

A RULE-BASED APPROACH
FOR SPATIAL OBJECT MODELLING AND TASK MANAGEMENT

Giming Chen

National Land Information System
National Bureau of Surveying & Mapping, Beijing, China

ABSTRACT

A rule-based object/task modelling approach for GIS's is proposed which is characterized by specifying database applications in terms of localized domain knowledge representation and explicit control knowledge representation, and by utilizing complementarily the techniques of database, logic and object-oriented programming to support inferencing and deductive quering.

INTRODUCTION

Relational Database (RDB) and Logic Programming (LP) techniques are moving fast towards a common destination. This holds true both in terms of functionality and performance, and is made possible both from their common ancestry of mathematic logic and the complementary benefits they can provide [Call 84] [Zani 84] [Parker 84] [Ullman 85] [Chen 86].

As we know, logic admits a declarative semantics, that is, logic clauses may be interpreted as statements of facts. In addition, logic admits an imperative semantics in which logic clauses may be interpreted as commands. The general form of clauses that will represent database facts and deductive rules is

$$P_1 \ \& \ P_2 \ \& \ \dots \ \& \ P_k \ \rightarrow \ R_1 \ ; \ R_2 \ ; \ \dots \ ; \ R_q$$

An attempt to answer a query is referred to an attempt to satisfy a goal (or to prove a goal). The correspondence between logical predicates and relations is that a predicate could reasonably be stored as a relation with a type hierarchy (such as an aggregation).

As a simplified geographic objects quering example, suppose there is a plan to install communication lines among towns within a local area, the information necessary is kept in relations (predicates) "plan" (indicating the central town assigned and the cable type selected), "town" (listing the location, population and area of each town), and "cable"

(giving the unit_cost for each type of cable), whose instances are viewed as facts of the following form :

```
plan (cen_town, cable_code).
town (town_name, x, y, population, area).
cable (cable_code, unit_cost).
```

The rules are stated in terms of Horn clauses, such as :

```
network (CEN_TOWN, TOWN, DIST, COST) :-
  plan (CEN_TOWN, CABLE_CODE),
  cable (CABLE_CODE, UNIT_COST),
  town (CEN_TOWN, X, Y, _, _), !,
  town (TOWN, X1, Y1, _, _),
  distance (DIST, X, Y, X1, Y1),
  COST is DIST * UNIT_COST,
  write (network (CEN_TOWN, TOWN, DIST, COST)), fail.
```

Then one may make deductive queries in a style not possible in traditional RDB systems, such as :

```
?- network (three_tree, TOWN, _, COST).
(what is the cost for installing communication lines, from town three_tree to every town in the local area).
```

From this example we can see that logic clauses can be employed as a very elegant formalism for representing deductive queries, where the details of which relation are actually stored is hidden from users by the "logic navigation", and rules can be thought of as a generalization of the derived view mechanism which provide for their efficient implementation via the usual mechanisms for query support.

However, in developing intelligent GIS's, the following two major problems have not been well solved :

- The lack of a rule based and well supported spatial object modelling mechanism that flexibly guides the use of a large data/knowledge base.
- The difficulty to express control structures for complex GIS applications, which describe the usage of domain data/knowledge and the linkage of action modules for constructing an operational scheme with desired run-time control.

The focus of this work is to explore possible solutions.

RULE-BASED OBJECT MODELLING

The rule-based object/task modelling approach described in this paper is characterized by the integration of object-oriented programming, LP and RDB.

In the object-oriented semantic modelling paradigm, reality is represented in terms of objects as well as their relationships [Cox 84]. Each object has an associated set of procedures called methods denoting its dynamic behaviors. The manipulation of objects is made by applying the methods on them, referred to as "sending the objects a message".

Unlike the usual operator/operand model, which treats operators and operands as if they were independent, in the above message/object approach, an object instance records its type (class) explicitly, which is used to determine the set of legal operations on the objects of this class, and the type dependencies become permanently encapsulated within classes. This concept is coincident with a "save" database design principle : localization [Rid 83].

Generally we identify an object structurally as :

- (1) a primitive one represented by an LP predicate without any nested predicate as its argument.
- (2) a compound one composed in a nested or recursive fashion from two or more objects, represented by an LP predicate with at least one predicate as its argument.

Then each method is stated as an LP clause. Passing a message is specified by using the infix operator ":", (the number of arguments may be zero). We classify two kinds of methods:

- (1) i-method (the individual-oriented method), which is defined on a single object instance (universally quantified).
 i-method : object (arg1, arg2, ...)
- (2) c-method (the class-oriented method), which is defined on a class of object instances (the simplest example is a relation with multiple tuples). Its application is represented as
 c-method : object ()

For example the object type "town" is specified as :

```
object_type town (TOWN_NAME, X, Y, POPULATION, AREA)
{
  structure { TOWN_NAME : c4,
              X         : r8,
              Y         : r8,
              POPULATION : i6,
              AREA      : i6,
              key : TOWN_NAME
            }
  constraints { POPULATION > 5000, POPULATION < 100000,
              AREA > 5
            }
}
```

```

i-methods { what_kind_town ([TOWN_NAME, POPULATION]) :-
            town (TOWN_NAME, _, _, POPULATION, _).
            density (DENS) :-
            town (TOWN_NAME, _, _, POPULATION, AREA),
            DENS is POPULATION/AREA.
        }
c-methods { candidate (CEN_TOWN) :-
            town (CEN_TOWN, _, _, POPULATION, AREA),
            DENS is POPULATION/AREA,
            POPULATION >= 20000,
            DENS <= 15000,
            write (candidate (CEN_TOWN)), fail.
        }
}

```

In our approach a constraint is specified similar to the body of a Horn clause. The specified constraint in this example means that

$$(\forall \text{TOWN_NAME})(\forall X)(\forall Y)(\forall \text{POPULATION})(\forall \text{AREA})$$

$$(\text{town}(\text{TOWN_NAME}, X, Y, \text{POPULATION}, \text{AREA}) \rightarrow$$

$$\text{POPULATION} > 5000 \ \& \ \text{POPULATION} < 100000 \ \& \ \text{AREA} > 5)$$

An instance of an object is a predicate with all the arguments been instantiated with specific values. Thus the goal

```
?- density (X) : town (thre_tree, 65.4, 34.7, 51000, 5).
succeeds with X = 10200.
```

And the query

```
?- candidate (X) : town ().
returns names of all the candidate towns.
```

In the object-oriented semantic modelling paradigm, an important feature is the inheritance network whereby an object can be declared as a specialization of other objects, therefore inheriting their behavioral properties. The inheritance network is declared by means of the infix operator "isa", such as

```
object_A isa object_AA & object_BB & ...
```

The interpretation of a message consists in the unification of the objects, the unification of the method and the proof of the method. If an object is declared by "isa" as an sub-object, then the unification of the method with the methods associated with the ancestors of the object in the "isa" network, is necessary. When an object has more than one ancestors, we first determine the order of the ancestors, then use a so called "upward-first" search strategy, starting from checking the first available ancestor, observe to see whether it is a root (no further ancestor remained), if not, move upward until either the unification succeeds or backtrack to try the next possible ancestor at the highest possible ancestry level.

VIRTUAL OBJECTS

Passing message implies that certain action may be taken ruled by the specified method clause, then the resulting predicate (on the left side of the clause) can be reasonably viewed as a virtual object (VO). The instance of a VO is derived from the instances of other(source) objects, and instantiated by passing message thus represents the results of the message passing. Thus VO's may be viewed as a link between data access and data processing. Different from actual objects, VO's may or may not be actually filed and physically stored until needed.

For instance, in the above example, the predicate network can be specified as a VO instantiated by applying the method (identified by the same name) on an association of objects :

```
network (CEN_TOWN, TOWN, DIST, COST) : plan(), town(), cable().
```

The declaration of a VO contains structure (which can be the same as for the usual actual object), source objects, mapping rules and so on. The source objects can be actual or VO or a mixture of both, thus a VO may be defined hierarchically. For example the VO "network" is specified as :

```
Virtual_object_type
network (CEN_TOWN, TOWN, DIST, COST)
{
  structure { CEN_TOWN : c8,
              TOWN      : c8,
              DIST      : r8,
              COST      : i8,
              key : CEN_TOWN, TOWN
            }
  source { plan, town, cable
        }
  mapping rules
  { network (CEN_TOWN, TOWN, DIST, COST) :-
    plan (CEN_TOWN, CABLE_CODE),
    cable (CABLE_CODE, UNIT_COST),
    town (CEN_TOWN, X, Y, _, _), !,
    town (TOWN, X1, Y1, _, _),
    distance||(DIST, X, Y, X1, Y1),
    COST is DIST * UNIT_COST,
    write (network (CEN_TOWN, TOWN, DIST, COST)), fail.
  }
  c-method
  { network_cost (CEN_TOWN, TOTAL_COST) :-
    assert (network_cost (CEN_TOWN, 0)), !,
    network (CEN_TOWN, _, _, COST),
    retract (network_cost (CEN_TOWN, SUBTOTAL)),
    TOTAL_COST is SUBTOTAL + COST,
    assert (network_cost (CEN_TOWN, TOTAL_COST)), fail.
  }
}
```

Unification is the way to make VO's physically accessible. In the case that a VO has virtual source objects, the latter must be instantiated in advance according to their own definitions, and such a process may spread upward in the virtual object network until all the actual and VO's involved are fully instantiated. Syntactically, the instantiation of a VO, such as "network", is represented by the clause

mapping (network).

The instance of a VO has a life time which extends over a single task. Its refreshment may adopt one of the following strategies : (a) when it is accessed after the alteration of its source objects; (b) by demand. The detailed VO management is similar to that we described in [Mel 83b].

network (CEN_TOWN, TOWN, DIST, COST) : plan(), town(), cable().

This approach offers the following advantages :

- (1) In the rule-based environment, a VO mapping is just one or a cluster of rules. However, a VO is a more feasible entity than a rule. A VO has a structure definition, can be directly queried at a high level, with a printing format; It can, flexibly, either be used as a source object for other VO's, or be involved in tasks.
- (2) VO provides a natural way to accomplish rule localization.
- (3) Duplicate unification for VO's in a task can be reduced.

RULE-BASED TASK SPECIFICATION

A task is an operational scheme for complex decision-making, problem-solving, simulation or control in a knowledge based environment. The task modelling concerns not only the domain knowledge, but also the control structure whose description may be viewed as certain meta-rules, determining the usage of domain knowledge and the linkage of actions which make up an operational scheme with desired run-time control [Mel 83a].

Our solution for handling tasks is characterized by using the combination of forward chaining (data-driven) and backtracking (goal-driven). The task scheme specification is basically a forward chaining network with a number of paths with three basic kinds of components : action, linker and net. Below we briefly describe this approach. For more detail, refer to author's another paper [Chen 85b].

In the object-oriented paradigm, actions are denoted by the messages (methods on objects) to be passed. An action specification gives the correspondence between the statically described message and the action as its execution. A message

passing at different stage of a task must be identified separately, except in the case of a loop. There is a one-to-many mapping from the space of methods to the space of actions.

Linkers describe the conditions for each possible path in the net denoting the task scheme, that is, the connection rules. Each linker has four arguments (attributes), as linker (LINKER_NAME, LINK_MODE, TERMINATE_MODE, CONDITION). The LINK_MODE may be 'r' (regular) or 'c' (conditional branch). The TERMINATE_MODE includes 'h' (hard termination if errors are detected, with system state recovery) and 's' (soft termination without state recovery).

A net definition specifies the internal linkage among the actions and the control flow (not data flow, as a rule, all inter-action communication must be carried out by accessing data/knowledge base, which may be viewed as "mailboxes").

The proposed net specification system, with its interpretation, is characterized by introducing the concepts of net-structures and net forming operations called path forms for constructing new nets from existing ones, which map net-structures to net-structures in general.

A path form is an expression denoting a combination of net-structures which depend on the actions, linkers and sub-nets that form the parameters of the expression. Instead of describing nets structurally, path forms represent nets functionally. Examples of path forms are

A > B (Composition),
A > (P)B (conditional composition through a linker P),
[A; B] (serialization),
(P){A; B; C} (conditional branch),
A >> {B; C; D} (compose-to-all, means [A>B; A>C; A>D]).

These path forms can be utilized hierarchically or recursively for representing a complex net as the combination of certain simpler (or lower level) nets. And in fact more additional path forms may be defined and implemented.

A naming mechanism, the first-meet rule, is introduced, describing how to handle a symbol for an action, a linker or a sub-net, which appears more than once in a net-expression, by going back to the leftmost position in the expression where the symbol appears.

In this approach, the "task" is treated as a special type of system objects with a set of operations defined on it, such as query, update, execute, ... etc. Thus a knowledge based "Task Libraries" could possibly be built easily.

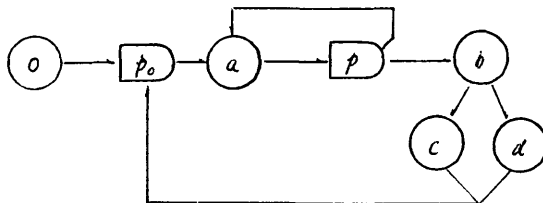
To illustrate the proposed approach, we will give a complete rule-based task specification for the telecommunication system planning example mentioned before, as following:

```

task analysis
{
  object_specification
  { actual_objects { town, cable
                    }
    virtual_objects { plan, network
                    }
    constraints      { plan with_cons
                      { CABLE_CODE [ "h201, h203, h205" ]
                      }
                    }
  }
  scheme_specification
  { action
    { o is candidate (CEN_TOWN) : town ().
      a is mapping (plan).
      b is mapping (network).
      c is listing (network).
      d is network_cost (CEN_TOWN, TOTAL_COST):network().
    }
    net
    { analysis is o > (p0)a > (p){b; a} > [c; d] > (p0)
    }
    linker
    { p is (c, (%{exist (candidate)}), cons_with plan
           {CABLE_CODE [ "h201, h203"}, %{})}).
      p0 is (r, s, (%{exist (candidate)})).
    }
  }
}
}

```

We assume that all the anticipating objects are previously declared. Within this task, the VO's only need to be instantiated once (by "mapping"). The net representation for the task is illustrated in Figure 1. Its performance can be described as follows: action 'o' selects all the possible candidates as the central town of the system; If there exist candidates, through action 'a', a plan is to be made using (then deleting) the first candidate town according to certain criteria; Then, if the plan satisfies certain task oriented constraints, 'b', 'c', 'd' carry out the processing in terms of computing virtual object "network" and applying c-method "network_cost" to it; After this run, the next candidate town is to be analyzed. The control rules are specified at linkers p0 and p.



INTERFACING TO GIS

Conceptually the simplest solution to the problem of using existing RDB is to extract a snapshot of the required data when the system begins to work on a set of related problems, then keep the snapshot in the internal RDB. However, such an approach faces at least the following major difficulties :

- Many GIS-oriented databases are not typical RDB's.
- This solution may not be general enough since the decisions for extracting must be made statically, which is not adaptable if the portion of the RDB to be extracted is not known in advance.

A more suitable solution is to redirect all queries, on predicates representing relations, to the stored data. The key issue is when and how to consult the database. In general this approach includes the following steps :

- (a) Generating queries form goals with optimization.
- (b) Submitting the queries to the GIS database.
- (c) Translating the queries into proper DBMS calls.
- (d) DBMS query evaluating.
- (e) Transforming the received data into the internal knowledge representation.
- (f) Storing the facts in the internal memory.

It is worth noting that :

(1) There may be mappings at two levels : the mapping from a non-relational GIS database to an RDB, and the mapping from relations to predicates.

(2) To reduce the complexity of database queries, it is reasonable to collect and jointly execute database calls, rather than executing them separately whenever required by the system (particularly for huge GIS databases).

(3) An LP system normally operates in a tuple-at-a-time fashion, which will generate a particularly inefficient version of a nested iteration query evaluation without getting the benefits of query optimization procedures of the DBMS to reduce the number of secondary storage access. The introduction of c-methods provides a high level representation, more dedicate interface must be developed at the system level.

CONCLUSIONS

Logic Programming and Functional Programming have set a bridge between AI and other fields [Chen 85a]. For developing more intelligent GIS's, in this work we have combined object-oriented paradigm, logic programming, semantic modelling and event modelling, providing therefore complementary benefits in inference, deductive query support, integrity control, and explicit control knowledge representation, towards a generalized management of data, action and operational schemes.

REFERENCES

- [Chen 86] Q. Chen, "A Rule-based Object/task Modelling Approach", Proc. of ACM-SIGMOD 86 International Conference, Washington D. C. 1986, USA.
- [Chen 85a] Q. Chen, "Extending the Implementation Scheme of Functional Programming System FP for Supporting the Formal Software Development Methodology", Proc. 8th International Conference on Software Engineering, London, 1985.
- [Chen 85b] Q. Chen, "Toward A Generalized Data/Action Management : An Approach for Specifying and Implementing Operational Schemes", Proc. 1st Pan Pacific Computer Conference, Melbourne, Australia, Sep. 10-13, 1985.
- [Cox 84] B. J. Cox, "Message/Object, An Evolutionary Change", IEEE Trans. On SE, pp. 50-61, Jan. 1984.
- [Gall 84] H. Gallaire, J. Minker and J. Nicolas, "Logic and Databases : A Deductive Approach", Computing Surveys, Vol. 16, No. 2, 1984.
- [Melk 83a] M. Melkanoff and Q. Chen, "An Experimental Database Which Combines Static and Dynamic Capabilities", Proc. of Engineering Design Applications, ACM-SIGMOD'83/Database Week, pp. 53-61, 1983.
- [Melk 83b] M. Melkanoff and Q. Chen, "Integrating Action Capabilities into Information Databases", Proc. 2nd International Conference on Databases (ICOD-2), Cambridge, UK, 1983.
- [Parker 84] D. Parker et al., "Logic Programming and Databases", Proc. Int. Workshop on Expert Database Systems, 1984.
- [Rid 83] D. Ridjanovic and J. Brodie, "Action and Transaction Skeletons : High level Language Constructs for database Transactions", Proc. ACM-SIGPLAN 83, 1983.
- [Ullman 85] J. Ullman, "Implementation of Logical Query Language for Databases", Proc. of ACM-SIGMOD 85, 1985.
- [Zani 84] C. Zaniolo, "Object-Oriented Programming in Prolog", Proc. Int. Logic Programming Symposium, IEEE 1984.