimensional coordinate system and has no way of knowing that the database changed to incorporate the Z-dimension. The task of the database is to recognize and transfer three-dimensional into four-dimensional identification by assigning the base member from the Z-dimension.

4 Implementation

Considering the goal of this research was to find the architecture for a database that meets the needs of volatile and multi-behavioral environments, the implementation stage was aimed only to provide proof of concept. Characteristics of a full-scale database, such as transaction support, language interface, or concurrent updates, were not implemented and will be considered for the next design phase. The current implementation functions only as an in-memory database without an option for permanent storage. This stage of implementation builds a new data manager and ability to query data. The implementation, programmed in Java, includes a storage manager for multidimensional cell values (cube), a dimensional manager module, and a relational manager.

4.1 Storage Manager for Multidimensional Cell Values (Cube)

The storage manager handles data organization in the database. It retrieves data along with dimensions and members, which serve as coordinates for identifying data in the cube. The dimensions are compared against a list of those present in the current data cube. If the sets do not match, the processor checks if the input statement contains new dimensions and updates the database cube (described in section 4.2). If the statement contains less dimensions than those present in the cube, pairs of missing dimensions and base members are added to the coordinate sets. Then, the data is saved in a cell within a cube organized as a hashmap. All the dimensions have an “ALL” member, representing aggregation of the data in a dimension. The manager updates the aggregation value each time data is modified. Since the dimension aggregations are calculated at each individual insertion, the time of retrieving an aggregation value is particularly efficient.

The novelty of this mechanism offers the ability to dynamically resolve dimensional coordinates compared to traditional multidimensional database where the coordinates in all dimensions in the cube must be fully specified before data can be updated.

4.2 Dimension Manager Module

The purpose of the dimensions manager is to self-modify dimensional structure of the database cube. The manager can add or remove dimensions when it receives a data update statement containing new dimensions or when the last database client (agent) that solely uses certain dimensions is disconnected. The rationale behind removing dimensions is to sustain cube update efficiency by maintaining only used dimensions by current database clients. However, the ultimate goal of the database is to preserve all data collected; the current cube is saved in an array of historic cubes.

Parsing the update statement, the dimension manager retrieves a list of dimensions and searches for those not present in the current database model. If any new dimensions are identified, the database structure is dynamically modified to include those dimensions (Figure 10). Each new dimension and its base member are added to existing cells’ coordinate sets. Cells that contain the new dimension’s aggregation data are also added to the cube. Then, the manager searches through an array of historic cubes for a cube with the same set of dimensions. If such a cube is found, cells from the historic cube are added to current cube, with exception of cells containing base and aggregation members of new dimension. As each new cell is added, the aggregations for dimensions are updated. Thus, the cube recovers previously saved data with the same dimensionality (Figure 5).

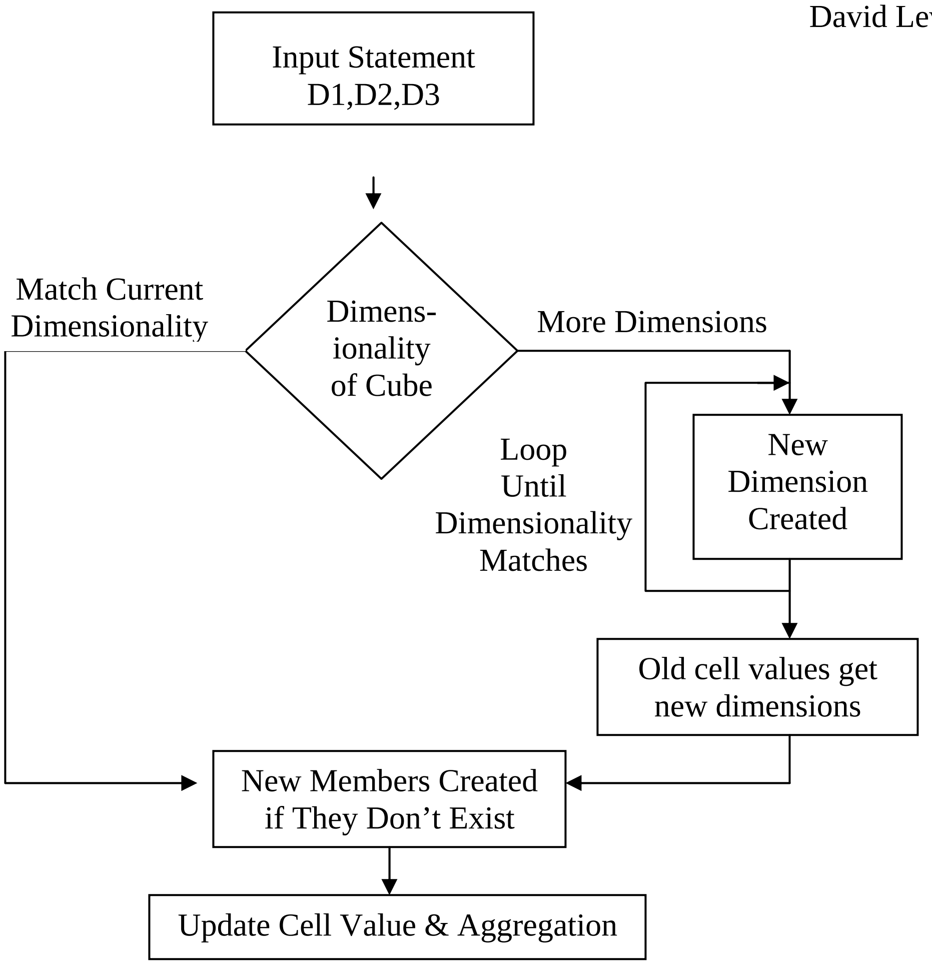
Input Statement D1,D2,D3
Dimens-ionality of Cube
Match Current Dimensionality
More Dimensions
New Dimension Created
Loop Until Dimensionality Matches
Old cell values get new dimensions
New Members Created if They Don't Exist
Update Cell Value & Aggregation

Figure 10 – Dimension manager module

The manager also obtains dimensions' members from an update statement. Members in a dimension object are presented as nodes in a linked list and are indexed through a binary tree. The manager searches the index tree for an equivalent member. If a member is not found, a new node is created to represent the member in the appropriate order, and the index is updated.

In case a dimension is removed, the cube is first saved to a historic array of cubes. The current cube's data cells not containing the dimension's base member are removed, and the manager removes the dimension from coordinate sets of all data cells left.

A similar operation in traditional multidimensional databases would require restructuring and rebuilding the cube and recalculating all the aggregations. Removing dimensions with associated cell values would lead to loss of data, since historic cubes are not maintained. Proposed architecture provides more flexibility in manipulation of dimensions and cell values.

4.3 Relational Manager

Relational manager is similar to the organization found in relational databases. The manager is capable of creating objects that utilize hashmaps as structures for storing relational data. The relational tables have unique names assigned at the time of their creation and store relational associations (edges) between members within the same dimension or between different dimensions. The relational associations may be characterized by multiple attributes (Figure 7 and Figure 8).

5 Motivating Applications

5.1 Artificial Intelligence Agents

Progress in the field of Artificial Intelligence (AI) has been primarily geared toward development of sophisticated algorithms, wherein storage of underlying data was designated only a supportive role. The rationale behind this data storage trend maybe attributes to limitations of memory and permanent storage resources available for autonomous agents. However, new innovations allow agents to integrate with database systems. For example, new data storage technologies increase memory capacity needed for database software on board of agents, or wireless communication allows data interchange between agents and remote database systems.

Another reason preventing AI agents from using database systems was limited agent functionalities and not well developed cross agent communication technology. Standalone agents were able to accomplish all necessary data manipulation by only incorporating data structures within loaded agent software. Absence of a need in concurrency control made it virtually unnecessary to include any ACID (atomicity, consistency, isolation, and durability) database properties as part of AI agent data management.

Advancements in AI led to significant changes in the field of automated and half-automated agents. AI agents' spectrum of capabilities increased; areas of employment widened.

AI systems have the potential to grow from a standalone agent to a complex conglomerate system where multiple AI agents would have different sub-tasks while underlining the mutual tasks of the whole system. Colonies of species, such as ants or bees, could serve as examples of such conglomerate systems in the real world. The species are grouped based on specific tasks, such as soldiers, builders, and so on, but share a common physical space and resources. Therefore, situations could arise wherein multiple agents unified in such a system would need a common database in which data could be shared between members of the system.

5.2 Other Applications

The proposed database can be adapted to systems other than those present in AI. Examples of such systems whose characteristics are erratic and can change rapidly, are combat and search-and-rescue operations.

A battlefield can prompt hasty changes in resources required. Combat forces may call units, similar to agents discussed earlier, equipped with wireless computing devices that are capable of interacting with a central database. By depending on data retrieved from a common database that stores real-time information, military units may immediately participate in a battle with minimum knowledge about location and disposition of own forces and enemies. Collected data from separate units can be easily shared with limited or no verbal communication. Air, naval, or ground units can operate with dimensions relevant only to their unique needs, and also share dimensions with other types of units.

Emergency management systems could have similar needs for databases. In case of natural disasters, rescue delegations have little time to prepare and must operate in chaotic conditions. Coordination efforts between rescue forces are complex tasks. To increase efficiency, teams will interact with a database to retrieve necessary data collected by all types of teams (police, firefighters, army, etc.). Such electronic coordination could be vital important in international rescue efforts, in which often languages are barriers between different international teams.

6 Discussion & Conclusion

The goal of this research was to analyze relational and multidimensional database systems and to design an optimal database architecture that would perform most effectively in real-time, volatile, and multi-behavioral environments. In such environments, database clients can modify the database structure in unpredictable ways to accommodate changes in the environment. Relational and multidimensional databases have strong fundamentals, however, they require human interaction for data structure alterations. The proposed design is based on a multidimensional model's fundamental structure, because of its ability to intuitively divide complex entities of the real world into basic constituents. The database architecture is sufficiently flexible to adequately self-manage its data model, allowing addition and subtraction of dimensions in multidimensional space. The current state of the database can be utilized in many applications where database in-memory would be sufficient to solve data management needs. The physical aspects such as concurrent control and transaction management as the next phase for future research.

References

[1] Connolly, Thomas and Carolyn Begg. *Database Systems*. Harlow: Pearson Educational Limited, 2002.

[2] Bagnall, Brian. *Core Lego Mindstorms Programming: Unleash the Power of the Java Platform.* Upper Saddle River: Prentice Hall PTR, 2002.

[3] Delaney, Kalen. *Inside Microsoft SQL Server 2000*. Redmont: Microsoft Press, 2001.

[4] Horstmann, Cay S. and Gary Cornell. *Core Java 2 Volume II – Advanced Features*. Palo Alto: Sun Microsystems, Inc., 2002.

[5] Jones, Tim M. *AI Application Programming*. Hingham: Charles River Media, Inc., 2003.

[6] Rafanelli, Maurizio. *Multidimensional Databases: Problems and Solutions.* Hershey: Idea Group Publishing, 2003.

[7] Russel, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Upper Saddle River: Pearson Education, Inc, 2003.

[8] Sedgewick, Robert. *Algorithms in C++: Fundamentals, Data Structures, Sorting, Searching.* Boston: Addison-Wesley Publishing Company, Inc., 1998.

[9] Sedgewick, Robert. *Algorithms in C: Graph Algorithms.* Boston: Addison-Wesley Publishing Company, Inc., 2002.

[10] Thomsen, Erik, George Spofford, and Dick Chase. *Microsoft OLAP Solutions*. New York: John Wiley & Sons, Inc, 1999.