Chapter 5
Design Methodology

It is noted in Section 1.10 that the field of GIS is beset with the following difficulties:

The majority of the current GISs are built around relational databases which have severe limitations in handling spatial data. Formulation of queries, even for the simplest of spatial operations in such GISs is cumbersome. Relational data model lacks the ability to specify explicitly the semantic information about relationships.

The motivation of the present work was to design a GIS that did not suffer from the drawbacks and limitations of the relational data model. Knowledge-based approach offers greater promise in realising that goal. In such an approach, a feature can be manipulated directly, and not via its attributes, as in relational models. An entity has an existence, separate from the descriptive thematic data. The knowledge content of the points, lines, arcs, polygons, and textual descriptions are stored in memory as rules that conform to the production system, OPS5. In formulating the rules, it is not necessary to make distinctions at the top level between spatial and aspatial components of the data. OPS5 can start with a set of overly specific rules and generate new rules that are more general and abstract. When the system over-generalises, it would backtrack or specialise some of the rules that are too general. The convenience and suitability of OPS5 in designing a knowledge-based GIS, have been explained in Chapter Three. This chapter explains the methodology adopted in the design of a knowledge-based GIS (KBGIS), as per the requirements specified in Chapter 4.
5. 1. Data Modelling in KBGIS

A VECTOR spatial data model is used for storing the graphic objects in the KBGIS where the basic unit is a pair of \((x, y, z)\) coordinates. Using these coordinate pairs, points, lines, and polygon (region) are represented. Thus, the map features can be identified in three basic graphic primitives:

\textbf{Point}: An \((x, y)\) coordinate pair with a unique identifier

\textbf{Boundary chain}: A set of ordered coordinate pairs with a chain number, and the unique polygon codes of the polygon that it bounds on the left and on the right. Only boundary chains can be used later to form region features automatically

\textbf{Network chain}: These chains represent the linear features such as roads, rivers, main pipelines, \textit{et cetera}. Every chain in the network is represented as a set of ordered coordinate pairs with a unique chain number. In network chains, both the junctions and the dead-ends are stored as network nodes with numbers assigned. These nodes are then named and stored along with the associated attributes in the spatial object KB [Jayaprakash and Menon 1995].

To make the searching of point objects much faster, locational values of point objects are stored directly as vector points in the spatial object KB. The non-graphic attribute data about point, line, and area features are also associated with each object in the spatial object KB.

Thus, the data model adopted in the design of KBGIS can be summarised in the following three steps:
Firstly, it is a vector representation strategy because the geographic data entities are stored as a vector of coordinate points that define the geometry and topology of these entities precisely.

Secondly, it is a logic based structure because two geographic entities, viz., polygon and line are represented as logic clauses, more specifically as predicate structures describing certain relationship between a series of objects that are attributes of space.

Thirdly, it is a relational structure because the polygon and the line predicate structures can be interpreted as implementation of two relations (tables). Clauses of predicates can be seen as the relations’ tuples. Polygon and line relations can be joined, when necessary, using common identifiers.

5. 2. THE ARCHITECTURE OF KBGIS

Based on the findings of previous research outlined in the earlier chapters, one can conclude that a good GIS should meet the following criteria:

- Ability to handle large, multi-layered, heterogeneous types of spatially indexed data;
- Ability to query such data sets interactively about the existence, location, and relevant properties of a wide range of spatial objects;
- Flexibility in configuring the system so as to accommodate a variety of applications and users;
- Ability of the system to learn, in a significant way, about the spatial objects in the knowledge and data bases during the use of the system.

The architecture of the knowledge-based GIS, reported in this work is shown in Figure 5.1.
5. 2. 1. **Spatial Object Knowledge-Base**

The spatial object KB contains knowledge about the high level spatial objects. The basic unit in the KBGIS implementation presented here is the entity representation, class. Each spatial information is treated as an object. An object should belong to one of the many classes which are pre-defined using the literalize declaration of OPS5. The class representation naturally captures the inherent structure of the data. The classes can be created at different levels of spatial resolutions, and information about the hierarchical links between classes can be stored along with classes themselves. Thus a spatial tree structure can be generated through class-subclass relationship formalism.
A class is defined with various attributes, and the instances of classes are stored as WMEs, in the spatial object knowledge-base, using the `literalize` declaration as in the following example:

\[
\text{(literalize education\_inst} \\
\quad \text{type} \\
\quad \text{name} \\
\quad \text{location} \\
\quad x\_cord \\
\quad y\_cord) \\
\]

where "literalize" is the OPS5 declaration specifying the format of a WME that has attributes, "education\_inst" is the name of the object, that has the attributes, "type", "name", "location", "x\_cord", and "y\_cord."

An instance of the above object "education\_inst" can be created as an element in the working memory by the "make" action as follows:

\[
\text{(make education\_inst} \\
\quad ^\text{type} \text{college} \\
\quad ^\text{name} \text{|University College|} \\
\quad ^\text{location} \text{Palayam} \\
\quad ^\text{x\_cord} 1583.0 \\
\quad ^\text{y\_cord} 2385.0) \\
\]

where the accent symbol ^ is used to denote an attribute that is immediately followed by its value.

Two different instances of the same object can be created similarly as follows:

\[
\text{(make education\_inst} \\
\quad ^\text{type} \text{school} \\
\quad ^\text{name} \text{|Cotton Hill High School|} \\
\quad ^\text{location} \text{|Cotton Hill|} \\
\quad ^\text{x\_cord} 12148.0 \\
\quad ^\text{y\_cord} 8647.0) \\
\]

and
In this manner, a spatial object knowledge-base of any arbitrary size, and consisting of any complex spatial objects can be created by the "literalize" declaration and the "make" action. Each element of the spatial object KB is thus a class (object) with a class name and a number of attribute value pairs. The attributes may be scalar or vector. The value of a scalar attribute is an atom, and the value of a vector attribute is a list of one or more atoms.

5.2.2. The Rule-Base

While the information about the spatial objects and their definitions are stored in the spatial object KB, the information about search heuristics, object classification, object complexity, et cetera, are stored in the Rule-base, which, as stated in Chapter 3, is an unordered set of if-then rules, called productions.

To find and display all the schools known to the system, we could use the following production:

```
(p find_school
  (education_inst
    ^type school
    ^name <schlName>
    ^location <loc1>)
  -->
  (wrte (CRLF) <schlName> <loc1>))
```
Here, "p" indicates a production, and "find_school" is the name of the production. The statement following the production name is the condition element of the production (LHS). This production is satisfied by all the objects in the spatial object KB whose class name is education and the value of the attribute "type" matches the pattern "school." If the production is satisfied, all the schools in the spatial object KB will be selected, and the action on the RHS will be executed whereby, the names and locations of all the schools known to the system get listed on the printer.

Constants and variables are used as attribute values in the condition elements on the LHS of a production to constrain the search. Constants can be strings, integers, or floating point numbers. A variable is enclosed in angle brackets < >.

The production for the query for finding the location of the hotel named "Luciya," will take the form:

```
(p find_hotel (hotel
  ^name Luciya
  ^location <loc>
  ^x_cord <x_cord1>
  ^y_cord <y_cord1>)
  -->
  (write (CRLF) <loc> <x_cord1> <y_cord1>))
```

Comparisons to be made for values that an attribute may take in a conditional element of a production is implemented using the relational operators, as illustrated below:

```
(block
  ^block_name <blk>
  ^area >= 200)
```
Conjunction is implemented by enclosing the list of conditional tests in braces {}, as shown below

(block
  ^block_name  <blk1>
  ^area      <area1>
  ^population  {>20000 <30000})

Compute and external functions are implemented to perform mathematical calculations and to assign attribute values to the conditional element as in the following:

(block
  ^block_name  <blk1>
  ^population  {>20000 <30000}
  ^area      <a1>

(place
  ^place_name  <blk1>
  ^area1      (compute <a1> — 1000))

5. 2. 3. The Query Processor

In query processing, the user can enter a query, either through the keyboard or by selecting options from the menu. The query entered is parsed and checked for syntactic correctness, and the user can modify the query, if required. The query processor module consists of Search and Analysis Routines. First the search routines are invoked which itself is divided into two levels of searching, viz., the High Level Object Search, and the Low Level Graphic Element Search. In answering a query, the high level object search is invoked first.
5. 2. 3. A. High Level Object Search

The high level object search is performed on the spatial object KB, and is implemented using the OPS5 inference engine which, as explained in Chapter 3, is a forward chaining inference mechanism. Exiting from IE occurs after the match state, either because of an explicit halt or because there are no more rule instantiations in the conflict set. It is also possible to have a limit on the number of rule-firings by setting break points on particular rules.

If the match step produces a conflict set containing more than one rule instantiation, the conflict resolution is performed to select one instantiation for firing. In the KBGIS, the MEA strategy is used in this step. (Details and alternative strategies are given in Appendix B) With an instantiation selected, the execution phase is entered. In this phase, variables are bound to values and the action on the RHS of the production associated with the selected instantiation, is executed. When a given query is satisfied, the answers to the query can be output by invoking the Display and Report Generation Module.

5. 2. 3. B. Low Level Graphic Element Search

If the search on the spatial object KB is unsuccessful in producing the required answer to the query, then a low level graphic element search procedure is invoked for a detailed search on the spatial graphic primitive entity database. If the search turns out to be a success, the Display and Report Generation module will handle the output.
5.2.3. C. Multiple Queries

An important feature of the user interface of the KBGIS is that, in addition to the keyboard input and the menu-based input of queries, it also provides a set of pre-stored queries pertaining to the domain city, with facility for inputting optional parameters through submenus. This is achieved by selecting the Standard Query option of the Main menu (To be described in Chapter 6). Seventeen such standard queries have been identified, labelled, and stored. When the Standard Query option is selected, these "intermediate queries" will be listed on the screen, starting with Query 1, and the submenu associated with it, for effecting the choice of parameters. Other queries in the series can be browsed (and selected) by manipulating the vertical scroll button on the right side of the window. Each one of the "intermediate queries" is associated with a submenu that enables the user to exercise different parameters that are appropriate to the query. This feature enables a novice to quickly get the hang of navigating the GIS and get the required information. Some of the standard queries, and the submenus associated with them, are illustrated in Chapter 7.

The exercise of options through parameters, serve another purpose viz., reducing the clutter on the display screen. With several roads criss-crossing, placenames and location identifiers can get crowded. For certain queries, therefore, a priority on the roads to be displayed is enforced so that, the display can be controlled selectively, layer-by-layer. The situation is the same with the placenames, and the same strategy is employed whenever display of placenames are called for. The discussion on Query 1 in Chapter 7 should clarify the points further.
5.2.4. Learning Module

The main purpose of learning procedures in the KBGIS is to reduce query search time. In a production system, incremental changes in performance can be effected by adding new production rules to an existing rule-base. Learning is accomplished in two ways, either by remembering the results of previous search or by learning the definition of an object more precisely so that the search space may be pruned rapidly. The details of learning, implemented as an integral part of OPS5, is given in section B.7 of Appendix B.

5.3. SUMMARY

THIS CHAPTER has enumerated the desiderata of a good GIS, borne out by the earlier researches in the field. The choice of the vector model for representing the geographical data for the particular application envisaged in the present work has been reasoned out. The relevance of using OPS5 in implementing the KBGIS has been highlighted. The chapter has described the architectural features of KBGIS, especially with respect to the rule-base, the spatial object facts-base, and the query processor.