An Infrastructure for Manipulating Multidimensional Semistructured Data

Vassilis Zafeiris[1], Christos Doulkeridis[1], Yannis Stavrakas[1,2], and Manolis Gergatsoulis[2,3]

[1] Knowledge & Database Systems Laboratory
National Technical University of Athens (NTUA), 15773 Athens, Greece.
[2] Institute of Informatics & Telecommunications,
National Centre for Scientific Research (N.C.S.R.) 'Demokritos',
15310 Aghia Paraskevi Attikis, Greece.
[3] Department of Archives and Library Sciences,
Ionian University, 49100 Corfu, Greece.
{bzafiris,cdoulk}@aueb.gr
{ystavr,manolis}@iit.demokritos.gr

Abstract. Multidimensional Semistructured Data (MSSD) are semistructured data that present different facets under different contexts (i.e. alternative worlds). For the representation of MSSD various formalisms have been proposed by the authors, both syntactic (such as mssd-expressions and MXML) as well as graphical (such as Multidimensional OEM). In this paper we present an infrastructure for handling MSSD. This infrastructure provides appropriate tools for building MSSD applications, and is independent from any particular application that uses it. We also present a graphical interface, called MSSDesigner, that provides access to the infrastructure, and we describe OEM History, an MSSD application that supports keeping track of temporal changes in semistructured databases.

Keywords: Semistructured Data, OEM Graph, OEM History, Multidimensional Semistructured Data, MOEM Graph.

1 Introduction

The nature of the Web poses a number of new problems [4]. While in traditional databases and information systems the number of users is more or less known and their background is to a great extent homogeneous, Web users do not share the same background and do not apply the same conventions when interpreting data. Such users can have different perspectives of the same entities, a situation that should be taken into account by Web data models. Those problems call for a way to represent information entities that manifest different facets, whose contents can vary in structure and value.

Multidimensional Semistructured Data (MSSD) paired with an extension of OEM called *multidimensional OEM*, have been proposed in [9]. MSSD and MOEM incorporate ideas from multidimensional programming languages [3] and associate data with *dimensions*, in order to tackle the aforementioned problems. In MSSD, variants of the same information entities, each holding under a specific world, have been consolidated to form *multidimensional entities*. Syntactic expressions called *context specifiers* are associated to pieces of data (facets of multidimensional entities), and specify sets of worlds under which these data hold.

In this paper, we present the overall architecture of an infrastructure that allows the management of multidimensional semistructured data. This infrastructure can be used for the development of new applications and MSSD tools that will be placed on top of it, by providing access to a number of operations on MSSD. We focus mainly on MOEM graphs that can be used to represent MSSD and we present MSSDesigner, a graphical interface for handling MOEM graphs, which is also a part of the infrastructure. Moreover, we present an interesting application concerning MSSD, called OEM History, that deals with accommodating temporal changes in semistructured databases.

The rest of the paper is organized as follows. In section 2, we present the basic notions behind multidimensional semistructured data. In section 3, an overview of the MSSD infrastructure is given. In section 4, we discuss the basic component of the infrastructure, which is the Multidimensional Data Manager. In section 5 we present MSSDesigner, a graphical interface for handling MOEM graphs. Section 6 presents OEM History, an application for accommodating changes in semistructured databases. Finally, in section 7 we conclude the paper.

2 Multidimensional Semistructured Data

Multidimensional semistructured data (*MSSD* in short) [9] are semistructured data [1, 10] which present different facets under different *contexts* (or sets of *worlds*). Information entities that assume different facets are called *multidimensional entities*. Each facet is associated with a context that defines the conditions under which that facet holds.

2.1 Contexts and Dimensions

The notion of *world* is fundamental in MSSD. A world represents an environment under which data obtain a substance. In the following definition, we specify the notion of world using a set of parameters called *dimensions*.

Definition 1. *Let $\mathcal{D}$ be a set of dimension names and for each $d \in \mathcal{D}$, let $\mathcal{V}_d$ be the domain of d, with $\mathcal{V}_d \neq \emptyset$. A* world *$w$ with respect to $\mathcal{D}$ is a set whose elements are pairs (d, v), where $d \in \mathcal{D}$ and $v \in \mathcal{V}_d$, such that for every dimension name in $\mathcal{D}$ there is exactly one element in w.*

The main difference between conventional and multidimensional semistructured data is the introduction of *context specifiers*. Context specifiers are syntactic constructs, expressing constraints on dimension values, that are used to qualify semistructured data expressions (*ssd-expressions*) [1] and specify sets of worlds under which the corresponding ssd-expressions hold. In this way, it is possible to have at the same time variants of the same information entity, each holding under a different set of worlds. An information entity that encompasses a number of variants is called *multidimensional entity*, and its variants are called *facets* of the entity. The facets of a multidimensional entity may differ in value and / or structure, and can in turn be multidimensional entities or conventional information entities. Each facet is associated with a context that defines the conditions under which the facet becomes a *holding facet* of the multidimensional entity. If a facet f of a multidimensional entity e holds under a world w (or, under every world defined by a context specifier c), then we say that e evaluates to f under w (under c, respectively).

Example 1. The use of dimensions for representing worlds is shown with the following three context specifiers:

```
(a) [time in {07:00..15:00}]
(b) [language=greek, detail in {low,medium}]
(c) [season in {fall,spring}, daytime=noon | season=summer]
```

In Example 1, context specifier (a) represents the worlds for which the dimension `time` can take any value between `07:00` and `15:00`, while (b) represents the worlds for which `language` is `greek` and `detail` is either `low` or `medium`. Context specifier (c) is more complex, and represents the worlds where `season` is either `fall` or `spring` and `daytime` is `noon`, together with the worlds where `season` is `summer`. Notice that, according to Definition 1, for a set of $(dimension, value)$ pairs to represent a world with respect to a set of dimensions $\mathcal{D}$, it must contain exactly one pair for each dimension in $\mathcal{D}$. Therefore, if $\mathcal{D} = \{\texttt{language}, \texttt{detail}\}$ with $\mathcal{V}_{language} = \{\texttt{english}, \texttt{greek}\}$ and $\mathcal{V}_{detail} = \{\texttt{low}, \texttt{medium}, \texttt{high}\}$, then $\{(\texttt{language}, \texttt{greek}), (\texttt{detail}, \texttt{low})\}$ is one of the six possible worlds with respect to $\mathcal{D}$. This world is represented by context specifier (b) in Example 1, together with the world $\{(\texttt{language}, \texttt{greek}), (\texttt{detail}, \texttt{medium})\}$.

Notice that it is not necessary for a context specifier to contain values for every dimension in $\mathcal{D}$. Omitting a dimension implies that its value may range over the whole dimension domain. The context specifier `[]` is called *universal context* and represents the set of all possible worlds with respect to a set of dimensions $\mathcal{D}$.

2.2 Multidimensional OEM

Multidimensional Object Exchange Model (*MOEM*) is an extension of *Object Exchange Model* (OEM) [2] suitable for representing multidimensional data. MOEM extends OEM with two new basic elements:

– *Multidimensional nodes* represent multidimensional entities, and are used to group together nodes that constitute facets of such entities. Graphically, multidimensional nodes have a rectangular shape to distinguish them from conventional circular nodes, which are called *context nodes*.
– *Context edges* are directed labeled edges that connect multidimensional nodes to their facets. The label of a context edge pointing to a facet p, is a context specifier defining the set of worlds under which p holds. Context edges are drawn as thick lines, to distinguish them from conventional (thin-lined) edges, called *entity edges*.

The existence of two kinds of nodes and two kinds of edges raises the question of which node - edge combinations are meaningful. Starting with what is not legal, a context edge cannot start from a context node, and an entity edge cannot start from a multidimensional node. Those two are the only constraints on the morphology of an MOEM graph.

Note that, as in OEM, a context node can be complex or atomic, depending on whether it has outgoing edges or not. Atomic nodes are leaves and have an atomic value of some predefined type (integer, string, etc).

An MOEM is defined as a *context-deterministic multidimensional data graph*. By context-deterministic, we mean a graph whose context edges that depart from the same multidimensional node have labels (context specifiers) that are mutually exclusive (define disjoint sets of worlds). In a context-deterministic graph, each multidimensional entity may evaluate to at most one facet under any given world. The definition of multidimensional data graph is given below.

Definition 2. *Let $\mathcal{C}$ be a set of context specifiers, $\mathcal{L}$ be a set of labels, and $\mathcal{A}$ be a set of atomic values. A* multidimensional data graph *is a finite directed edge-labeled multigraph $G = (V, E, r, \mathcal{C}, \mathcal{L}, \mathcal{A}, v)$, where:*

1. *The set of nodes V is partitioned into* multidimensional nodes *and* context nodes *$V = V_{mld} \cup V_{cxt}$. Context nodes are further divided into* complex nodes *and* atomic nodes *$V_{cxt} = V_c \cup V_a$.*
2. *The set of edges E is partitioned into* context edges *and* entity edges *$E = E_{cxt} \cup E_{ett}$, such that $E_{cxt} \subseteq V_{mld} \times \mathcal{C} \times V$ and $E_{ett} \subseteq V_c \times \mathcal{L} \times V$.*
3. *$r \in V$ is the* root*, with the property that there exists a path from r to every other node in V.*

4. v is a function that assigns values to nodes, such that: $v(x) = M$ if $x \in V_{mld}$, $v(x) = C$ if $x \in V_c$, and $v(x) = v'(x)$ if $x \in V_a$, where M and C are reserved values, and v' is a value function $v' : V_a \rightarrow \mathcal{A}$ which assigns values to atomic nodes.

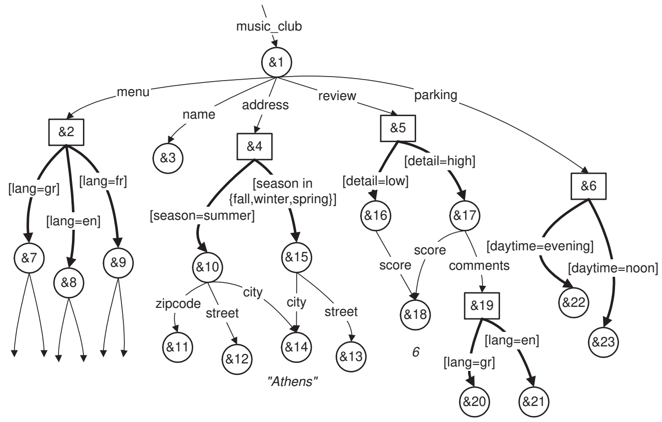

Fig. 1. A multidimensional music-club.

As an example consider the (part of an) MOEM graph in Figure 1 which represents context-dependent information about a music-club. The graph is not fully developed and some of the atomic objects do not have values attached. The `music_club` with oid `&1` operates on a different address during the summer than the rest of the year (in Athens it is not unusual for clubs to move south close to the sea in the summer period, and north towards the city center during the rest of the year). Except from having a different value, context objects can have a different structure, as is the case of `&10` and `&15` which are variants of the multidimensional object `address` with oid `&4`. The `menu` of the club is available in three languages, namely English, French and Greek. In addition, the club has a couple of alternative `parking` places, depending on the time of day as expressed by the dimension `daytime`.

Two fundamental concepts related to multidimensional data graphs are the notions of *explicit* and *inherited contexts*. The explicit context of a context edge is the context specifier assigned to that edge, while the explicit context of an entity edge is the universal context specifier `[]`. The explicit context can be considered as the "true" context only within the boundaries of a single multidimensional entity. When entities are connected together in an MOEM graph, the explicit context of an edge is not the "true" context, in the sense that it does not alone determine the worlds under which the destination node holds. The reason for this is that, when an entity e_2 is part of (pointed by through an edge) another entity e_1, then e_2 can have substance only under the worlds that e_1 has substance. This can be conceived as if the context under which e_1 holds is inherited to e_2. The context propagated in that way is combined with (constraint by) the explicit context of each edge to give the *inherited context* for that edge. In contrast to edges, nodes do not have an explicit context; like edges, however, they do have an inherited context. The inherited context of a node or edge is the set of worlds under which the node or edge is taken into account, when reducing the MOEM graph to a conventional OEM graph (as explained later in section 4).

Multidimensional entities are not obliged to have a facet under every possible world. However, they must provide enough coverage to give substance to each incoming edge under at least one world. The notion of *validity* of an MOEM graph ensures that edges pointing to multidimensional nodes do not exist in vain. In particular, an edge h leading to a node q is invalid if the inherited context of h has no common world with the context union of the worlds represented by the explicit contexts of the edges that depart from q.

2.3 Multidimensional XML

Besides MOEM which models MSSD as a graph, a notation for expressing MSSD has been also proposed in [9]. The notation extends ssd-expression [1] with context specifiers, and is called mssd-expression. Another way to describe MSSD is *Multidimensional XML* (MXML) [7, 6] which is an extension of XML that incorporates context specifiers [6]. In MXML, elements and attributes may depend on a number of dimensions. A multidimensional element is denoted by preceding its name with the special symbol "`@`", and encloses one or more *context elements* that constitute facets of that multidimensional element, holding under the worlds specified by the corresponding *context specifier*. Context elements have the same form as conventional XML elements.

MXML suggests a new way for designing Web pages which encode context-dependent data. We refer to the new paradigm as *multidimensional paradigm* [6]. The multidimensional paradigm allows a single document to have a number of *variants*, each holding under a specific *world*. Information in such a document is encoded in *MXML*. Once a world is specified, such an MXML document can be reduced to a conventional XML document that constitutes the holding facet under that world. An MXML document may be associated with a Multidimensional XSL stylesheet (*MXSL* in short) containing instructions on how to present information in XML documents. An MXSL stylesheet encodes a set of conventional XSL stylesheets, each being the facet of the MXSL under a specific world. For each possible world, the holding XSL is applied to the holding XML to give the view of the information under that world.

3 Architecture of an MSSD Infrastructure

An MSSD infrastructure is a set of tools and processes that create, manipulate, and query MSSD, and are used directly, or by applications that need the support of an MSSD framework.

This section presents such an infrastructure for manipulating multidimensional semistructured data, which can also be used for implementing additional tools and applications. The infrastructure consists of the following components, depicted in Figure 2: MOEM Graph, Multidimensional Data Manager, Manager GUI, Repository, Manager API.

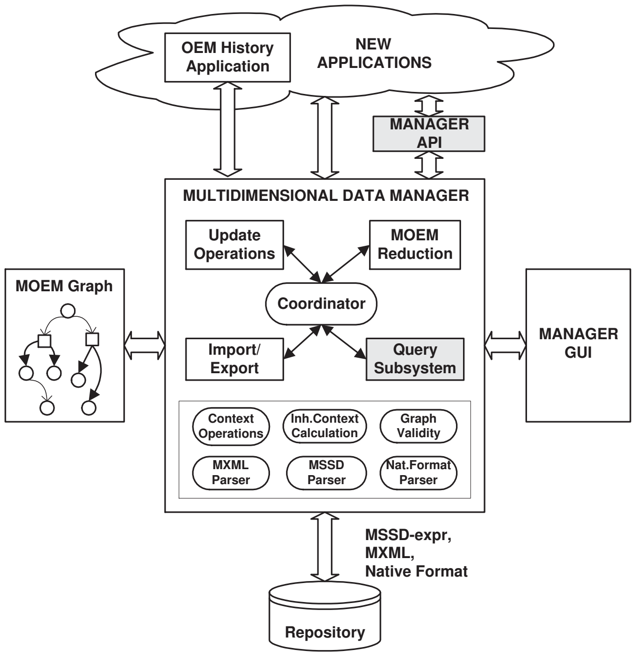

Fig. 2. Architecture of an MSSD infrastructure

MOEM Graph consists of the main memory data structures which actually hold graph representations of MSSD.

Multidimensional Data Manager (MDM) is responsible for managing MOEM graphs. It comprises a set of modules that allow the creation, maintenance, and querying of multidimensional semistructured data. Various modules of MDM can be accessed through graphical user interfaces offered by the Manager GUI.

Manager GUI comprises a number of user interfaces, which provide access to various functions of MDM, like MOEM graph creation and maintenance, and MOEM graph querying. MOEM graph creation and maintenance can be performed through MSSDesigner, which is described in detail in section 5.

Repository is the physical storage medium that supports the MDM needs for loading and saving MSSD and MOEM graph representations. Note that a number of formats able to represent semistructured data can be used when storing MOEM in files. At this moment, mssd-expressions, MXML and Native Format expressions are supported.

Manager API aims at providing an application programming interface for new applications that will need to use the functionality of the system. This module enables applications to use the existing infrastructure by issuing commands in an especially made script-like language. However, an application can directly use the MDM, as is the case of OEM History, described in section 6.

4 Multidimensional Data Manager

Multidimensional Data Manager (MDM) is the most important component in Figure 2. It comprises a set of utility processes which appear inside a box placed at the bottom of MDM in Figure 2 and are accessible to all other MDM modules. Those utility processes are explained below.

4.1 General Purpose Functions

Context Operations, implements the necessary operations on context specifiers defined in [9], including *context intersection* ($\cap_c$) and *context union* ($\cup_c$). Context intersection and context union correspond to conventional set operations on the set of worlds defined by the context specifiers. Specifically, if c_1, c_2 are context specifiers and $W_D(c_1)$, $W_D(c_2)$ are the sets of worlds they represent with respect to a set of dimensions D, then the context intersection $c_1 \cap_c c_2$ gives a context specifier that represents the intersection of the worlds that c_1 and c_2 represent, i.e. $W_D(c_1 \cap_c c_2) = W_D(c_1) \cap W_D(c_2)$. Similarly, for the context union we have: $W_D(c_1 \cup_c c_2) = W_D(c_1) \cup W_D(c_2)$.

Inherited Context Calculator is used in order to compute the inherited context of a graph. The *inherited context* of a node p is given by a context specifier ic_p which is the context union of the inherited contexts of the edges that lead to p. Let h be an edge that departs from node p. Then the *inherited context* ic_h of h is the context intersection of the inherited context of p with the explicit context of h. Note that the root of an MOEM graph is assumed to hold under every possible world, thus its inherited context is the universal context. For calculating the inherited context the graph is traversed in a breadth-first manner, starting from the root.

The *Graph Validator* checks the validity of the graph.

Furthermore, there exist some processes that deal with parsing expressions in various formats, namely *MSSD Parser*, *MXML Parser*, and *Native Format Parser*, which parse mssd-expressions, MXML, and native MOEM expressions respectively. Except for the utility processes, MDM consists of a number of modules that are described in the following sections.

4.2 Coordinator

External components communicate with MDM through the coordinator. Its job is to decompose incoming requests into a number of basic operations, and to assign these operations to appropriate modules in MDM.

As an example, consider the case of a user of a graphical interface who removes an edge from an MOEM graph through a graphical interface. The coordinator accepts the request from Manager GUI, and diverts it to the *Update Operations* module, which is responsible for carrying out such modifications. After that, the coordinator notifies the corresponding GUI to display the updated graph.

4.3 Update Operations Module

Modifications of an MOEM graph are carried out by this module, which implements the following basic change operations:

createCNode(*cid*, *val*): a new context node is created. The identifier *cid* is new and the value *val* can be an atomic value of some type, or the reserved value C, that denotes a complex object.

updateCNode(*cid*, *val*): changes the value of *cid* to *val*. The node must not have any outgoing arcs.

createMNode(*mid*): a new multidimensional node is created with the new identifier *mid*.

addEEdge(*cid*, *l*, *id*): creates a new entity edge with label l from context node *cid* to node *id*.

remEEdge(*cid*, *l*, *id*): removes the entity edge (cid, l, id) from the MOEM graph.

addCEdge(*mid*, *context*, *id*): creates a new context edge labeled with the context specifier *context* from multidimensional node *mid* to node *id*.

remCEdge(*mid*, *context*, *id*): removes the context edge $(mid, context, id)$ from the MOEM graph.

As in conventional OEM, in MOEM object deletion is achieved through arc removal, since the persistence of an object is determined by whether or not the object is reachable from the root.

4.4 Import/Export Module

An MOEM graph can be encoded and stored in a file in a number of different formats. We have used three different formats namely *mssd-expressions*, *Multidimensional XML* and *Native Format expressions*. The Import/Export module handles the storing and the loading processes concerning a graph and all the different representations that describe it.

MSSD-Expressions The grammar of mssd-expressions is given in Extended Backus-Naur Form (EBNF) in [9]. Here we give as an example the mssd-expression that describes the `address` object with oid `&4` in Figure 1:

```
&4 ([season=summer]:
        &10 {zipcode: &11,
             street: &12,
             city: &14 "Athens"},
    [season in {fall,winter,spring}]:
        &15 {city: &14,
             street: &13})
```

MXML Representations MXML has been defined in [7]. The following MXML extract describes the same `address` object as the above mssd-expressions example:

```
<@address>
    [season=summer]
        <address>
            <zipcode>...</zipcode>
            <street>...</street>
            <city id="c1"> Athens </city>
        </address>
    [/]
    [season in {fall,winter,spring}]
        <address>
            <city idref="c1" />
            <street>...</street>
        </address>
    [/]
</@address>
```

Native Format Expressions Native format expressions are used to describe MOEM graphs, and have the property to retain the exact position of the graph nodes. They are formed by describing every node with a line of the form:

[nodeId, nodeType, xPos, yPos, value, isRoot, bHung]

where *nodeId* is the node identifier, *nodeType* is the node type (atomic, complex, multidimensional), *xPos* and *yPos* its coordinates on the screen, *value* its value. *isRoot* is true if the node is the root, and *bHung* is true if the node is not reachable from the root. Every edge is described with a line of the form:

(fromNode; toNode; edgeType; value)

where *fromNode* is the starting node, *toNode* is the destination node, *edgeType* specifies if the edge is context edge or entity edge, and *value* contains the edge's explicit context (for context edges) or its label (for entity edges).

4.5 MOEM Reduction Module

This module is responsible for two jobs: (a) reduction of a MOEM graph to a conventional OEM graph holding under a specific world, and (b) partial reduction of a MOEM graph to another MOEM graph holding under a set of worlds.

Reduction to OEM Given a specific world, it is always possible to reduce an MOEM graph to a conventional OEM graph holding under that world. By specifying different worlds, the same MOEM can be reduced to different OEMs. The graph to be reduced must be *context deterministic*, i.e. for every multidimensional entity in the graph the context specifiers of that entity must be mutually exclusive. This ensures that two different facets of a multidimensional entity cannot hold under the same world. A procedure which performs reduction to OEM is presented below, and it is based on the idea that inherited contexts identify the parts of the graph that do not hold under a world.

The facet of an MOEM graph G under a world w, is an OEM graph G_w that holds under w. Given a world w expressed as a context specifier c_w, the graph G_w can be obtained from G through the following process:

Procedure reduce_to_OEM (G, c_w, G_w) is

Step 1: Remove every node and edge with $c_w \cap_c ic = \emptyset_c$, where ic gives the inherited context of the node or edge respectively.

Step 2: For every entity edge (p, l, m_1) with m_1 a multidimensional node, follow the path of consecutive context edges (m_1, c_1, m_2),…,(m_n, c_n, q), $n \geq 1$, until no more context edges can be followed. Then, if q is a context node add a new entity edge (p, l, q) in the set of entity edges.

Step 3: Remove all multidimensional nodes. Remove all edges departing from or leading to the removed nodes.

As an example, consider the OEM graph in Figure 3 obtained by applying the reduction_to_OEM procedure on the MOEM graph of Figure 1 for the world $w = \{(\texttt{season}, \texttt{summer}), (\texttt{daytime}, \texttt{noon}), (\texttt{lang}, \texttt{gr}), (\texttt{detail}, \texttt{low})\}$.

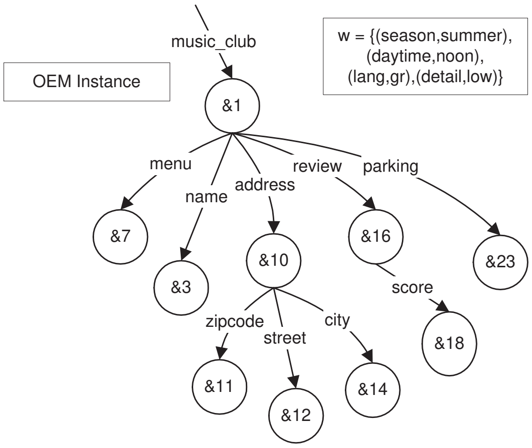

Fig. 3. The OEM instance, holding under the world w, of the MOEM graph in Figure 1.

Partial Reduction Partial reduction is in fact a generalization of the procedure reduce_to_OEM given above. In partial reduction a context specifier that represents a set of (generally more than one) worlds is given. The MOEM graph is reduced to a new MOEM graph containing only the nodes and edges that hold under any of the specified worlds. In order to obtain the reduced MOEM graph the inherited context of nodes and edges is used, and a process similar to step 1 of reduce_to_OEM is performed.

4.6 Query Subsystem Module

Multidimensional Query Language (MQL) is a query language for MOEM databases. It resembles to query languages for semistructured data like Lorel, but incorporates context specifiers in path expressions, and additional clauses in queries that cater for context operations. MQL has been specified and is currently being implemented. The query module is responsible for executing MQL queries on an MOEM graph and producing new MOEM graphs as a result.

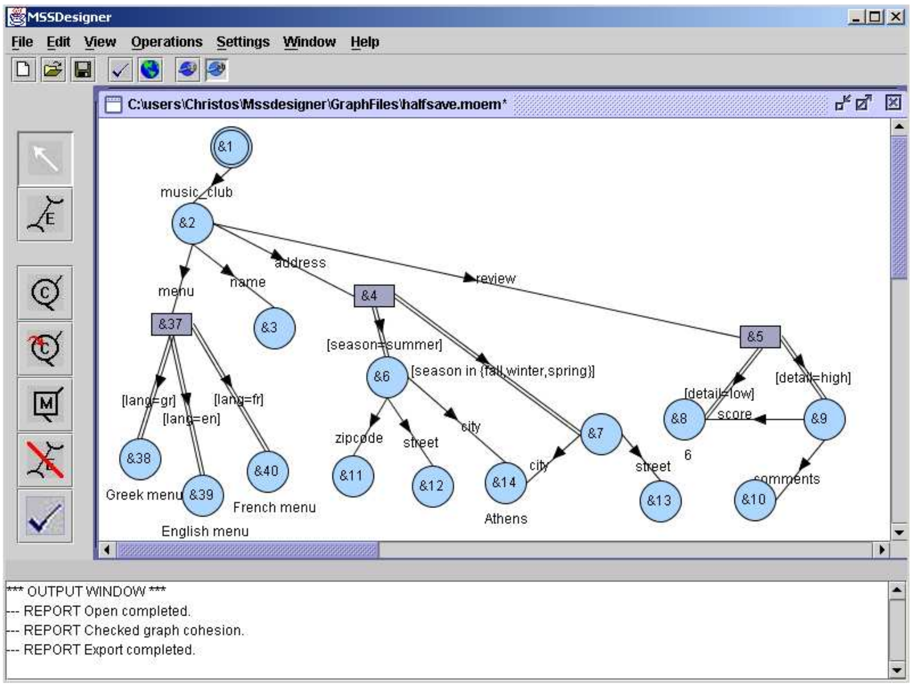

Fig. 4. A sample image of MSSDesigner.

5 MSSDesigner

The infrastructure described in Figure 2 has been implemented, except from the *Query Subsystem* and the *Manager API*, which are under development. *MSSDesigner* is a graphical interface (part of the *Manager GUI*) that gives access to the functionality of MDM. A sample image of MSSDesigner displaying a simple graph about a multidimensional music club is depicted in Figure 4.

MSSDesigner employs a multi-document interface (MDI) where each document-frame corresponds to a data graph. All the operations, performed by the various control buttons of the application, have effect to the currently focused frame. The control buttons of MSSDesigner are positioned in two toolbars placed in the upper and left sides of the main window. The toolbar on the left side contains buttons performing the basic operations that modify the structure and layout of the graph. Beginning from the top of the toolbar, the arrow-labeled button allows the multiple selection and transposition of nodes in the display area. The following five buttons correspond to the MOEM basic change operations for adding and updating nodes, and adding and removing edges. The last button performs consistency check of the graph, and removes nodes that are not accessible from the root.

The upper toolbar contains seven buttons. The first one creates a new window for designing an MOEM graph. The second button corresponds to the import operation concerning Native Format

Expressions, whereas the next button exports the graph contained in the focused window as a Native Format Expression. The button with the tick symbol calls the Graph Validator module to check the validity of the graph, while the next button is used to reduce the MOEM graph to an OEM graph holding under a given world. Partial reduction can be performed through the application menu. In order to display or hide the explicit contexts and labels on edges, the following button is used. Finally, the last button calls the Inherited Context Calculator, and causes the inherited contexts to appear on the screen. An alternative way of performing the aforementioned operations is via the menu bar of the application.

Through MSSDesigner it is possible to import a graph from an MSSD expression or MXML representation, and export a graph to one of those formats. These operations are activated by the File Import/Export menus respectively. Native format imprint the position of nodes as well, and is useful for saving and loading unfinished graphs, as the process of saving and loading in native format does not perform consistency checking, and consequently it does not remove hanging nodes and subgraphs. Note that the graph is always checked for consistency when importing or exporting MSSD expressions or MXML.

MSSDesigner has been implemented in Java (requires JDK 1.3.1), and is available at:

`http://www.dbnet.ece.ntua.gr/~ys/moem/moem.html`

6 An Application Example: OEM History

As stated in [8], MOEM can be used to keep track of changes over time, in OEM databases. The process is as follows. The necessary operations for changing an OEM are *addNode*, *updateNode*, *addEdge*, and *removeEdge*, as stated in [5]. Each OEM basic change operation is associated with a timestamp and is trapped to a sequence of MOEM operations, in such a way that new facets of an object are created in MOEM, whenever an object changes in OEM. MOEM facets are associated with contexts comprised of a dimension d whose domain is time. In this way, the resulting MOEM represents the history of the OEM database.

OEM History is an example application that uses the MSSD infrastructure, depicted in Figure 2, in order to represent histories of semistructured databases. OEM History uses directly the MDM subsystem, and the general architecture of the application is shown in Figure 5.

The user interacts with the graphical interface of the application, which employs a MDI (Multi Document Interface). Each document-frame of the application depicts a graph corresponding to the current database state, the history representation of the database, or to database snapshots at previous time instances. The history of the database is represented by an MOEM graph, which is the only graph maintained internally by the application. Coexistence of more than one MOEM

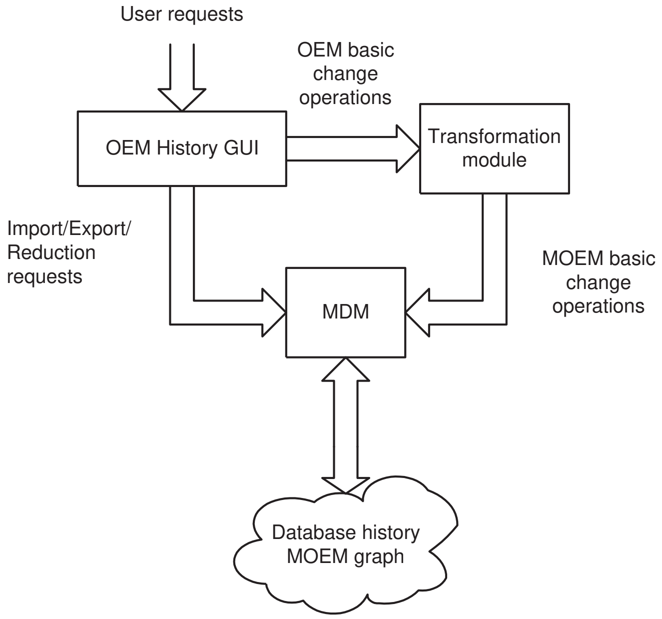

Fig. 5. Architecture of OEM History application.

graphs in the same application instance is not allowed, as OEM History tracks the history of a single database. The graph depicting the current state of the database and the graphs depicting database snapshots, are OEM graphs produced by reducing the MOEM graph for a world defined by specifying a time instance for the dimension d.

The user cannot directly affect the MOEM graph or the database snapshot graph. It is only allowed to modify the OEM graph which corresponds to the current state of the database. Modifications take place through OEM basic change operations, initiated from the user interface. Requested OEM operations are passed to the "Transformation module" and are mapped to MOEM basic change operations, that incorporate the current snapshot of the database as a new facet into the MOEM graph. The operations are applied to the MOEM graph by the MDM subsystem.

A screenshot of the OEM History's main window is presented in Figure 6. The functionality of the application, as well as concepts concerning its theoretical background will be explained by the use of an example. The example refers to a database of a company that records its employees and their salaries. Figure 6 presents the database in its initial state. The left frame shows the current state of the database while the right shows the MOEM representation of its history. Notice that, except node 0, the two graphs are identical, as no changes have occurred yet to the database.

The changes to the database are initiated from buttons located in the left toolbar of the main window, as well as from the menu. The button with the arrow allows the multiple selection of nodes

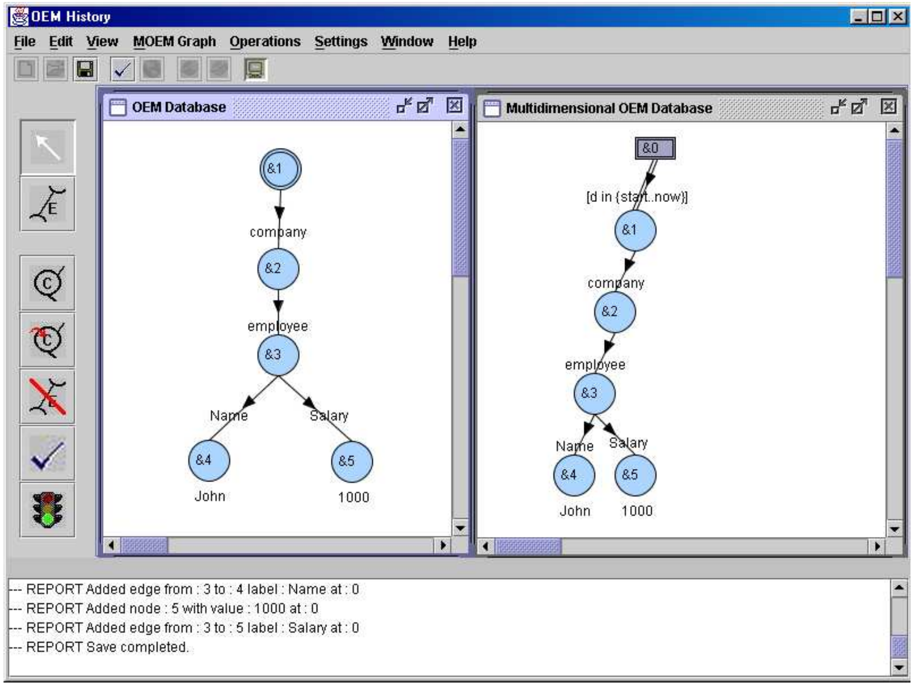

Fig. 6. The main window of OEM History depicting the initial state of the database.

and their transposition on the design area. The following four buttons correspond to the four OEM basic change operations. The 'tick' button performs coherence check of the graph, removing nodes that are not accessible from the root of the OEM database. Finally, the last button marks the end of a sequence of basic change operations and commits all changes to the database under a common timestamp. Notice that the toolbar is active, only when the graph representing the current state of the database is focused.

The upper toolbar contains buttons that load or save the database and its history. It also contains a button that creates snapshots of the database at previous time instances by reducing the MOEM graph to OEM. Those actions are executed by submitting appropriate requests to MDM.

Figure 7, depicts the MOEM history graph after updating 'Salary' to value 2000 at timestamp 10. The comparison of MOEM graphs in Figures 6 and 7, gives an intuition of the MOEM basic change operations that correspond to an update operation on the OEM database. Specifically, the 'Salary' atomic node is surrogated by a multidimensional node that groups together its new and previous facets. Node 5 holds for timestamps before 10 while node 7 for timestamps after 10.

In Figure 8, a new employee called *Peter* is registered in the database at timestamp 20. The right frame shows the history graph after the changes have been committed to the database. The new elements added to the graph are: a new 'Company' facet with two employees and a

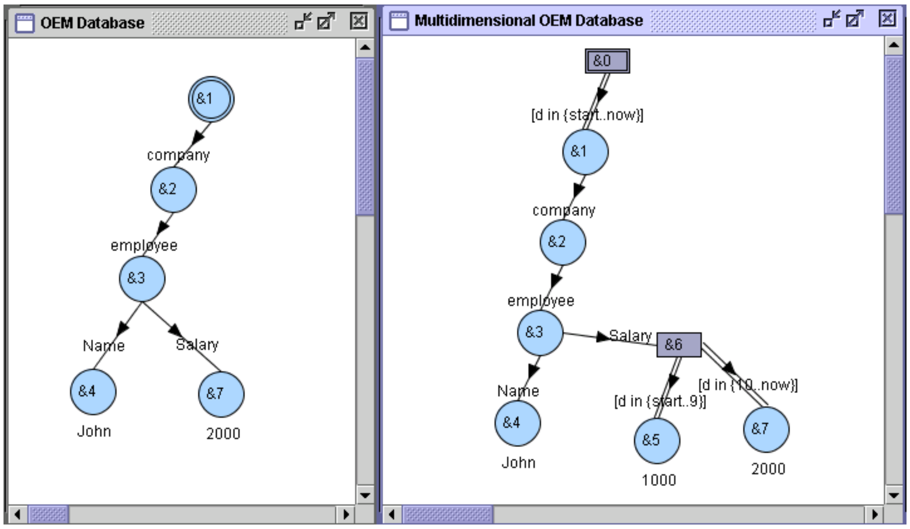

Fig. 7. Update of the 'Salary' attribute to value '2000' at d = 10.

multidimensional node that surrogates the initial 'Company' sub-graph with its alternative facets before and after timestamp 20. This procedure will be repeated in case a new employee is added or removed from the database. Figure 9 shows the MOEM graph after updating the salary of Peter to 4000 at d = 30, and subsequently removing employee Peter from the database at d = 40. The left part of Figure 9 shows a snapshot of the database at timestamp d = 5. The OEM graph is identical to the initial state of the database as the first change took place at timestamp 10.

OEM History is implemented in Java (requires JDK 1.3.1) and is available at:

`http://www.dbnet.ece.ntua.gr/~ys/moem/moem.html`

7 Conclusions

In this paper we proposed an architecture for manipulating MSSD that can be used as an infrastructure for the development of new MSSD tools and applications. We showed the capabilities of this infrastructure and we presented MSSDesigner, a graphical user interface for designing MOEM graphs, that is a part of the GUI of this infrastructure. Furthermore we explained how a new application can exploit this functionality, and we presented OEM History, an application that uses the infrastructure and aims at accommodating temporal changes in semistructured databases.

We believe that MOEM has a lot of potential, and can be used in a variety of fields, among which: in information integration, for modeling objects whose value or structure vary according to sources; in digital libraries, for representing metadata that conform to similar formats; in representing geographical information, where possible dimensions could be *scale* and *theme*.

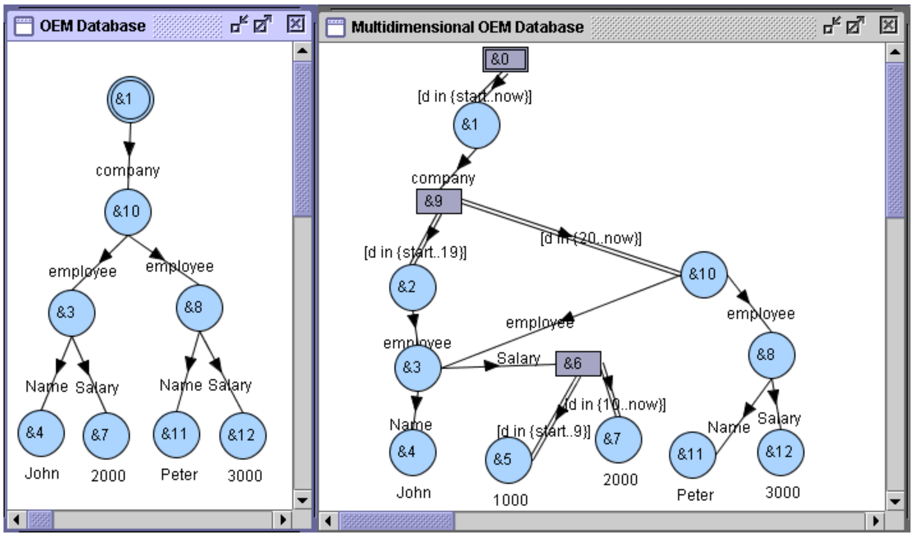

Fig. 8. The database after the insertion of a new employee at d = 20

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000.
2. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
3. E. A. Ashcroft, A. A. Faustini, R. Jagannathan, and W. W. Wadge. *Multidimensional programming*. Oxford University Press, 1995.
4. Ph. 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, and J. D. Ullman. The Asilomar Report on Database Research. *SIGMOD Record*, 27(4):74–80, 1998.
5. S. S. Chawathe, S. Abiteboul, and J. Widom. Managing historical semistructured data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999.
6. M. Gergatsoulis, Y. Stavrakas, D. Karteris, A. Mouzaki, and D. Sterpis. A Web-based System for Handling Multidimensional Information through MXML. In A. Kaplinskas and J. Eder, editors, Advances in Databases and Information Systems (ADBIS' 01), Proceedings, Lecture Notes in Computer Science (LNCS), Vol. 2151, pages 352–365. Springer-Verlag, 2001.
7. M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating dimensions to XML and DTD. In H. C. Mayr, J. Lanzanski, G. Quirchmayr, and P. Vogel, editors, Database and Expert Systems Applications (DEXA' 01), Munich, Germany, September 2001, Proceedings, Lecture Notes in Computer Science (LNCS), Vol. 2113, pages 646–656. Springer-Verlag, 2001.

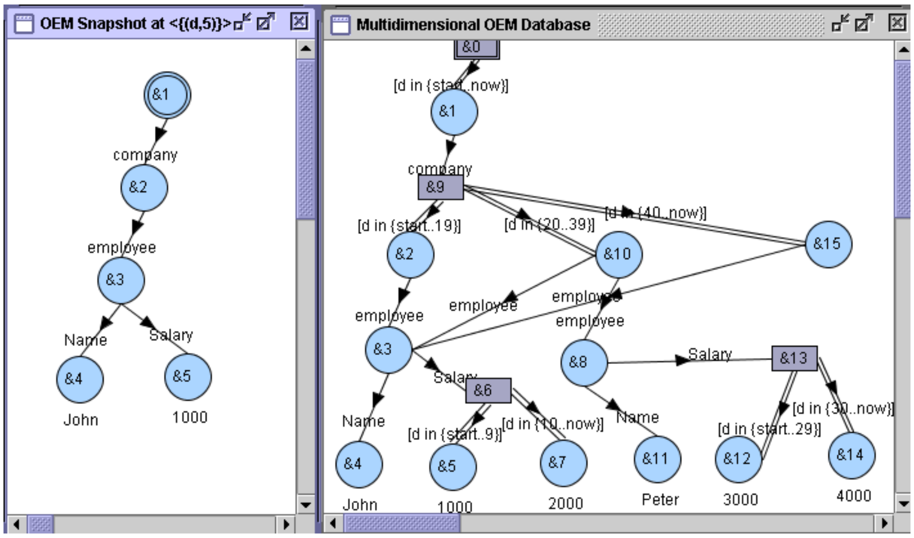

Fig. 9. The MOEM database after the update of Peter's salary to 4000 at d = 30 and the removal of employee Peter at d = 40. On the left, snapshot of the database at timestamp 5.

8. Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, and V. Zafeiris. Accomodating changes in semistructured databases using multidimensional OEM. In *Advances in Databases and Information Systems (ADBIS' 02), Proceedings*, Bratislava, Slovakia, September 2002 (to appear).
9. Y. Stavrakas and M. Gergatsoulis. Multidimensional Semistructured Data: Representing context-dependent information on the web. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering, Toronto, Ontario, Canada, May 2002*, Lecture Notes in Computer Science (LNCS), Vol. 2348, pages 183–199, Springer-Verlag, 2002.
10. D. Suciu. An overview of semistructured data. *SIGACT News*, 29(4):28–38, December 1998.