Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web

Yannis Stavrakas[1,2] and Manolis Gergatsoulis[2]

[1] Knowledge & Database Systems Laboratory, National Technical University of Athens (NTUA)
157 73, Athens, Greece

[2] Institute of Informatics & Telecommunications, National Centre for Scientific Research (N.C.S.R.) 'Demokritos'
153 10 Aghia Paraskevi Attikis, Greece
{ystavr,manolis}@iit.demokritos.gr

Abstract. In this paper, we address a problem common in the frame of WWW, namely, representing information that assumes different facets under different *contexts* (sets of *worlds*). For expressing context-dependent (or *multidimensional*) data, we introduce *Multidimensional Semistructured Data*, where context is defined through $(dimension, value)$ pairs. An extension of OEM called *Multidimensional Object Exchange Model* (*MOEM*) is introduced, for representing multidimensional data. We discuss the properties of MOEM and define a transformation that, for any given world, reduces MOEM to a conventional OEM holding under that world. As a case study, we show how MOEM can be used to represent changes over time in an OEM database.

1 Introduction and Motivation

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. Similar problems appear when integrating information from various sources [10], where the same conceptual entity may exhibit different structure, or contain conflicting data.

Those problems call for a way to represent information entities that manifest different facets, whose contents can vary in structure and value. As a simple example imagine a report that must be represented at various degrees of detail and in various languages. A solution would be to create a different report for every possible combination of variations. Such an approach is certainly not practical, since it involves excessive duplication of information. What is more, different variants are not associated as being parts of the same entity. Although existing web data models, such as OEM [7], are in principle capable to represent such multi-facet entities, they fall short for a number of reasons, namely (a) hidden semantics: by not addressing directly the issue of multiple facets, it is the responsibility of an application to assign semantics in an ad-hoc manner, (b) cumbersome notation: representing multiple facets of an entity cannot be done in an elegant way, and (c) duplication of information: information that is common can be duplicated, which is undesirable.

In this paper we introduce *multidimensional semistructured data* and an extension of OEM called *multidimensional OEM*, that incorporate ideas from multidimensional programming languages [3] and associate data with *dimensions*, in order to tackle the aforementioned problems. We show how multidimensional OEM can be reduced to OEM under a specific *world*. As an example application of multidimensional OEM, we consider the problem of representing histories of changes in an OEM database. This problem has been also investigated in [8], where the need for an OEM extension has been recognized. In Section 5 we discuss [8] further. A model for semistructured data that deviates from OEM has been proposed in [6], where edge labels are themselves pieces of semistructured data. The model we propose views labels as containing metadata rather than data. In [9], an extensible semistructured data model has been proposed that uses sets of properties of the form "property_name: property_value" as edge labels. By attaching properties at will, the graph becomes rich in metadata. Different properties may have different semantics, which results in a model of increased generality, but on the other hand makes the formulation of queries more complicated. The goal of our approach is to represent information that presents different facets. This leads to common semantics for the metadata, which is used solely to embody *context* information. In addition, our model retains OEM labeled edges and attaches context metadata to a new type of edges; graph models such as OEM become special cases of the model we propose.

Our work was influenced by *Intensional HTML* [16], a Web authoring language that incorporates ideas presented in [13] and allows a single Web page to have different variants and to dynamically adapt itself to a user-defined context. Our previous work on *Multidimensional XML* (MXML) [14,11,12] has also played a major role in shaping the data model we describe in this paper. MXML is an extension of XML that treats context as first class citizen. In MXML, elements and attributes can assume different values or structure, depending on the context. A multidimensional DTD (MDTD) has also been proposed for defining constraints on MXML documents.

In this paper, we address the representation of context-dependent semistructured data in general. We define an extended formalism for contexts, and investigate various aspects of a multidimensional model for semistructured data. The structure of the paper is as follows. In Section 2 we give a formalism for contexts. In Section 3 we introduce a graph data model and syntax for multidimensional semistructured data, we discuss validity issues of the proposed model, and define a process for obtaining conventional graphs from a multidimensional graph. Querying multidimensional semistructured data is briefly discussed in Section 4.

In Section 5, a way to model OEM histories using multidimensional OEM is presented. Finally, Section 6 concludes the paper.

2 A Formalism for Contexts

Multidimensional semistructured data (*MSSD* in short) are semistructured data (*SSD* in short) [15] which present different facets under different worlds. The notion of *world* is fundamental in our approach. 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 nonempty 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 semistructured data and multidimensional semistructured data is the introduction of *context specifiers*, that are used to qualify semistructured data expressions (*ssd-expressions*) [1] with contexts. The context of an ssd-expression can be seen as a set of worlds under which that ssd-expression holds; it becomes therefore 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 (also called *facets*) is called *multidimensional entity*. If the facets $e_1, e_2, \ldots, e_n$ of a multidimensional entity e hold under a world w (or, under every world defined by a context specifier c), then we say that e evaluates to $e_1, e_2, \ldots, e_n$ under w (under c, respectively).

In order to define context specifiers and explain their relation to worlds, we first discuss *context specifier clauses*.

Definition 2. *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 $\mathcal{V}_d \neq \emptyset$. Then a* dimension specifier *s of a dimension d is a pair (d, V) where $d \in \mathcal{D}$ and $V \in 2^{\mathcal{V}_d}$. A* context specifier clause *cc is a set of dimension specifiers, such that for any dimension $d \in \mathcal{D}$ there exists at most one dimension specifier (d, V) in cc.*

A context specifier clause cc is called *empty* and is denoted by $\emptyset_{cc}$, if for some dimension d, $(d, \emptyset) \in cc$. The empty context specifier clause $\emptyset_{cc}$ represents the empty set of worlds $\mathcal{E}_c$ (empty context). If $cc = \emptyset$, then cc is called *universal* context specifier clause, and represents the set of all possible worlds $\mathcal{U}_c$ (universal context). Any number of dimensions can be combined to form a context specifier clause, in other words, it is not necessary for a context specifier clause to contain an element (dimension specifier) for every possible dimension. The meaning of omitting dimensions becomes evident in what follows, where we explain how a context specifier clause is interpreted as a set of worlds.

Definition 3. *The* extension $\otimes cc$ *of a context specifier clause* cc *is defined as follows: if* $cc{=}\emptyset$*, then* $\otimes cc{=}\mathcal{U}_c$*; if* $cc{=}\emptyset_{cc}$*, then* $\otimes cc{=}\mathcal{E}_c$*; else if* $cc{=}\{(d_1,V_1),(d_2,V_2),\ \ldots,\ (d_n,V_n)\}$*, then* $\otimes cc = \{\ \{(d_1,v_1),(d_2,v_2),\ldots,(d_n,v_n)\}\mid v_i \in V_i$ *with* $1 \le i \le n\}$*.*

The extension of a context specifier clause gives a set of sets of *(dimension, value)* pairs. To determine whether each set of pairs represents a world or not, the set $\mathcal{D}$ of all the dimensions must be taken into account. For a dimension in $\mathcal{D}$ that does not exist in a context specifier clause, the inclusion of all values in its domain is implied. The following defines an expansion of context specifier clauses in order to take into account the set of dimensions $\mathcal{D}$.

Definition 4. *Let* $\mathcal{D}$ *be a set of dimensions and for each* $d \in \mathcal{D}$*, let* $\mathcal{V}_d$ *be the domain of* d*, with* $\mathcal{V}_d \neq \emptyset$*. Let* cc *be a context specifier clause. Then the* expansion of cc with respect to $\mathcal{D}$ *is a context specifier clause, denoted by* $exp(cc)$*, that contains the following elements: (a) if* $(d,V) \in cc$*, then* $(d,V) \in exp(cc)$*; (b) if* $d \in \mathcal{D}$ *and* $(d,V) \notin cc$ *for any* V*, then* $(d,\mathcal{V}_d) \in exp(cc)$*.*

Assuming a set of dimensions $\mathcal{D}$, the worlds specified by cc with respect to $\mathcal{D}$ are given by the extension of the expansion of cc, $W^{cc}_{\mathcal{D}}(cc) = \otimes(exp(cc))$. Consequently, (a) what represents a world with respect to a set of dimensions D, represents a set of worlds with respect to every $D' \supset D$, and (b) what represents a world with respect to a set of dimensions D, also represents a world with respect to every $D' \subset D$.

The intersection of two context specifier clauses cc_1 and cc_2, is a context specifier clause that represents the worlds specified by both cc_1 and cc_2.

Definition 5. *Let* cc_1*,* cc_2 *be two context specifier clauses. The* context specifier clause intersection $cc_1 \cap_{cc} cc_2$*, is a context specifier clause* cc_3 *such that:*

$cc_3 = \{(d,V) \mid (d,V) \in cc_1$ *and there is no element* (d,V') *in* $cc_2\} \cup \{(d,V) \mid (d,V) \in cc_2$ *and there is no element* (d,V') *in* $cc_1\} \cup \{(d,V) \mid (d,V_1) \in cc_1$ *and* $(d,V_2) \in cc_2$ *and* $V = V_1 \cap V_2\}$

Note that $cc \cap_{cc} \emptyset_{cc} = \emptyset_{cc}$, and $cc \cap_{cc} \emptyset = cc$.

Consider the dimension *language* ranging over *English*, *French*, *Greek*, the dimension *detail* ranging over *low*, *medium*, *high*, and the dimension *format* ranging over *ps*, *pdf*. Consider also the context specifier clauses $cc_1 = \{(lang,\{en,gr\}),(detail,\{medium,high\})\}$ and $cc_2 = \{(lang,\{gr,fr\})\}$. Then, $cc_1 \cap_{cc} cc_2 = \{(lang,\{gr\}),\ (detail,\{medium,high\})\}$, and represents the worlds: $\{(lang,gr),\ (detail,medium),\ (format,ps)\}$, $\{(lang,gr),\ (detail,medium),\ (format,pdf)\}$, $\{(lang,gr),\ (detail,high),\ (format,ps)\}$, $\{(lang,gr),\ (detail,high),\ (format,pdf)\}$.

Definition 6. *A* context specifier c *is a nonempty set of context specifier clauses.*

A context specifier $c = \{cc_1,cc_2,\ldots,cc_n\}$, $n \ge 1$, represents the set of worlds $W^{c}_{\mathcal{D}}(c) = W^{cc}_{\mathcal{D}}(cc_1) \cup W^{cc}_{\mathcal{D}}(cc_2) \cup \ldots \cup W^{cc}_{\mathcal{D}}(cc_n)$. In analogy to context specifier clauses, the *empty context specifier* $\emptyset_c$ is a context specifier that contains only empty clauses $cc_1 = cc_2 = \ldots = cc_n = \emptyset_{cc}$, and does not represent any world. A context specifier that contains at least one universal clause represents the set of all possible worlds, is called *universal context specifier*, and is denoted by $\{\emptyset\}$.

We now define how the intersection and union of worlds is performed at the level of context specifiers. The intersection of context specifiers $\cap_c$ is based on the intersection of clauses $\cap_{cc}$, while the union of context specifiers $\cup_c$ is not different from conventional set union, and is introduced for uniformity of notation.

Definition 7. *Let c_1, c_2 be two context specifiers. Then the* context specifier intersection $c_1 \cap_c c_2$*, is a context specifier c_3 such that: $c_3 = \{cc_i \cap_{cc} cc_j \mid cc_i \in c_1, cc_j \in c_2\}$. The* context specifier union $c_1 \cup_c c_2$*, is a context specifier c_4 such that: $c_4 = c_1 \cup c_2$.*

Consider the context specifiers $c_1 = \{\{(lang, \{en, gr\}),\ (detail, \{high\})\}\}$ and $c_2 = \{\{(lang, \{en\}),\ (detail, \{low\})\},\ \{(lang, \{gr\})\}\}$. Then $c_1 \cap_c c_2 = \{\emptyset_{cc}\ , \{(lang, \{gr\}), (detail, \{high\})\}\} = \{\{(lang, \{gr\}), (detail, \{high\})\}\}$, and $c_1 \cup_c c_2 = \{\{(lang, \{en, gr\}),\ (detail, \{high\})\},\ \{(lang, \{en\}),\ (detail, \{low\})\},\ \{(lang, \{gr\})\}\}$.

It is easy to show that the context specifier intersection and union are equivalent to the intersection and union of the corresponding sets of worlds. More formally, if c_1 and c_2 are context specifiers, then:

$W^c_{\mathcal{D}}(c_1) \cap W^c_{\mathcal{D}}(c_2) = W^c_{\mathcal{D}}(c_1 \cap_c c_2)$

$W^c_{\mathcal{D}}(c_1) \cup W^c_{\mathcal{D}}(c_2) = W^c_{\mathcal{D}}(c_1 \cup_c c_2)$

A context specifier may contain clauses that define overlapping sets of worlds. In the example above, the worlds matching $\{(lang, gr), (detail, high), \ldots\}$ are covered by the first and the third context specifier clause of c_4. The context specifier c_4 can be simplified as follows: $c_4 = \{\{(lang, \{en\}), (detail, \{high\})\}, \{(lang, \{gr\}), (detail, \{high\})\}, \{(lang, \{en\}), (detail, \{low\})\}, \{(lang, \{gr\})\}\} = \{\{(lang, \{en\}), (detail, \{high\})\}, \{(lang, \{en\}), (detail, \{low\})\},\ \{(lang, \{gr\})\}\} = \{\{(lang, \{en\}), (detail, \{low, high\})\}, \{(lang, \{gr\})\}\}$.

Definition 8. *Two context specifier clauses cc_1, cc_2 are said to be* mutually exclusive *iff $cc_1 \cap_{cc} cc_2 = \emptyset_{cc}$. Two context specifiers c_1, c_2 are said to be* mutually exclusive *iff $c_1 \cap_c c_2 = \emptyset_c$.*

Mutually exclusive context specifier clauses and context specifiers define disjoint sets of worlds. The context specifier clauses $cc_1 = \{(lang, \{en\})\}$ and $cc_2 = \{(detail, \{low\})\}$ are not mutually exclusive, since the worlds matching $\{(lang, en), (detail, low), \ldots\}$ are covered by both cc_1 and cc_2. In contrast, the context specifier clause $cc_3 = \{(lang, \{gr, fr\}), (detail, \{high\})\}$ is mutually exclusive with both cc_1 and cc_2.

Section 3.2 defines a syntax for context specifiers in ssd-expressions, which will be used throughout the paper. As an example, consider the expressions:

```
[time=07:45]
[language=greek, detail in {low,medium}]
[season in {fall,spring}, daytime=noon | season=summer]
```

The last context specifier contains two clauses, and represents the worlds where it is either summer or fall/spring noons. The universal context specifier is denoted by `[]` while the empty context specifier is denoted by `[-]`.

3 Supporting Contexts in SSD

In this section we propose a graph model for representing MSSD, specify a syntax for expressing multidimensional semistructured data, and discuss some properties of multidimensional data graphs.

3.1 Multidimensional OEM

A predominant graph model for SSD is *Object Exchange Model* (OEM) [2], that was originally designed in Stanford as part of the TSIMMIS project [7]. OEM is a rooted directed labeled multigraph, flexible enough to tolerate the irregularities of SSD. We retain that flexibility, and extend OEM with two new basic elements:

– *Multidimensional nodes*: a multidimensional node represents a multidimensional entity, and is used to group together nodes that constitute facets of that entity. Facets of entities can use multidimensional nodes to connect to each other, as a multidimensional node plays the role of a surrogate for its facets. In our graph model, multidimensional nodes have a rectangular shape to distinguish them from conventional circular nodes.
– *Context edges*: context edges are directed labeled edges that connect a multidimensional node to its variants. The label of a context edge pointing to a variant p, is a context specifier that defines the set of worlds under which p holds. Context edges are drawn as thick or double lines, to distinguish them from conventional edges.

We call the new model *Multidimensional Object Exchange Model* (*MOEM* in short). In MOEM the conventional circular nodes of OEM are called *context nodes* and represent variants associated with some context. Conventional (thin) OEM edges are called *entity edges* and define relationships between objects.

As in OEM, all MOEM nodes are considered objects, and have a unique *object identifier* (oid). In what follows, the terms *node* and *object* will be used interchangeably in the frame of MOEM. Context objects are divided into *complex objects* and *atomic objects*. Atomic objects have a value from one of the basic types, e.g. integer, real, strings, etc. The value of a complex object is a set of *object references*, represented by entity edges. The value of a multidimensional object is also a set of object references, represented by context edges.

The MOEM in Figure 1 is an example of a context-dependent recreation guide. For simplicity, the graph is not fully developed and some of the atomic objects do not have values attached. The dimensions and their respective domains in Figure 1 are as follows: `season` ranging over {`summer, fall, winter, spring`}, `daytime` ranging over {`noon, evening`}, `detail` ranging over {`high,`

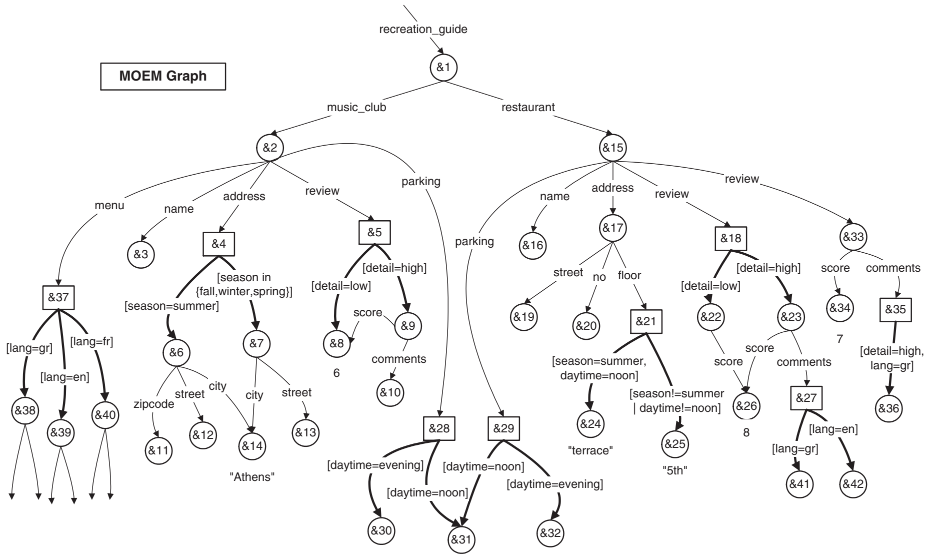

Fig. 1. A multidimensional recreation guide

low}, and lang ranging over {en, fr, gr}. The restaurant with oid &15 normally operates on the fifth floor, but at summer noons it operates on the terrace. Therefore, floor with oid &21 is a multidimensional object whose (atomic) value depends on dimensions season and daytime. Except from having a different value, context objects can have a different structure, as is the case of &6 and &7 which are variants of the multidimensional object address with oid &4. In this case, the music_club with oid &2 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). The menu of the club is available in three languages, namely English, French and Greek. The restaurant and the club have a number of reviews that can be detailed or brief, depending on the dimension detail. In addition, each has a couple of alternative parking places, depending on the time of day as expressed by the dimension daytime.

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.

Figure 2 depicts some legal non-trivial MOEM constructs. In Figure 2(b) more than one context edges connect a multidimensional node with the same context node. The multidimensional node A evaluates to B under the union of the worlds specified by $c1$ and $c2$. The two context edges can, therefore, be replaced by a single one with context specifier $c3 = c1 \cup_c c2$. Figure 2(c)

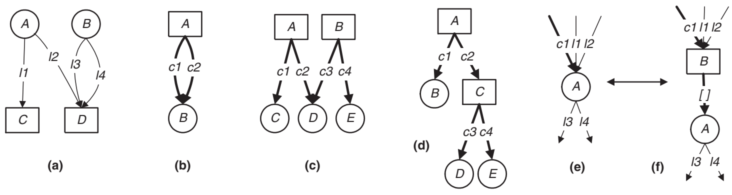

Fig. 2. Some interesting MOEM constructs

shows a context node D that is part of two multidimensional nodes, A and B. This is the case of the context object with oid `&31` in the example of Figure 1. A multidimensional entity may be part of another multidimensional entity, as demonstrated in Figure 2(d). The construct of Figure 2(e) is equivalent to that of Figure 2(f): from the perspective of $l1$ and $l2$, the context node A is assumed to be the only variant of the multidimensional entity represented by B, holding under every possible world.

We now give some formal definitions for the concepts that have been discussed in this section. We start by introducing *multidimensional data graphs.*

Definition 9. *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.*

It is easy to recognize that OEM is a special case of multidimensional data graph, where there are no multidimensional nodes and context edges.

An important issue is whether or not, given a specific world, it is always possible to reduce a multidimensional data graph to a conventional graph holding under that world. To be able to safely "disassemble" a multidimensional data graph, each multidimensional entity in the graph must evaluate to at most one variant under any world. This leads to the definition of *context deterministic* multidimensional data graphs, where the context specifiers of each multidimensional entity are mutually exclusive.

Definition 10. *A multidimensional data graph $G = (V, E, r, \mathcal{C}, \mathcal{L}, \mathcal{A}, v)$ is* context-deterministic *iff for every (p, c_1, q_1), (p, c_2, q_2) in E_{cxt}, with $q_1 \neq q_2$, $c_1 \cap_c$

$c_2 = \emptyset_c$. *An* MOEM graph *is a context-deterministic multidimensional data graph.*

Note that, in any case, a multidimensional entity may evaluate to no variant under some world(s). A context-deterministic graph merely assures that an entity cannot evaluate to more than one variants under any specific world.

For the rest of this paper we assume context-deterministic graphs. This does not imply that context-nondeterministic graphs are of no interest; however, the investigation of context-nondeterministic graphs is out of the scope of this paper.

3.2 MSSD-Expressions

As pointed out in [1], the cornerstone of SSD syntax is *ssd-expression*. We define *mssd-expression* by extending SSD syntax to incorporate context specifiers. The grammar of mssd-expression is given below in Extended Backus-Naur Form (EBNF), where symbols that can be defined by a regular expression start with a capital letter.

```
mssd-expr ::= value | Oid value | Oid
value ::= Atomicvalue | "{" complexvalue "}" | "(" multidimvalue ")"
complexvalue ::= Label ":" mssd-expr ("," complexvalue)?
multidimvalue ::= contspec ":" mssd-expr ("," multidimvalue)?
```

It is evident that `multidimvalue` corresponds to the multidimensional node of MOEM, while `atomicvalue` and `complexvalue` correspond to context nodes. Note that `multidimvalues` can have object identifiers, just like complex and atomic values. The conventions concerning the syntax of labels, atomic values, and object identifiers, as well as the requirements for the consistency of ssd-expressions [1] also hold for mssd-expressions.

A context specifier is of the form:

```
contspec ::= "[" contspecclause ("|" contspecclause)* "]"
contspecclause ::= "" | "-" | dimlist
dimlist ::= dimspec ("," dimspec)*
dimspec ::= dimname (atomicop Dimvalue | setop "{" setdimvalue "}")
atomicop ::= "=" | "!="
setop ::= "in" | "not in"
setdimvalue ::= Dimvalue ("," Dimvalue)*
```

As an example, consider the following mssd-expression that describes the `music_club` object with oid &2 in Figure 1:

```
&2 {menu: &37 ([lang=gr]: &38 {...},
              [lang=en]: &39 {...},
              [lang=fr]: &40 {...}),
    name: &3,
    address: &4 ([season=summer]:
                     &6 {zipcode: &11, street: &12, city: &14 "Athens"},
              [season in {fall,winter,spring}]:
                  &7 {city: &14, street: &13}),
   review: &5 ([detail=low]: &8 6,
              [detail=high]:
                  &9 {score: &8, comments: &10}),
   parking: &28 ([daytime=evening]: &30,
                 [daytime=noon]: &31)
  }
```

In this paper we assume finite dimension domains, which the proposed syntax describes by enumerating their elements. Other ways of representation as well as infinite domains may be useful and are not excluded, they are, however, out of the scope of this paper. In Section 5.2 we introduce a shorthand for representing intervals over a bounded, discrete, and totally ordered time domain.

3.3 Properties of Multidimensional Data Graphs

Multidimensional data graphs present interesting properties, a discussion of which cannot be exhausted in the present section. In what follows, we introduce some basic concepts starting with the definitions of *explicit context* and *inherited context*.

Definition 11. *Let $G = (V, E, r, \mathcal{C}, \mathcal{L}, \mathcal{A}, v)$ be a multidimensional data graph. The* explicit context *of an edge $h = (p, k, q) \in E$ is given by the context specifier ec, defined as follows: if $h \in E_{cxt}$ then $ec = k$; otherwise, $ec = \{\emptyset\}$.*

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 a multidimensional 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 to 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 (constrained 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 multidimensional graph to a conventional graph (as explained later in this section).

Definition 12. *Let $G = (V, E, r, \mathcal{C}, \mathcal{L}, \mathcal{A}, v)$ be a multidimensional data graph, ic_r be a context specifier giving the inherited context of the root r, and p, q be nodes in V with $p \neq r$. The* inherited context of node *p is given by the context specifier $ic_p = ic_1 \cup_c ic_2 \ldots \cup_c ic_n$, with $n \geq 1$, where $ic_1, ic_2, \ldots, ic_n$ give the inherited contexts of the edges in E that lead to p. Let ic_q be a context specifier giving the inherited context of node q, h be an edge in E that departs from q, and*

ec_h be a context specifier giving the explicit context of h. The inherited context *of edge h is the least set of worlds given by a context specifier ic_h, such that $ic_h = ic_q \cap_c ec_h$.*

If the root r of an MOEM graph G is assumed to hold under every possible world, the inherited context of the root becomes the universal context. A point that requires attention, is that the inherited context of an edge which constitutes part of a cycle is eventually defined in terms of itself. It is easy, however, to show that in such cases there exists a least fixed point, which gives the inherited context of the edge.

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 validity of a multidimensional data graph ensures that edges pointing to multidimensional nodes do not exist in vain. As an example, consider the MOEM in Figure 3(a), which is a variant of a part of the MOEM in Figure 1 where the context specifier c of the context edge (`&18`, c, `&23`) has been changed to `[detail=high,lang=fr]`. Now, the entity edge (`&23`, "`comments`", `&27`) does not hold under any world and is *invalid*, since there does not exist any world under which `&23` holds together with one of `&41`, `&42`. If node `&27` pointed also to a third context node through a context edge with explicit context `[lang=fr]`, the entity edge in question would be valid.

Definition 13. *Let $G = (V, E, r, \mathcal{C}, \mathcal{L}, \mathcal{A}, v)$ be a multidimensional data graph. Let $h = (p, k, q)$ be an edge in E that leads to a node q in V, let ic_h give the inherited context of h, and let $ec_1, \ldots, ec_n$, with $n \geq 1$, give the explicit contexts of the edges in E that depart from q. Then, the edge h is* invalid *iff $ic_h \neq \emptyset_c$ and $ic_h \cap_c (ec_1 \cup_c \ldots \cup_c ec_n) = \emptyset_c$. The multidimensional data graph G is* valid *iff none of its edges is invalid.*

Each MOEM comprises a number of conventional OEM graphs. 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 Initialize G_w to G. With G_w do the following.

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

Step 2: For every edge $(p, l, m_1) \in E_{ett}$ with $m_1 \in V_{mld}$, 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 $q \in V_{cxt}$ add a new entity edge (p, l, q) in E_{ett}.

Step 3: Remove all multidimensional nodes in V_{mld}. Remove all edges in E departing from or leading to the removed nodes. □

Intuitively, explicit contexts can be seen as defining constraints that are accumulated from the root to the leaves to form inherited contexts. Inherited contexts

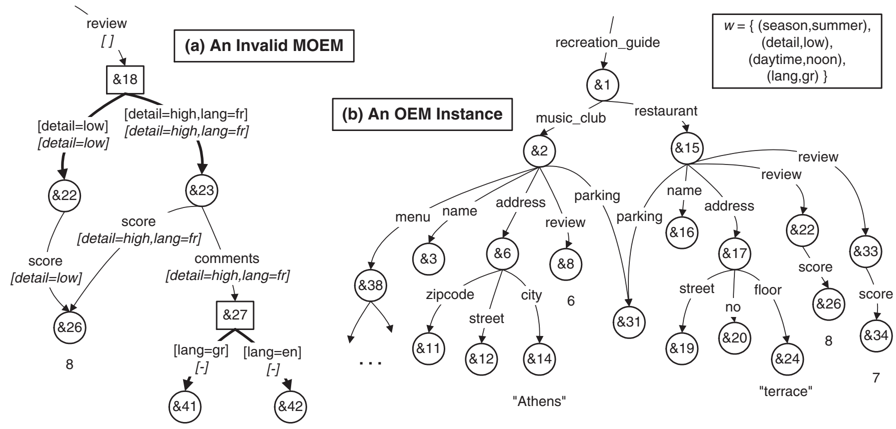

Fig. 3. Figure 3(a) depicts an invalid MOEM, annotated with the inherited contexts of edges (second line of labels), and Figure 3(b) the OEM instance, holding under the world w, of the MOEM in Figure 1

are used to identify parts of the graph that are unreachable under some context, and that can be removed in order to obtain the facet corresponding to a specific world. As an example, consider the MOEM in Figure 1. By applying the above steps on the MOEM for the world $w = \{(season, summer), (detail, low), (daytime, noon), (lang, gr)\}$, we get the OEM in Figure 3(b). Notice how the two `parking` multidimensional entities with oids `&28` and `&29` are represented by the object with oid `&31`. Also, notice how the `comment` object with oid `&35` is excluded from the resulting OEM.

4 Querying Multidimensional Data Graphs

An issue that arises is how to query [5,2] multidimensional semistructured data. In this section, we discuss briefly what a "multidimensional" query is, mention the directions of our ongoing work, and argue that an MOEM graph is something more than the sum of the OEMs it can be "decomposed" to.

Similarly to an OEM database, we define an *MOEM database* as a database whose model is an MOEM graph. Suppose that an MOEM database M is reduced to an OEM database O_w under the world w. Then, a "multidimensional" query $q = (q_w, w)$ on M can be expressed as a query q_w on O_w. For example, consider the query q "*give me the addresses of restaurants at summer noons in low detail in Greek*" on the MOEM database M of Figure 1. Then q is equivalent to q_w "*give me the addresses of restaurants*" on the OEM facet O_w of M for $w = \{(season, summer), (detail, low), (daytime, noon), (lang, gr)\}$.

However, reducing M to O_w is not a necessary step for evaluating q; the processing of q can take place directly on M. Intuitively, q_w can be used for navigating through MOEM entity edges, while w can guide the navigation through MOEM context edges.

In addition, the fact that multidimensional data graphs group variants of entities together, allows a "cross-world" type of queries. As an example, consider the `music_club` in Figure 1, and the query: "*give me the name and the address in winter of a club whose summer address is given*". We believe that such queries show the potential of multidimensional data graphs, and that query processing for multidimensional data graphs is an interesting research direction.

5 Using MOEM to Represent Changes

In this section, we will give an example of how MSSD can be applied to a tangible problem: we will use MOEM to represent changes in an OEM database. In short, the problem can be stated as follows: given a static OEM graph that comprises the database, we would like a way to represent dynamically changes in the database as they occur, keeping a history of transitions, so that we are able to subsequently query those changes.

The problem of representing and querying changes in SSD has been studied in [8] where *Delta OEM*, which extends OEM with annotations, is proposed. Our approach, although quite different, is based on the same framework, which we outline in Section 5.1. The approach that we propose will be developed in Section 5.2. An important advantage of MOEM is that a single model can be applied to a variety of problems from different fields; representing valid time is a problem that we discuss here as a case study.

5.1 Basic Concepts

In order to modify an OEM database O, four basic change operations were identified in [8]:

creNode(*nid*, *val*): creates a new node, where nid is a new node oid ($nid \notin V$), and val is an atomic value or the reserved value C.

updNode(*nid*, *val*): changes the value of an existing object nid to a new value val. The node nid must not have any outgoing arcs (in case its old value is C, the arcs should have been removed prior to updating the value).

addArc(*p*, *l*, *q*): adds a new arc labeled l from object p to object q. Both nodes p and q must already exist in V, and (p, l, q) must not exist in E.

remArc(*p*, *l*, *q*): removes the existing arc (p, l, q). Both nodes p and q must exist in V.

Arc removals can be used for deleting objects, as in OEM the persistence of an object is determined by whether or not the object is reachable from the root. Sometimes the result of a single basic operation u leads to an inconsistent state: for instance, when a new object is created, it is temporarily unreachable from the root. In practice however, it is typical to have a sequence $L = u_1, u_2, \ldots, u_n$ of basic operations u_i, which corresponds to a higher level modification to the database. By associating such higher level modifications with a timestamp, an OEM history H is defined as a sequence of pairs (t, U), where U denotes a set of basic change operations that corresponds to L as defined in [8], and t is the associated timestamp. Note that within a single sequence L, a newly created node may be unreachable from the root and still not be considered deleted. At the end of each sequence, however, unreachable nodes are considered deleted and cannot be referenced by subsequent operations.

5.2 Modeling OEM Histories with MOEM

Given an MOEM database M, the following MOEM basic operations are introduced: **createCNode** for creating a new context node, **updateCNode** for changing the value of an atomic context node, **createMNode** for creating a new multidimensional node, **addEEdge** for creating a new entity edge, **remEEdge** for removing an entity edge, **addCEdge** for creating a new context edge, and **remCEdge** for removing a context edge.

We will now use the framework outlined in the previous section and the MOEM operations introduced above to represent changes in an OEM database using MOEM. Our approach is to map the four OEM basic change operations to MOEM basic operations, in such a way, that new variants of an object are created whenever changes occur in that object. In this manner, the initial OEM database O is transformed into an MOEM graph, that uses a dimension d whose domain is time to represent an OEM history H valid [8] for O. We assume that our time domain T is linear and discrete; we also assume: (1) a reserved value *now*, such that $t < now$ for every $t \in T$, (2) a reserved value *start*, representing the start of time, and (3) a syntactic shorthand $v_1..v_n$ for discrete and totally ordered domains, meaning all values v_i such that $v_1 \leq v_i \leq v_n$. The dimension d denotes, for each context node it qualifies, the time period during which this context node is the holding node of the corresponding multidimensional entity.

Figure 4 gives an intuition about the correspondence between OEM and MOEM operations. Consider the sets U_1 and U_2 of basic change operations, with timestamps t_1 and t_2 respectively. Figure 4(a) shows the MOEM representation of an atomic object, whose value “A” is changed to “B” through a call to the basic change operation *updNode* of U_1. Figure 4(b) shows the result of *addArc* operation of U_1, while Figure 4(c) shows the result of *remArc* operation of U_2, on the same multidimensional entity. It is interesting to notice that three of the four OEM basic change operations are similar, in that they update an object be it atomic (*updNode*) or complex (*addArc*, *remArc*), and all three are mapped to MOEM operations that actually update a new facet of the original object. Creating a new node with *creNode* does not result in any additional MOEM operations; the new node will subsequently be linked with the rest of the graph (within the same set U) through *addArc* operation(s), which will cause new object variant(s) to be created. It is worth noting that the changes induced by the OEM basic change operations affect only localized parts of the MOEM graph, and do not propagate throughout the graph.

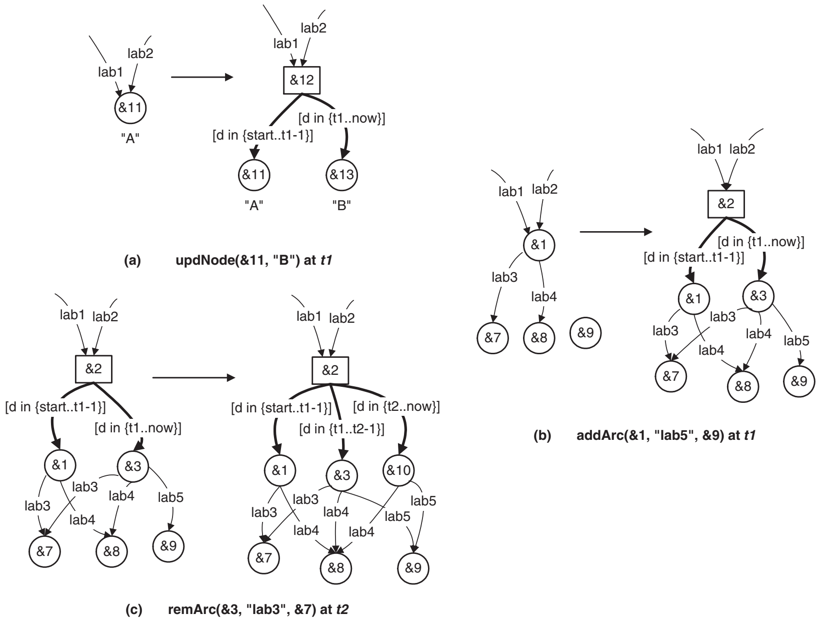

Fig. 4. Modeling OEM basic change operations with MOEM

Based on the above, each OEM basic change operation can be mapped to a procedure implemented through calls to MOEM basic operations, thus defining a process for encompassing an OEM History into an MOEM database. Given an MOEM database M created through such a process, it is possible to specify a time instant and get an OEM database O which is a temporal instantiation of M. In other words, a time instant t is a world for M and we can apply the process described in Section 3.3 to reduce M to an OEM holding under t.

6 Conclusions

In this paper, we presented a formalism for *contexts*, we introduced *multidimensional semistructured data*, specified their syntax, and proposed a new data model called *multidimensional OEM*, which is a graph model that extends OEM by incorporating *dimensions*. We defined the concept of a *multidimensional data graph*, gave a validity criterion for that graph, and specified a process for reducing multidimensional OEM to a conventional OEM under a specific *world*. As a case study of multidimensional OEM, we showed how it can be used to represent the history of an OEM database.

The implementation of the above comprises two applications: "MSSDesigner" that allows to design, validate and reduce MOEM graphs, and "OEM History" that models the history of an OEM database and allows to get OEM temporal instantiations. Both applications can be reached at:

`http://www.dblab.ntua.gr/∼ys/moem/moem.html`

We believe that MOEM has a lot more 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*.

Acknowledgements

Special thanks to Prof. Timos Sellis and Prof. Panos Rondogiannis, for their valuable comments and suggestions. Also, to Christos Doulkeridis and Vassilis Zafeiris for implementing the ideas presented in this paper.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, 2000. 185, 191
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. 188, 194
3. E. A. Ashcroft, A. A. Faustini, R. Jagannathan, and W. W. Wadge. *Multidimensional Programming*. Oxford University Press, 1995. 184
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. 183
5. P. Buneman, M. Fernandez, and D. Suciu. UnQL: A Query Language and Algebra for Semistructured Data Based on Structural Recursion. *The VLDB Journal*, 9(1):76–110, 2000. 194
6. Peter Buneman, Alin Deutsch, and Wang-Chiew Tan. A Deterministic Model for Semistructured Data. In *Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1998. 184
7. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The TSIMMIS project: Integration of Heterogeneous Information Sources. In *Proceedings of IPSJ Conference, Tokyo, Japan*, pages 7–18, October 1994. 184, 188
8. S. S. Chawathe, S. Abiteboul, and J. Widom. Managing Historical Semistructured Data. *Theory and Practice of Object Systems*, 24(4):1–20, 1999. 184, 195, 196
9. C. E. Dyreson, M. H. Böhlen, and C. S. Jensen. Capturing and Quering Multiple Aspects of Semistructured Data. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 290–301, 1999. 184
10. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997. 183

11. M. Gergatsoulis, Y. Stavrakas, and D. Karteris. Incorporating Dimensions to XML and DTD. In *Database and Expert Systems Applications (DEXA' 01)*, Munich, Germany, September 2001, LNCS Vol. 2113, pages 646–656. 184
12. M. Gergatsoulis, Y. Stavrakas, D. Karteris, A. Mouzaki, and D. Sterpis. A Web-based System for Handling Multidimensional Information through MXML. In *Advances in Databases and Information Systems (ADBIS' 01)*, September 2001, LNCS Vol. 2151, pages 352–365. 184
13. J. Plaice and W. W. Wadge. A New Approach to Version Control. *IEEE Transactions on Software Engineering*, 19(3):268–276, 1993. 184
14. Y. Stavrakas, M. Gergatsoulis, and T. Mitakos. Representing Context-Dependent Information Using Multidimensional XML. In *Research and Advanced Technology for Digital Libraries, 4th European Conference ECDL'2000*, LNCS 1923, pages 368–371, 2000. 184
15. D. Suciu. An Overview of Semistructured Data. *SIGACT News*, 29(4):28–38, December 1998. 185
16. W. W. Wadge, G. D. Brown, M. C. Schraefel, and T. Yildirim. Intensional HTML. In *Proceedings of the Fourth International Workshop on Principles of Digital Document Processing (PODDP '98)*, March 1998, LNCS 1481, pages 128–139. 184