Representation of Map Objects with Semi-Structured Data Models

Emmanuel Stefanakis

Department of Geography, Harokopio University, 70 El. Venizelou Ave., 17671 Athens, Greece, e-mail: estef@hua.gr

Abstract

Mapping agencies world-wide produce a range of different types of maps, in efforts to meet different user requirements. These maps are usually designed in a digital environment and some of them are distributed through the World Wide Web. One important issue for those agents is the appropriate modelling and handling of map objects from which the digital maps are composed. , This paper focuses on this aspect and as such addresses two aspects. First, the semi-structured nature of data composing a map is discussed. Second, the Object Exchange Model (OEM), a popular model for semi-structured data, is adopted to represent a map. Several configurations regarding the representation of a map in OEM are proposed. Finally, it is shown how useful information can be extracted from those configurations using Lorel query language for semi-structured data.
Keywords: semi-structured data, OEM, Lorel, XML

1 Introduction

Semi-structured data has recently emerged as an important topic of study for a number of reasons (Buneman, 1997): (a) there's a need to handle data sources that cannot be constrained by a schema, through the web (b) there's a need for a flexible format for data exchange between remote databases, and (c) there's a need to assist the browsing on (structured or semi-structured) data based on a self-describing schema. The use of semi-structured data models has been investigated in the past mostly in more traditional data, such us data for bibliographic databases. Expressive data models for representing semi-structured data have been proposed and efficient languages to query semi-structured data have been developed (Abiteboul, 1997; Suciu, 1998).

This study focuses on the use of semi-structured data models for representing *map objects*. Currently there's an increased demand from mapping agencies to

produce digital products and distribute them electronically through the web (Kraak and Brown, 2001). When applicable, map objects are modelled using proprietary models of software systems (CAD, GIS, etc.) which are applied to generate the digital maps. Although, some of those systems already use sophisticated models, such as the object-oriented model, several deficiencies emerge. First, map data is usually only semi-structured and traditional database models are not well suited to represent it, primarily because they assume a constrained schema. Second, the database schema is strict and when data is exchanged well described metadata should accompany the data so that the recipient is well advised of the detail.

The scope of this paper is twofold. First, to discuss the advantages of using semi-structured data for composing a map, and second, to show how semi-structured data models can be adapted to model map data. Taking into account the current developments on the web technology and the rapid spread of *XML* (eXtensible Mark-up Language) (W3C, 1998) and its extensions, the reader would expect the above specification to be adopted for modelling map objects. However, this is not the case in this paper, and *OEM* (Object Exchange Model) (Papakonstantinou *et al.*, 1995) has been chosen instead. There are several reasons for making this choice.

First, OEM is a pure database model for semi-structured data, in contrast to XML, which has been built for the exchange of structured data over the web. It is argued (Abiteboul *et al.*, 2000) that the XML specification and its extensions, although suitable for modelling semi-structured data, will not solve the problem of efficiently extracting the required portion of the underlying data, in other words, XML does not efficiently query large sets of data and extract the required information. The solution to this problem is in part through the development of parallel databases that have semi-structured data. It is therefore well recognised that the combination of XML with semi-structured data models (OEM is the most popular of them) will yield a new technology for web data (Abiteboul *et al.*, 2000) and generally for information exchange across agencies.

Second, considerable efforts have been given to advancing or improving the representation of geographic information with XML, and *GML* (Geographical Mark-up Language) (OpenGIS, 1999). On the other hand, the representation of geographic information with semi-structured data models does not, as yet, appear to have been fully investigated.

Third, it has already been shown that an OEM representation can be generally mapped to an XML file (Suciu, 1998; Abiteboul *et al.*, 2000). XML does not have an associated data model, OEM on the other hand, provides a more suitable and convenient environment to model data.

The discussion is organized as follows. Section 2 briefly presents the application requirements, i.e., the modelling of map objects, and provides a simplified example of a map object collection, to demonstrate the discussion. Section 3 briefly presents the *Object Exchange Model* (OEM), which is the most popular semi-structured data model found in the literature (Papakonstantinou *et al.*, 1995) and has been adopted in this study. Section 4 elaborates on the semi-structured nature of data involved in a map. Section 5 shows how an individual map can be modelled in OEM, while section 6 is focused on the operational

issues. Specifically, it shows how *Lorel* language, designed to query semi-structured data (Abiteboul *et al.*, 1997; Quass *et al.*, 1995), can be used to extract information from an OEM representation of a map. Finally, Section 7 concludes the discussion by summarising the contribution of the paper and giving hints for future research.

2 The Application Domain

A map, either on paper or in digital form, can be seen as a database. It consists of a set of objects (or entities), the *map objects*, which represent real world entities, physical (e.g., a road, a river, etc.) or conceptual (e.g., poverty, criminality, among other aspects.), on the map. The representation of an entity on the map may take different forms, depending on the map scale, the map content, or the map scope (Keates, 1989).

Any map object, presented either in analogue or digital form, can be considered as the *representation instance* of some real world entity. A map object has several dimensions. First, it has a unique identifier. Second, it is assigned the identifier of the real world entity it represents. Third, it has a specific *cartographic symbol*. Fourth, it is positioned in a specific location on the medium, namely it has *geometry*. Both the symbol and geometry are characterised by the *symbol type*, which may be (Keates, 1989): (a) point, (b) line, (c) area, and (d) text, or combinations of them. Each symbol, depending on its type, is described by appropriate graphic *parameters*, such as form, dimension, colour, orientation, pattern, font, etc.

Fig. 1a presents a simplified example map. This map consists of a set of map objects, which represent real world entities on a paper medium. These objects are assigned unique identifiers (Fig. 1b). Letters *P*, *L*, *A*, and *T* refer to the type of the symbol, i.e., point, line, area and text, respectively. The bridge located at the crossing between main road *L2* and lake *A1* (LakeA) is a point symbol, with parameters: colour = “black”, orientation = 30deg, dimension = 8mm, form = “II-like” (a geometric description might be given instead). Main road *L2* is a line symbol, with parameters: colour = “red”, dimension (i.e., width) = 2mm, form = “solid line". LakeA, represented by *A1*, is an area symbol, with parameters: colour = “cyan”, pattern = “plain”. Finally, annotation “LakesB”, represented by *T2*, is a text symbol with parameters: colour = “black”, font = “Times bold”, dimension = “16pt”, orientation = -45deg.

The map objects geometry, is a function of symbol type. In other words, the map object geometry can be seen as an ordered set of primitive geometric elements. An element is the basic building block of geometry, i.e., point, line (or polyline) and area (or polygon). This schema, also adopted in spatial database management systems (DBMS), such as Oracle Spatial (Oracle, 2000), is capable of modelling simple and complex map object geometries. Text objects can be handled as objects of type either point or line. In the former case, text centroid is

the point of reference. In the latter case, a polyline models text curvature (Keates, 1989).

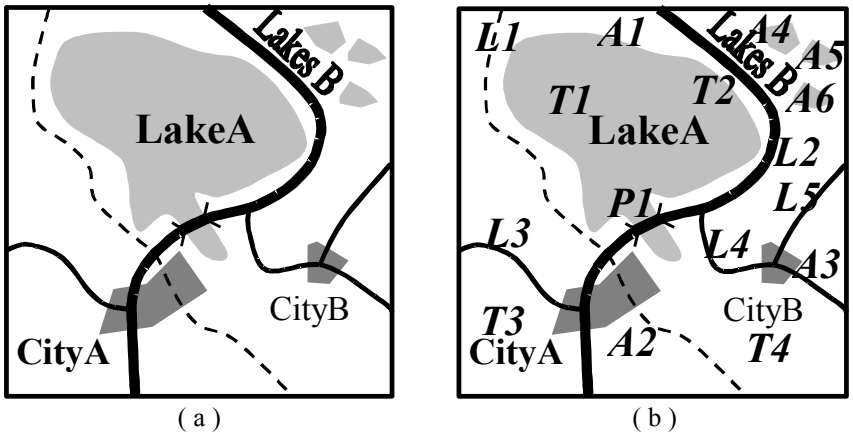


Fig. 1(a) A simplified example map, **(b)** with map objects labelled

Although the map in Fig. 1 contains only a few simple symbols and by no means can be considered as a complex map representation, it is adequate to assist the discussion on the highlights of semi-structured data necessary for composing a map. Further, it will show how semi-structured data models can be used to efficiently handle map objects.

3 Semi-Structured Data and the Object Exchange Model

The observation that much of today's electronic data does not conform to traditional relational or object oriented models, combined with the need to integrate heterogeneous data, has led to the development of data models for handling semi-structured data. Additionally, query languages for handling this data have recently been proposed and investigated (Abiteboul, 1997; Buneman, 1997; Suciu 1998).

Semi-structured data is characterised by an irregular structure. Some objects may have missing attributes, while others may have multiple occurrences of the same attribute. Further, a single attribute may share different objects, and semantically related information may be represented differently for various objects.

The most popular model of semi-structured data is the *Object Exchange Model* (OEM) (Papakonstantinou *et al.*, 1995), originally introduced for the Tsimmis data integration project (Chawathe *et al.*, 1994). OEM is a simple, self-describing, model, where data is represented by a collection of objects. There is no need to define the structure of an object in advance, and there is no notion of a fixed schema or object class. In a sense, each object contains its own schema.

Every object in the OEM data model has an *identifier*, a *label*, a *type*, and a *value*. The *identifier* uniquely identifies the object among all objects in the domain of interest. The *label* is a string (the tag) presumably denoting the "meaning" of the object. Labels may be used to group objects by assigning the same label to related objects. Labels are usually drawn from a universe of label names. The *type* refers to the data type of object value. The *value* can be of a scalar type, such as an integer or string, or it can be a set of (sub)objects.

Each object may be *atomic* or *complex*. The value of an atomic object is of some base type (e.g., integer, string, image, sound, etc.). The value of a complex object is a set of subobjects, i.e., a set of object identifiers.

OEM is a collection of tagged values and is usually represented as a graph, where the nodes are the objects, the edges are labelled with object labels, leaf nodes are atomic objects associated with an atomic value, and non-leaf nodes are complex objects pointing to their subobjects. The graph also has a root, which is a distinguished object with access to all other objects.

OEM is much simpler than object-oriented models. It supports only *object nesting* and *object identity*, while other features such as classes, methods, and inheritance are omitted. When dealing with semi-structured data the OEM has several advantages as compared to relational or object-oriented models (Abiteboul, 1997; Buneman, 1997; Suciu, 1998).

A representative general-purpose data management system for semi-structured data, which uses OEM, is *Lore* (Lightweight Object Repository), which provides *Lorel* query language (Quass *et al.*, 1995). Notice that contrary to traditional data models, in OEM it is not required to be aware of the schema in order to pose a query. The schema can be discovered as queries are posed (Papakonstantinou *et al.*, 1995).

4 Map Objects are Semi-Structured in Nature

Data involved in a map forms a special category of semi-structured data. The following paragraphs explore this issue.

The symbols used to represent real world entities on the map may have a complex type described by many and variable parameters. Specifically, there are four basic symbol types, i.e., point, line, area or text, which can also be combined. Each type is described by different parameters (Keates, 1989). For instance, a point symbol is usually described by three parameters: form, colour, and dimension. There are, however, cases where more parameters are applied, such as orientation for non-symmetric symbols, or additions, extensions and iconic forms depending on the choice of the cartographer. The use of images, in place of point symbols, is also possible. Things are more complicated when dealing with line or area symbols, where additional parameters may be present, such as multiple lines, or area patterns with variable orientations and densities. Text symbols, used in annotations, are also hard to model, provided that a text is described by its colour,

type (or font), size, rotation and curvature. The whole problem of modelling symbols becomes much more complex if the requirement calls for symbols that combine two or more types, e.g., point and area types, or all three point, line and area types. Hence, it is hard to generate a schema in advance that can describe every symbol used in the map, especially in a cartographic organisation, where symbology is a subject of continuous revision.

The granularity of space, i.e., the precision of coordinates describing geometry, is variable and depends on source data and map scale. Hence, the geometry of a map object is expressed with variable precision. Even further, in some cases the coordinate reference system for geometries may not be the same for all map objects. This situation may occur when geometric data is derived from different sources (e.g., land surveys, photogrammetric detail, and paper map digitisation, among others) and they are maintained in the database without being transformed to a common reference system. The simultaneous and integrated visualisation of this type of data can only be obtained by performing appropriate registration procedures.

The units used to express the parameters of symbols may not be unique. For instance, line width may be expressed in units on the map or on the ground. In addition, the units in each case may vary between symbols. For example, the use of mm/inches or meters as integers/meters as floats with two decimal figures, respectively. Another example concerns the colour value, which can be expressed as a triple in a RGB system, as another triple in a HSL system or simply as a standard string, such as "dark red".

All features above render objects involved in a map to form a category of semi-structured data. It is apparent that defining a schema, that uses traditional data models, and that will meet the future modelling requirements of the larger mapping agencies (e.g., national cartographic organisation) is a complex and demanding task. It seems that a more convenient configuration is the adoption of a *semi-structured database* (Suciu, 1998), which does not require an a-priory schema, and is based on a self-describing model that is readily extensible.

The following discussion is focused on how a semi-structured data model, and specifically OEM, can be applied to model map objects.

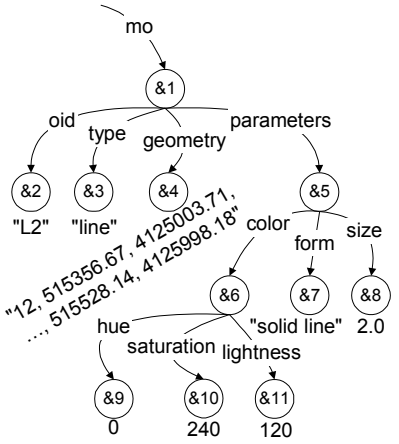
5 Representation of a Map in OEM

Consider the map in Fig. 1. This map can be modelled in OEM. The map consists of complex and atomic objects. An example of an atomic object is the dimension (i.e., width) of a linear symbol, e.g., "1.5 mm". Another example is a point location described by a string that has two double numbers, the coordinate values for X and Y in a reference system, separated by a comma; for example, the UTM coordinates "515456.67, 4125113.71".

An example of a complex object is the main road, the thick solid line, represented by *L2*, in the map. This object has four sub-objects (with labels) which are as follows:

(a)"oid", an atomic object with a value that the user identifier assigns to the representation instance, i.e., *L2*;
(b) "type", an atomic object that accommodates the value "line" and describes the symbol type used to represent the road;
(c) "geometry", an atomic object with a value of type string, e.g., "12, 515356.67, 4125003.71, 515358.23, 4125110.29, …, 515528.14, 4125998.18", where the first integer, i.e., 12, refers to the number of vertices composing the polyline, and the twelve pairs of numbers that follow are the UTM coordinates of polyline vertices, separated by commas, and;
(d) "parameters", a complex object, that points to three sub-objects, with labels "colour", "form" and "size".

Objects labelled as "form" and "size" are atomic, with values "solid line" (a string) and 2.0 (a float) respectively, while an object labelled as "colour" is a complex object pointing to three atomic objects labelled as "hue", saturation", and "lightness" with values 0, 240, and 120 respectively, representing in this case, red.. Fig. 2 presents the graphical and textual representation of the main road in OEM. Notice that *mo* stands for "map object".



<&1, mo, set, {&2, &3, &4, &5}>
<&2, oid, string, "L2">
<&3, type, string, "line">
<&4, geometry, string, "12, 515356.67, 4125003.71,…,515528.14, 4125998.18">
<&5, parameters, set, {&6, &7, &8}>
<&6, colour, set, {&9, &10, &11}>
<&9, hue, integer, 0>
<&10, saturation, integer, 240>
<&11, lightness, integer, 120>
<&7, form, string, "solid line">
<&8, size, float, 2.0>

Fig. 2. Example representation of a map object in OEM

It is clear, that in the above case, the OEM representation is based on some hidden semantics. For example, the size of line symbol refers to its width and is expressed in millimetres. If units are not the same, the label may be used to convey this fact. For example, label "size" can be replaced by label "size-in-mm" (Papakonstantinou, 1995), and similar aspects apply to the "form", of the coordinate reference system, among other aspects.

In addition, the representation of polyline geometry as a string assumes that the parser is aware of the string syntax. An alternative way to represent this geometry is by a set of point objects, i.e., the vertices of the polyline. At this point the problem of ordering the vertices, in an effort to form the polyline, arises. One way

to overcome this inefficiency is to use appropriate labels on point objects, which can depict the sequence of vertices. An alternative way is to represent each point as a complex object with two sub-objects, which consist of the point order in polyline and the point location. A third solution is to extend OEM, so that complex objects accommodate ordered lists of sub-objects instead of sets. This last solution has been already proposed for mapping an OEM to an extensible mark-up language (XML) document (Suciu, 1998b). The latter is by definition ordered, while the former is not. As for the point geometry representation, the "X,Y" string can be readily replaced by two atomic objects with labels "x" and "y", and of type double.

In the following, geometry in 2D space is represented according to the configuration shown in Fig. 3. Specifically, *point geometry* is a complex object, which points to two atomic sub-objects, with labels "x" and "y". These sub-objects accommodate projection or geographic coordinates of the point location in space. *Line geometry* is a complex object, which points to as many sub-objects (with labels "vertex") as the number of vertices composing the line. Each sub-object is in turn a complex object, which points to three atomic sub-objects, with labels "x", "y" and "order", where the first two accommodate the coordinates of a line vertex, whereas the third is the order of this vertex in the polyline. In a similar way, *area geometry* is represented. The only difference, which does not affect OEM representation, is that area outline has one more line segment, connecting the last vertex in the sequence with the first. Notice that this geometry representation scheme is compatible with original OEM definition, where no ordering of sub-objects is assumed.

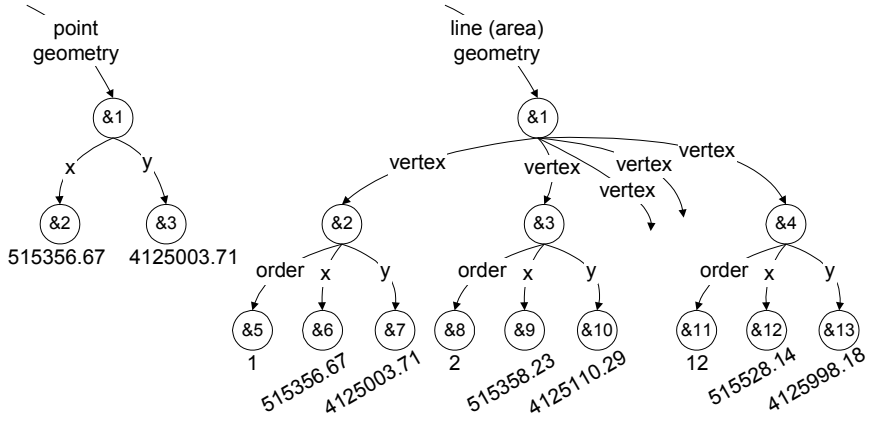


Fig. 3. Geometry representation in OEM

Coming back to the map in Fig. 1, it can also be seen as a complex object, consisting of a set of sub-objects, the representation instances of cartographic entities (i.e., the map objects). Hence, OEM can be used to model this map. In the graphical representation of OEM, the root of the graph is the map, while the leaves accommodate the user identifier (oid), the geometry, the symbol type and the

symbol parameters of the map objects. Fig. 4 presents a portion of the OEM diagram used to represent the map. For simplicity, several symbol parameters are relatively abstract, e.g., a unique string represents colour (e.g., "red").

As mentioned previously, the object label in an OEM can, by definition, be used to group related objects. Hence, objects may primarily be grouped based on their type, i.e., point, line, area, and text. Further, they may be grouped based on their properties. For example, all linear objects representing roads can be nested in a super-object labelled as "road network". Obviously, this configuration approaches a cartographic database (or GIS database) organisation.

The use of symbols in representing cartographic entities is not arbitrary. It is usually based on some standards adopted by organisations or companies producing maps. In most cases, those standards define the symbol types and parameters to represent a cartographic entity at a specific scale. Such a standard forms a *symbol library* (Robinson *et al.*, 1995). It is a common practice to model a symbol library in a separate schema, where map objects point to.

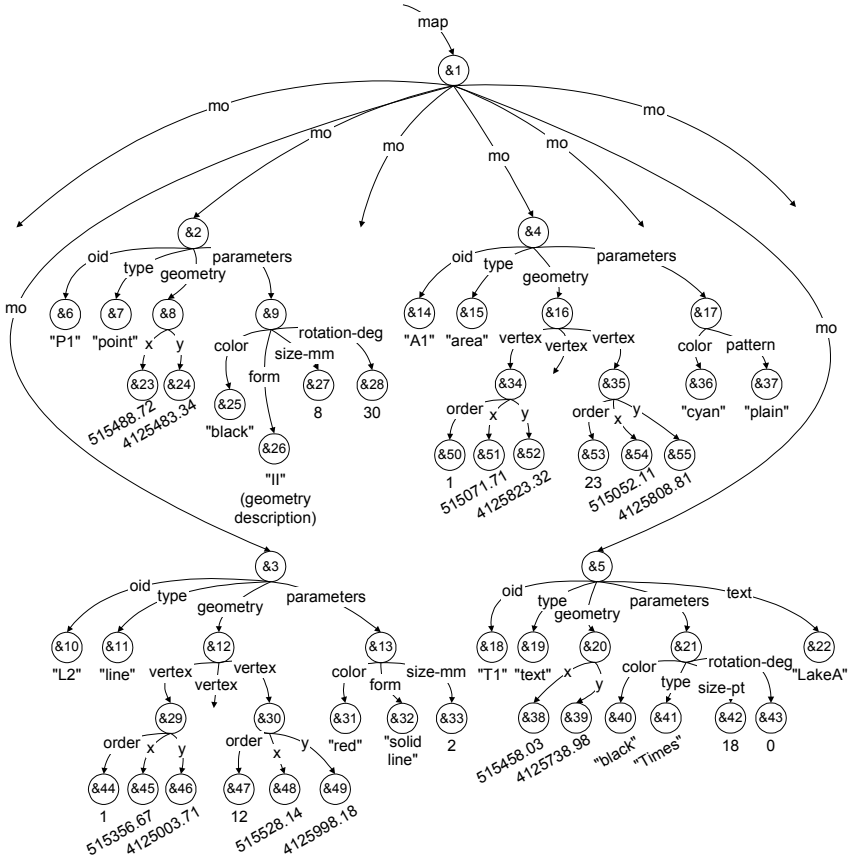


Fig. 4. A portion of the OEM diagram used to represent the map of Fig. 1

When an OEM is adopted to represent the maps of an organisation, a separate OEM can be adopted to accommodate the common symbols used in these maps. Fig. 5 presents a portion of an OEM diagram used to model the symbol library adopted in the map of Fig. 1. The same figure shows an alternative version of OEM in Fig. 4, whose map objects point to the symbol library. Geometry sub-objects are omitted for simplicity.

This configuration is flexible, when graphic limitations or other reasons require the representation of a cartographic entity and it's symbol, which does not conform to the symbol library standard or is not included in the symbol library. In this case, the specific symbol parameter object is attached to the map object directly as a sub-object. An example is given in Fig. 5, for the bridge map object *P1*.

6 Operational Issues

The discussion so far has focused on the representation of a map in OEM. It is argued that this is a convenient practice, due to the semi-structured nature of the map objects involved. The rest of this discussion is focused on handling issues such as extracting information from an OEM representation of a map. The task to be accomplished requires the existence of a query language, able to handle semi-structured data.

In the past, several languages have been proposed to query semi-structured data (Abiteboul, 1997). This study adopts *Lorel*, the query language supported by *Lore system* (Quass *et al.*, 1995), which makes use of the OEM data model. Lorel query language is a compatible extension to the OQL object-oriented query language (Cattell, 1994), with new features designed specifically for querying semi-structured data. Briefly, those features are: partially specified path expressions, wildcards, automatic type coercion in comparisons, and a special semantics for disjunction. Unlike OQL, Lorel does not enforce strong typing, and allows querying and schema browsing when the object structure is unknown or partially known. In other words, the uniqueness of Lorel includes: (a) the extensive use of *coercion*, and (b) the powerful *path expressions* (Abiteboul *et al.*, 1997).

The scope of this Section is to show how Lorel can be used to support some basic queries in the application domain of handling a map, which has been modelled using a semi-structured data model, specifically OEM. This task is accomplished through a series of examples. For brevity, it is assumed that the reader is familiar with Lorel query language syntax (Abiteboul *et al.*, 1997). All example queries refer to the database (fragment) in Fig. 6. The database refers to the map shown in Fig. 1. The graphical OEM representation of this database is given in Fig. 4.

The textual syntax adopted in Fig. 6 has been simplified according to Abiteboul *et al.* (1997). Tabs have been used to represent the nesting of objects. Each object has a unique *object identifier* (obj_id). Some objects are atomic and contain a value from one of the disjoint basic types, e.g., integer, real, string, gif, html,

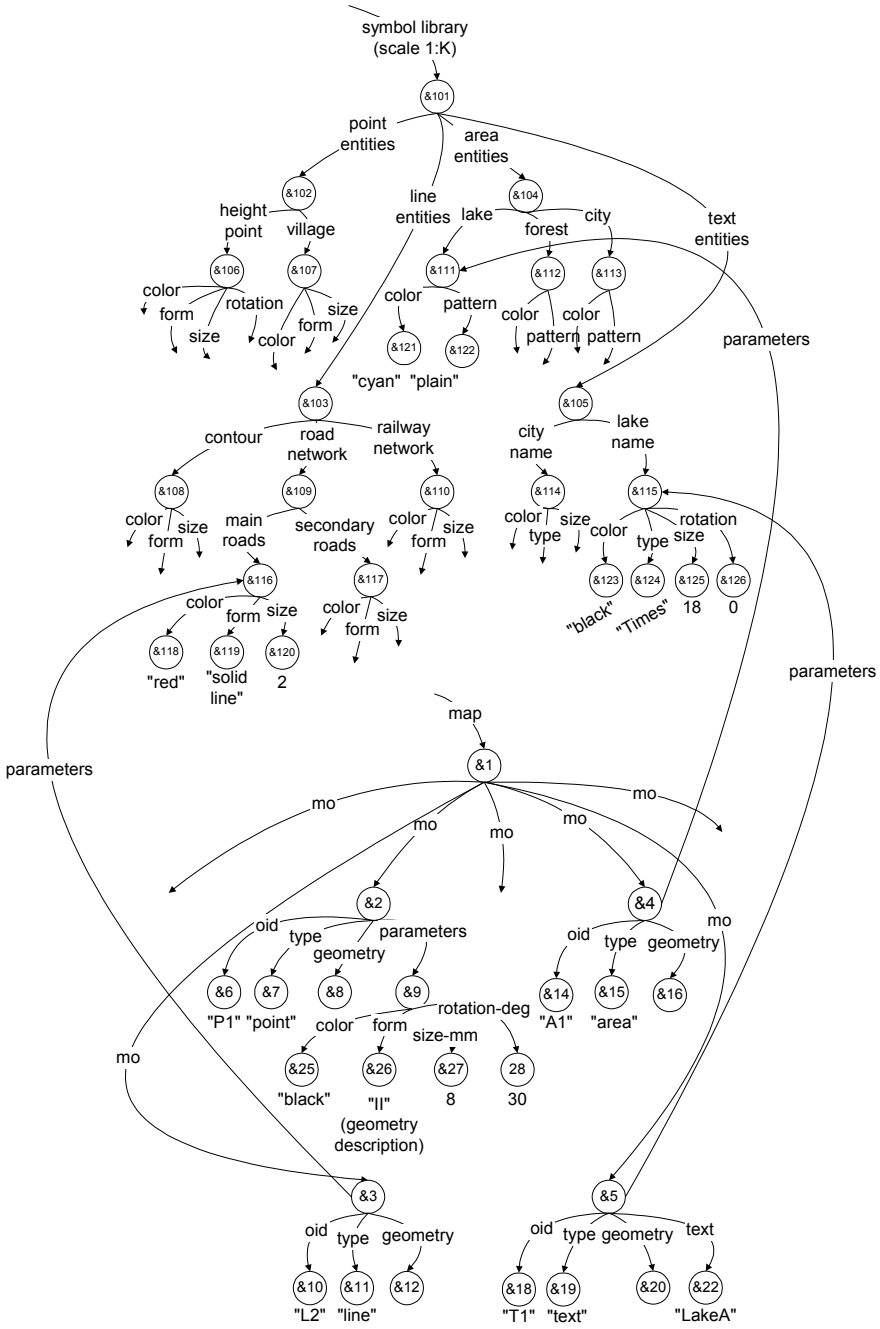


Fig. 5. The symbol library and an alternative version of OEM in Fig. 4

audio, java, etc. All other objects are complex; their value is a set of object references, denoted as a set of (*label, obj_id*) pairs. The labels are all taken from the atomic type string.

Notice that "object identifiers" retrieved in most of the query examples that follow and denoted by "oid" refer to the user-defined labels assigned to map objects in Fig. 1b (e.g., *P*1, *L*3, *A*5) and differ from those system-generated object identifiers (denoted by "obj_id") described in the previous paragraph.

```
map &1
    mo &2
        oid &6 "P1"
        type &7 "point"
        geometry &8
            x &23 515488.72
            y &24 4125483.34
        parameters &9
            color &25 "black"
            form &26 "II"
            size-mm &27 8
            rotation-deg &28 30
    mo &3
        oid &10 "L2"
        type &11 "line"
        geometry &12
            vertex &29
                order &44 1
                x &45 515356.67
                y &46 4125003.71
            …
            vertex &30
                order &47 12
                x &45 515528.14
                y &46 4125998.18
        parameters &13
                color &31 "red"
                form &32 "solid line"
                size-mm &33 2
    mo &4
        oid &14 "A1"
        type &15 "area"
        geometry &16
            vertex &34
                order &50 1
                x &51 515071.71
                y &52 4125823.32
            …
            vertex &35
                order &53 23
                x &54 515052.11
                y &55 4125808.81
        parameters &17
            color &36 "cyan"
            pattern &37 "plain"
    mo &5
        oid &18 "T1"
        type &19 "text"
        geometry &20
            x &38 515458.03
            y &39 4125738.98
        parameters &21
                color &40 "black"
                form &41 "Times"
                size-pt &42 18
                rotation-deg &43 0
        text &22 "LakeA"
```

Fig. 6. A textual OEM database (graphical representation in Fig. 4)

Query 1: "Find the identifiers (oid) of all line or area objects present in the map"

Lorel expression:

```
select A.oid
from map.mo A
where A.type = "line"
or A.type = "area"
```

Answer object:

```
answer &101
    oid &10 "L2"
    oid &14 "A1"
```

This is a simple query example. It involves two conditions which are both included in a 'where clause'.

Query 2: "Find all black-colored map objects and list their type"

Lorel expression:

```
select A, A.type
from map.mo A
where A.parameters.color = "black"
```

Answer object:

```
answer &102
    mo &2
        type &7 "point"
    mo &5
        type &19 "text"
```

With the query above, map objects can be separated based on their colour. This operation is usually applied in the cartographic production process (Keates, 1989).

Query 3: "Find the identifiers (oid), along with coordinates, of all point objects that fall inside the rectangular window with lower left (south-west) corner (515000, 4125000) and upper right (north-east) corner (515500, 4125500)" (this is the south west quadrangle of the map in Fig. 1).

Lorel expression:

```
select B.oid, A.x, A.y
from map.mo{B}.geometry A
where B.type = "point"
and A.x >= 515000
and A.x <= 515500
and A.y >= 4125000
and A.y <= 4125500
```

Answer object:

```
answer &103
    oid &6 "P1"
        x &23 515488.72
        y &24 4125483.34
```

This is a simple *window query* example. This query is commonly applied in map browsers and GIS software packages and is well known as a *zoom-in* operation. The extension of this query to select line and area objects involves spatial operators, which are not provided by Lorel query language. Notice that this is one of the open issues for future research.

Query 4: "Find the identifiers (oid) of all point objects and line/area objects that have a vertex (for line and area objects) at position (515488.72, 4125483.34) (i.e., the position of the bridge in Fig. 1).

Lorel expression:

```
select B.oid
from map.mo{B}.geometry(.vertex)? A
where A.x = 515488.72
and A.y = 4125483.34
```

Answer object:

```
answer &104
    oid &6 "P1"
    oid &10 "L2"
```

(*L2* has a vertex on the bridge)

or alternatively,

```
select B.oid
from map.mo{B}.geometry A
where A.#@P.x = 515488.72
and A.#@Q.y = 4125483.34
and P == Q
```

This is a simple *point query* example. This query is also commonly applied in map browsers and GIS. Usually, the user is prompted to select an object by clicking with the mouse on the graphical screen. The query as expressed above restricts mouse clicks to coincide with a point object location or a line/area object

vertex location. The extension of this query to select point, line and area objects with a tolerance of, e.g., 0.5mm (i.e., distance less than 0.5mm on the screen, from the mouse click position), involves spatial operators, which are not provided by the Lorel query language. The path expression given in the 'from' clause of the first alternative query guarantees a path to x and y coordinates of both point objects and line/area object vertices.

The second alternative query is a more complex expression, but highlights the flexibility provided by Lorel language such that :

(a) it involves the use of wildcard (#), which matches to any data path of length 0 or more (here, 0 for point objects and 1 for line/area objects); and
(b) the use of path variables P and Q, which assure that x and y coordinate values are examined each time and are assigned to the same line/area vertex.

The last condition in the 'where' clause eliminates from the answer object cases where a vertex of a line/area object has the desired X value and another vertex of the same object has the desired Y value.

Query 5: "Find identifiers (oid) of all line objects that are composed by less than 20 vertices".

Lorel expression:

```
select A.oid
from map.mo A
where A.type = "line"
and A SATISFIES
    20 > COUNT(select B.geometry.vertex
                  from map.mo B)
```

Answer object:

```
answer &105
    oid &10 "L2"
```

This query uses a sub-query as operand to the aggregation operator COUNT. What the sub-query does is to count the number of vertices per line object. Notice that B already has its mapping fixed by the object assignment A (Quass *et al.*, 1995) in the enclosing query. The latter reports line objects, which satisfy the condition posed by the query.

Query 6: "Find the identifiers (oid) of all line objects intersecting area objects. Also report the corresponding area object identifier".

Lorel expression:

```
select distinct A.oid, B.oid
from map.mo A, map.mo B, A.geometry AG,
      B.geometry BG
where A.type = "line"
and B.type = "area"
and AG.x = BG.x
and AG.y = BG.y
```

Answer object:

```
answer &106
    oid &10 "L2"
    oid &14 "A1"
```

This query is a simple example of *join query*. Notice that the key word "distinct" eliminates duplicates from the answer object, e.g., *L*2 intersect twice *A*1, and however the pair appears once in the answer object. This query assumes the existence of coincident vertices on both objects at their intersections. Otherwise,

no map objects will be included in the answer object (due to the lack of spatial operators).

7 Conclusions and Future Research

This paper argues that objects involved in a map form a category of semi-structured data. It adopts OEM, the most popular semi-structured data model that exists in the literature, and shows how objects of an individual map can be represented using alternative configurations. In addition, some basic application domain queries for handling these data are expressed in Lorel, also a popular query language for semi-structured data.

Several issues remain open for future research. One important issue is the integration of topological relations and constraints, which characterise cartographic entities of the real world, in the same model. This information is of significant value to the cartographer when performing several cartographic design and composition processes, such as the generalisation processes (Keates, 1989).

Another direction for future research is the mapping of the OEM representation of a map into XML variants (Suciu, 1998b) and specifically to GML (OpenGIS, 1999). Use of XML is expanding rapidly, and XML data will be exchanged freely in the near future between applications, belonging to the same or different organisations, in the same way HTML documents are currently exchanged. Hence, geographic and cartographic data exchange using XML must be seriously considered. Additionally, as already mentioned, the combination of XML with semi-structured data models will give as result a new technology for web data (Abiteboul *et al.*, 2000).

An issue that requires future examination is the extension of the Lorel language with spatial operators. This will enhance the functionality of the language and enrich the possibilities for spatial semi-structured data handling.

References

Abiteboul S (1997) Querying semi-structured data. In: Proceedings of the Sixth International Conference on Database Theory (ICDT'97). Delphi, Greece, pp 1-18

Abiteboul S, Quass D, McHugh J, Widom J, Wiener JL (1997) The Lorel query language for semistructured data. International Journal on Digital Libraries 1:68-88

Abiteboul S, Buneman P, Suciu D (2000) Data on the Web: From Relations to Semi-Structured Data and XML. Morgan-Kaufmann

Buneman P (1997) Semistructured data. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97). Tucson, Arizona, pp 117-121

Cattell RGG (1994) The Object Database Standard ODMG-93. Morgan Kaufmann

Chawathe SS, Garcia-Molina H, Hammer J, Ireland K, Papakonstantinou Y, Ullman JD, Widom J (1994) The TSIMMIS project: integration of heterogeneous information

sources. In: Proceedings of the Tenth Meeting of the Information Processing Society of Japan, Tokyo, Japan, pp 7-18

Keates JS (1989) Cartographic Design and Production, 2nd Edition. Longman

Kraak MJ, Brown A (2001) Web Cartography: Developments and Prospects. Taylor & Francis

OpenGIS (1999) Open GIS Consortium (OGC) [online]. Available from: http://www.opengis.org/

Oracle (2000) Oracle Spatial user's guide and reference. Oracle Corporation

Papakonstantinou Y, Garcia-Molina H, Widom J (1995) Object exchange across heterogeneous information sources. In: Proceedings of the Eleventh International Conference on Data Engineering (ICDE'95). Taipei, Taiwan, pp 251-260

Quass D, Rajaraman A, Sagiv Y, Ullman JD, Widom J (1995) Querying semistructured heterogeneous information. In: Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases. Singapore, pp 319-344

Robinson AH, Morrison JL, Muehrcke PC, Kimerling AJ, Guptill SC (1995) Elements of Cartography, 6th Edition. Wiley

Suciu D (1998) An overview of semistructured data. ACM SIGACT News 29:28-38

Suciu D (1998b) Semistructured data and XML. In: Proceedings of the 5th International Conference of Foundations of Data Organization (FODO'98). Kobe, Japan

W3C (1998) The World Wide Web Consortium (W3C) [online]. Available from: http://www.w3c.org/