A Data Model to Deal with Multi-Scaled Networks

M. MAINGUENAUD
X.T. SIMATIC

FRANCE TELECOM- Institut National des Télécommunications
9 Rue Charles FOURIER, F91011 EVRY - FRANCE
Tel : + 33 1 60 76 47 82 Fax : + 33 1 60 76 47 80
Email : MAINGUENAUD@FRINT51.bitnet
SIMATIC@FRINT51.bitnet

Abstract :

This paper presents the merge of graph theory concepts and the object oriented paradigm to model network oriented data. The introduction of the master nodes (resp. master edges) allows to define a node (resp. an edge) as an abstraction of a sub-network. A network can be defined using several levels of definition (detail) following the importance of the nodes (resp. the edges) within an application. The object oriented concepts are applied to the alphanumerical parts of the data and to model the topology of the graph. We define in this paper the structure (but not the behavior) of the multi-level networks.

I. INTRODUCTION :

Transport networks (i.e. train, plane, water, telecommunication) may be modeled with the graph theory concepts. Early 1970, numerous works were based on the optimization of the graph operators. The underlying data model was only composed of nodes and edges. The graph theory applied to operations research represents an important theoretical base [4] , [14] for efficient data manipulation operators. Nevertheless, this formalism has a lack of semantic power if we compare it to the semantic data models [6, 9, 10, 15] or the object oriented data models [2, 11].

The main advantage of a Data Base Management System (DBMS) is its ability to handle a huge amount of data. The underlying data model tries to avoid the redundancy of the data [17] because this can involve incoherences while querying or updating the data base. A view mechanism allows to define part of the data base as a user's world. The data model we present in this paper is well suited to the partial definition of a network (view). In fact object oriented concepts such as classification, aggregation, generalization/specialization [16] can be applied to the alphanumerical information parts of the network but also to the topology of this network.

The aim of this paper is to present a data model allying the power of the graph theory and the object oriented concepts (focussing on the structure but not on the behavior). We do not consider the modelization of the alphanumerical information parts since it depends on the application (however important it may be, this work is fully application dependent). Section 2 presents a short recall of the basic graph theory and object oriented notions, section 3 presents the basic elements of the data model, section 4 presents the different levels of abstraction of the networks and section 5 deals with the conclusions.

IGU - UGI Conf.
Brno Tchecoslovaquie
Avril 91 22-25

II. BASIC NOTIONS :

This part presents very shortly the basic central notion of the data model concepts. Further informations can be obtained for example in [4] for the graph theory and in [18] for the object oriented paradigm.

II.1 The basic graph theory concepts :

We model the network with the concept of graph. A graph is a couple (N, E) where N is a set of nodes and E is a sub-set of the cartesian product N x N.

$$N = \{n_1, \ldots, n_p\}$$
$$E = \{ (n_i, n_j) / n_i \in N, n_j \in N \}$$

(n_i is said to be the initial node and n_j is said to be the end node for an edge (n_i, n_j))

In this paper, the graph is considered as oriented. The order, initial node, end node, is relevant. Several edges may be defined within the same initial and end nodes. We do not limit this number but we do not allow an edge (n_i, n_i).

A node is used to model for example a town or a crossing point between two roads. An edge is used to model for example a link between two towns. The nodes and the edges are labeled. We do not consider here in detail the label. Let us suppose we can define a labelling function which allows to name and to give some characteristics to the nodes and to the edges.

II.2 The basic object oriented concepts :

We assume the reader to be familiar with the object oriented paradigm. A class is defined as a group of objects (characterized with some attributes and identified by an OID - Object IDentifier), with the same structure and the same behavior. A type represents the structure of a class.
In this paper, a type is defined using the tuple constructor (aggregation) [], the set constructor { } and with the classical basic types (i.e. integer, string).

The constructors are used to model alphanumerical parts (the labelling function of the previous sub-section). But in this paper they are used to model the topology of a graph in order to allow various levels of definition. Therefore we do not consider the attributes such as the population for a town or the departure and arrival time for an edge.

III. THE BASIC TYPES OF THE MODEL :

The classes are organized following a tree. A node of the tree has a unique "father" (execept the root node). This node may or may not have some "child". This tree represents the relationships between classes and super-classes and may be seen as an inheritance tree of the attributes. The predefined types of the model are : NODE, EDGE and NETWORK. For each of them we give its notation (without taking into account the OID - detailled in IV.3), an example of use and its graphical representation.

III.1 The nodes :

The type NODE is defined by an aggregation of one attribute (name). Its notation is :

NODE = [name : string]

Let us suppose we want to represent a town (node) labeled "Paris". So an object of type NODE is defined. Its representation is :

(oid_n_1, [name = "Paris"]) or shortly (oid_n_1, ["Paris"])

Such a node is graphically represented by :

III.2 The edges :

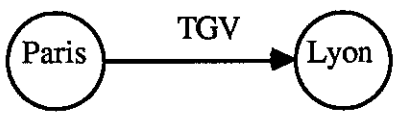
The type EDGE is defined by an aggregation of three attributes (name, initial_node, end_node). Its notation is :

EDGE = [name : string, initial_node : NODE, end_node : NODE]

Let us suppose we want to represent a link (edge) labeled "TGV" between two towns (nodes) labeled "Paris" (oid_n_1, ["Paris"]) and "Lyon" (oid_n_2, ["Lyon"]). So an object of the type EDGE is defined. Its representation is :

(oid_e_1, [name = "TGV", initial_node = oid_n_1, end_node = oid_n_2])
or shortly (oid_e_1, ["TGV", oid_n_1, oid_n_2])

Such an edge is graphically represented by :



Remark : the OID makes the difference between two edges with the same initial and end nodes.

III.3 The network :

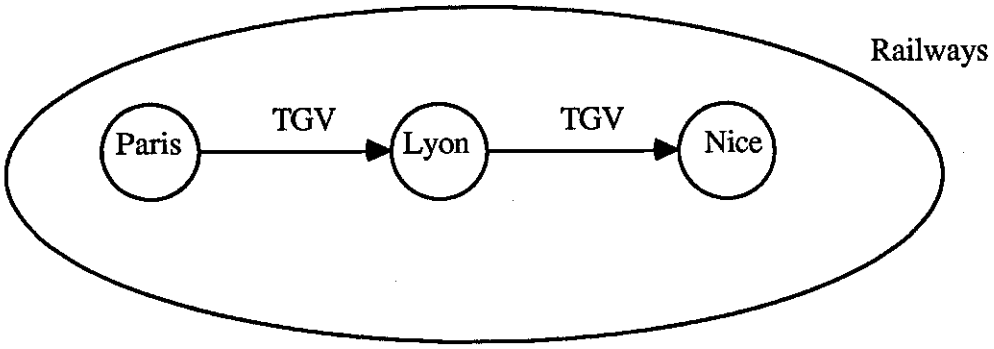
The type NETWORK is defined by an aggregation of two attributes (name, graph). Its notation is :

NETWORK = [name : string, graph : [nodes : {NODES}, edges : {EDGE}]]

Let us suppose we want to represent a network labeled "Railways" with two edges (oid_e_1, oid_e_2). Its representation is :

(oid_N_1, [name = "Railways", graph = [nodes ={oid_n_1, oid_n_2, oid_n_3}, edges = {oid_e_1, oid_e_2}]])

Such a network is graphically represented by an hypergraph [5] :



IV. THE ABSTRACTION :

The decomposition following several levels of abstraction is a natural way of structuring network-oriented data. These levels (or layers) do not introduce redundancy since the inheritance mechanism of the object oriented paradigm gives the propagation of the attributes. Furthermore, a logical decomposition of a network allows to free the conceptor from the physical coordinates in order to evaluate a local path.

It is of prime importance to introduce various levels of abstractions to handle network oriented data [7]. Two levels of abstraction may be considered : (1) defining a node as an abstraction of a sub-network, and (2) defining an edge as an abstraction of a sub-network. Using these two notions and the graphical notations of the previous section, figure 1 and figure 2 represent two ways of defining a network.

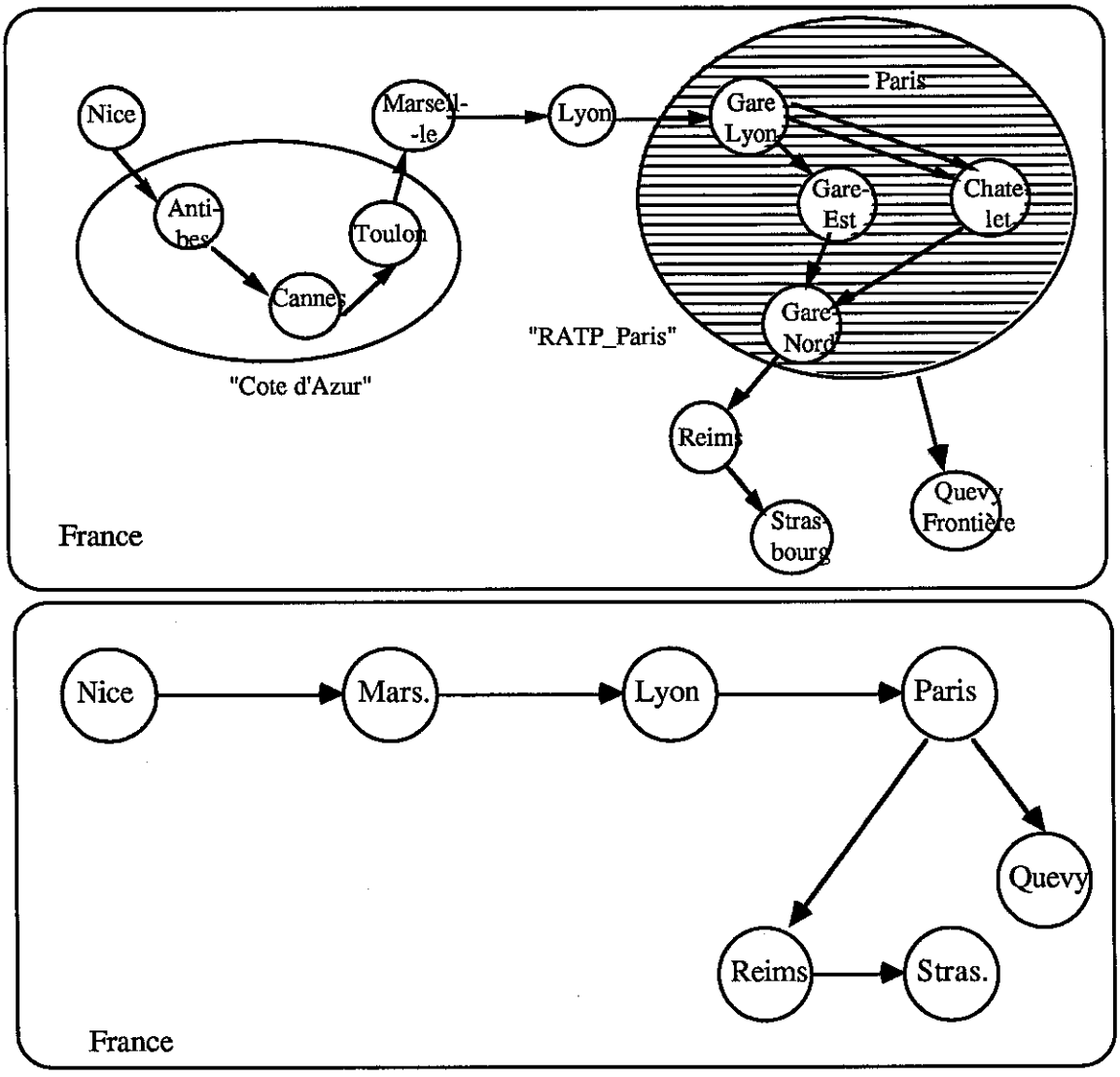


figure 1 and figure 2

This section presents the abstraction of node, the abstraction of edge and the methodology defined to handle these various levels of detail.

IV.1 The MASTER_NODE :

The type MASTER_NODE is a sub_type of the type NODE. This type is defined by an aggregation of four attributes (name, associated_network, in_edges, out_edges). The attribute name is inherited from the type NODE. Its notation is :

MASTER_NODE = [associated_network : NETWORK,
in_edges : {EDGES},
out_edges : {EDGES}
]

Associated_network may be defined as a graph G (N, E). The elements of N are called the specialized nodes. The attribute in_edges (resp. out_edges) represents the set of edges "arriving in" (resp. "leaving") this graph.

$$\text{in\_edges} = \{ (n_i, n_j) / n_i \notin N \wedge n_j \in N \}$$
$$\text{out\_edges} = \{ (n_i, n_j) / n_i \in N \wedge n_j \notin N \}$$

In the example, figure 1, the edge (Lyon, Gare_lyon) belongs to the set in_edges for the master node Paris. The edge (Gare_nord, Reims) belongs to the set out_edges. The attribute associated_network inherits the attribute name from the NETWORK type (in this example "RATP_Paris").

A master_node (i.e. Paris), called a generalized node, is graphically represented by :

IV.2 The MASTER_EDGE :

A similar notion is defined for the edges. The type MASTER_EDGE is a sub-type of the type EDGE. This type is defined by an aggregation of six attributes (name, initial_node, end_node, associated_network, in_edges, out_edges). The attributes name, initial_node, end_node are inherited from the type EDGE. Its notation is :

```
MASTER_EDGE =   [   associated_network : NETWORK,
                    in_edges : {EDGE},
                    out_edges : {EDGE}
                ]
```

Associated_network may be defined as a graph G (N, E). The elements of E are called the specialized edges. The attribute in_edges (resp. out_edges) represents the set of edges "arriving in" (resp. "leaving") this graph.
In the example, figure 1, the edge (Nice, Antibes) belongs to the set in_edges for the master_edge "Cote d'Azur", and the edge (Toulon, Marseille) belongs to the set out_edges.

A master_edge, called a generalized edge, is graphically represented by :

Remark : The main difference between the master_node and the master_edge is that the in_edges and the out_edges of the master_edge belong to the associated network since the edges do not belong any more to the aggregated graph (see figure 2).

Figure 3 presents the network of the figure 1 with the introduction of the master_node and the master_edge.

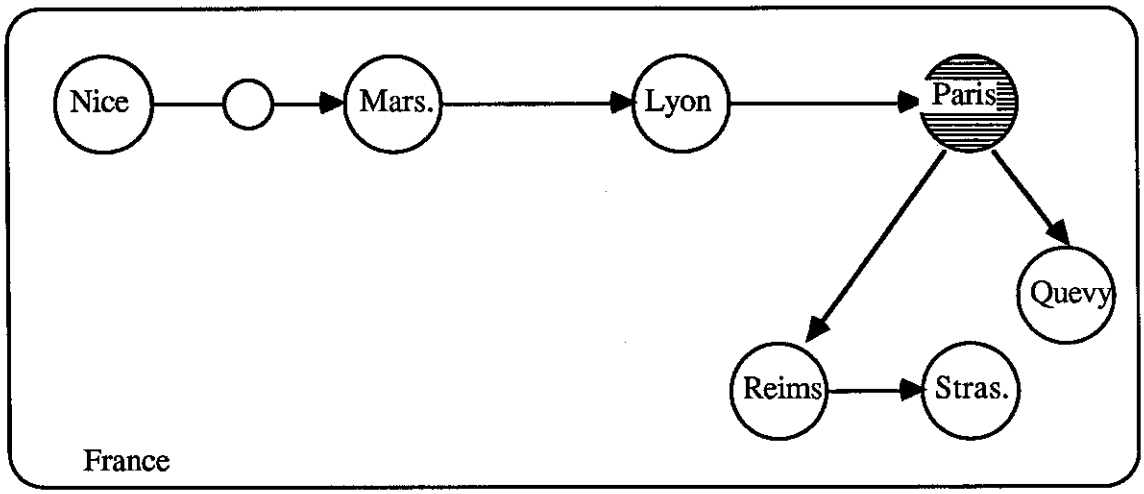


Figure 3

IV.3 The representation of the various levels of abstraction :

Each object (i.e. node, edge) has a logical identifier, the OID. This OID is stratified to take into account the various level of abstraction. The main goals of this structure are (1) to determine very quickly the level of abstraction of an object, and (2) to identify very easily the generalized (resp. specialized) objects. This section presents the node OID, the edge OID and the introduction of the view concept.

The node OID :

A node belongs to a hierarchy of node. Let N be the depth of this hierarchy. The OID has N layers (from 1, the lower level of abstraction, to N, the higher level of abstraction). Let n (a node) belong to the layer k. Two cases may happend :
- n is located at the top level of abstraction (k = N). The first layer has the local OID and the other layers are null.
- n is not located at the top level ($1 \leq k < N$). The (N - k) first layers contain each local OID of the generalized nodes. The layer k contains the local OID. The (k - 1) last layers are null.

Remark : Let n a node of the layer k ($k \leq N - 2$). The generalized node of the node k may not belong to the layer k+1. Therefore a shadow node is created with an arbitrary local OID at the level k+1.

The edge OID :

The OID of an edge is divided into three parts. Part 1 (resp. 2) contains the OID of the initial node (resp. end node). These OID are built following the rule defined in the previous sub-section. With the same philosophy an edge belongs to a hierarchy of edges. The layer is obtained following the layers of the initial and end nodes. Part 3 contains the local OID of an edge (since several edges with the same initial and end node may exist).

The view concept :

The view (or snapshot) concept allows to define a part of the database as the user's world. Within an application a master-node (i.e. Paris in figure 1) may have to be considered as an atomic datum (i.e. Paris in figure 2). To do so, data manipulations are performed over the OID to change a master-node into a node (an object of the class c_1 is always an object of the subclass of c_1).

V. CONCLUSIONS :

This paper presents the basic data structures to manipulate network with various level of abstraction. Such a definition is an alternative way of manipulating networks in comparison with a dynamic aggregation [3]. This model relies on the concept of master-nodes and master-edges which represent an abstraction of sub-networks. These sub-networks are opened to the "world" using the concept of in-edges and out-edges. This paper focusses on the data structure. [12] presents the behavior of the basic objects of this model. The main binary manipulations of these objects are : the union, the intersection, the difference and the graph traversal.
We are currently implementing a toy application using an object oriented Data Base Management System, O2 [1]. The next step is to support this data model in the context of the CIGALES prototype [13] and to compare the object-oriented philosophy with an Extended Relational Data Base Management System based [8] philosophy to handle network-oriented data.

REFERENCES :

1. Bancilhon F. and al : The design and Implementation of O2, an Object Oriented Database System, Proc. of the 2nd Int. Work. on Object-Oriented Data Base Systems, K. Dittrich (Ed) Bad-Munster, FRG, September 1988
2. Banejeree J. : Data Model Issues for Object Oriented Applications, ACM T.O.I.S., vol 5, January 1987

3. Becker R.A. and al : Network Visualization, 4th Spatial Data Handling Symp., Zurich, Switzerland, 24-27 July 1990
4. Berge C. : Graphes, Gauthier-Villars, 1983
5. Berge C. : Hypergraphes, Gauthier-Villars, 1987
6. Brodie M.L., : On the Development of Data Models, On conceptual modelling perspectives from artificial intelligence, databases, and programming languages, Brodie, Myropolous, Schmidt Eds, Springer-Verlag 1984
7. Eck (van) J.R., De Jong T. : Adapting datastructures and algorithms for faster transport network computations, 4th Spatial Data Handling Symp., Zurich, Switzerland, 24-27 July 1990
8. Gardarin and all : Managing Complex objects in an Extensible Relational DBMS, 15th Int. Conf. on Very Large Data Bases, Amsterdam, The Netherlands, 22-25 August 1989
9. Hammer M., Mc Leod : Database Description with SDM : a Semantic Data Model, ACM T.O.D.S., vol 6, n° 3, Sept 1981
10. Hull R., King R. : Semantic Database Modelling : Surveys, Applications and Research Issues" ACM Computing Surveys, Vol 19, Sept 1987
11. Lécluse C., Richard P., Vélez F. : O2, an Object-Oriented Data Model, Proceedings of the ACM-SIGMOD Conference, Chicago, USA, June 1988
12. Mainguenaud M., Raffy J.L., X.T. Simatic : Graph Manipulations for Network Oriented Management - Application to a Telecommunication Network, 14th Urban Data Management Symp., Odense, Denmark, 29-31 May 1991
13. Mainguenaud M., Portier M.A. : CIGALES : a Graphical Query Language for GIS, 4th Spatial Data Handling Symp., Zurich, Switzerland, 24-27 July 1990
14. Nato Conference : The Application of Operations Research to Transport Problems, Sandefjord, Norway, 14-18 August 1972
15. Peckham J. , F. Maryanski F. : Semantic Data Models, ACM Computing Surveys, Vol. 20, N° 3, Sept 1988
16. Smith J.M., Smith D. C. P. : Database Abstraction: Aggregation and Generalization, ACM T.O.D.S., 1977
17. Ullman J. : Principles of Databases, Academic Press, 1980
18. Zdonic S., D. Maier (Eds), Readings in Object-Oriented Database System, Morgan Kaufmann, 1989