An object-oriented geographical modeling system

David A. Bennett
*Southern Illinois University Carbondale*

Marc P. Armstrong
*The University of Iowa*
AN OBJECT-ORIENTED GEOGRAPHICAL MODELING SYSTEM*

David A. Bennett
Department of Geography
Southern Illinois University
Carbondale, Illinois 62901-4514

Marc P. Armstrong
Departments of Geography and Computer Science
The University of Iowa,
316 Jessup Hall
Iowa City, IA 52242

ABSTRACT

This paper describes a geographical modeling system (GMS) that provides users with the atomic objects and object managers needed to construct sophisticated spatial simulation models. As such, the system provides a digital environment in which spatial knowledge can be stored and managed, theory can be modeled and tested, and alternative resource management strategies can be evaluated. To develop this system we used object-oriented analysis and design methods to integrate modelbase management and geographic information system technologies into a single system.

INTRODUCTION

Commercial geographic information system (GIS) software does not fully support the study and management of complex spatial systems because it provides users with only limited ability to represent and model dynamic spatial processes. This research takes a step toward overcoming this limitation by providing a work space in which analysts can quickly construct geographical models of dynamic spatial systems. This capability is developed by integrating modelbase management techniques with GIS technology in an object-oriented software system.

OBJECT-ORIENTED GEO-PROCESSING SYSTEMS

Work in object-oriented GIS (Worboys et al., 1990) and spatial decision support systems (SDSS) (Armstrong et al., 1989) illustrates the utility of the object-oriented paradigm in the representation of geographic phenomena. In such systems, geographical features are represented as cartographic objects that encapsulate thematic information and are accessed through an object-oriented database management system. Topological relationships between geographic features are modeled by linking objects into a class composition hierarchy. Communication between objects in this hierarchy can occur by passing messages through these topological connections.

An object-oriented approach to GIS software development also allows users to encapsulate spatial structure and behavior into the class definition of geographic features. Defining features as aggregate objects that possess both spatial structure and behavior facilitates: 1) code reusability, 2) the representation of polymorphic behavior (e.g. a geographic feature that exhibits multiple forms of the same behavior), and 3) model management. In addition, by incorporating feature behavior into GIS the temporal dynamics of geographic systems can be modeled.

These capabilities can support the construction of spatial decision support systems within the more general framework of GIS software. A SDSS helps decision-makers by providing them with access to domain specific knowledge, spatial models, and spatial data handling techniques (Armstrong and Densham, 1990). With these tools the decision-maker and/or analyst can explore the problem domain, gain

* portions of this paper appear in the Proceedings of GIS/LIS '93
greater insight into the behavior of a spatial system and determine how it will respond to management. The literature (see for example Armstrong and Densham, 1990) provides a blueprint for SDSS. However, a fully functional SDSS that includes all defining characteristics does not yet exist. A major impediment to the construction of such a system is the difficulty of integrating spatial database and modelbase management techniques.

Modelbase Management Systems.

Modelbase management systems (MBMS) store, manipulate, and retrieve models (Dolk and Konsynski, 1984). By managing models like data, algorithms can be shared between applications and, thus, model redundancy is reduced and the consistency and extensibility of the modelbase is enhanced. For the analysis of spatial systems, modelbase management software must seamlessly integrate mathematical models that capture the behavior of spatial processes with digital models that represent spatial structure.

Existing modelbase management systems are rare. One notable example, however, is EDSS-1 a prototype environmental decision support system proposed by Guariso and Werthner (1989). This system is comprised of five components: knowledgebase, database, modelbase, system manager, and dialog menu. In this framework, compound models can be constructed by coupling input and output variables of applicable sub-models. This provides the flexibility needed to construct sophisticated nested and linked models. Jankowski (1992) used a similar approach (see also Zeigler, 1990) to develop a water quality model. However, both approaches lack support for geographic data structures and, thus, the ability of this software to represent spatial structure is reduced.

GMS DESIGN AND IMPLEMENTATION

The GMS is designed for experts who study and manage dynamic spatial systems. Although these individuals may possess considerable knowledge about a particular problem domain, it cannot be assumed that they have sufficient expertise in computer science and/or GIS to develop sophisticated simulation software. The challenge in designing the GMS, therefore, was to develop a software environment in which users could translate their conceptual and mathematical models of spatial systems into the digital domain without requiring them to construct complex computer software systems. To accomplish this goal, the GMS was developed using an object-oriented approach to software development.

Within the GMS framework, geographic models (e.g. watershed models) are decomposed into sub-models (e.g. to simulate sub-watersheds) and geographic features (e.g. streams and ridges). Geographic features, in turn, can be decomposed into structural objects that capture topographic form and behavioral objects that represent the flow of material and energy across the landscape. Note that these features can exhibit several forms of the same behavior depending on their current state. The flow behavior of a stream, for example, can be modeled as a clear, sediment laden, hyperconcentrated, or debris flow depending on sediment concentration. Since mathematical models are often valid only within a specific range of physical conditions, it may be necessary to provide more than one mathematical representation to model the full range of behavior associated with a particular feature. The behavior of a geographic feature, therefore, is modeled as a set of individual behavioral models and the GMS must ensure that the appropriate model is used throughout the simulation.

To enforce the valid use of simulation models, two forms of knowledge are encoded into the GMS. It is necessary to know 1) the physical conditions under which the application of each mathematical model is valid and appropriate and 2) how to restructure the simulation model when a mathematical representation is no longer appropriate for a given geographic feature. Given this knowledge, a comparison can be made between the current state of a feature and the assumptions built into the model that simulates its behavior. If assumptions are violated, then a more appropriate model must be found before the simulation can proceed.
New Model Construction.
The design and implementation of a simulation model can be viewed as a four step process: 1) abstract from reality those features that influence the spatial processes of interest, 2) design a conceptual model that captures the form and behavior of individual features, and illustrates how these features interact to form a single geographic system, 3) cast this form, behavior, and interaction in terms of data structures, mathematical equations, and topological relationships, and 4) translate this representation into computer code. Note that these steps closely parallel the process of object-oriented analysis and design as defined by Rumbaugh et al. (1991).

At this time the GMS supports steps 3 and 4 in the model development process. The materials from which new models are constructed consist of abstract classes that represent the fundamental form and behavior of geographic objects. The tools that assist in the creation and use of environmental models are stored as three classes that provide modelbase management capabilities (constructors, implementors, and interfaces). Constructor objects accept the design entered by the modeler and create the code needed to implement the model, implementor objects manage the use of simulation models, and interface objects provide an icon-based user interface that supports drag and drop model creation.

Classes Representing Geographic Objects.
Nine class hierarchies are used to represent the form and behavior of geographic systems. When new environmental models are created, the GMS extends these class structures to construct objects that match the users conceptual and mathematical representation of the natural system under study. The class hierarchies used to represent geographic systems in the GMS are:
- **CCartography**: defines a spatial data structure (e.g. a chain) and/or topological linkage.
- **CParameter**: stores a particular datum needed to calculate a mathematical model (e.g. Manning's $n$).
- **CBehavioralModel**: stores a mathematical model and knowledge about its assumptions (e.g. Manning's flow equation).
- **CBehavior**: captures the range of behavior exhibited by a geographic feature as a set of one or more instances of CBehavioralModel (e.g. stream flow).
- **CStructure**: captures the form and location of a geographic feature (e.g. a stream channel).
- **CFeature**: represents the structure and behavior of a geographic feature (e.g. a stream segment).
- **CFlow** and **CMWayFlow**: represent the flow of energy and material through the landscape.
- **CGeoModel**: models a geographic system (e.g. a watershed).
- **CProject**: a simulation model.

In the GMS, the design of a computer simulation model is analogous to the construction of a class composition hierarchy with each lower level in this hierarchy representing an increasingly simple spatial object (Figure 1). By decomposing geographic systems into collections of simple spatial objects, the system has the flexibility to meet diverse user needs. At the lowest level in this decomposition, objects represent cartographic primitives, spatial relationships, inter-object flows of material and energy, mathematical models, and descriptive parameters. At the next higher level, the physical structure of geographic features (e.g. a stream reach) is captured by classes composed of a cartographic primitive and parameters that provide input to mathematical models. The dynamic behavior of these features is represented by mathematical models and descriptive information that documents the conditions under which the model can be applied. A geographic feature, therefore, is represented by a class composed of a structural object, topological relationships, inter-object flows, and one or more behavioral objects. At the next level of abstraction are simulation models that capture the form and behavior of the geographic system under study (e.g. a drainage basin). These models are represented by a class composed of geographic features and sub-models. Finally, CProject objects store and execute root models (models that are not sub-models).
Figure 1. Simulation models of geographic systems can be defined as class composition hierarchies.

**Model Manager Classes.** Three class hierarchies assist the user in the creation and use of environmental simulation models:

- **CCManager:** Constructor managers interact with the user to create new environmental simulation models.
- **CIManager:** Implementor managers read in and help execute simulation models.
- **CInterface:** Interface classes interact with the user during the construction and use of simulation models.

**Interface Classes.** The user interacts with the system through an icon-based graphical user interface. This interface was built by extending the window and document handling classes provided with Think C, and the resource library associated with the Macintosh operating system. Within a window the programmer can establish icons, dialog boxes, buttons and panes. For the GMS, the Think C class hierarchy was extended to include view windows that provide the user access to currently defined classes, build windows that facilitate the creation of new classes, and simulation windows that support model use.

**Class CCManager.** New models are created by extending the **CBehavior**, **CStructure**, **CFeature**, and **CGeoModel** class hierarchies to reflect a particular environmental system. Each new class begins as a proto-object. A class description for these objects is created by dragging an icon of another object into the icon of the proto-object. For example, a user may create a structural object to represent a stream channel. To indicate that this channel will be represented by a chain data structure, the user would drag the icon of class **CChain** (or **CSurfaceChain**) over the icon associated with the prototypical channel and release the mouse button. The **DropIn** method checks for object compatibility and redundant definitions. The prototypical objects created by this process are also responsible for constructing the code that defines their structure and behavior. This process is analogous to engineering a new organism. A new, and hopefully desirable, structure and behavior result from the integration of the "genetic" code extracted from several dissimilar objects.

The process of model development proceeds in a stepwise fashion, with increasingly complex objects being derived from collections of simple objects. Model construction proceeds until users have incorporated their conceptual models into the GMS class hierarchy. The way in which class hierarchies are extended and class definitions propagated, is class dependent. As such, there is a subclass of **CCManager** for each
When the user has completed this process of model development these \textit{CCManagers} direct all new proto-objects to generate the header and source code files necessary to define and implement the new class.

Class CIManager. Implementor managers assist in the execution of fully defined simulation models. These involves two primary tasks, reading information into the system and supporting the simulation process.

One of the more difficult tasks associated with implementing spatial data structures is the establishment of topological connections. This task becomes even more problematic when users are allowed to construct new connections to meet application-specific needs. This problem is solved in the following manner. Objects are stored in a file by class name and identification number. When an object is read, a new instance of that class is created and added to a list of objects stored in the appropriate \textit{CIManager}. A reference to all objects contained in, or linked to this object is then read and stored as a generic object that possess only a class name and identification number. After all objects in the system have been read and created, \textit{CIManagers} can use these generic objects to establish topological connections. To launch a simulation, the user selects a project icon. The associated project object sends \texttt{ReadIn} messages to all \textit{CIManagers} and, then, \texttt{Simulate} messages to all root models.

\textbf{CONCLUSION}

Our goal is to provide a digital environment in which spatial knowledge can be stored and managed, theory can be modeled and tested, and alternative resource management strategies can be evaluated. Such systems will allow users to construct spatial decision support systems within the more general framework of GIS. The research presented above provides a first step toward the creation of such a system.

\textbf{BIBLIOGRAPHY}


