Geographic Data Handling in a Deductive Object-Oriented Database

Alia I. Abdelmoty, Norman W. Paton, M. Howard Williams,
Alvaro A.A. Fernandes, Maria L. Barja, Andrew Dinn

Department of Computing and Electrical Engineering,
Heriot-Watt University
Riccarton, Edinburgh EH14 4AS, Scotland, UK
e-mail: <alia,norm,howard,alvaro,marisa,andrew>@cee.hw.ac.uk
phone: +44-31-449-5111 ; fax: +44-31-451-3431

Abstract.
This paper describes how a deductive object-oriented database (DOOD) can be used to support the storage and management of data which is typical of that found in geographic information systems (GIS). This is done with two aims in mind: to illustrate how a combination of deductive and object-oriented facilities can be applied effectively in an advanced application, thereby motivating the development of DOOD systems; and to show how geographic database systems stand to gain from the utilisation of advanced data modelling and inference facilities as supported by a DOOD. The paper describes the DOOD system which has been used for prototyping a range of geographic concepts, presents a framework for the structural organisation of GIS data using an object-oriented data model, and shows how a logic query language can be used within this structural framework to perform a range of analyses. [a]

[a] In: Proc. 5th Int. Conf. on Databases and Expert Systems Applications (DEXA), D. Karagiannis (ed), 445-454, Springer-Verlag, 1994.

1 Introduction

Geographic information systems (GISs) present a major challenge to the developers of advanced databases. The complexity of the data to be modelled – which may be richly structured, spatially distributed, time-variant and subject to complex operations – strains the relational model, which has been shown to be inadequate for many geographic applications [WHM90, Ege92]. Systems have been proposed for managing geographic data which add explicit support for spatial data types into the relational model [Gut88, AS91], but such systems retain the widely recognised weaknesses of relational systems for handling the aspatial aspects of geographic applications, thereby diluting the gains achieved as a result of incorporating spatial data types and operations.

This paper presents an approach to the combined use of deductive and object-oriented facilities for managing geographic data. It is shown how object-oriented modelling constructs can be used to capture the structural semantics of a range of geographic concepts, and how deductive rules can be used to infer the existence of complex relationships within a geographic database.

The paper is structured as follows. Section 2 outlines the principal facilities of the DOOD system which is being used with geographic applications. Section 3 indicates how the modelling facilities of the DOOD can be used to describe a range of geographic concepts. Section 4 shows how deduction can be used in this context to infer the existence of additional relationships between geographic concepts. Conclusions are presented in section 5.

2 ROCK & ROLL - A Deductive Object Oriented Database

This section presents an informal overview of the ROCK (Rule Object Computation Kernel) & ROLL (Rule Object Logic Language) DOOD system. ROCK & ROLL has three principal components – an object-oriented data model (OM), a logic query language (ROLL), and an imperative manipulation language (ROCK). In ROCK & ROLL, the object-oriented data model has been used as the basis for the design of the other components. The data model describes the structural characteristics of the data which can be processed by both the logic and imperative languages, and thus the two programming languages can be integrated in a way which avoids the introduction of the impedance mismatch [BPF+94]. Support for the two languages enables the user to select the programming paradigm which is most suitable for particular tasks within a complex application, and thus allows the benefits of both a deductive database and an imperative programming language in a single system.

2.1 Data Model

This section gives an informal overview of the constructs used to model an application domain both structurally and behaviourally.

Atomic values and compound data items are called *primary objects* and *secondary objects* respectively. Each object is assigned an *object type*, and must conform to the structure associated with that type. Every secondary object has a unique object identifier.

A type definition can describe references of two kinds. The first kind of (optional) reference definition is used to model the *properties* of the type. This results, for each type, in a possibly empty set of type names which are the attributes of the type. For example, the definition in figure 1 indicates that a `road` has a single property of type `roadName`.

```
type road
    properties:
        roadName;
    { roadSegment };
    ...
end-type
```

Figure 1: Definition of type road.

The second kind of (optional) reference definition, which is referred to as the *construction* of the type, is used to distinguish the fundamental structural characteristic of a type from its other stored properties. A type can be structured by *association, sequentiation* or *aggregation*, which support the modelling of sets, lists and tuples, respectively. For example, the definition in figure 1 indicates that a `road`, is constructed as an association of `roadSegment` objects (represented by curly brackets).

Schema diagrams for the data model can be constructed using the following notation. Secondary object types are represented using rectangles, primary object types using ellipses, and operations as round-cornered rectangles. Labelled directed edges represent modelling features thus: ○ – attribution, ⍋ – specialisation, ⊗ – aggregation, ⊛ – association, and ⊚ – sequentiation. Unlabelled directed edges represent the aliasing of type names. This notation is used to describe part of a geographic database in figure 2. Shaded types in this figure are discussed later.

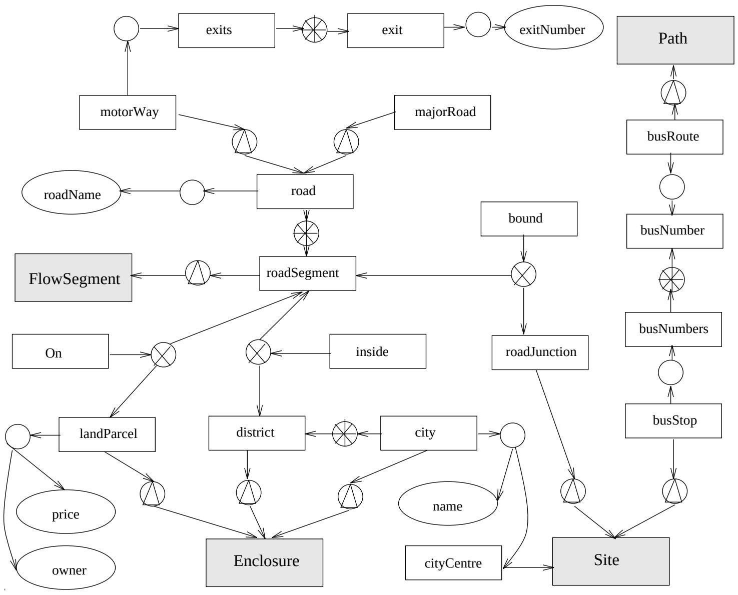

Figure 2: Application schema showing the structure of some of the involved objects.

The behavioural properties of objects are defined using methods. The signatures of public methods are included as part of the definition of a type. Every type is associated with a class of the same name as the type. The class specifies the private properties of the type and gives the code which implements the methods. The intention is that the type specifies all that a user of the type needs to know in order to use the type, while method code and other implementation details are specified in the class. Methods can be implemented using either the imperative language ROCK or the logic language ROLL.

2.2 Logic Language

ROLL is a Horn clause language. Familiarity with the latter subclass of first-order languages is assumed at the level of [CGT90].

The *ROLL alphabet* consists of logic variables, constant symbols and predicate symbols, but not function symbols. The set of *constant symbols* is the set of names of values (e.g. `5`, `"Edinburgh"`, `true`). There is no need for object identifiers to appear as constant symbols in ROLL expressions, as specific objects are either passed into a ROLL expression from ROCK, or are retrieved from the extensional database.

The set of *predicate symbols* is the set of operation names declared by operation interfaces. It follows that ROLL queries abide by strict encapsulation.

A ROLL *term* τ is either a ROLL *constant* or a *(logical) variable*. A ROLL *atom* has the form $\beta(\tau_1, \ldots, \tau_{n-1})@\alpha == \tau_n$, which is read as "send the *message* β with the arguments $\tau_1, \ldots, \tau_{n-1}$ and the result τ_n to the object α". An operation interface $\beta(T_1, \ldots, T_{n-1})@\alpha == T_n$, is assumed to be defined, such that each τ_i denotes an instance of T'_i, where $T'_i \leq T_i$. If the operation β has no distinguished result (as is the case with ROLL methods), then the term has the form $\beta(\tau_1, \ldots, \tau_n)@\alpha$. If $n = 0$ then $\beta@\alpha =_{def} \beta()@\alpha$.

A *ROLL clause* is a Horn clause, and the usual convention is followed of rewriting a clause as a reverse implication, i.e. *head* '`:-`' *body*.

The following query over the schema presented graphically in figure 2 retrieves as bindings for `LP` the `landParcel` objects which are associated with the `owner` `"Acme Products"`, and which have a `price` greater than 1000.

```
get_owner@LP == "Acme Products", get_price@LP == P, P > 1000.
```

This example uses a number of facilities which bear explanation: methods with the prefix `get_` are generated automatically by the system to allow access to the structural characteristics of an object; the results of the messages `get_owner` and `get_price`, each of which are sent to `landParcel` objects bound to the logic variable `LP`, are unified with the constant `"Acme Products"` and the variable `P` respectively; ROLL queries and methods are strongly typed, and a type inference system is used to infer types for logic variables – for example, that the logic variable `LP` is associated with the type `landParcel` is inferred from the fact that the methods `get_owner` and `get_price` are applicable to objects of type `landParcel`; in this example there is no explicit iteration over the instances of `landParcel` – the ROLL query evaluator optimises a query, and plants iterators within the evaluation graph wherever there is no other way of obtaining a binding for a logic variable associated with an object type.

The following method definition introduces a method called `roads` attached to the class `district`, which retrieves as bindings for `Road` the `road` objects which pass through the `district` which is the recipient of the message, namely `District`.

```
class district
    ...
    roads(Road)@District :-
        get_roadSegment@Inside == Segment,
        get_district@Inside == District,
        get_member@Road == Segment.
end-class
```

In this example, it can be seen that rules are associated with object classes, thereby allowing the rule base to be organised in an object-oriented manner. That the variable `District` is of type `district` is established from the fact that it fulfills the role of message recipient in the method definition which is specified within the class `district`. The system generated method `get_member` is used to retrieve the objects which are stored within an object constructed by association, in this case the association `Road`.

3 Object-Oriented Modelling of the Geographic Domain

In defining a geographic entity, two kinds of property can be distinguished: descriptive properties and spatial properties. The former refers to textual and numeric information (e.g. the name and rate of flow of a river, the density of population of a region), while the latter specify spatial characteristics of that entity, i.e. its shape and location. The current generation of GISs [Mor86] is characterised by the presence of:

- Distinct underlying representations: a dichotomy exists between the systems used for the representation and manipulation of the spatial and aspatial aspects of an application (e.g. a relational DBMS for managing aspatial data and specialised systems for managing the spatial data) with an integration layer to link them.

- Data source dependence: two categories of geographic data source can be recognised, which yield vector data and raster data. The spatial representations of geographic objects in these two forms are different, both in terms of structure and in the geometric operations which can be efficiently supported. Current GISs are restricted to only one of these categories due to the strong link between the representation of the application level concepts and the geometric data structures.

With a view to overcoming the above problems a three level conceptual model for the geographic domain is proposed, in which a level of geographic abstract data types acts as a buffer between the geometric level of representation, where only the graphical aspects of the data are modelled, and the level where application concepts are modelled. Figure 3 illustrates the mapping between the three levels, which is discussed in more detail later.

In what follows a brief overview is given of the three levels and their representation using the ROCK & ROLL data model.

3.1 Application Level Concepts

Figure 2 shows a sample schema of part of a geographic application. The shaded object types in the figure are sample GADTs which are discussed in the next subsection. Application objects are modelled as specializations of GADTs, from which structure and behaviour are inherited.

Spatial hierarchies of geographic concepts are represented using the modelling constructs aggregation, association and sequentiation. For example, a

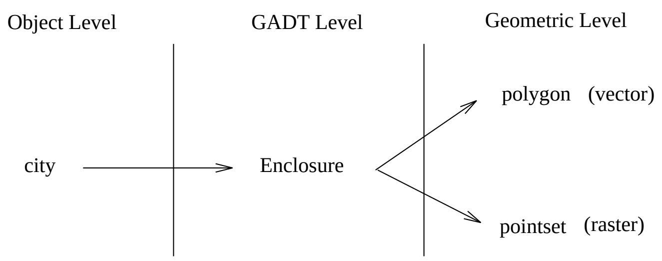

Figure 3: Example types from the three levels of representation showing the indirect mapping between the object and the geometric levels.

`city` is a set of `districts`. Objects can have complex object types as attributes, for example, the type `motorWay` has an attribute `exits` which is a set of `exit` objects.

The example schema shows both the flexibility and directness of representation of the geographic domain concepts using the semantic constructs provided by the data model. Furthermore, lower level details are hidden from the application level using the GADTs introduced in the next subsection.

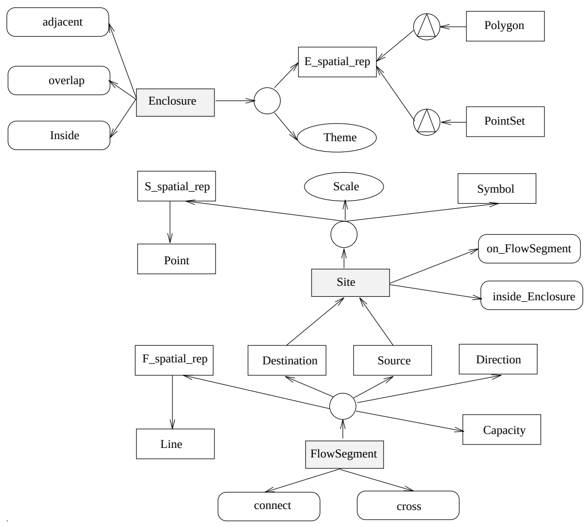

Figure 4: Examples of structure of some GADTs.

3.2 Geographic Abstract Data Types

Figure 4 presents the structure of some of the GADTs used in figure 2. For example, an `Enclosure` is a representation of a real world object enclosing a property or an activity which has an areal spatial representation. Examples include *house*, *district*, *city*, etc. The spatial representation of the enclosure is hidden as one of its attributes, and is chosen in this case to generalize both the polygon (vector) and the pointset (raster) representations, thus allowing objects from different data sources to be homogeneously modelled. Spatial operations such as *overlap*, *inside* and *adjacent* are defined on the class `Enclosure`, and implemented on both its spatial representations.

More complex objects can be built using the above GADTs, such as the `Path` in figure 2, which represents a list of `FlowSegments`, where a `busRoute` is a specialisation of `Path` representing an ordered set of `roadSegments`.

3.3 The Geometric Level of Representation

Types from the geometric level of representation represent concepts such as points, lines and polygons. Spatial operations on the application level objects map to operations on the geometric level through the GADTs. For example, the *overlay* operation between two thematic maps, e.g. soil types and rain-fall maps, is interpreted as an overlap operation between their component regions, represented as `Enclosures`. Such an operation is implemented as an intersection operation over the spatial representation of the `Enclosure` objects, where the method of implementation may be different for the two spatial representations. Spatial indexes can be used at this level to enhance spatial search.

4 Deductive Inference in a Geographic Database

Section 3 has shown that the rich semantic modelling constructs offered by the object data model have allowed the realisation of a three level conceptual model for the geographic domain. This model also provides an effective framework within which operations representing spatial relationships can be developed.

By virtue of possessing a location in space, geographic objects exhibit spatial relationships with all other objects in this space. Ad-hoc spatial relationships between objects in a GDB, which are a consequence of their relative positions, are very common in GIS queries. For example, the following are typical: find the dwellings which are adjacent to a post office; find the dwellings which are down-stream of a given chemical factory on a river and south-of a given city; find the dwellings which are within 1/2 hour driving distance from a given shopping centre.

It is generally accepted that it is not feasible to store all the relationships required to answer such queries explicitly. Considerations such as storage overheads, frequency of usage and cost of derivation are used in assessing which spatial relationships to store and which to derive. In most GISs, such as ARC/INFO [Mor86], some topological relationships are stored at the geometric level, and computational geometry is used for the dynamic evaluation of others. More recently, qualitative spatial reasoning over some of the above relationships has been proposed [Ege91, AWP93] as a complementary mechanism to computational geometry for the derivation of spatial relationships. This section illustrates how deduction can be used to support the inference of spatial relationships.

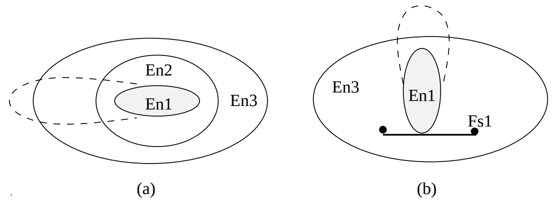

Figure 5: Defining the overlap operation on two Enclosures using a third Enclosure in (a) and a FlowSegment in (b).

4.1 Inference of Spatial Relationships

Spatial relationship composition, i.e. the deduction of a relationship between two objects given their relationship with a common intermediate object, is used below to illustrate spatial reasoning in a GDB.

For example, figure 5 shows different situations where an enclosure `En1` can be said to overlap the enclosure `En3`. In 5(a) `En1` is held to overlap `En3` because it overlaps `En2` and `En2` is inside `En3`. In figure 5(b) `En1` is held to overlap `En3` because `En1` is on the `flowSegment Fs1` and `Fs1` is inside `En3`. These observations can be incorporated into a definition of `overlap` as shown below:

```
class enclosure
    ...
    overlap(En3)@En1 :-
       inside(En1)@En2, inside(En2)@En3:enclosure.
    overlap(En3)@En1 :-
       on_flowSegment(En1)@Fs1, inside(Fs1)@En3:enclosure.
end-class
```

In the above example, the suffix `:<typename>` associated with some of the variables is used to give hints to the type inference system, which is otherwise unable to infer an unambiguous type for each variable in this case.

An example of the use of recursive rules in method definitions is illustrated in the `path` relationship between two `flowSegment`s as follows:

```
class flowSegment
    ...
    path(Fs2)@Fs1 :-
       get_Destination@Fs1 == Connection, get_Source@Fs2 == Connection.
    path(Fs2)@Fs1 :-
       path(Fs3)@Fs1, path(Fs2)@Fs3.
end-class
```

4.2 Using Rules on Application Level Concepts

Methods over application level classes in figure 2 can also be defined using rules. For example, `motorWaysInCity` is an operation defined on the object class `city` to determine which instances of the class `motorWay` pass through a particular `city` object.

```
class city
    ...
    motorWaysInCity(M)@City :-
       get_member@M:motorWay == RS, get_roadSegment@In:inside == RS,
       get_district@In == District, get_member@City  == District.
end-class
```

In the above example, the class `inside` is used to represent a relationship between `roadSegments` and `districts`. As ROLL queries and methods are optimised, the order in which the subgoals within a query or rule are entered does not determine how it is evaluated. In practice, ROLL queries are evaluated bottom-up after being optimised using a technique based upon Static Filtering [CGT90].

A query which calls this method to retrieve the `motorWays` passing through the city of `Athens` can be written as follows:

```
var mset := [{M} | get_name@City == "Athens", motorWaysInCity(M)@City]
```

This example shows how the result of a ROLL query can be assigned to a variable in the imperative programming language ROCK [BPF+94]. The embedded query is enclosed in square brackets, the result being the set of objects `M` which satisfy the goal expressed to the right of the `|`. It is possible to process the set assigned to `mset` using the normal facilities of ROCK. For example, the following program would print out the `roadName` of every `motorWay` in `mset`:

```
foreach o in mset do
    write get_roadName@o, nl;
```

Note that as ROLL is a logic language, it is possible to run the method `motorWaysInCity` with different variable bindings, i.e. to find the `motorWay` objects associated with a given `city`, to find the `city` objects through which a given `motorWay` passes, to find out if a given `motorWay` passes through a given `city`, or to find out which `motorWays` pass though which cities.

5 Conclusions

This paper has presented a new approach to the modelling and management of geographic data in a GIS, using a deductive object-oriented database system. The approach has two main features:

- A three level conceptual model for structuring geographic data, with the underlying geometric layer accessed through a collection of geographic abstract data types, which are in turn referenced by application level concepts. The model facilitates the effective modelling of complex geographic phenomena with possibly different underlying spatial representations.

- Spatial reasoning rules for the derivation of implicit spatial relationships which are too numerous to store in the database. Rules for the derivation of such relationships are complementary to computational geometry algorithms which are usually associated with large computation costs.

This approach to geographic data modelling is being prototyped using the deductive object-oriented database system ROCK & ROLL, where the following features have been exploited: an expressive object-oriented data model which can directly capture the structural characteristics of a range of geographic concepts at different levels of abstraction; a fully integrated logic language, which can be used to infer the existence of spatial or aspatial relationships between geographic concepts, which are typically too numerous to store; and a fully integrated imperative database programming language, which can be used for manipulating the database, but which also complements the logic language for retrieval tasks in this domain, where the execution of complex geometric algorithms is central to the efficiency of the overall system.

Acknowledgements The work that resulted in this paper has been funded by the Science and Engineering Research Council through the IEATP programme. We are also grateful to Dr. Keith G. Jeffery of RAL for useful discussions on the subject of this paper, and Dr. J.M.P. Quinn representing ICL and Mr Neil Smith of Ordnance Survey as the industrial partners in the project.

References

[AS91] W.G. Aref and H. Samet. Extending a DBMS with Spatial Operations. In O. Gunther and H.J. Scheck, editors, *Advances in Spatial Databases, SSD'91*, LNCS 525, pages 299–318, ., 1991. Springer-Verlag.

[AWP93] Alia I. Abdelmoty, M.Howard Williams, and Norman W. Paton. Deduction and Deductive Databases for Geographic Data Handling. In *Design and Implementation of Large Spatial Databases, Third International Symposium, SSD '93*, LNCS 692, pages 443–464. Springer Verlag, June 1993.

[BPF+94] M.L. Barja, N.W. Paton, A.A.A. Fernandes, M.H. Williams, and A. Dinn. An Effective Deductive Object-Oriented Database Through Language Integration. In J. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. on Very Large Data Bases (VLDB)*, pages 463–474. Morgan-Kaufmann, 1994.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases*. Springer-Verlag, Berlin, 1990.

[Ege91] M.J. Egenhofer. Reasoning About Binary Topological Relations. In O. Gunther and H.J. Scheck, editors, *Advances in Spatial Databases, 2nd Symposium, SSD'91*, Lecture Notes in Computer Science, 525, pages 143–161, Zurich, Switzerland., 1991. Springer-Verlag.

[Ege92] M.J. Egenhofer. Why not SQL! *Int. J. Geographic Information Systems*, 6(2):71–85, 1992.

[Gut88] R.H. Guting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In J.W. Schmidt and M. Missikoff, editors, *Advances in Database Technology- EDBT'88*, Lecture Notes in Computer Science, pages 506–527, Venice, Italy, 1988. Spriger Verlag.

[Mor86] S. Morehouse. ARC/INFO: A Geo-Relational Model for Spatial Information. In *Proceedings of 7th Int. Symposium on Computer Assisted Cartography*, pages 388–398, Washington, DC, 1986.

[WHM90] M.F. Worboys, H.M. Hearnshaw, and D.J. Maguire. Object-Oriented Data Modelling for Spatial Databases. *Int. J. Geographic Information Systems*, 4(4):369–383, 1990.