A Query Evaluation System for Multidimensional Semi-Structured Data

Antonis Efandis[1], Kostis Pristouris[1], and Yannis Stavrakas[1,2]

[1] Knowledge & Database Systems Laboratory
National Technical University of Athens (NTUA), 15773 Athens, Greece
[2] Institute of Informatics & Telecommunications,
National Center for Scientific Research (N.C.S.R.) 'Demokritos',
15310 Aghia Paraskevi, Greece
Email: {aefan,kprist}@freemail.gr ystavr@iit.demokritos.gr

Abstract. *Multidimensional semi-structured data* (*MSSD* for short) are semi-structured data where information entities may present different *facets* under different *contexts*. Context is expressed using variables called *dimensions* and represents a set of alternative *worlds*, under which data obtain a substance. In previous work, a graph data model for MSSD called *Multidimensional OEM* (*MOEM* for short) has been proposed, which is an extension of OEM that incorporates the notion of context and supports multifaceted entities. Moreover, a query language for MOEM called *Multidimensional Query Language* (*MQL* for short) has been defined, which treats context as first class citizen and is able to formulate *cross-world* queries. In this paper, we present an MQL query evaluation system, which is part of a more general platform for managing MSSD. We describe the system architecture and discuss its modules in detail. We demonstrate the operation of the system by following an MQL query example through its evaluation steps. The evaluation process is based on representing MOEM graphs as OEM graphs and translating MQL queries to equivalent Lorel queries which are subsequently evaluated by Lore. This allows a comparison between the expressiveness of the above query languages and data models when *context-driven* queries are involved.

1 Introduction

The diversity of Web users has raised new issues [1] to be considered by the database research community. Web users may have different perception of the same entities, therefore different variants of the same information may become relevant in different situations. Information providers have to take into consideration this diversity and maintain different variations of their services. This raises the need for data models that support different facets of the same information entity, whose contents can vary in structure and value, as well as for ways to query such models.

To deal with those issues, *multidimensional semi-structured data* [2] (MSSD for short) have been proposed, which are semi-structured data [3] that present different *facets* under different *contexts*. The main difference between conventional and multidimensional semi-structured data is the introduction of *context specifiers*. Context

specifiers are syntactic constructs that are used to qualify pieces of semi-structured data and specify sets of *worlds* under which those pieces 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. Each facet is associated with a context that defines the conditions under which the facet becomes a *holding facet* of the multidimensional entity. *Multidimensional Query Language* [4] (MQL for short) treats context as first-class citizen, and can express *context-driven* queries on MSSD, 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* for short), an extension of OEM [3] that is suitable for MSSD. Context-aware data models and query languages like MOEM and MQL can be applied on a variety of cases and domains; in [6, 7] we show how MOEM and MQL can be used to represent and query histories of OEM databases.

In this paper we present a prototype system for MQL query evaluation. The architecture of the system is presented in detail, as well as the way that its various modules cooperate in order to carry out the query evaluation. Part of this process involves the transformation of MOEM to OEM databases and the translation of MQL to Lorel queries. This allows a direct comparison between the two data models and the two query languages. The increased complexity of the OEM model, introduced to support context information, shows the benefits of MOEM. Moreover, the comparison of two equivalent queries, one in Lorel and one in MQL, demonstrates the increased syntactic complexity of the former and the elegance and expressiveness of the latter.

This paper is structured as follows: Section 2 reviews preliminary material on MSSD, MOEM, and MQL. Section 3 presents the architecture of the system and describes each module. In Section 4, a step by step example of the evaluation process of an MQL query is presented. Section 5 deals with general implementation issues. Finally, Section 6 concludes the paper.

2 Multidimensional Semi-Structured Data

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 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 of pairs (d, v), where $d \in \mathcal{D}$ and $v \in \mathcal{V}_d$, such that for every $d \in \mathcal{D}$ exactly one (d, v) belongs to w.*

In MSSD, sets of worlds are represented by *context specifiers* (*contexts* for short), which can be seen as constraints on dimension values. A simple context specifier example is `[year=1979]`, which represents the worlds for which the dimension `year` has the value `1979`. The context specifier `[language=greek, format in {html,pdf}]` corresponds to those worlds where `language` is `greek` and `format` is either `html` or `pdf`. The context specifier `[season=summer | season=spring, daytime=noon]` is more complex, and represents the worlds where `season` is `summer`, together with the worlds where `season` is `spring` and `daytime` is `noon`.

The context specifier `[ ]` is a *universal context* specifier and represents the set of all possible worlds with respect to any set of dimensions $\mathcal{D}$, while `[-]` is an *empty context* specifier, and represents the empty set of worlds with respect to any set of dimensions $\mathcal{D}$. [2] defines 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. In addition, comparison operations like *context equality* and *context subset* have been defined, as well as the transformation of a context specifier to the set of worlds that it represents with respect to a set of dimensions $\mathcal{D}$.

2.1 Multidimensional OEM

Multidimensional Data Graph (*MDG*) is an extension of *Object Exchange Model* (OEM) [5], suitable for representing multidimensional semi-structured 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 labelled 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 may hold. Context edges are drawn as thick lines, to distinguish them from conventional (thin-lined) OEM edges.

In MDG, the conventional circular nodes of OEM are called *context nodes*, while 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 MDG.

The Multidimensional Data Graph shown in Figure 1 represents a "multidimensional" club. This club operates on different addresses during the summer period and the winter period, a fact that is represented in the graph by the multidimensional node with oid `&4`, that has two facets (oids `&5` and `&6`), depending on the value for the `period` dimension. Moreover, reviews for the club are provided in two languages, through nodes `&13` and `&14` of the graph. These nodes constitute facets for the multidimensional node `&12` and may hold under the contexts `[lang=gr]` and `[lang=en]`, respectively.

Two fundamental concepts related to MDGs are *explicit context* and *inherited context* [2]. 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 effective 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 effective context, in the sense that it does not alone determine the worlds under which the destination node

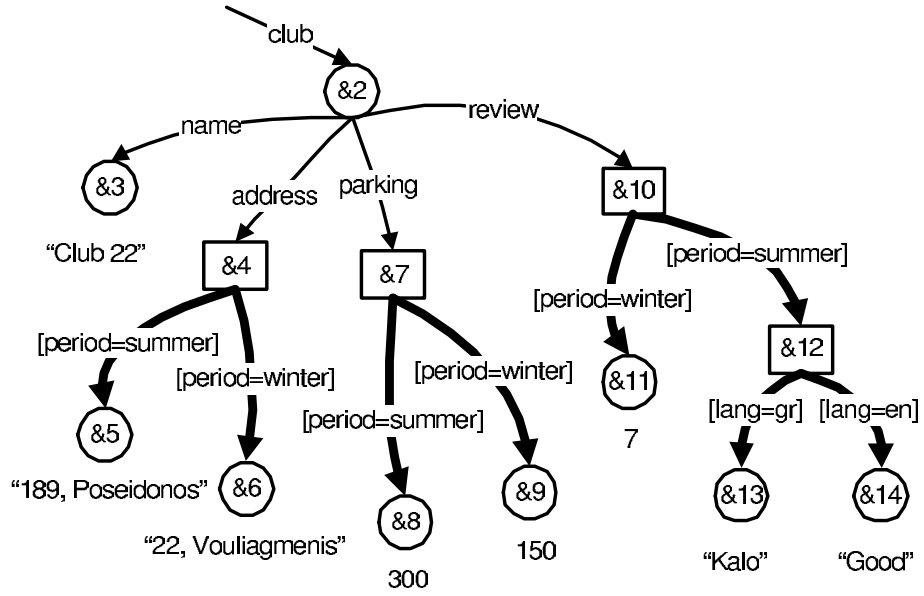

Fig. 1. A multidimensional club

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 MDGs 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. 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 is therefore the effective context under which a node or edge holds [4]. 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.

Every Multidimensional Data Graph can be transformed to a *canonical form*, which is a graph that contains the same information as the original, however it exhibits certain additional properties. In the canonical form of a graph a context edge can only point to context nodes, while an entity edge can only point to multidimensional nodes. Therefore, *every possible path in a graph that is in canonical form is formed by a repeated succession of one entity edge and one context edge.*

The graph in Figure 2 is the canonical form of the graph in Figure 1. Notice the insertion of the multidimensional node with oid `&53`, and of the context edge with

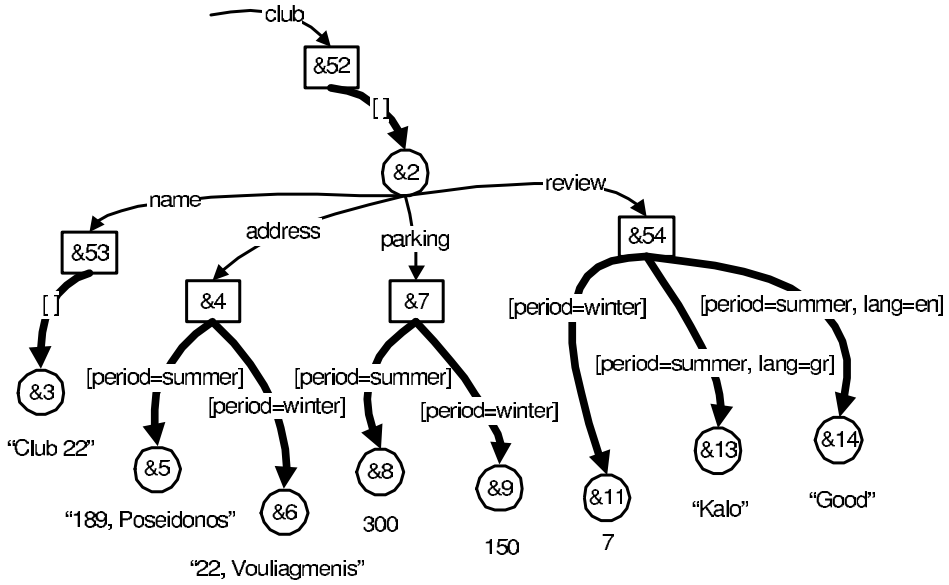

Fig. 2. Canonical form of the MDG in Figure 1

explicit context `[ ]`(universal context) after the entity edge labelled `name`. Moreover, the two consecutive multidimensional nodes with oids `&10` and `&12` have been merged into a single node (oid `&54`) and the respective context specifiers have been adjusted accordingly.

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). A *Multidimensional OEM* graph, 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 graph all nodes and edges hold under at least one world, and all leaves are atomic nodes. The Multidimensional Data Graphs in Figures 1 and 2 are MOEM graphs.

2.2 Multidimensional Query Language

Multidimensional Query Language [4], or *MQL* for short, is a query language specifically designed for MOEM databases. MQL is based on a "core language" for semi-structured data described in [8], and is effectively an extension of Lorel [5].

An essential feature of MQL is *context path expressions*, which are path expressions qualified by context specifiers and *context variables*. Context variables bind to the path inherited coverage of the path to which they apply. Context path expressions take advantage of the canonical form of Multidimensional Data Graphs, and are formed by a number of *entity parts* and *facet parts* succeeding one another. Entity parts start with

'.' (except for the entity edge pointing to the root) and are matched against entity edges, while facet parts start with '::' and are matched against context edges.

```
[period=winter]club::[-].address::[-] Z
```

In the context path expression above, `club` and `.address` are entity parts, while the two expressions containing empty context specifiers `::[-]` are facet 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, a facet part with the empty context `::[-]` matches any context edge. The context specifier `[period=winter]` is an *inherited coverage qualifier* and is matched against the path inherited coverage of the `club::[-].address::[-]` 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:

```
[period=winter]club.address Z
```

Evaluated on the graph of Figure 2, this context path expression causes the *context object variable* `Z` to bind to node `&6`.

Example 1. The following MQL query returns the name and summer address of a club whose winter address is known:

```
select name: N, summer_address: Y
from   [P]club.address Y,
       [period=winter]club.address Z,
       club.name N
where  Z = "22, Vouliagmenis"
within [P]*[period=summer] != [-]
```

In the query of Example 1, the `from` clause is the first to be considered during evaluation. This clause contains context path expressions that result in bindings of object variables `Y`, `Z`, and `N` and of the context variable `[P]`. Tuples of variable bindings are filtered in the `where` clause, which requires that `Z` has the value ''22, Vouliagmenis''.

Context path expression `[P]club.address` contains the context variable `[P]`, which is subsequently used in the condition `[P]*[period=summer] != [-]` in the `within` clause of the query. The `within` clause is used in MQL to express conditions on contexts; in this case, it requires that the context intersection of context `[period=summer]` with the path inherited coverage bound to `[P]` must result in a non-empty set.

3 System Architecture

In this section we present the architecture of our prototype MQL query evaluation system, which is in fact the query subsystem of an infrastructure for manipulating

MSSD [2, 9]. This infrastructure saves and loads MSSD in a number of formats, supports the graphical creation of MOEM models, and provides a number of functions that perform context operations. The graphical interface that gives access to this infrastructure is called *MSSDesigner*.

The query evaluation system presented here uses the data structures and the API of this infrastructure for implementing the operations required for query evaluation. Apart from that, it integrates with and extends MSSDesigner with options for submitting queries and presenting results.

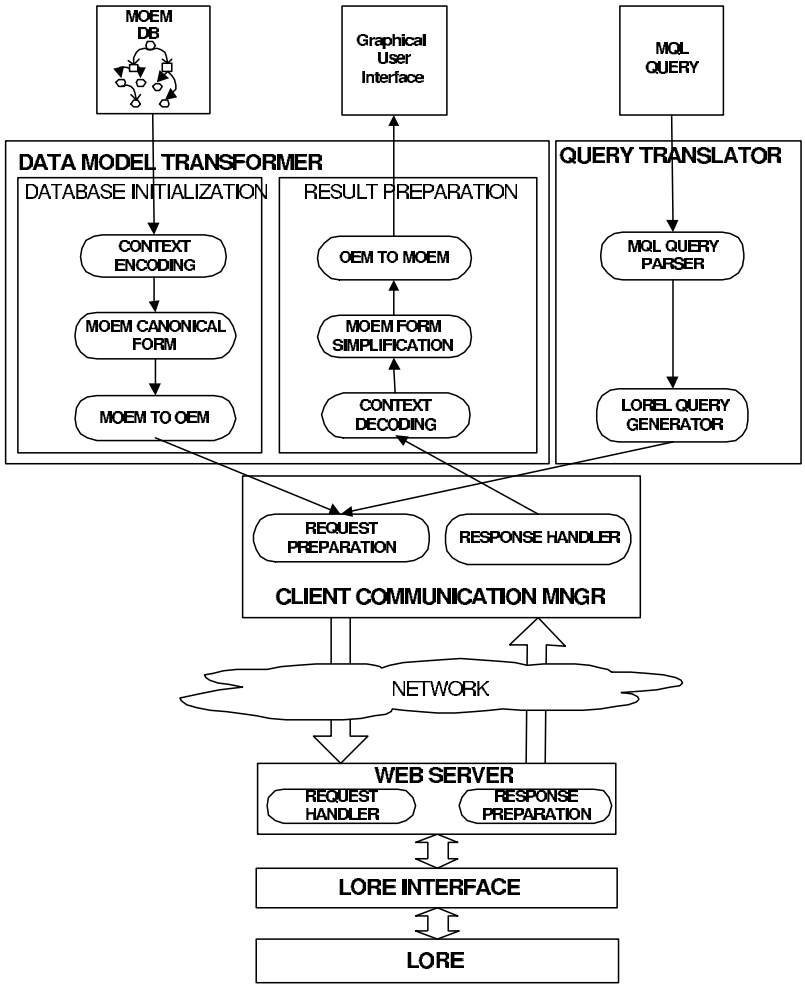

Fig. 3. Architecture of a query system for MOEM data.

3.1 Overall Architecture

The system is developed on top of Lore [10], which is a DBMS for semi-structured data that uses OEM [5] as a data model and Lorel [5] as a query language. Since MOEM is an extension of OEM and MQL is based on Lorel, it was sensible to develop a mechanism for transforming MOEM graphs into corresponding OEM graphs and translate MQL queries to "equivalent" Lorel queries.

As depicted in Figure 3, the system runs over a network in order to achieve platform independence between the *Server Side* and the *Client Side*. Communication is performed through HTTP requests. The following reside on the Server Side: a web server for HTTP request handling, a module that interfaces with Lore, and Lore itself. The Client Side consists of modules that comprise the query subsystem of the MSSD infrastructure, as well as the GUI extension to MSSDesigner.

3.2 System Modules

Figure 3 shows all the modules of the system, and the sequence they are used during the principal system operations. Important operations include database initialization, submission and evaluation of an MQL query, and results construction and presentation. The following provide a description of the modules and their role in the system.

The Server Side consists of *Lore*, the *Lore interface*, and the *Web Server* modules.

Lore: Lore provides a generic user interface for database creation and management, as well as an API for application development. Multiple OEM databases can be managed simultaneously, which are provided as input in some textual representation. Lorel queries can be submitted programmatically or through a generic user interface, and the results are returned either in some textual format, or as a data structure residing in the system memory.

Lore Interface: This module constitutes the interface through which our system uses Lore. It includes: a) a method for creating an OEM database, b) a method for dropping the current Lore database, c) methods for connecting to the Lore database and for query submission, and d) a method for the retrieval of the OEM result graph in XML format, after the evaluation of a Lorel query.

Web Server: This module accepts HTTP requests and sends back HTTP responses. It implements methods specific to the query system, which decode the requests accepted by the client and encode the responses, so that they can be utilized accordingly by the appropriate Client Side modules.

The Client Side consists of the *Client Communication Manager*, the *Data Model Transformer* and the *Query Translator* modules.

Client Communication Manager: This module is responsible for communicating with the Web Server module. Through the *Request Preparation* and the *Response Handler*, requests are formed and transmitted to the Web Server, and responses are received and decoded. Together with the Web server, this module makes up the communication channel between the Client Side modules and the Lore Interface. In this way, the Client Side modules can submit commands (db creation, db destruction, query submission) to Lore, and receive the appropriate results.

Data Model Transformer: The Data Model Transformer is responsible for transforming an MOEM database to a corresponding OEM graph, and for the reverse process of transforming OEM results back to MOEM. The MOEM to OEM transformation is carried out during database initialization. The produced OEM is saved in a suitable textual format and is sent through an HTTP request at the Server Side for loading into Lore. The reverse process of OEM to MOEM transformation is carried out on OEM results that are sent back by the server.

Query Translator: This module consists of the *MQL Query Parser* and the *Lorel Query Generator*. The input is an MQL query submitted by the user. After the syntactical analysis the equivalent Lorel query is produced and submitted to the *Client Communication Manager*.

As implied by the flow of arrows in Figure 3, the above modules are involved in three main processes: database initialization, query evaluation, and presentation of results. Those processes are explained in the next section by using an example MQL query evaluated on a sample MOEM database.

4 A Comprehensive Example of MQL Query Evaluation

This section provides a walk-through in the evaluation of a sample MQL query by the system. We present the intermediate data that is generated by the system during this process, and associate this data with the modules that produce or consume it.

As a sample database we will consider the MOEM database in Figure 1.

4.1 MOEM Database Initialization

As described in Section 3, before the system can perform any query evaluation operations, the database needs to be initialized. The first step of this procedure is the construction of the canonical form of the current MOEM database. This canonical form, which is depicted in Figure 2, is subsequently transformed to a corresponding OEM, which will become the database of Lore. All operations that regard this transformation are handled by the Data Model Transformer module.

Context information is represented in this OEM graph using integer values that are mapped to every possible world. Table 1 lists the encoding of possible worlds for the MOEM database in Figure 1.

World specifier	Code
[lang=gr, period=summer]	0
[lang=gr, period=winter]	1
[lang=en, period=summer]	2
[lang=en, period=winter]	3

Table 1. Encoding worlds

The OEM graph that corresponds to the MOEM graph of Figure 2 is too complex to be presented in its entirety, however in Figure 4 we illustrate a portion of the MOEM graph, together with the corresponding portion of the OEM graph. Note that before this transformation can take place, the inherited coverage of all the edges in the MOEM graph needs to be calculated.

The entity edge (&2, address, &4) in the MOEM graph of Figure 4(a) is represented in the corresponding OEM graph of Figure 4(b) by two successive edges and a new intermediate node (*hub node*) with oid &65. The first edge starts from the same node and has the same label as the original edge, but points to the hub node, whereas the second edge (labelled _ett to state that the respective MOEM edge is an entity edge) starts from the hub node and points to the destination node of the original MOEM edge. This construct provides a means to represent the inherited coverage of the edge in the OEM model. This is achieved by introducing nodes with integer values from Table 1 that uniquely identify worlds (*world nodes*). Edges labelled _icw connect the hub node with the world nodes that constitute the inherited coverage of the original MOEM entity edge. For the entity edge in question, hub node(oid &65) is connected to world nodes &w0, &w1, &w2, &w3, since the inherited coverage of the edge is the universal context.

Similar constructs are used to represent context edges as well, e.g. edge (&4, [period=summer], &5). In this case, the edges connecting to the hub node are labelled _facet and _cxt, and the hub nodes are connected to the world nodes not only by _icw edges (inherited coverage), but also by _ecw edges that are used to represent the explicit context of the edge (entity edges also have explicit context, but by definition it is always the universal context [ ]).

4.2 Query Translation and Evaluation

After database initialization, the system can receive and evaluate MQL queries. The sample query used in this section is the one of Example 1. With the submission of this MQL query, the Query Translator module of the system uses the correspondence denoted in Table 1, to construct the equivalent Lorel query that follows.

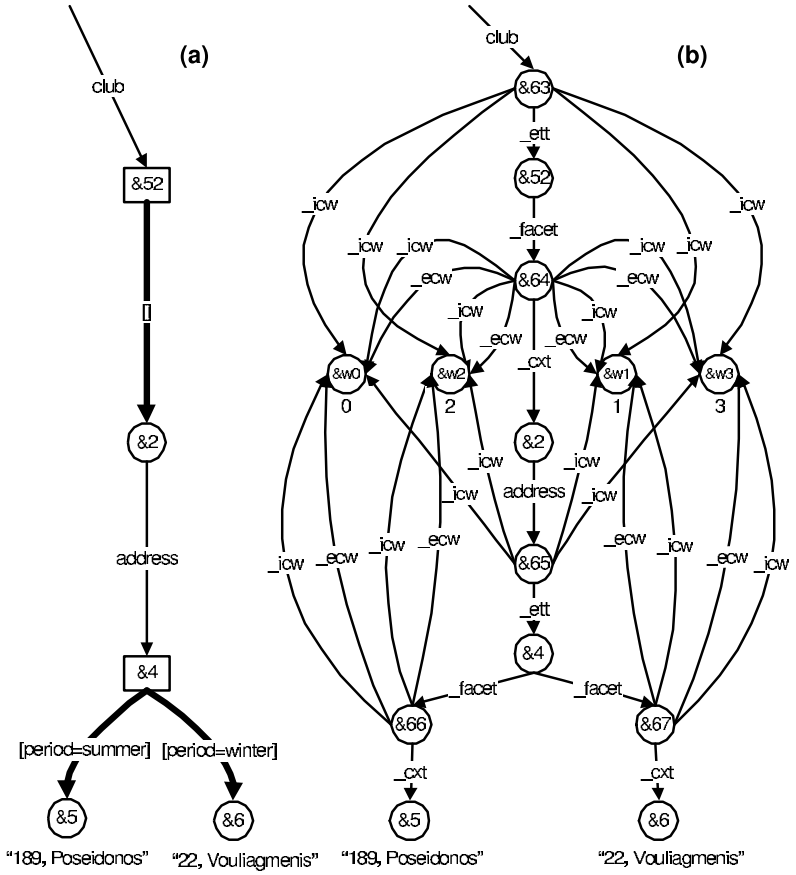

Fig. 4. MOEM to OEM conversion example

```
1    select name: N, summer_address: Y
2    from   club._ett._facet _Y0cxt, _Y0cxt._cxt Y0,
3           Y0.address _Yett, _Yett._ett._facet _Ycxt, _Ycxt._cxt Y,
4           club._ett._facet _Z0cxt, _Z0cxt._cxt Z0,
5           Z0.address _Zett, _Zett._ett._facet _Zcxt, _Zcxt._cxt Z,
6           club._ett._facet _N0cxt, _N0cxt._cxt N0,
7           N0.name _Nett, _Nett._ett._facet _Ncxt, _Ncxt._cxt N
8    where  (Z = "22, Vouliagmenis")
9           and (_Z0cxt._icw_W1 = "1" and _Z0cxt._icw_W3 = "3" )
10          and (_Zcxt._icw{_W1} = "1" and _Zcxt._icw{_W3} = "3" )
11          and (_Zett._icw{_W1} = "1" and _Zett._icw{_W3} = "3" )
12          and (exists( ((_Y0cxt._icw) intersect (_Ycxt._icw)
13                                  intersect (_Yett._icw))
14                       intersect (select W from guide._icw W
15                                  where  W="0" or W="2") ) )
```

The primary goal is to express context operations appearing in MQL queries in a way that Lorel can handle and process. Each one of the context path expressions in the `from` clause of the MQL query is translated to a number of Lorel path expressions. The context path expression `[period=winter]club.address Z` can be broken down to two simpler context path expressions: `[period=winter]club::[-] Z0` and `Z0.[period=winter]address::[-] Z`.

The first expression corresponds to the path `club._ett._facet._cxt` of the graph in Figure 4. It is translated in line 4 of the Lorel query to a series of path expressions that form its equivalent OEM path. These expressions bind to variable `Z0`, that corresponds to the root node of the original graph (oid `&2`). The path expressions in line 5 of the Lorel query are equivalent to `Z0.[period=winter]address::[-] Z`. These expressions form the path `Z0.address._ett._facet._cxt` that binds to variable `Z` (corresponding to oids `&5` and `&6`). The three additionally declared variables of lines 4, and 5 correspond to the hub nodes of the paths: `_Z0cxt` to oid `&64`, `_Zett` to oid `&65`, and `_Zcxt` to oids `&66` and `&67`. These three variables are included in the `where` clause of the Lorel query, in order to express the restrictions on context posed by the MQL context path expression.

In this case, it is required that the path inherited coverage be superset of the context `[period = winter]`. This condition translates into Lorel using the expressions in lines 9-11. The expression `(_Zett._icw{_W2} = "1" and _Zett._icw{_W3} = "3" )` states that two distinct objects (variables `_W1` and `_W3`) must exist with values `"1"` and `"3"`. These objects must be connected to the hub node (`_Zett`) via `_icw` labelled edges, as they are in fact the world nodes with oids `&w1` and `&w3`. Consequently, they constitute the context defined by the inherited coverage qualifier `[period=winter]`. The effect of this expression is therefore to determine whether the inherited coverage of the `address` entity edge of the MOEM graph is a subset or not of the given context. A similar expression is used for variables `_Z0cxt` and `_Zett`, that correspond to the two context edges of the path. In this way, variable `Z` binds to oid `&6`. This node satisfies the additional restriction `(Z = "22, Vouliagmenis")` in line 8, so the binding of the variable is successful.

In the case of the expression `[P]club.address Y` in the MQL query, the `from` clause of the Lorel query declares the appropriate variables similarly to the previous example, while the `where` clause of the same query contains in lines 12-15 the translation of the MQL `within` condition `[P]*[period=summer] != [-]`. The Lorel expression `select W from club._icw W where W="0" or W="2"` evaluates to the set of the world nodes corresponding to `[period=summer]`, that is nodes `&w0` and `&w2`. This set is intersected with the sets of world nodes connected to the hub nodes (`_Y0cxt`, `_Yett`, and `_Ycxt`) of the three edges in the path. Variable `_Yett` binds to node `&65`, whereas `_Ycxt` to nodes `&66` and `&67`. The intersection is nonempty only when node `&66` is considered, therefore variable `Y` binds only to node `&5`.

The `club.name` context path expression evaluates to node `&3`, where variable `N` binds. None of the hub nodes defined during the translation of this context path expression is present in the `where` clause of the Lorel query, because it does not include any restrictions on context. Since the `select` clause of the MQL query states that variables `Y` and `N` be returned, the result expected from the system is the set of nodes `&5` and `&3`.

4.3 Results Presentation

The translated Lorel query is sent by the Client Communication Manager to the Web Server as depicted in Figure 3. It is then submitted to Lore through the Lore Interface, and the result is returned to the Client Side in the form of an OEM graph. Using the reverse process of that described in Section 4.1, the OEM graph is transformed by the Data Model Transformer module to the MOEM that constitutes the answer. The final result of the query is presented to the user through the graphical interface of the client application. In the case of our example, the answer returned is depicted in Figure 5, and is an MOEM that happens to have only context nodes and entity edges.

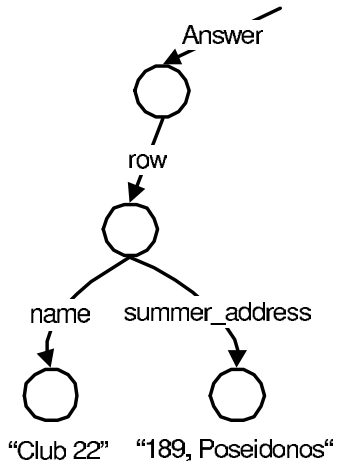

Fig. 5. Query result as an MOEM graph

5 Implementation

The programming language used for the development of all the Client Side components in Figure 3 as well as the Web Server is Java. The Lore Interface module was developed in C++, as it uses the C++ API of Lore, which runs exclusively on Linux. The client-server architecture offers to the query evaluation system cross-platform operability, since the client side runs on any operating system supporting Java, and is not obliged to run on Linux like Lore.

Figure 6 displays a screenshot of the Query Interface of MSSDesigner. Through the "Query System" menu, the user is able to initialize the database for query submission, as described in Section 4.1. After successful initialization the user can submit queries to the system through the window "MQL Query", that for each submitted query displays in the bottom the equivalent Lorel query that is sent for evaluation to Lore. The result returned is displayed in a separate window entitled "Query Result".

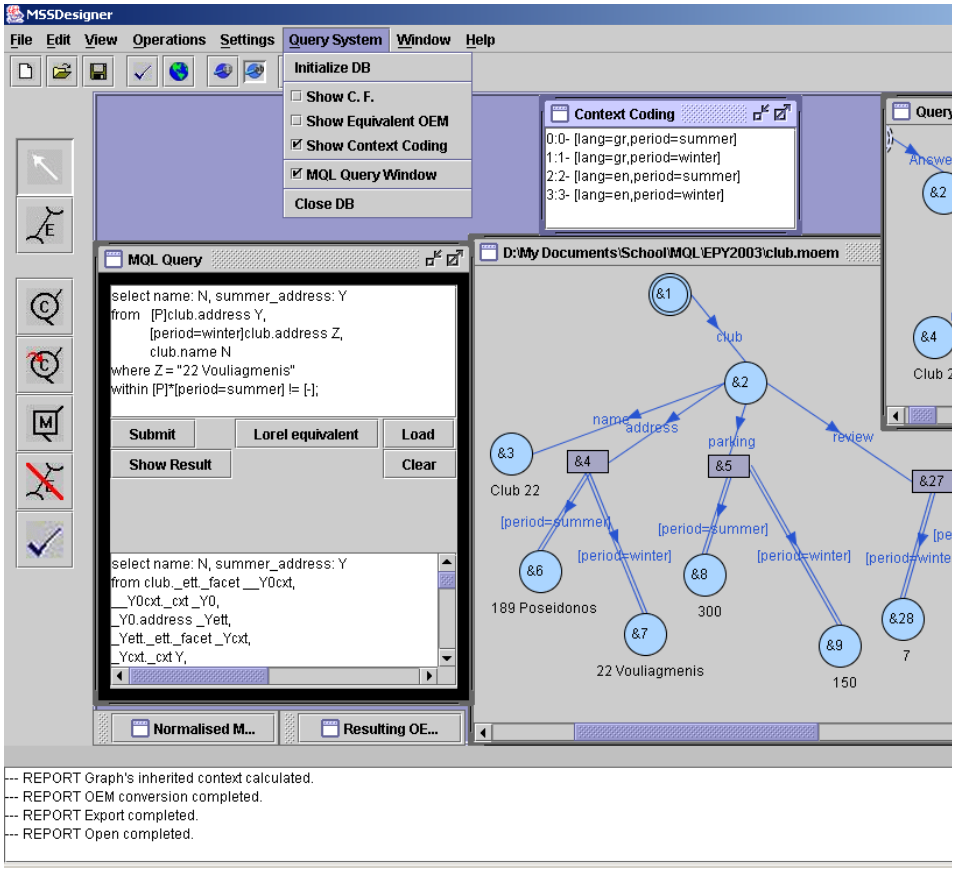

Fig. 6. Evaluating an MQL query through MSSDesigner.

6 Conclusions

In this paper, we presented an MQL query evaluation system, which is part of a platform for managing MSSD. We described the system architecture and discussed its modules in detail. Using a sample MOEM database, we demonstrated the operation of the system modules by following an MQL query example through its evaluation steps. MQL directly support cross-world queries, which have no counterpart in context-unaware query languages. By utilizing the Lore DBMS and the Lorel query language for evaluating MQL queries, we were able to intuitively compare the expressiveness of these data models and query languages when context-driven queries are involved. In future work we plan to re-implement MQL from scratch, and investigate performance and optimization issues.

References

1. P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. V. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, J. Ullman. The Asilomar Report on Database Research. *ACM SIGMOD Record*, 27(4): 74-80, (1998)
2. Y. Stavrakas, M. Gergatsoulis. Multidimensional Semistructured Data: Representing Context-Dependent Information on the Web. *Proceedings 14th International Conference on Advanced Information Systems Engineering (CAiSE)*, Toronto, Canada, (2002)
3. D. Suciu. An Overview of Semistructured Data. *ACM SIGACT News*, 29(4):28-38, (1998)
4. Y. Stavrakas. Multidimensional Semistructured Data: Representing and Querying Context-Dependent Multifaceted Information on the Web. Ph.D. Dissertation, Department of Electrical and Computer Engineering, National Technical University of Athens, Greece, (2003)
5. S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68-88, (1997)
6. Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, V. Zafeiris. Accomodating Changes in Semistructured Databases Using Multidimensional OEM. *Proceedings 6th East European Conference on Advances in Databases and Information Systems (ADBIS)*, Bratislava, Slovakia, (2002)
7. Y. Stavrakas, M. Gergatsoulis, C. Doulkeridis, V. Zafeiris. Representing and Querying Histories of Semistructured Databases Using Multidimensional OEM. *Information Systems*, in print.
8. S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, (2000)
9. V. Zafeiris, C. Doulkeridis, Y. Stavrakas, M. Gergatsoulis. An Infrastructure for Manipulating Multidimensional Semistructured Data. *Proceedings 1st Hellenic Data Management Symposium (HDMS)*, Athens, Greece, (2002)
10. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, J. Widom. Lore: a Database Management System for Semistructured Data. *ACM SIGMOD Record*, 26(3):54-66, (1997)