# Implementing a Query Language for Context-dependent Semistructured Data

Yannis Stavrakas, Kostis Pristouris, Antonis Efandis, and Timos Sellis

National Technical University of Athens, 15773 Athens, Greece
{ys, timos}@dblab.ntua.gr
{kprist, aefan}@freemail.gr

**Abstract.** In today's global environment, the structure and presentation of information may depend on the underlying *context* of the user. To address this issue, in previous work we have proposed *multidimensional semistructured data* (*MSSD*), where an information entity can have alternative variants, or *facets*, each holding under some *world*, and *MOEM*, a data model suitable for representing MSSD. In this paper we briefly present *MQL*, a query language for MSSD that supports *context-driven queries*, and we attempt to motivate the direct use of context in data models and query languages by comparing MOEM and MQL with equivalent, context-unaware forms of representing and querying information. Specifically, we implemented an evaluation process for MQL during which MQL queries are translated to equivalent Lorel queries, and MOEM databases are transformed to corresponding OEM databases. The comparison between the two query languages and data models demonstrates the benefits of treating context as first-class citizen. We illustrate this query translation process using a *cross-world* MQL query, which has no direct counterpart in context-unaware query languages and data models.

## 1 Introduction

The Web posed a number of new problems to the management of data, and the need for metadata at a semantic level was soon realized [9]. One such problem is that, 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 and query languages. A related issue is that information providers often need to manage different variations of essentially the same information, which are targeted to different consumer groups.

Those problems call for a way to represent and query information entities that manifest different facets, whose contents can vary in structure and value. As a simple example imagine a product (car, laptop computer, etc.) whose specification changes according to the country it is being exported. Or a Web page that is to be displayed on devices with different capabilities, like mobile phones, PDAs, and personal com-

puters. Another example is a report that must be represented at various degrees of detail and in various languages.

In previous work we proposed *multidimensional semistructured data* (*MSSD*) [2,14,16], which are semistructured data [3] that present different *facets* under different *contexts*. We argued [4] that Web data should be able to adapt to different *contexts*, and that this capability should be managed in a uniform way at the level of database and information systems. Context-aware data models and query languages can be applied on a variety of cases and domains; in [6,7,15] we showed how they can be used to represent and query histories of semistructured databases that evolve over time.

Context has been used in diverse areas of computer science as a tool for reasoning with viewpoints and background beliefs, and as an abstraction mechanism for dealing with complexity, heterogeneity, and partial knowledge. A formal framework for reasoning upon a subset of a global knowledge base can be found in [17], while examples of how context can be used for partitioning an information base into manageable fragments of related objects can be found in [8,13]. Our perception of context has also been used in *OMSwe*, a web-publishing platform described in [1,12]. OMSwe is based on an Object DBMS, which has been extended to support a flexible, domain-independent model for information delivery where context plays a pivotal role.

In this paper, we give a short overview of *Multidimensional Query Language* (*MQL*) [4,7] that treats context as first-class citizen and can express *context-driven queries*, in which context is important for selecting the right data. MQL is based on key concepts of Lorel [5], and its data model is *Multidimensional OEM* (*MOEM*) [2], an extension of OEM [3] suitable for MSSD. We present an evaluation process for MQL queries that we have implemented in a prototype system. As part of this evaluation process, MQL queries are translated to "equivalent" Lorel queries, and MOEM databases are transformed to corresponding OEM databases. Our purpose is to intuitively compare the two query languages and data models: although OEM and Lorel are not aware of the notion of context, they are in principle capable of handling context-dependent information encoded in a graph. Through this comparison, we demonstrate the benefits of directly supporting context as first-class citizen: MQL and MOEM are much more elegant and expressive when context is involved, while they become as simple as Lorel and OEM when context is not an issue. The query translation process is illustrated using a *cross-world* MQL query. Cross-world queries relate facets of information that hold under different worlds, and have no counterpart in context-unaware query languages and data models.

The paper is structured as follows. Section 2 reviews preliminary material on MSSD. Section 3 introduces MQL. Section 4 presents in detail the MQL evaluation process: the transformation of MOEM to OEM is specified first, and then the translation of MQL to Lorel is explained. Finally, Section 5 summarizes the conclusions.

## 2 Multidimensional Semistructured Data

The main difference between conventional and multidimensional semistructured data is the introduction of *context specifiers*. Context specifiers are syntactic constructs that

are used to qualify pieces of semistructured data and specify sets of *worlds* under which those pieces hold. In this way, it is possible to have 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. Each facet is associated with a context that defines the conditions under which the facet becomes a *holding facet* of the multidimensional entity.

### 2.1 Dimensions and Worlds

The notion of *world* is fundamental in MSSD. A world is specified using parameters called *dimensions*, and represents an environment under which data obtain a substance. The notion of world is defined [2] with respect to a set of dimensions **D** and requires that every dimension in **D** be assigned a single value.

In MSSD, sets of worlds are represented by *context specifiers* (or simply *contexts*), which are constraints on dimension values. The use of dimensions for representing worlds is shown through the following context specifiers:

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

Context specifier (a) represents the worlds for which the dimension `time` has the value `07:45`, 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`. For a set of (*dimension*, *value*) pairs to represent a world with respect to a set of dimensions **D**, it must contain exactly one pair for each dimension in **D**. Therefore, if **D** = `{language, detail}` with domains $\mathbf{V_{language}}$ = `{english, greek}` and $\mathbf{V_{detail}}$ = `{low, medium, high}`, then `{(language, greek), (detail, low)}` is one of the six possible worlds with respect to **D**. This world is represented by context specifier (b), together with the world `{(language, greek), (detail, medium)}`. It is not necessary for a context specifier to contain values for every dimension in **D**. Omitting a dimension implies that its value may range over the whole domain.

The context specifier `[]` is a *universal context* and represents the set of all possible worlds with respect to any set of dimensions **D**, while the context specifier `[-]` is an *empty context* and represents the empty set of worlds with respect to any set of dimensions **D**. In [2,4] we have defined operations on context specifiers, such as *context intersection* and *context union* that correspond to the conventional set operations of intersection and union on the related sets of worlds. We have also defined how a context specifier can be transformed to the set of worlds it represents with respect to a set of dimensions **D**. Moreover, *context equality* and *context subset* allow to compare contexts based on their respective set of worlds.

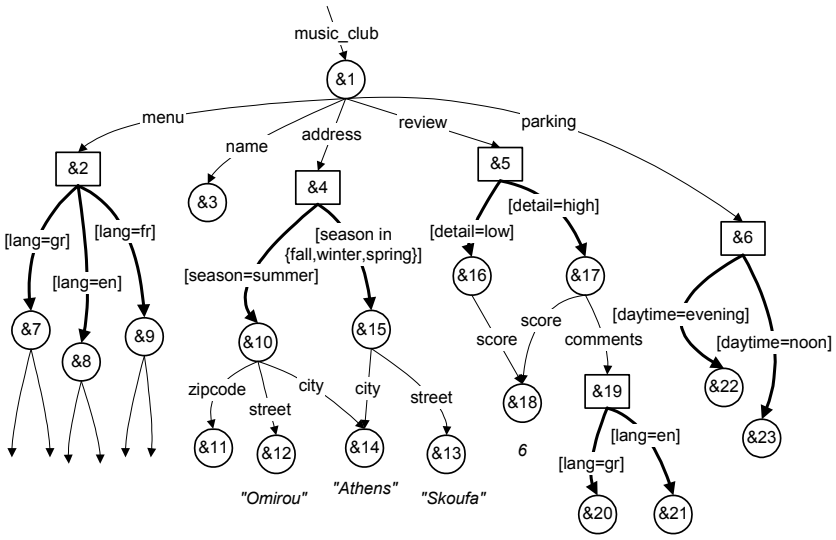


**Fig. 1.** A multidimensional music-club

### 2.2 Multidimensional OEM

*Multidimensional Data Graph* [4] is an extension of *Object Exchange Model* (OEM) [3,5], suitable for representing multidimensional semistructured data. Multidimensional Data Graph extends OEM with two new basic elements:

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

In Multidimensional Data Graph the conventional circular nodes of OEM are called *context nodes* and represent facets associated with some context. Conventional OEM edges (thin-lined) are called *entity edges* and define relationships between objects. All nodes are considered objects, and have unique *object identifiers* (*oid*s). 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. 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 Multidimensional Data Graph.

As an example, consider the simple Multidimensional Data Graph in Figure 1, which represents context-dependent information about a music club. For simplicity, 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 usual 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 facets 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`. The music-club review has two facets: node `&16` is the low detail facet containing only the review score with value `6`, while the high detail facet `&17` contains in addition review comments in two languages. In what follows we formally define Multidimensional Data Graph.

*Let CS be the set of all context specifiers, L be the set of all labels, and A be the set of all atomic values. A* ***Multidimensional Data Graph*** *G is a finite directed edge-labeled multigraph* $G = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$*, where:*

1. *The set of nodes V consists of* ***multidimensional nodes*** *and* ***context nodes****,* $V = V_{mld} \cup V_{cxt}$*. Context nodes are divided into* ***complex nodes*** *and* ***atomic nodes****,* $V_{cxt} = V_c \cup V_a$*.*
2. *The set of edges E consists of* ***context edges*** *and* ***entity edges****,* $E = E_{cxt} \cup E_{ett}$*, such that* $E_{cxt} \subseteq (V_{mld} \times CS \times V)$ *and* $E_{ett} \subseteq (V_c \times L \times V)$*.*
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* $v': V_a \rightarrow A$ *which assigns values to atomic nodes.*

Two fundamental concepts related to Multidimensional Data Graphs are *explicit context* and *inherited context* [2,4]. The explicit context of a context edge is the context specifier assigned to that edge, while the explicit context of an entity edge is considered to be 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 a 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. The inherited context of a node is the union of the inherited contexts of incoming edges (the inherited context of the root is `[]`). As an example, node `&18` in Figure 1 has inherited context `[detail in {low, high}]`. Worlds where `detail` is

`low` are inherited through node `&16`, while worlds where `detail` is `high` are inherited through node `&17`.

In Multidimensional Data Graph leaves are not restricted to atomic nodes, and can be complex or multidimensional nodes as well. This raises the question under which worlds does a path lead to a leaf that is an atomic node. Those worlds are given by *context coverage*, which is symmetric to inherited context, but propagates to the opposite direction: from the leaves up to the root of the graph. The context coverage of a node or an edge represents the worlds under which the node or edge has access to leaves that are atomic nodes. The context coverage of leaves that are atomic nodes is `[]`, while the context coverage of leaves that are complex nodes or multidimensional nodes is `[-]`. The context coverage of node `&19` in Figure 1 is `[lang in {gr, en}]` (all leaves in Figure 1 are considered atomic nodes).

For every node or edge, the context intersection of its inherited context and its context coverage gives the *inherited coverage* of that node or edge. The inherited coverage represents the worlds under which a node or edge may actually hold, as determined by constraints accumulated from both above and below. A related concept is *path inherited coverage*, which is given by the context intersection of the inherited coverages of all edges in a path, and represents the worlds under which a complete path holds.

A *context-deterministic* Multidimensional Data Graph is a Multidimensional Data Graph in which context nodes are accessible from a multidimensional node under mutually exclusive inherited coverages (hold under disjoint sets of worlds). Intuitively, context-determinism means that, under any specific world, at most one context node is accessible from a multidimensional node. A *Multidimensional OEM*, or *MOEM* for short, is a context-deterministic Multidimensional Data Graph whose every node and edge has a non-empty inherited coverage. In an MOEM all nodes and edges hold under at least one world, and all leaves are atomic nodes. The Multidimensional Data Graph in Figure 1 is an MOEM.

Given a world *w*, it is possible to *reduce* an MOEM *to a conventional OEM* graph holding under *w*, by eliminating nodes and edges whose inherited coverage does not contain *w*. A process that performs such a *reduction to OEM* is presented in [2]. In addition, given a set of worlds, it is possible to *partially reduce* an MOEM into a new MOEM, that encompasses exactly the OEM facets for the given set of worlds.

## 3 Multidimensional Query Language

*Multidimensional Query Language* (*MQL*) [4], is a query language designed especially for MOEM databases, and is essentially an extension of Lorel [5]. An important feature of MQL is *context path expressions*, which are path expressions qualified with context specifiers and *context variables*. Context path expressions take advantage of the fact that every Multidimensional Data Graph can be transformed to a *canonical form* [4,7], where every context node is child of solely multidimensional node(s), and vice-versa. The canonical form of the MOEM in Figure 1 contains an additional multidimensional node which is pointed by the entity edge labeled `name`, and whose only

facet under every possible world (explicit context `[]`) is the context node `&3`. It also contains similar multidimensional nodes for the context nodes `&11`, `&12`, `&13`, `&14`, `&18`, and `&1` (the root of a graph in canonical form is always a multidimensional node). If a graph is in canonical form, every possible path is formed by a repeated succession of one context edge and one entity edge. Context path expressions are built around the canonical form, and therefore consist of a number of *entity parts* and *facet parts* succeeding one another. Entity parts follow a dot (`.`) and are matched against entity edges, while facet parts follow a double colon (`::`) and are matched against context edges:

```
[detail=high]music_club::[-].review::[-]  X
```

In this context path expression, `music_club` and `review` are entity parts, while the two empty context specifiers `[-]` are facets parts. A facet part matches a corresponding context edge, if it is subset of the explicit context of the edge, in other words, if every world it defines is covered by the explicit context of the edge. Consequently, the empty context `[-]` as a facet part matches any context edge. The context specifier `[detail=high]` is an *inherited coverage qualifier* and is matched against the *path inherited coverage* of a path. For a path to match an inherited coverage qualifier, it must hold under every world specified by the qualifier. An inherited coverage qualifier may precede any entity part or facet part in a context path expression. Facet parts can often be omitted, implying the empty context `[-]`. Therefore, the above context path expression can also be written as:

```
[detail=high]music_club.review  X
```

Evaluated on the graph of Figure 1, this context path expression causes the *context object variable* `X` to bind to node `&17`. Had we used a *multidimensional object variable*, denoted `<X>`, we would have caused it to bind to the multidimensional node `&5`.

Consider the following MQL query:

```
select name: P, winter_street: Y
from music_club X,
  X.[season=winter]address.street Y,
  X.[season=summer]address.street Z,
  X.name P
where Z="Omirou"
```

This is a *cross-world* query, which returns the name and the street address in winter of a music club whose summer address is known. Evaluated on the database of Figure 1, variable `P` binds to node `&3` and `Y` binds to `&13`. The result of an MQL query is always a Multidimensional Data Graph in the form of an *mssd-expression* [2].

## 4 Evaluating MQL Queries with LORE

We have implemented MQL on top of LORE [10], analogously to Lorel, which has been implemented on top of an object database [5]. We have chosen LORE as a basis for implementing MQL, because our purpose was to see: (a) how an MQL query com-

pares with an "equivalent" Lorel query, and (b) how an MOEM can be expressed through a conventional OEM.

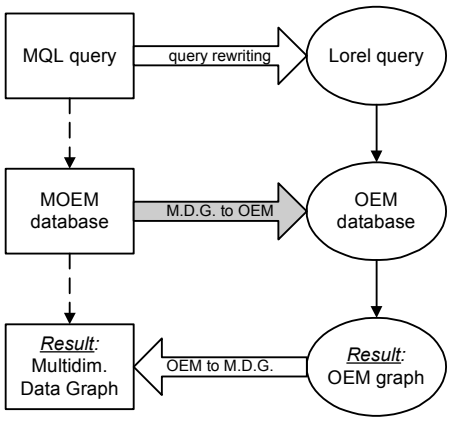


**Fig. 2.** Evaluating MQL queries using LORE

The overall architecture is shown in Figure 2. The process we want to implement is depicted as a dashed line, which starts from an MQL query, passes through an MOEM database, and concludes with a Multidimensional Data Graph that is the result of the query.

The process that actually takes place is depicted as a normal line, and shows a Lorel query evaluated on an OEM database that returns an OEM graph as a result. This line together with the ellipse-shaped boxes is part of LORE, which is controlled by our system through the programming interface that it provides.

The main issue is to define a transformation T from Multidimensional Data Graphs M to OEMs O = T(M), with the following properties:

- The reverse transformation $T^{-1}$ exists, and if O is given then $M = T^{-1}(O)$ can be recovered.
- It is possible to translate an MQL query $q_M$ to an "equivalent" Lorel query $q_L$.

By equivalent we mean that if $q_M$ evaluated on M returns M′ and $q_L$ evaluated on O returns O′, then T(M′) = O′. Then, the answer to $q_M$ can be computed by evaluating $q_L$ on T(M), and by applying the reverse transformation $T^{-1}(O')$ to the results of $q_L$.

Those transformations and the MQL query translation are depicted in Figure 2 as thick horizontal arrows. The system, among other things, implements those arrows and performs the following key steps:

1. Converts an MOEM database to an OEM database, which becomes the database of LORE.
2. Translates an MQL query to a Lorel query, which is passed over to LORE for evaluation on the OEM database.
3. Gets the results from LORE, and converts them from OEM back to Multidimensional Data Graph.

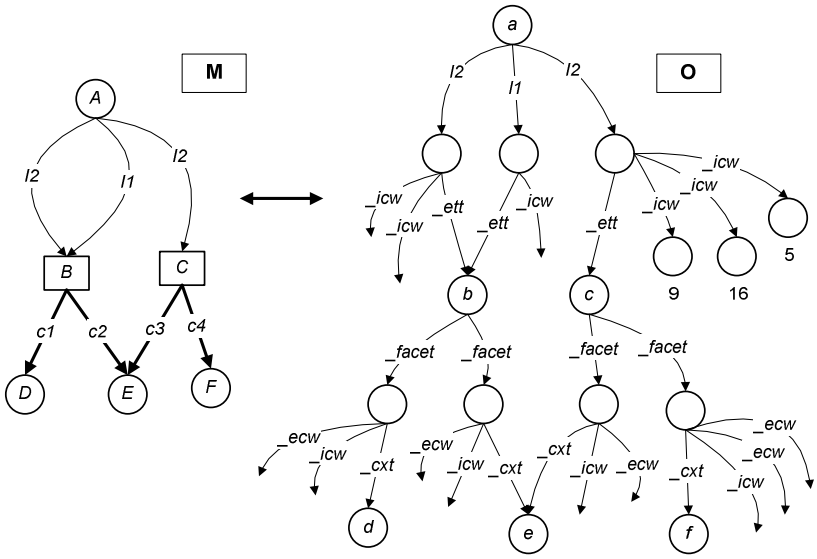


**Fig. 3.** Representing Multidimensional Data Graph using OEM

Step 1 initializes the database and corresponds to the gray horizontal arrow in the middle of Figure 2, while steps 2 and 3 are carried out every time an MQL query is submitted. The Multidimensional Data Graph of step 3 is the result of the MQL query of step 2 evaluated on the MOEM database of step 1.

In the following sections, we specify the three key steps listed above. The actual application that implements them is part of a more comprehensive platform [11] for MSSD, and is presented in Section 4.4.

### 4.1 Transforming MOEM Databases to OEM

In order to transform MOEM to OEM, we must use special OEM structures to represent MOEM elements that do not have a counterpart in OEM, namely context edges and multidimensional nodes: context edges can be represented by OEM edges that have some special label, while multidimensional nodes correspond to OEM nodes from which these special edges depart. Moreover, we must encode context in OEM in a way Lorel can understand and handle. Contexts that must be encoded include explicit contexts and inherited coverages of edges.

Figure 3 gives an intuition of the transformation. It presents a simple Multidimensional Data Graph M together with its OEM counterpart O. Nodes with a capital letter in M correspond to nodes with the respective lowercase letter in O. Observe that for each edge in M an additional node exists in O, splitting the edge in two OEM edges. The role of this node is to group the encoded context(s) for the corresponding Multidimensional Data Graph edge. A number of reserved labels with special meaning are

used. All reserved labels start with an underscore, and they are: `_ett`, `_facet`, `_cxt`, `_icw`, and `_ecw`. An entity edge is represented by an edge with the same label and a following edge labeled `_ett`. A context edge is represented by an edge labeled `_facet` and a following edge labeled `_cxt`. Explicit contexts and inherited coverages of edges are converted to the worlds they represent, and all possible worlds are mapped to integers. Edges that are labeled `_icw` point to the enumerated worlds (integer-valued nodes) that belong to inherited coverages, while edges labeled `_ecw` point to the enumerated worlds that belong to explicit contexts.

The actual transformation process of a Multidimensional Data Graph $M = (V_{mld}, V_{cxt}, E_{cxt}, E_{ett}, r, v)$ to an OEM O is given below.

**O ← MDGToOEM (M) is:**

1. For every world, add a new atomic node w to $V_{cxt}$ that corresponds to that world, having as value the integer mapped to the world.
2. Move all (multidimensional) nodes from $V_{mld}$ to $V_{cxt}$ (complex nodes).
3. For every edge $h = (q, l, p) \in E_{ett}$ add a new complex node u to $V_{cxt}$. Then, replace h with the new edges (q, l, u) and (u, `_ett`, p) in $E_{ett}$. Next, for every world represented by the inherited coverage of h, add an edge (u, `_icw`, w) to $E_{ett}$, where w corresponds to that world.
4. For every edge $h = (q, c, p) \in E_{cxt}$ add a new complex node u to $V_{cxt}$. Then, remove h from $E_{cxt}$, and add the edges (q, `_facet`, u) and (u, `_cxt`, p) to $E_{ett}$. Next, for every world represented by the inherited coverage of h, add an edge (u, `_icw`, w) to $E_{ett}$, where w corresponds to that world. Moreover, for every world represented by the explicit context of h, add an edge (u, `_ecw`, w) to $E_{ett}$, where w corresponds to that world.
5. Return $O = (V_{cxt}, E_{ett}, r, v)$.

### 4.2 Translating MQL Queries to Lorel

MQL queries can be translated to "equivalent" Lorel queries, which are evaluated on the OEM given by the transformation defined in the previous section. For this translation to work, the MOEM database *must be in canonical form* when the transformation to OEM takes place. This allows context path expressions, which are built around the canonical form, to be translated to "equivalent" Lorel path expressions.

In this section we specify such a translation that supports the major features of MQL. We have not addressed some features of MQL that seemed difficult or impossible to translate, like regular expressions in context path expressions (*general context path expressions* [4]).

**Converting Context Path Expressions to Path Expressions.** To facilitate the comprehension of translated Lorel queries, we use `E-HUB` as identifier for nodes from which a `_ett` edge departs, `MLD` for nodes from which a `_facet` edge departs, and `C-HUB` for nodes from which a `_cxt` edge departs. In addition, we use `w1`, `w2`, ... to denote the integers that have been mapped to worlds.

We start with a very simple MQL `from` clause:

```
from X.[c]label Y
```

We assume [c] is a context specifier representing the worlds that correspond to $w_4$, $w_7$, and $w_9$. As shown in [4], [c] is implied throughout the path it qualifies, and the clause can be written as:

```
from X.[c]label::[c][-] Y
```

This from clause can also be written in MQL using a multidimensional object variable <V> as:

```
from X.[c]label <V>,
     <V>::[c][-] Y
```

The equivalent Lorel expression is:

```
from X.label E-HUB, E-HUB._ett MLD,
     MLD._facet C-HUB, C-HUB._cxt Y
where     E-HUB._icw{W4} = w4
      and E-HUB._icw{W7} = w7
      and E-HUB._icw{W9} = w9
      and C-HUB._icw{W4} = w4
      and C-HUB._icw{W7} = w7
      and C-HUB._icw{W9} = w9
```

The where clause states that the inherited coverages of the two MOEM edges must contain all the worlds specified by [c]. Observe the use of the variables W4, W7, and W9, which declare that it is *not* the same node that must be equal to $w_4$, to $w_7$, and to $w_9$ (otherwise the condition would always be false).

We now use the MQL query example of Section 3, and apply the same process to its from clause. For brevity, we use [c1] to denote the context specifier [season=winter], and [c2] to denote [season=summer]. The MQL query can now be written as:

```
select name: P, winter_street: Y
from [-]music_club <V1>, <V1>::[-][-] X,
   X.[c1]address <V2>, <V2>::[c1][-] V3,
   V3.[c1]street <V4>, <V4>::[c1][-] Y,
   X.[c2]address <V5>, <V5>::[c2][-] V6,
   V6.[c2]street <V7>, <V7>::[c2][-] Z,
   X.[-]name <V8>, <V8>::[-][-] P
where Z="Omirou"
```

The equivalent Lorel query is:

```
select name: P, winter_street: Y
from
   music_club E-HUB1, E-HUB1._ett V1,
   V1._facet C-HUB1, C-HUB1._cxt X,
   X.address E-HUB2, E-HUB2._ett V2,
   V2._facet C-HUB2, C-HUB2._cxt V3,
   V3.street E-HUB3, E-HUB3._ett V4,
   V4._facet C-HUB3, C-HUB3._cxt Y,
   X.address E-HUB4, E-HUB4._ett V5,
```

```
    V5._facet C-HUB4, C-HUB4._cxt V6,
    V6.street E-HUB5, E-HUB5._ett V7,
    V7._facet C-HUB5, C-HUB5._cxt Z,
    X.name E-HUB6, E-HUB6._ett V8,
    V8._facet C-HUB6, C-HUB6._cxt P
where
    Z="Omirou"
    and predicate(E-HUB2) and predicate(C-HUB2)
    and predicate(E-HUB3) and predicate(C-HUB3)
    and predicate(E-HUB4) and predicate(C-HUB4)
    and predicate(E-HUB5) and predicate(C-HUB5)
```

The expressions *predicate*(VAR) ensure that the corresponding edges have a proper inherited coverage. Therefore, each expression *predicate*(VAR) must be replaced by

```
        VAR._icw{W1} = w1
    and VAR._icw{W2} = w2
    and VAR._icw{W3} = w3
    and ...
```

where $w_1$, $w_2$, $w_3$, ... are the integers that correspond to worlds of the respective inherited coverage qualifier: for E–HUB2, C–HUB2, E–HUB3, and C–HUB3 the inherited coverage qualifier is [season=winter], while for E–HUB4, C–HUB4, E–HUB5, and C–HUB5 the inherited coverage qualifier is [season=summer]. The edges that correspond to variables E–HUB1, C–HUB1, E–HUB6, and C–HUB6 can have any inherited coverage because their implied inherited coverage qualifier is the empty context [-], thus they are not included in where.

Using the above framework, it is straightforward to translate MQL queries that contain multidimensional object variables. Actually, the analytical form of our MQL query example contains the multidimensional object variables <V1>, <V2>, <V4>, <V5>, <V7>, and <V8>, which correspond to the variables V1, V2, V4, V5, V7, and V8 of the equivalent Lorel query. In addition, it is easy to accommodate explicit context qualifiers. A facet part ::[$c_I$][$c_E$] will result in a predicate of the form:

```
        VAR._icw{W1} = w1
    and VAR._icw{W2} = w2
    and VAR._icw{W3} = w3
    and ...
    and VAR._ecw{W2} = w2
    and VAR._ecw{W6} = w6
    and ...
```

where $w_1$, $w_2$, $w_3$, ... correspond to the worlds of the inherited coverage qualifier [$c_I$], and $w_2$, $w_6$, ... correspond to the worlds of the explicit context qualifier [$c_E$].

**Context Variables and "within" Clause.** MQL uses an additional `within` clause to express conditions on contexts. Consider the MQL query:

```
select comments: Y
from music_club.[X]review.comments Y
within [X] * [detail=high] <= [lang=gr]
```

The *context variable* `[X]` binds to the path inherited coverage of the path

```
review::[-].comments::[-]
```

and the condition in `within` requires that the context intersection (denoted `*`) between this path inherited coverage and `[detail=high]` be context subset (denoted `<=`) of `[lang=gr]`. Consequently, this condition ensures that the query returns `comments` facets in Greek in high detail (node `&20` in Figure 1). Note that there are more intuitive ways to express the same query in MQL, but with less demonstrative value.

The first step is to express context specifiers as Lorel queries. Suppose that `[detail=high]` represents the worlds that correspond to the integers $w_1$, $w_2$, and $w_3$. The following Lorel query evaluates to the respective nodes:

```
select W
from music_club.#._icw W
where W=w1 or W=w2 or W=w3
```

Lets use $L_{[detail=high]}$ to refer to this query, and $L_{[lang=gr]}$ to refer to an analogous Lorel query that expresses `[lang=gr]`. In addition, we use the symbol $L_{CXT\_VAR}$ to refer to a Lorel query expressing the path inherited coverage bound to the context variable `[X]`. This Lorel query is:

```
E-HUB2._icw intersect C-HUB2._icw
intersect
E-HUB3._icw intersect C-HUB3._icw
```

The query evaluates to the "worlds" under which all edges of the path hold. Now that we have expressed all contexts as queries evaluating to sets of nodes that represent worlds, we can express context subset as a relation between the queries. Assuming that *query1* expresses a context `[c1]` and *query2* a context `[c2]`, the condition `[c1] <= [c2]` (`[c1]` context subset of `[c2]`) is implemented by the predicate:

```
for all LEFT in (query1):
    exists RIGHT in (query2): LEFT = RIGHT
```

where `LEFT` and `RIGHT` are Lorel variables that range over the "worlds" to the left and to the right side of the symbol `<=`, respectively.

The MQL query can now be translated to the following Lorel query:

```
select comments: Y
from
  music_club E-HUB1, E-HUB1._ett V1,
  V1._facet C-HUB1, C-HUB1._cxt V2,
```

```
    V2.review E-HUB2, E-HUB2._ett V3,
    V3._facet C-HUB2, C-HUB2._cxt V4,
    V4.comments E-HUB3, E-HUB3._ett V5,
    V5._facet C-HUB3, C-HUB3._cxt Y
  where
    for all LEFT in
    (L_CXT_VAR intersect L_[detail=high]):
         exists RIGHT in (L_[lang=gr]):
             LEFT = RIGHT
```

By combining in similar ways Lorel queries that express sets of worlds, it is straightforward to implement any context condition in the `within` clause.

### 4.3 Transforming OEM Results to M.D.G.

LORE returns the result of a Lorel query as an OEM graph. As stated, this OEM graph can be transformed to a Multidimensional Data Graph, which is the result of the original MQL query.

The process that transforms an OEM O = (V, E, r, v) to a Multidimensional Data Graph M is given below.

**M ← OEMToMDG (O) is:**

1. Represent O as a Multidimensional Data Graph M = ($V_{mld}$, $V_{cxt}$, $E_{cxt}$, $E_{ett}$, r, v), where $V_{cxt}$ = V, $E_{ett}$ = E, and $V_{mld}$, $E_{cxt}$ are empty sets.
2. For every edge h = (q, l, u) ∈ $E_{ett}$ where l is not a reserved label, remove u from $V_{cxt}$. Then remove h and (u, `_ett`, p) from $E_{ett}$, and add the edge (q, l, p) to $E_{ett}$. Remove all edges (u, `_icw`, w) from $E_{ett}$.
3. For every edge h = (q, `_facet`, u) ∈ $E_{ett}$, move q from $V_{cxt}$ to $V_{mld}$ (if not already moved), and remove u from $V_{cxt}$. For all nodes w, where (u, `_ecw`, w) ∈ $E_{ett}$, apply context union to the corresponding worlds to get a context specifier c. Then remove h and (u, `_cxt`, p) from $E_{ett}$, and add the edge (q, c, p) to $E_{cxt}$. Remove all edges (u, `_ecw`, w) and (u, `_icw`, w) from $E_{ett}$.
4. Remove from $V_{cxt}$ all nodes that correspond to worlds (unreachable from the root at this time).
5. Return M = ($V_{mld}$, $V_{cxt}$, $E_{cxt}$, $E_{ett}$, r, v).

Notice that, in order to reconstruct context specifiers, step 3 needs the *same* mapping of worlds to integers that was initially used while transforming the MOEM database to OEM.

### 4.4 Prototype Implementation

Our prototype system is implemented in Java and interfaces with LORE, which is used as a back-end. The system is initialized with an MOEM database, receives MQL queries, and returns Multidimensional Data Graphs as results. The system actually constitutes the Query Subsystem of *MSSDesigner* [11], a more general platform for manag-

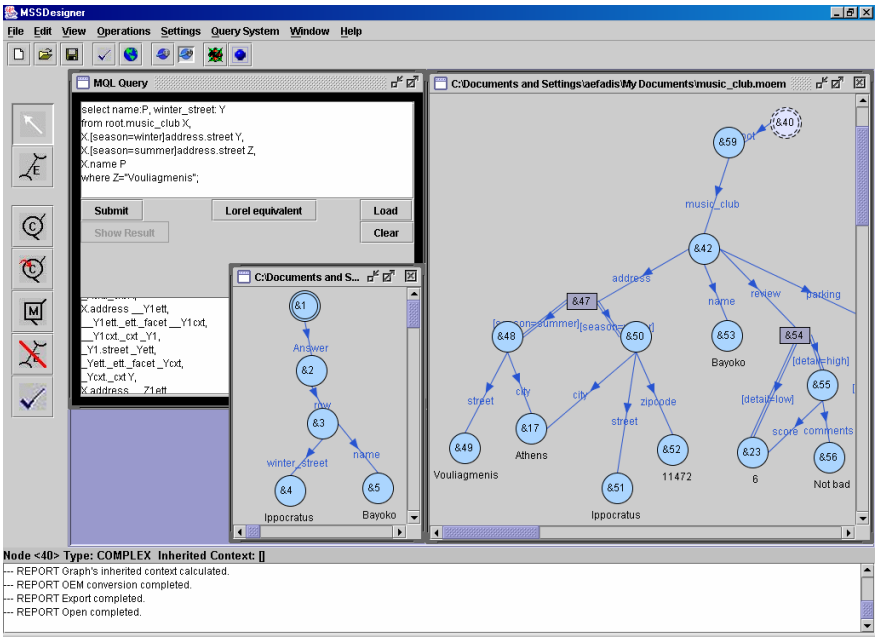


**Fig. 4.** Evaluating an MQL query and displaying the results in *MSSDesigner*

ing multidimensional semistructured data, and is shown in Figure 4. It relies on this platform to perform basic functions, like carrying out context operations and calculating the inherited coverage of MOEMs.

## 5 Conclusions

In this paper we demonstrated some of the benefits of treating context as first-class citizens in Web data models and query languages. We briefly introduced MQL, a context-aware query language for semistructured data, and discussed in detail an evaluation process for MQL queries. We defined a transformation from MOEM graphs to corresponding OEM graphs and vice-versa, and specified how MQL queries can be translated to equivalent Lorel queries. We presented a prototype system that implements the above, and uses LORE to evaluate MQL queries. This evaluation process gave an opportunity for an intuitive comparison between the two query languages and data models: MQL and MOEM are much more elegant and expressive when context is involved, while they become as simple as Lorel and OEM when context is not an issue. Moreover, MQL and MOEM directly support cross-world queries, which have no counterpart in context-unaware query languages.

## References


1. M. C. Norrie, and A. Palinginis. From State to Structure: an XML Web Publishing Framework. In the *15th International Conference on Advanced Information Systems Engineering (CAiSE'03)*, Austria, June 2003.
2. Yannis Stavrakas, and Manolis Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. In the *14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, Toronto, Canada, May 2002.
3. Dan Suciu. An Overview of Semistructured Data. *SIGACT News*, 29(4): 28-38, 1998.
4. Yannis Stavrakas. *Multidimensional Semistructured Data: Representing and Querying Context-Dependent Multifaceted Information on the Web*. PhD Thesis, Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, June 2003.
5. 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, April 1997.
6. Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, and V. Zafeiris. Accommodating Changes in Semistructured Databases Using Multidimensional OEM. In the *6th Conference on Advances in Databases and Information Systems (ADBIS'02)*, Slovakia, September 2002.
7. Yannis Stavrakas, Manolis Gergatsoulis, Christos Doulkeridis, and Vassilis Zafeiris. Representing and Querying Histories of Semistructured Databases Using Multidimensional OEM. To appear in *Information Systems* journal.
8. John Mylopoulos, and Renate Motschnig-Pitrik. Partitioning Information Bases with Contexts. In the *3rd International Conference on Cooperative Information Systems (CoopIS'95)*, pages 44-55, Vienna, Austria, May 1995.
9. The World Wide Web Consortium (W3C). Resource Description Framework (RDF) Schema Specification, 1999. `http://www.w3.org/ TR/PR-rdf-schema`
10. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. LORE: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3): 54-66, September 1997.
11. Vassilis Zafeiris, Christos Doulkeridis, Yannis Stavrakas, and Manolis Gergatsoulis. An Infrastructure for Manipulating Multidimensional Semistructured Data. In the *1st Hellenic Data Management Symposium (HDMS'02)*, Athens, Greece, July 2002.
12. Moira C. Norrie, and Alexios Palinginis. Empowering Databases for Context-Dependent Information Delivery. *CAiSE'03 Workshop on Ubiquitous Mobile Information and Collaboration Systems (UMICS'03)*, Austria, June 2003.
13. Manos Theodorakis, Anastasia Analyti, Panos Constantopoulos, and Nikos Spyratos. Context in Information Bases. In the *3rd International Conference on Cooperative Information Systems (CoopIS'98)*, New York City, 1998.
14. Manolis Gergatsoulis, Yannis Stavrakas, and Dimitris Karteris. Incorporating Dimensions to XML and DTD. In the *12th Conference on Database and Expert Systems Applications (DEXA'01)*, Munich, Germany, September 2001.
15. Manolis Gergatsoulis, and Yannis Stavrakas. Representing Changes in XML Documents Using Dimensions. In *XML Database Symposium (XSym 2003)* in Conjunction with VLDB 2003, Berlin, Germany, September 2003.
16. M. Gergatsoulis, Y. Stavrakas, D. Karteris, A. Mouzaki, and D. Sterpis. A Web-based System for Handling Multidimensional Information through MXML. In *the 5th Conference on Advances in Databases and Information Systems (ADBIS 2001)*, Lithuania, 2001.
17. Chiara Ghidini, and Fausto Giunchiglia. Local Model Semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence* 127, pages 221 - 259, 2001.