Rule-Based Modelling In GIS

Dr. David Pullar
Geographical Sciences and Planning
The University of Queensland. Qld. 4072
AUSTRALIA
+61 7 33656522
D.Pullar@mailbox.uq.edu.au

Presented at the second annual conference of GeoComputation '97 & SIRC '97, University of Otago, New Zealand, 26-29 August 1997

Abstract

Geographic Information Systems (GIS) provide functionality for visualising, managing and manipulating spatially referenced data. For straight forward spatial queries users perform these functions through a graphical user interface, for more advanced spatial models the user is often required to write a program as a formulation of the problem. The objective of this paper is to find out what type of programming language is suitable for users who do not have extensive programming experience, and yet provides powerful modelling capability. The paper reviews different types of programming paradigms and the abstractions they support. It argues that a combination of decision rules and object-oriented paradigms offer a clear and effective language interface to GIS. The rules are structured and presented in a tabular form to simplify their specification. The approach emphasises a style of programming that is attuned to spatial data models and programs are easy to comprehend.

1. Introduction

A necessary part of solving problems with computers is to express them in a formal way. The appropriate computer tool to solve geographical problems is a Geographic Information System (GIS). Such systems provide the basic functionality for visualising, managing and manipulating spatially referenced data. Problem solving is expressed using a computer language either provided by the system or one that interoperates with the system. Users of GIS are faced with the task of writing programs as a concrete formulation of their particular problem for not only advanced spatial analysis problems, but also for many *ah hoc* queries. Given that most users of GIS are professionals in a land related discipline, and not professional programmers, then it is important to make the language interface to a GIS as easy to use and intuitive as possible.

Novice users often find writing programs to be a daunting task. Assuming there is a systematic or scientific approach to the problem being solved, expressing this in a program is still a difficult intellectual activity. This paper has identified two fundamental reasons for this:

1 *Representation mismatch* between the object level representation of spatial data in the programming language and the application view. This problem occurs when the application presents information in one way but the programming environment to access and manipulate that information is different. A popular way to present information in GIS is as a map organised into thematic layers, whereas in a programming environment the user is presented with tables containing records. This also affects the way queries are expressed. For instance the application interface provides associative access by querying feature properties, whereas the programming language may provide data access by retrieving records based upon relative record numbers in a table.

2 *Problem specification mismatch* between the way a user expresses a problem and how programming languages implement the solution. This occurs when a users expresses a problem in set theoretic terms, yet is forced to resolve the problem in an application program as a sequence of operations on individual records. Many

programming languages compel a programmatic approach to solve a problem involving detailed actions in strict operational order.

This paper examines programming-language integration with GIS. We consider elements of programming including control structures, data structures, arithmetic, and so forth. In particular we explore the potential of decision tables to express query and modelling problems in a conceptually intuitive way. Decision tables express condition-action clauses in a tabular form. Arentze *et al.* (1995) show this to be a flexible technique for decision support in facility planning. We further explore their integration with operational semantics of GIS. We propose that decision tables be used in combination with object-oriented methods to provide a very direct and concise way to solve geographical problems. To demonstrate the concept, a prototype development is described where decision tables are integrated with the programming framework used by a commercial GIS.

The outline of this paper is as follows. Sections 2 and 3 characterise programming languages by the type of information abstractions they support. It is concluded that a combination of decision rules and object-oriented access to spatial features provides a uniform and convenient notation. Section 4 advocates the use of decision tables as a presentation style. Examples for a non-trivial spatial query demonstrate the concepts.

2. Background

Programming paradigms can be characterised according to: i) the data abstractions, and ii) the procedural abstractions they support.

1 *Data abstraction* refers to the way information content is represented. Low level languages use simple value types, like integers and reals, while high level languages support more abstract object types and data modelling relationships. Objects types impose a class definition to describe structural properties (state information) and behavioural properties (operations). Data modelling relationships define how objects may be related. This includes associative relationships for structural linkages between objects, composition relationships where objects form part of an aggregation hierarchy, and generalisation relationships where objects share common semantics in an inheritance hierarchy.

2 *Procedural abstraction* refers to the way actions are defined and controlled within the programming language. Low level languages solve problems in terms of *imperative* commands. The program code describes exactly how to solve the problem as a rigidly controlled set of detailed actions. Program control is expressed by either sequential instructions, repetition or branching conditional constructs. Examples of low level languages include FORTRAN and C. High level languages solve problems in a *declarative* fashion. The program code specifies the desired outcome or goal. Program control is less rigid and may be based upon reasoning to prove a hypothesis. Examples of high level languages include C++ and PROLOG.

So what are the best language characteristics to use in GIS? We reduce the scope of this question by focussing on a programming paradigm that is designed for a typical GIS user, namely one who does not have a significant amount of training in programming techniques. Any procedural abstractions would need to be implicit to harmonise with the way a user attempts to solve a problem, and should exercise a reasonably obvious method of control over spatial features. It would support a range of queries on spatial databases without the user needing to understand the intricacies of computer algorithms. The data abstractions used within the language need to be tightly interweaved with the way information is managed and manipulated at the user level. In modern systems this means the language must harmonise with geographic data modelling methods and with user interface paradigms used to manipulate geographic information.

3. Programming Paradigms

Programming languages employ different types of data abstractions and procedural abstractions. Different types of languages stress one characteristic over another. Four main paradigms are identified:

- Logic Programming
- Functional Programming
- Rule-Based Programming
- Object-Oriented Programming

3.1 Logic Programming

Logic programming applies rules of exact logic to solve problems, or to be more exact it applies rules of first order predicate logic. Problems are expressed as statements to represent things that we believe about the world. The statements are composed of a set of logical terms and logical connectors. The rules for evaluating statements are given by a truth table shown in Figure 1.

In first order predicate logic all objects belong to a single universe. This leads to a characteristic of "flatness" in pure logical languages. All objects are universal and so are the axioms by which they are related. There is no procedural abstraction in first order predicate logic.

In practice, logic programming languages use some procedural mechanisms to interpret logical statements. The most popular of these programming languages is PROLOG (Bratko, 1990). A logical statement is expressed as a Horn clause consisting of a conclusion head "*C*" and several conditional terms "*B*" in the body. They have the form:

"B_1 and B_2 and B_3 … and B_N implies C"

Different combinations of a head and body create three types of clauses: queries, rules and facts. The fundamental form of programming control is a query that is answered by searching for matching facts, or rules whose heads match the query and whose body may be proven. This ability to search through a set of facts and to further deduce relations from rules gives PROLOG its deductive capability.

Terms		*AND connector*	*OR Connector*	*Implication*
p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$
true	*true*	*true*	*true*	*true*
true	*false*	*false*	*true*	*false*
false	*true*	*false*	*true*	*false*
false	*false*	*false*	*false*	*true*

Figure 1: Truth Table.

The power of PROLOG-like languages to express both spatial queries and spatial models has been well demonstrated. LOBSTER is an early example of a prototype system that used PROLOG as the language interface to query a spatial DBMS (Egenhofer, 1990). The prototype provided a high level language to manipulate symbolic representations of spatial features. This was possible because the DBMS was able to handle complex record structures, and user defined functions could be programmed as builtins to the PROLOG interpreter. Spatial data types for points, lines, areas, and surfaces were defined in the DBMS and manipulated at a semantic level by the rules and facts expressed in Horn clauses. All low level access to spatial data and spatial manipulation is handled by the builtin functions. This ability to include declarative expressions of spatial queries within a logic language is viewed as a key requirement by other researchers (Abdelmoty *et al*., 1993).

3.2 Functional Programming

Functional programming is based upon mathematical concepts of mapping functions. A function maps object values from one domain to another. This is expressed formally $f:X \rightarrow Y$, the function f maps object values from the domain X to the domain Y. The object returned by a function depends only on its arguments. In addition functions do not induce any side effects so all state information evolves in an explicit and controlled way. This trait is known as *referential transparency*. Any transformations on objects are handled by explicitly returning new objects. This has a bearing on the data and procedural abstractions used by functional languages. Both rely upon mapping functions to express structural and behavioural relationships.

Advanced functional languages have a powerful expressive quality with the ability to use higher order functions (a function of a function of a .....) to perform symbolic manipulation and proofs in programs. Functions are also treated as first class objects so they may be used as arguments and may be the return value from a function. A mathematical style of programming is ob-

tained by using algebra expressions instead of function names. Examples of functional languages include LISP, this manipulates objects as a list of unitype symbols, and ML that is a language that manipulates objects with more advanced data types (Paulson, 1996).

A GIS database perceived and manipulated by a functional language is viewed as a collection of objects together with a collection of functions. This has not proven to be a very attractive quality for feature-based GIS applications as there is not sufficient selective distinction between the different operations permitted on various types of spatial features (ie. point, linear, and area features). However, GIS applications that use a simple image-based structure are more predisposed to this type of manipulation. Map algebra is an example of a function-oriented language used in GIS for manipulating and analysing surface data (Tomlin, 1991). Map algebra uses a set of conventions to provide finer interpretation of the geographic locations (ie. local, neighbourhood, zonal) but these are still manipulated by functional transformations. Map algebra has the advantage of a straight forward notation and is very useful for developing models of spatial interpretations.

3.3 Rule-Based Programming

Rule-based programming is a special case of logic programming. The language is based on a procedural scheme with the canonical condition-action form:

IF condition-pattern THEN actions.

The left-hand side consists of several conditions that return a logical result. The right-hand side consists of several actions. Actions can fire other rules, establish new facts, and perform procedural operations. Rules express relationships and meta-information. Rules are grouped in rulesets known to the inference engine. The engine works in a continuous loop, at each cycle a rule that matches some condition-pattern is chosen and the related actions are fired. The execution stops when no more rules are fireable.

Rule-based programming uses a simple procedural abstraction to search for goals that satisfy the condition-pattern and then subsequently firing the action clauses. Queries are solved as proofs computed from the facts and rule set. Rule-based programming does not directly support data abstractions but relationships can be expressed by meta-rules.

Rule-based programming provides a model of the decision process that suits a range of problems used for spatial reasoning (Scarponcini *et al*., 1995). The techniques have been used in several *ad hoc* system developments for decision support (Lowes and Bellamy, 1994) (Davis and McDonald, 1993).

3.4 Object-Oriented Programming

Object-oriented programming (OOP) is based on concepts for *objects*, *classes*, and the *inheritance* mechanism between classes. An object is an instance of a class to hold all related state information. Since objects can reference other objects, it is possible to build compositions of more complex objects. The classes in a program define categories of objects which share the same state information and procedural interfaces. Inheritance provides a relationship between classes based upon a taxonomy hierarchy. These organising principles are formally based upon classification theory.

OOP has become very popular as it provides a mental leverage for designers to encapsulate the structure and behaviour of design problems as objects. Data abstraction is supported through associative references to express structural relationships between objects, and class inheritance. Procedural abstractions are provided in two ways. The permissible actions on an object, and a configuration of objects, are integrated as part of the object class description. But the final implementation code still uses low level procedural mechanisms to perform operations in sequence, by conditional branching, or within an iteration. A disadvantage is that these control constructs involve the introduction of state variables to hold computational values between operations and procedures.

Writing a program in an OOP language does not necessarily make the program object-oriented. But in general programs incorporate object-oriented design principles

(Rumbaugh *et al.*, 1991). OOP is especially suited to problems where these is a large number of entities to be modelled, each with complex structural relationships and operational semantics. In recent years OOP has made a significant impact on graphical user interfaces (GUI's) and the application programming environment. Desktop GIS's often use object-oriented concepts in the user interface and application programming environment. But in most cases spatial data handling is still based upon a geo-relational model, and so data abstractions such as association and inheritance are not applied to the spatial data. Morehouse (1990) discusses the implications and difficulty of having true object-oriented modelling semantics for spatial databases. The OpenGIS Specification (OGC, 1997) incorporates object-oriented geo-processing concepts. The full development of models to allow user defined schemas will require information representation specified by data dictionaries, schematic catalogues, geometry rules, etc. This technology specification will have an important impact on the adoption of object-oriented data abstractions within GIS programming languages.

3.5 Summary

Different programming paradigms may be characterised by the data and procedural abstractions employed. The four paradigms and the types of abstractions supported are summarised in Figure 2. Note that most language implementations use a combination of programming paradigms, or programmers adopt a style suited to one or another programming paradigm. Therefore in practice this taxonomy is less well defined.

Our objective was to find a language that is easy to understand and is able to express a solution in a very direct and concise manner. To avoid any representation mismatch the data model must be consistent between the application user interface and the programming language. If the prescribed view of geographic information is feature-based, then the programming language must support data access and manipulation using feature structures. Likewise to avoid any problem specification mismatch the style of expression must be consistent between the application user interface and the programming language. If the prescribed view of geo-processing is set-theoretic operations then the programming language must support set queries and set operations on map features.

We believe the geo-relational model is easily comprehended. Users assume that programming a GIS requires manipulating attributes for defined map feature sets. In an object-oriented programming environment this type of data and procedural abstraction is easily supported for queries on a single data set. But for compound queries involving several data sets one quickly finds that intricate control constructs and intermediate state information (record numbers, lists of attribute names and values, iterating variables, etc.) are needed. Most users are not accustomed to this programming style, and find it difficult to reconcile the program code with the problem at hand. We believe that a combination of some aspects of functional, object-oriented and rule-based paradigms offers a better solution. An object-oriented interpretation of spatial features provides a simple semantic interpretation of geographical data, all features belong to a themes in a map with appropriate op-

Programming Paradigms	*Procedural Abstraction*	*Data Abstraction*	*Example of Use in GIS*
Logic	-	Meta-rules	*ad hoc*
Functional	Functional mappings	Functional mappings	Map Algebra
Rule-Based	Conditional pattern and actions	-	*ad hoc*
Object-Oriented	Sequential, repetition, and conditional branching	Objects, class, inheritance	OpenGIS

Figure 2: Programming paradigms and the types of abstractions employed.

erations on member features. The rule-based paradigm provides a simple mechanism to control program flow. Users can easily grasp the IF-THEN rules and see how it is applied generally. Programmers do not need to construct or navigate between objects, this is inferred from the pattern and action syntax of the IF-THEN rules.

One disadvantage of rules is that they become unyielding and their specification is difficult to understand for nontrivial problems. To simplify the way rules are structured we have explored decision tables. The next section shows how structured rule-sets are organised into a tabular form.

4. Decision Tables

Rule-sets are difficult to interpret for any reasonably sized knowledge base. An alternative technique for representing decision rules is as decision trees (Giarratano and Riley, 1994) or decision tables (Reilly *et al*., 1987).

The different forms for representing rules can be shown by example. The example describes rules for choosing the best wine to have with a meal.

Given the following rule-set:

IF (main_course is beef) THEN (wine is red)
IF (main_course is fish) THEN (wine is white)
IF (main_course is poultry) AND (meat is light) THEN (wine is white)

C1	main_course	beef	fish	poultry	
C2	meat	-	-	dark	light
A	wine	red	white	red	white

Figure 4: Decision Table.

IF (main_course is poultry) AND (meat is dark) THEN (wine is red)

This can be represented in a graph form as a decision tree shown in Figure 3.

This can also be represented in tabular form as a decision table shown in Figure 4.

Some of the advantages of decision tables include compactness, self-documentation, modifiability and completeness checking (Reilly *et al*., 1987). Given that information is stored and viewed in a tabular form in geo-relational databases, it seems fortuitous to represent the rules in a similar form. This presents the user with a very consistent representation of data and procedures.

4.1 Prototype Implementation

The concept of using decision tables for spatial query and modelling was explored by implementing a prototype tool. The tool needed to either interoperate or to be programmed with a GIS that offered object-oriented language features. ArcView (ESRI, 1994) was used because it provided a comprehensive application development environment that included an object-oriented programming language.

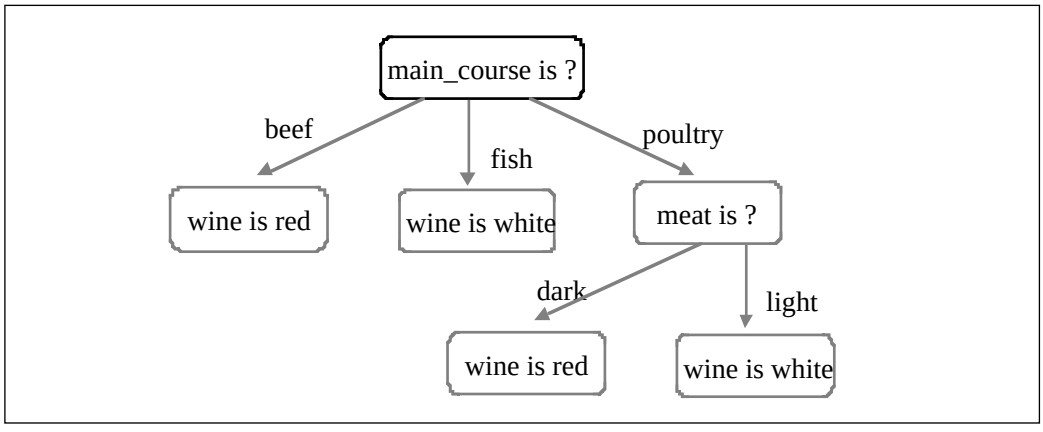

Figure 3: Decision Tree

The organising principle in ArcView is that thematic spatial information is defined and rendered within a geographical portal called a view. A *view* contains a set of themes all registered to the same geographical space. Each *theme* represents a defined set of geographic features with their own distinct display characteristics. Each theme corresponds to a geo-relational model of a data source. The geo-relational model (Morehouse, 1985) is based upon a relational data model where recognised tables have one column with values for a spatial domain. A set of these tables each modelling some thematic set of spatial features which share a common geographical extent are the basis of the layered database concept.

The prototype tool was implemented in ArcView using the native programming environment. The algorithm to implement the rule-based approach is a backward chaining inference engine as described in Giarratano and Riley (1994, p:566). Goal objects were matched against the set of features in a theme. The pattern matching was performed on feature-attributes for specified subject clauses and associated values in the columns of the decision table. The syntax adopted was that a table was identified by its thematic name, this was placed in brackets to indicate it represents a free variable that ranges over the set of features in a theme.

For example:

[tree].growth_rate	is a free variable that ranges over the set of features in the "tree"theme with the named attribute "growth_rate".

Rule inference worked by attempting to match against feature-attributes. The combination of a theme name and attribute domain name identifies feature-attributes in data tables. If a matching feature-attribute could not be found then this was treated as a new attribute derived as part of the inference process. This allowed new derived facts linked to features to be inferred in a natural way. It also relieved the programmer from the burden of setting up variables to hold this state information which was only used during the inference process.

4.2 Example

Two examples of decision tables are described. The first example shows a simple query that may be expressed using an advanced query tool provided within desktop GIS. The second example demonstrates a more complex query that would not be readily represented by any table query tool.

The example is based on a public works problem. Trees located near powerlines need to be periodically trimmed to avoid interference with electrical cables. An application view would include a feature table for tree locations and powerlines.

In the first example a decision table, see Figure 5, is used to express the following query:

check trees within 10 meters of a powerline and have not been trimmed for 2 years.

In the second example a decision table, see Figure 6, is used to develop a more realistic query to account for different growth rates in trees:

> *check trees within 10 meters of a powerline where the growth from last trim height is now within one meter of powerline height.*

Tree height obviously varies over time as a function of recorded height plus growth that has occurred since it

C1	[tree].shape.*DistanceTo*([powerline].shape)	< 10		-
C2	[tree].*SinceTrim*	< 2	-	-
A	[tree].check	true	false	false

Figure 5: Decision Table for first query example.

C1	[tree].shape.*DistanceTo*([powerline].shape)	< 10		-
C2	[powerline].height - [tree].height	< 1	-	-
A	[tree].check	true	false	false

Figure 6: Decision Table for second query example.

C	[tree].type	pine	oak	ash
A1	[tree].growth	0.9 * [tree].*SinceTrim*	0.2 * [tree].*SinceTrim*	0.7 * [tree].*SinceTrim*
A2	[tree].height	[tree].trimHeight + [tree].growth		

Figure 7: Decision Table to deduce tree height.

was last trimmed. The second condition in the decision table specifies a [tree].height which is not a persistent attribute of tress. With pattern matching a decision table is found that lists this attribute (goal) in its action clause. Therefore it is able to infer this information from the decision table shown in Figure 7 and calculate the growth based upon the type of tree.

Growth is given by growth period and a multiplier that varies depending upon the tree type;

$$growth = rate_{tree\_type} * peroid_{since\_last\_trim}$$

Note that both feature attributes for [tree].growth and [tree].height are derived for the purpose of the pattern inference and therefore are virtual attributes defined programmatically.

5. Conclusion

The paper has reviewed the different programming paradigms used in computer languages. The goal is to assess what programming paradigm would best suit integration into the user environment of a desktop GIS. We conclude that a combination of a rule-based and object-oriented programming paradigms delivers an easy way for users to perform relatively complex queries and formulate models in a GIS. Researchers have demonstrated that these methods follow the way humans model information and follow human problem solving (Newell and Simon, 1972). We believe that decision tables significantly simplify the layout specifications of rules, and are consistent with the way information is already presented in tabular form in most GIS's.

To support this conclusion we have implemented a prototype query interface to operate with a commercial desktop GIS. Presently the prototype is able to demonstrate the concepts, but has not been developed to the extent that a graphical interface is provided for users to express and execute queries. The next milestone will be to test the tool on a wide range of queries and distribute a robust version of the tool.

6. Bibliography

Abdelmoty A.I., Williams M.H. and Paton N.W. (1993) Deduction and Deductive Databases for Geographic Data Handling. *3rd International Symposium on Large Spatial Databases*, SSD'93, Singapore, pp.443-464

Arentze T.A., Borgers A. and Timmermans H. (1995) The Integration of Expert Knowledge in Decision Support Systems for Facility Location Planning. *Computers, Environment and Urban Systems* 19(4), pp.227-247

Bratko Ivan (1990) *Prolog Programming for Artificial Intelligence*. Addison Wesley.

Davis J.R. and McDonald G. (1993) Applying a Rule-Based Decision Support System to Local Government Planning. In: *Expert Systems in Environmental Planning*, Editors J.R. Wright, et. al. Springer-Verlag, pp.23-45

ESRI (1994) *Avenue - Customization and Application Development for ArcView*. Environmental Systems Research Institute Inc, Redlands, CA.

Egenhofer M. and Frank A. (1990) LOBSTER: Combining AI and Database Techniques in GIS. *Photogrametrmetric Engineering & Remote Sensing* 56(1), pp.919-926

Frank A.U. and Kuhn W. (1995) Specifying Open GIS with Functional Languages. *Advances in Spatial Information Systems*, Proceedings SSD'95, Portland, pp.184-195

Giarratano J. and Riley G. (1994) *Expert Systems - Principles and Programming*. PWS Publ. Co., Boston.

Lowes, D. and Bellamy J.A. (1994) Object Orientation in a

Spatial Decision Support System for Grazing Land Management. *AI Applications* 8(3), pp.55-66

Morehouse S.D. (1985) ARC/INFO - A Geo-Relational Model for Spatial Information. *Proceedings Auto-Carto 7*, Washington, pp.388-397

Morehouse S.D. (1990) The Role Of Semantics In Geographic Data Modelling. *Proceedings 4th International Symposium on Spatial Data Handling*, Zurich, pp.689-698

Newell A. and Simon H. (1972) *Human Problem Solving*, Prentice-Hall.

Paulson L.C (1996) *ML For Working Programmers*. Cambridge University Press.

OGC (1997) *The Open GIS Consortium*, http://www.opengis.org

Reilly K.D., Salah A., and Yang C. (1987) A Logic Perspective on Decision Table Theory and Practice. *Data and Knowledge Engineering* (2), pp.191-210

Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W. (1991) *Object-Oriented Modeling and Design*, Prentice-Hall.

Scarponini P., Clair D., and Zobrist G. (1995) An Inferencing Language for Automated Spatial Reasoning About Graphic Entities. *Advances in Spatial Information Systems*, Proceedings SSD'95, Portland, pp.259-278

Tomlin C.D. (1991) Cartographic Modelling. In: *Geographical Information Systems* - Vol. 1, Editors D. MacGuire, M. Goodchild, and D. Rhind, Longman Scientific & Technical, pp.361-374