COOPERATIVE MODELING IN APPLIED GEOGRAPHIC RESEARCH*

GUSTAVO ALONSO†
Computer Science Department, University of California
Santa Barbara, CA 93106, USA

and

AMR EL ABBADI
Computer Science Department, University of California
Santa Barbara, CA 93106, USA

Recieved
Revised

ABSTRACT

The characteristics of geographic data and the nature of geographic research require the participation of many agents. Data is generated by multiple sources (satellites, ground observation, weather stations, photography, etc.), accessed, processed and transformed by many users and available for use to an even larger population of users. Lack of coordination among all these different agents may render large amounts of work useless. Most existing GIS (Geographic Information Systems) do not provide any support for cooperative work, which adds to the problem. To overcome this serious limitation while still allowing users to take advantage of GIS technology, we propose GOOSE, a system implemented as a top layer for existing GIS. GOOSE provides the tools for constructing large geographic models in a cooperative environment with potentially many users and participants.

Keywords: Cooperation, Modeling, Geographic Information Systems, Geographic Research, Databases, Metadata, Object-Oriented Systems

1. Introduction

Geographic applications are known for the size and volume of the data involved. It has been estimated that in less than ten years, satellites around the earth will produce one terabyte of information every day [GM93]. Systems such as EOSDIS [GM93] or SEQUOIA 2000 [Kea93] are being designed and built to deal with the problem of accessing and storing the data. Moreover, Geographic Information Systems (GIS) have the added problem of the multiple formats in which the information is represented. This often results in additional information required to interpret the data. Agencies developing standards for spatial data handling have defined a whole

*This research was partially supported by the NSF under grant number IRI-9117904.
†E-mail: {gustavo, amr}@cs.ucsb.edu

set of directives about the nature of this metadata [Moe88, Mor88, SDT92]. In addition to this, research in this area often overlooks the fact that geographic data is almost never used "as is". Most activities related to geographic research involve modeling geographic phenomena in one form or another. Modeling takes place by applying successive transformations to some input data, which results in the creation of derived data. Derived data can not be interpreted correctly without knowledge about how it was created and the global model of which it is part. Furthermore, the complex models used require the collaboration of several researchers and involve using the model by people that may not be its original designers. To complicate matters even further, the models themselves undergo changes as more precise knowledge and data becomes available and, often, those changes are introduced by other users. Hence the need for tools to support cooperative modeling. Few GIS, if any, provide support for modeling, let alone for the cooperative environment in which this activity takes place.

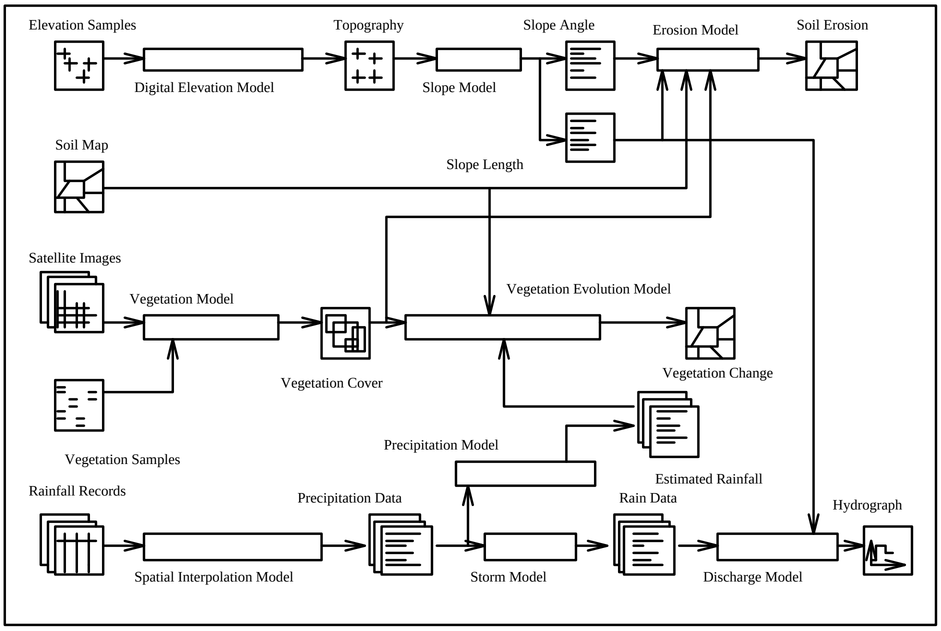

Figure 1: Modeling geographic phenomena as a set of transformations

Within this framework cooperation has several aspects. The main one is that multiple agents interact with each other to create complex structures/models. The diversity of the agents is quite ample, ranging from data sources (satellites, aerial photography, ground observations) and the data repositories (databases, tertiary storage, file systems) to the researchers (hydrologist, geologist, geographers, computer scientist) and the tools involved (telemmetry, photography, image processing). Each of these agents may have an entirely different view of the data/models, and their interactions are not necessarily coordinated, both novel issues in cooperative environments. Hence, geographic modeling becomes the vehicle and the ultimate goal of the cooperation among all these agents. Furthermore, there is also the issue of temporal cooperation. As in any *legacy* system, the datasets may be used and accessed a long time after they are created. This raises the issue of defining cooperation between actions executed maybe years apart. Finally, the success of the modeling effort depends on the ability to integrate diverse platforms and media into a whole that bridges the inherent incompatibilities of the many parts involved. Thus, a cooperative information system that supports this activity is a tool to build a common model in which all agents can access the different submodels. This tool must also address and provide solutions to issues such as temporal cooperation, interoperability of multiple platforms, different views of the same model, and coordination of all the actions that take place within the system.

In this paper we present the architecture of GOOSE, a cooperative information system for GIS along the lines of the previous definition, based on an object oriented paradigm. GOOSE has been designed to provide the high level layer lacking in most GIS to support modeling and cooperative research work. GOOSE makes several novel contributions such as the notion of *projects*, which are used to mirror the activities of geographic modeling and to allow identifying complex sets of transformations as reproducible single entities. Different users can work concurrently in different parts of a model and then combine their efforts into a single project. GOOSE provides the mechanisms to create *objects* that are self contained data units whose interpretation and history is incorporated as part of the objects' attributes. In this way, for instance, an object's *lineage* (how it was created, using what sources and so forth) is always reproducible from information contained in the system. GOOSE is also intended to provide a bridge between different GIS systems, which is done by supporting multiple views of the data regardless of the characteristics of the underlying storage system.

The paper is organized as follows: in the next section we discuss a simple example of geographic model. This example will be used throughout the paper. Section 3 lists a set of minimal features that GOOSE must provide to support cooperative modeling. Section 4 contains a description of the architecture of the system. Section 5 explains in detail how this architecture is used to implement the features discussed in Section 3 in light of the example. Section 6 concludes the paper.

2. Geographic Research

In this section we present an example of geographic modeling to illustrate the characteristics of these applications. This is also intended to motivate the discussion of the features that the system should provide. The example, shown in Figure 1 has been simplified for clarity and has been adapted from [ZG91, SS93].

The purpose of the overall model is to study the changes in the erosion patterns, vegetation and hydrographic characteristics of the area. It can be divided in three major, interleaved, parts. The erosion model takes information about the slopes of the area, its soil characteristics and vegetation cover to produce an estimate of the erosion of the terrain. Note that the soil information is obtained directly from

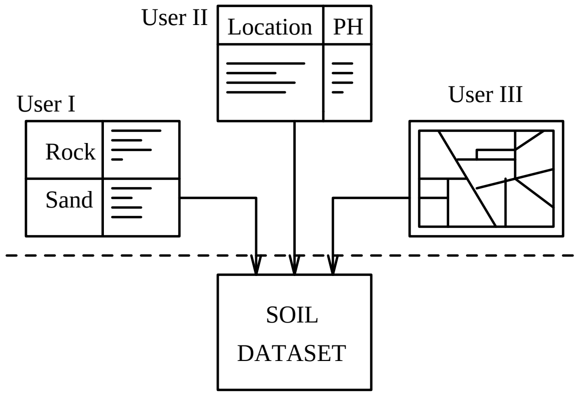

Figure 2: Multiple views of the same dataset

available data. However, the slope information is not readily available and requires taking elevation samples and processing them to get the desired information. This is done by using two more models, the Digital Elevation Models and Slope Models, also shown in the figure. The data about vegetation changes is the result of a vegetation evolution model. This model takes several inputs, some of them primitive, i.e. raw data such as the soil map, and some of them derived (by applying other models). Finally the discharge model involves interpolating rainfall records, calculating the storm coverage and applying a flow analysis algorithm to define an hydrograph (showing the flow of water at a given point).

3. Computer Supported Cooperative Modeling in Geographic Research Environments

The requirements of GIS environments can now be discussed in light of the example. We start by first noting that the system should support several concurrent modeling efforts. This means to allow multiple users to access the same set of data, which is essentially the same as supporting multiple views of the underlying storage system [BCL86, BLT86, LHM86, Han87]. Consider for instance the two uses made of the soil map. The erosion model is likely to focus more on certain properties of the terrain such as its physical composition. The vegetation model, on the other hand, probably uses more the chemical properties of the terrain such as its *ph* (in both cases the characteristics considered are not exclusive, this is only meant as an example). For efficiency reasons, the soil information is stored only once, but the system must provide different views of it. Figure 2 shows three different views of the soil dataset. A user interprets this dataset as a collection of terrain types and their characteristics. Another user sees the data as a list of locations and the corresponding *ph*. A third user reads the data as a map of areas with the same soil characteristics. With complex datasets, the number of possible views can be very large and the system should be flexible enough to support all of them.

Lineage (Soil-Erosion) :

A) Erosion Model, based on

- Slope-Length
- Slope-Angle
- Soil-Map
- Vegetation Cover

B) Erosion=-Model, based on

- Slope-Length, from Slope-Model
- Slope -Angle, from Slope Model
- Soil-Map, primitive
- Vegetation Cover, from Vegetation Model, based on
  - Satellite Image, primitive
  - Vegetation Samples, primitive

Figure 3: Lineage of the Soil-Erosion dataset a two different resolutions

In a model of moderate complexity, involving several submodels, the outputs of individual models can not be correctly interpreted without information about the model itself. Similarly, if views are materialized and stored, they become effectively new datasets whose meaning may not be immediately obvious to users other than the creator of the view. Hence a mechanism is needed to describe the history of the data: how it was created, by which models, source data, etc. This process, known as lineage tracking [Lan88, LV90], is of outmost importance for the correct interpretation of the data. In a legacy system, multiple users "cooperate" during a long period of time by adding information to the system. If this data is to be of any use, it is necessary to record why and how each dataset was created. There is no room for cooperative work if the data interdependencies can not be interpreted and the meaning of the data can not be found in the data itself. For the datasets in the example lineage tracking can be done at different levels. Figure 3 shows two lineages for the Soil-Erosion dataset of the example. Lineage A specifies the model used to create Soil-Erosion, the Erosion Model, and the inputs to that model. Lineage B is a more detailed description. The inputs are also described in terms of the models used to create them. For the Vegetation Cover dataset the lineage specifies the model used and the inputs to that model. In general there are several possible descriptions of the lineage of an object, depending on the amount of detail included. The user should be allowed to select the level of description that is most suitable.

Most of the modeling done using computer programs (ultimately the models in the example are computer programs in one form or another) takes place through iterative refinement executions of the model until it is considered to be correct. This involves adjusting model parameters, changing inputs and so forth. To some extent these models are also used to answer "what if" type of questions with new parameters, more accurate inputs, and more complex algorithms. To support such a basic aspect of modeling, the system must provide automatic change propagation (a term borrowed from design databases [BK85, CK86]), i.e., when the input to a model changes, the entire model is run again to get a new version of the output. Furthermore, these changes occur as versions of the same dataset so the system must also provide version control capabilities. For instance, as new rainfall records are incorporated to the database, the predicted rainfall dataset resulting from the precipitation model can be updated. This, in turn, triggers a reevaluation of the vegetation evolution model which produces a new vegetation change dataset. By recording the vegetation change datasets as versions of each other, it is possible to study the effects of the predicted rainfall on the vegetation evolution model.

Many of these issues are related and can be seen as different aspects of the same basic problem: how to keep track of complex data interdependencies. In a sense, what is needed is a mechanism to identify sets of models, like the one in the example, as single entities that can be manipulated as a whole. For instance, the entire example could be considered as a resource monitoring model. Some users will be interested in seeing it as a black box with five input datasets (elevation samples, soil map, satellite image, vegetation samples and rainfall records) and three output datasets (soil erosion, vegetation change and hydrograph). Other users may be interested only in the vegetation evolution model or in the erosion model. Hence, it is clear that the system should also provide different views of the modeling activity itself, allowing nested models to be able to describe them at different levels of resolution.

From this discussion, we can summarize the following set of basic requirements a GIS must meet to support cooperative geographic modeling:

- Multiple views of the data: access to the data should be independent of the storage system. Support for concurrent access by several users with different semantic interpretations of the data. Multiple views is a basic requirement to have a multiuser system.

- Data history records: the lineage of a dataset must be available at all times. This includes models and sources used to create the dataset. Such information is crucial to guarantee the correct interpretation of legacy and derived data.

- Automated change propagation: create new versions of derived data when new versions of the sources become available. This amounts to the system being able to reproduce user models, thus facilitating the task of updating information across the database.

- Version control: maintain information about the existing versions of a given dataset and provide consistency checks. Version control is necessary to keep track of the changes introduced in large models and it is a basic feature of any cooperative system.

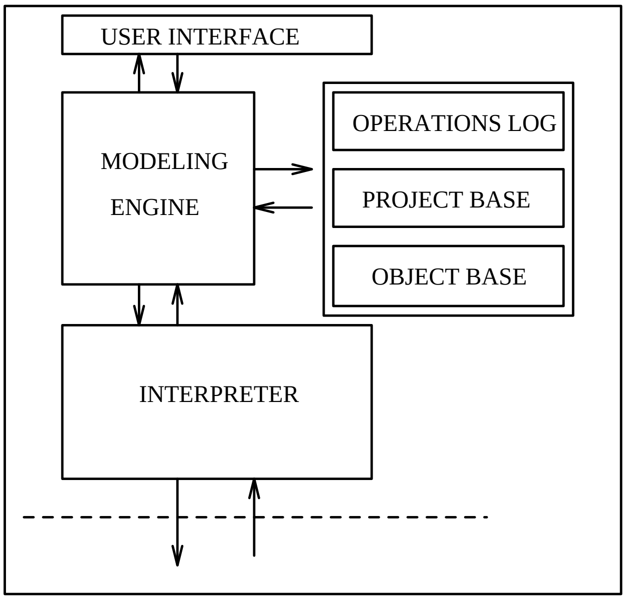

Figure 4: Block diagram of the architecture of GOOSE

- Multiple views of the modeling activity: a model must be divisible into smaller units that can be manipulated separately. A model should be suitable to be treated as a whole or as a set of individual parts. The idea is to allow structured modeling and to ensure a useful organization of the users cooperating in the design of the model.

In what follows we describe the architecture of GOOSE and how it implements features to meet these requirements.

4. System Architecture

Users accessing the data through GOOSE are provided with a uniform interface to the storage system (by storage system we mean the underlying GIS), modeling capabilities (with models that can be physically executed in different underlying systems, transparent to the user) and the high level support to maintain multiple views of the data and the models. GOOSE is not a stand alone system. Functions such as spatial data management, data storage or spatial query processors are not provided. GOOSE relies on existing GIS to provide these functions. GOOSE implements the modeling capabilities the underlying system lacks.

GOOSE is conceptually organized at two levels: the object level and the project level. The object level is intended to provide transparent access to the datasets. An object ties together the access reference to the data and the metadata that allows its correct interpretation. The project level implements the modeling aspects using the object level as an intermediate step to unify access to datasets and keep track of the data interdependencies. The implementation reflects to a certain extent this conceptual division. Figure 4 shows a high level description of the architecture. The main components are the *interpreter*, basically an interface between GOOSE and the underlying system; the *modeling engine*, where most of the functionality is implemented; the *object base*, which provides persistent storage for the object level entities; the *project base*, which provides persistent storage for the project level entities; and the *operations log*, where the system stores the mappings between GOOSE actions and the corresponding commands in the underlying system.

4.1. Interpreter

Since GOOSE does not provide data storage, it is necessary to rely on an interface to access the system that actually performs these functions. The interface is intended to be as general as possible and is considered a module in itself to ensure this. Datasets in GOOSE are represented as objects (explained in detail in the next sections). An object is not the dataset but a reference to it. This reference can be as simple as the name of a file and as complex as a full query to some database system. The role of the interpreter is to translate these references to commands that can be understood by the underlying system. For instance, the current prototype of GOOSE works on top of the UNIX file system, thus references to datasets are translated into UNIX commands. This allows defining objects independently of the way they are stored in files. An object can be defined as a file, part of a file, or several files.

Similarly, GOOSE does not support or encourage any particular programming language or execution environment. The generality of GOOSE is guaranteed by relying on the interpreter module to translate GOOSE commands into calls to programs or commands from the underlying GIS. A model is stored in GOOSE as a project (explained in detail in the next sections). A project involves the execution of one or more transformations of input data to produce output data. This results in the execution of a program or a procedure call to the underlying system. The current prototype uses C and FORTRAN programs as transformations, thus the execution of a model in GOOSE results in the execution of a C program with the appropriate inputs.

The role of the interpreter will become clearer as we discuss other features of GOOSE. The reader should keep in mind that the design target is to achieve as much generality as possible, not tying GOOSE to any particular GIS. For example GOOSE will be eventually incorporated into a larger GIS project [SSAE93] that defines its own language for data manipulation, MDBL, and an execution model [SS93]. This adaptation will require only to redefine the interpreter. Once several interpreters are available, GOOSE can be effectively used as a tool binding together heterogeneous GIS, allowing cooperative modeling across different platforms.

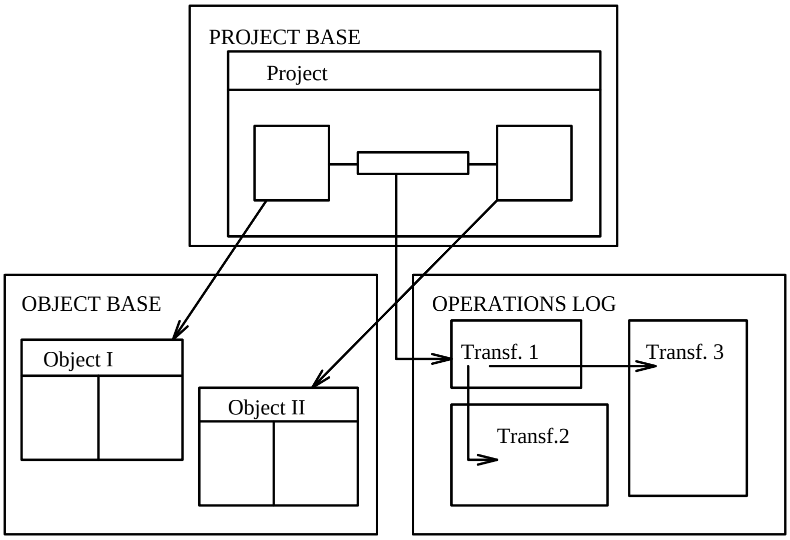

Figure 5: Main data structures within GOOSE

4.2. Object Base

The object base is part of the internal database GOOSE uses to implement the project and object level. An object, in GOOSE terminology, has a name (given by the user), an identifier (generated by the system and for internal use only) and two sets of attributes, data and metadata attributes [AE93]. The data attributes contain the reference to the actual dataset and how to retrieve the data. This reference can be a pointer, a file name, a query, a shell script that extracts data from a set of files, etc. The data attributes are the mechanism used in GOOSE to support multiple views of the data. Since an object contains only the specification of how to retrieve the data, several objects can refer to the same dataset without creating conflicting accesses.

The actual retrieval of the data is performed by the interpreter, which redirects the commands to the appropriate system. For instance, an object can be defined as a concatenation of UNIX files by referencing the data using a UNIX "cat" command. Similarly, an object could be defined as database query, a GIS command, and so forth. Furthermore, these objects could coexist in the same GOOSE system, which provides an effective way to combine heterogeneous systems.

The metadata attributes of an object store information describing the data interdependencies of that particular object. To model these interdependencies, we have identified [AE92] a class of interactions between transactions in a database (similar to the conventional reads-from, over-writes and reads-before relations [BHG87]). This class can be captured using graphs but, for efficiency reasons, the information has been encoded within each object. In this way an object can be described entirely and correctly interpreted solely using information contained in the object itself. This information forms the metadata attributes of the object and contains references to the sources used to create this object, dependencies with other objects and version information. Figure 6 shows how an object is described within GOOSE. The object name is assigned by the user. The object identifier is a system generated identifier used internally to index the data. The data attributes contain the *access method* (a description of the underlying system) and an *access mask* that is the actual procedure to access the dataset. The metadata attributes specify the *sources* used in the creation of the object, its *versions* and any use that may have been made of it (as a source to create other objects).

The object level implements objects as "labels" that are used to access datasets in a variety of systems. The "label" approach allows multiple views of the datasets and establishes the object as a self-contained semantic unit. In GIS environments, the meaning of the data is often exogenous and not found in the data itself [Tob79]. Moreover, the ability to create new data is far greater than that of keeping track of how the data was created [Lan88]. In this framework, the concept of an object that can be treated as a semantic unit independently of the underlying storage system is an important contribution.

4.3. Project Base

Projects in GOOSE are seen as a transformation from a set of input objects to a set of output objects. These transformations can be *primitive* (executable) or *composite*. Primitive transformations correspond to user programs, procedure calls, commands in some GIS and, in general, any transformation of objects that is performed outside GOOSE. Composite transformations, on the other hand, are internal constructions that allow combining several primitive transformations, or other composite transformations, into higher level transformations. Composite transformations can not be executed as such. They do not correspond to any executable code, rather, they can be decomposed into several primitive transformations that can be executed.

A transformation, plus its inputs and outputs, corresponds to a model in the geographic world. Primitive transformations are models that have been implemented and can be executed. Composite transformations corresponds to higher level models that are executable by executing each of their submodels. Using this approach, GOOSE can store the structure of a complex geographic model by mapping transformations to *projects*.

Projects are stored in the project base as a collection of input, output and transformation attributes. The input and output attributes contain the identifiers of the objects involved in the transformation. Thus, when a project is accessed, the objects are accessed through two levels of indirection, one to fetch the object identifier from the project data structure, and a second to access the dataset through the data attributes of the object. The transformation attribute specifies the transformation applied to the input objects to get the output objects. If it is a primitive transformation then GOOSE can call the underlying system to execute it using the operations log and the interpreter. If it is a composite transformation, the attribute contains references to the corresponding subprojects (where eventually the primitive

OBJECT NAME: Rainfall Records
OBJECT ID: system generated id
DATA:
    Access Method: UNIX file system
    Access Mask: 'cat rain_data_1990.*'
METADATA:
    Sources: primitive
    Versions: none
    Usage: Precipitation Data

Figure 6: Description of an object within GOOSE

functions that can be executed can be found).

The project level allows to structure and register the modeling process by identifying each model with a project. When models are combined, as in the example, to form higher level models, function composition is used to create higher level projects. The notion of projects allows GOOSE to reproduce the execution of the entire model, repeat the execution step by step to examine intermediate results, replace submodels by new submodels, produce new version of the results when new version of the inputs are available, etc. Projects capture the modeling activities and the cooperation inherent to it (models developed concurrently and combined to form higher level models) so the system can record these activities and reproduce them at will.

4.4. Operations Log

The transformation attribute of a project indicates how the output objects were generated from the input objects. If it refers to a primitive function, GOOSE needs to be able to execute it. Since GOOSE relies on the underlying system to perform the execution of transformations, and this may require several steps, a mapping must be done from the transformations specified in the project to the actual sequence of commands to be executed by the underlying system. Note that the interpreter acts as the interface, however, to place the burden of translating every transformation into the proper sequence of commands on the interface would make it unnecessarily complex and affect its generality. Instead, transformations are seen as "script" names. These scripts, or sequence of commands to send to the interpreter, are stored in the *operations log*. Similarly, when the user defines a project using composite functions, the operations log is used to order the execution of each subproject.

The combination of the project base and the operations log corresponds to the implementation of the project level. The project base stores the references to the project and its different components (inputs, outputs, transformations). The operations log provides the means for accessing the projects and executing them. Figure 5 summarizes the data structures described so far.

PROJECT NAME: Erosion Model
PROJECT ID: system generated id
INPUT: Slope Length
Slope Angle
Soil Map
Vegetation Cover
OUTPUT: Soil Erosion
TRANSFORMATION: Erosion Model (primitive)

TRANSFORMATION NAME: Erosion Model
TRANSFORMATION ID: system generated id
CODE: primitive

Figure 7: Description of an project within GOOSE

5. Cooperative Modeling

To illustrate the features of GOOSE, in this section we discuss how the example presented in Section 2 is represented within the system.

5.1. Object Level

Each dataset created by the user either as a result of a transformation or entered as "raw" data, is represented in GOOSE as an object. Note, however, that there may be several objects that refer to the same dataset and objects that refer to several datasets. For the example at hand, each dataset corresponds to an object. For instance, the rainfall records, stored as a set of files is represented within GOOSE as shown in Figure 6.

When an object is created, the user has to give it a name for future reference, although the system uses an internal identifier for the object. The user must also specify the access method and the access mask. The rest of the attributes are dynamically maintained by GOOSE. If the object is the result of a transformation, the user needs only to give it a name since GOOSE will record the way it was created and the access method and mask. The information stored in the attributes is used to support multiple views of the same dataset and to have a complete description of the object (its lineage, version, uses, dependencies, etc.). Note also that the object level is functionally complete. The project level does not add any extra functionality *per se*, only a better interface and a clearer presentation of the information stored in GOOSE. To illustrate this point, assume the user needs to trace the lineage of the vegetation change dataset in the example. This can be done entirely within the object level, however, for this particular object, the lineage comprises twelve objects and seven transformations. This sort of output would make the analysis and interpretation very tedious. It would be much better to provide a high level description of the lineage and go into details only upon request from the user. This is exactly what the project level does. The object level is used internally. The attributes of the objects are manipulated by algorithms that maintain the proper cross references between objects and provide information to the project level. From the user point of view, objects contain too detailed and extensive information. An object can be seen as a sophisticated "reference card" that points to where the data can be found and how the data is being used. This information is crucial to keep track of the activities of other users in a cooperative environment.

5.2. Projects and Transformations

Projects are used to allow the user to access the information within GOOSE in a way that mirrors the modeling activities. Projects can be *primitive* or *composite* depending of the type of transformation they contain. In the example, every model is represented as a project, and we assume that every step considered is a primitive function, therefore, the representation will be done using primitive projects. For instance, the erosion model is represented within GOOSE as shown in Figure 7.

For efficiency reasons, the input and outputs are actually encoded using the system generated object identifiers. The same applies to the transformations. Transformations are stored in a similar fashion (Figure 7), the code section is the sequence of commands to send to the underlying system (name of a UNIX shell script or a C program, for instance). Once each individual primitive model has been defined in GOOSE, they can be combined into higher level projects. This is one of the major features of GOOSE and what makes it a suitable tool for modeling. For instance, the whole example can be represented as shown in Figure 8.

In this way, the entire example can be run by just specifying the project "Example". The exact representation of the transformation "T-Example" depends on how the project has been defined. If it has been defined as a combination of primitive models then the operations log will contain the references to these primitive projects and the order of execution. However, there are better ways to structure the example. Since it produces three different results, we could divide the example into three different projects that are then combined into a bigger project (the "Example" project). This is shown in figures 9 and 10.

Note that the projects in Figure 9 are overlapping, in the sense that they share many intermediate results. A further and perhaps more useful subdivision of the example could be as shown in Figure 10.

There are many possible combinations. It is up to the user which ones to choose. The idea is to allow the user to structure large conceptual models in sets of submodels that can be worked out separatedly. This is one place where GOOSE offers

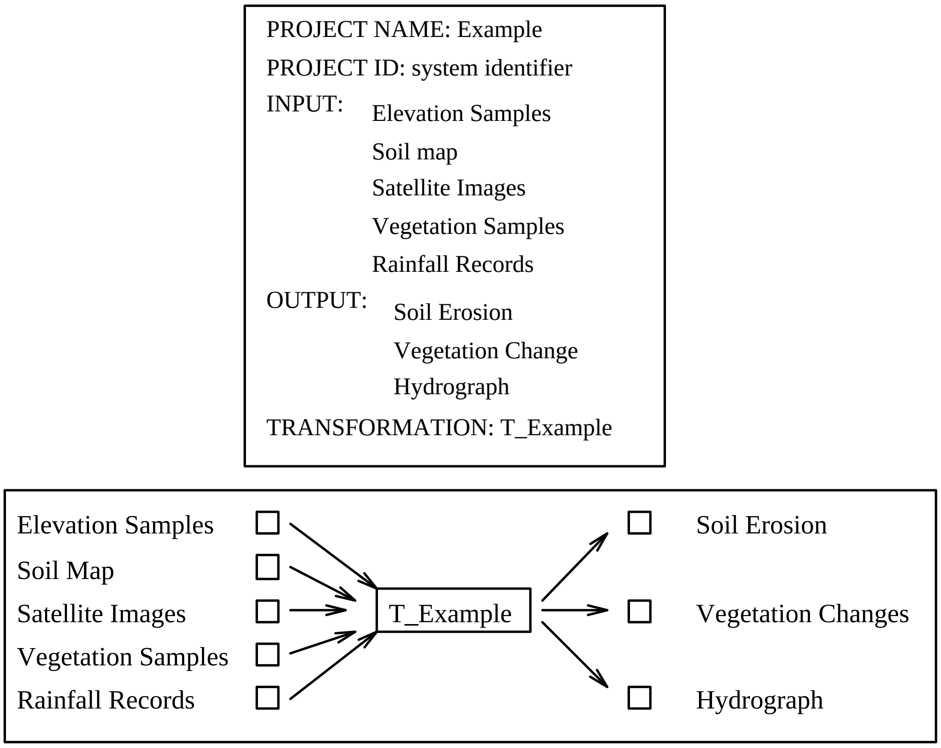

Figure 8: Description of the example of Figure 1 within GOOSE

help for cooperative modeling. Projects offer a similar function as that of CASE environments: structured modeling and controlled interaction among the different agents. Once the overall project and its subdivisions have been defined, users can work on the subprojects in a coordinated fashion. This is where the metadata attributes on the objects become useful. Cooperative work in this environment means to establish a common model in which all users can access the different submodels. A global view as well as a restricted set of partial views are maintained. Data must be interpreted both within the global model and within each submodel, and the interpretation must be accessible to all users of the system.

For instance, a user may come across an object "Soil-Erosion", of which s/he knows nothing. The first step is to query GOOSE to find the lineage of the object. By default, GOOSE replies with the highest level description available. This is shown at the top of Figure 11. At this point, the user may know the research project "Example" and that is perhaps enough information. If it is not, GOOSE can be requested to provide more and more detailed information. The lower part of Figure 11 shows a more detailed description of the lineage of the object.

Similar mechanisms are used to manipulate projects. For instance, once the

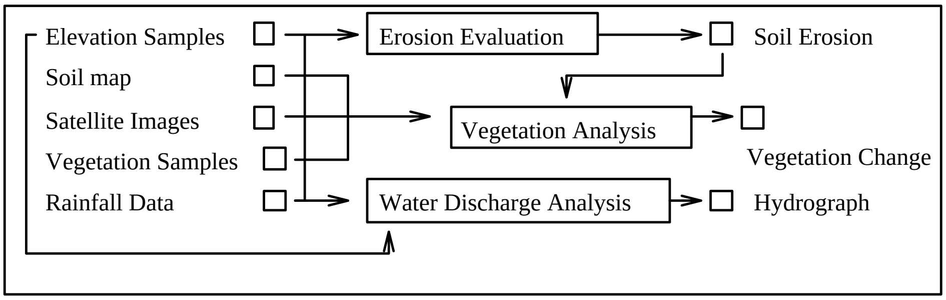

Figure 9: The example of Figure 1 divided into three projects

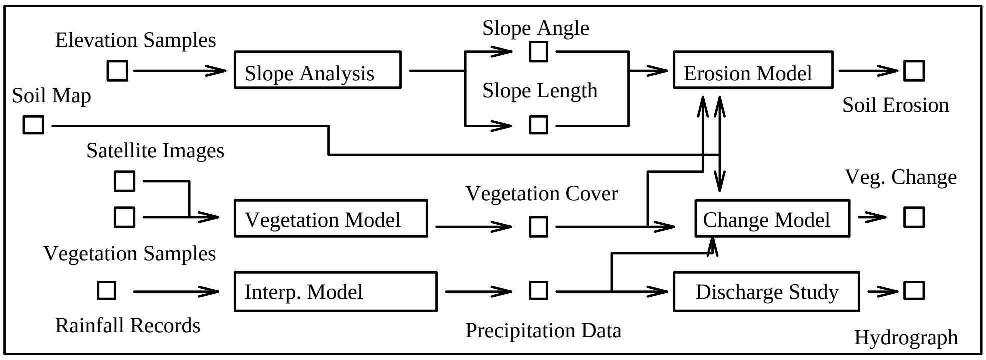

Figure 10: A further subdivision of the example of Figure 1 into projects

entire "Example" project has been completed, it is possible to apply it to different inputs. Consider, for instance, that a more accurate soil map becomes available. The user only needs to specify the new input to the project "Example" to get an updated version of the output. Since lazy evaluation is used, only the results that need to be reevaluated are modified. This change propagation mechanism can be executed at different levels, the user need only to specify which projects are to be used.

When change propagation takes place, new versions of existing objects are created. GOOSE performs version control automatically, storing the necessary information in the object attributes. For instance, the new version of the soil map creates a new version of the soil-erosion result. Figure 12 shows how this is represented within GOOSE.

If the user so wishes, the mere creation of a new version of the inputs will result in an automatic rerun of the project creating the corresponding new versions of the outputs. Users can query GOOSE for the version information regarding an object.

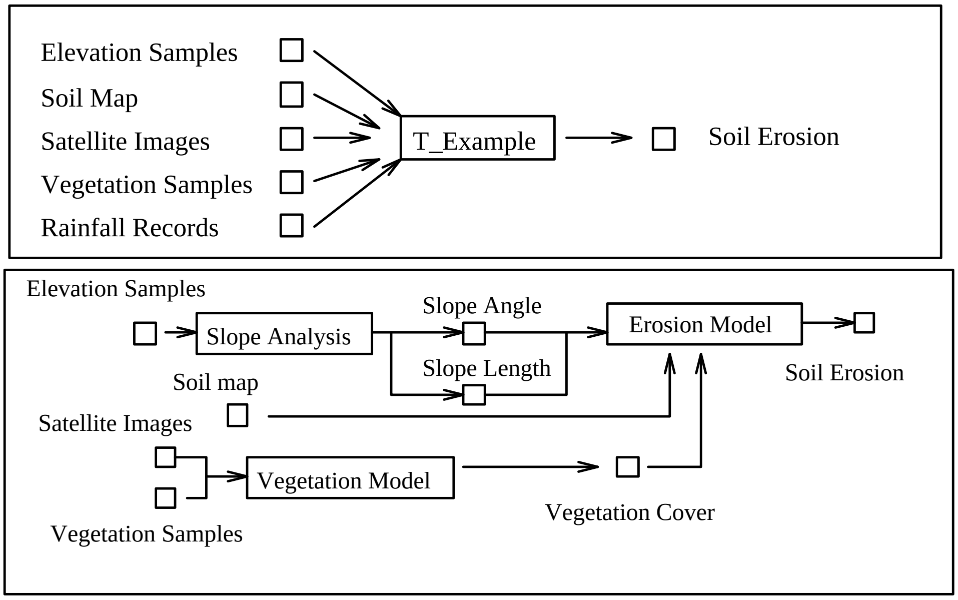

Figure 11: Lineage of Soil-Erosion, at two levels of resolution

GOOSE responds with the partial order formed by the different versions. Then lineage queries can be executed on each particular version.

These mechanisms are intended to enhance concurrent work on large modeling efforts. By restricting the change propagation to subprojects, groups of users can work on subprojects until they are completed. Then the final results are incorporated into the overall project. Any user interested in the overall project can treat it as such without having to deal with the details of every intermediate step. But it is also possible to execute a global project step by step, which allows the analysis of intermediate results.

6. Related Work

A rough taxonomy of the approaches chosen when designing GIS shows three major trends: GIS as databases, GIS as catalogs and GIS as development tools. In the first case, database-like functionality can be achieved by restricting the role of the GIS to that of a customized display tool. This approach restricts the amount of data manipulation that can take place and there is no provision to support large models that may involve several interrelated datasets. An examples of this approach is GEO++ [vOV91, dHvO92]. When the large volumes of existing geographic data are considered, it becomes clear that some means are necessary to locate relevant data. Some GIS concentrate on the task of allowing users to navigate through different data repositories. Most of these system assist only in locating the data

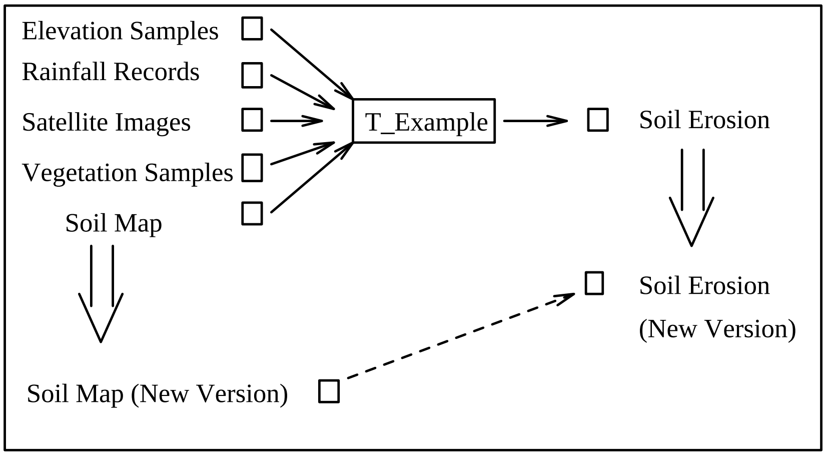

Figure 12: Automatic propagation of version changes

and are therefore incomplete for modeling purposes. An instance of catalog GIS is HEM [Rad91]. Finally, when the GIS focuses on the processing of data information, it becomes a development tool. They can be more or less complete, but their architecture reflects the fact that the data is used as input to a complex set of transformations. REINAS [Lea92] and VIMSYS [GWJ91] are examples of these development tools. GOOSE is related to this third approach, development tools, since we believe it is the most complete. However, it is difficult to compare GOOSE with existing systems since it is not a GIS in itself.

Research on GOOSE is part of a larger project in GIS currently being developed at UCSB [SSAE93]. Many of the concepts it implements are common to advanced database applications, especially design databases where ideas such as change propagation were first proposed [BK85, CK86]. The theoretical background is related to transaction processing in that the data interdependencies created by the application of transformations are seen as a set of interactions between transactions, although this work is beyond the scope of this paper, most of the ideas on metadata originate from it [AE92]. The issue of cooperation between different agents appeared in the early stages of design, when it became clear that the complexity of the geographic models treated required the participation of many users. The notion of data attributes of an object are related to ideas borrowed from traditional database design and view management techniques. The metadata attributes were motivated by the need to avoid the problem of uninterpreted legacy data, so common in geographic applications [Arm88, Xia89, Rad91, Cea93], and the need to provide some mechanisms to record the activities of multiple users collaborating in the same model. The actual implementation and semantics of the metadata are strongly related to research in modeling work environments [BM85, Kun88, MC90, Ham90] and systems modeling in general [Hir81, Che85, Gre91, KRW93]. The idea of projects has its origins in concepts of structured programming techniques, since most of the modeling efforts in geographic environments can be seen as large programming efforts. However, the idea of projects can be extended to incorporate many of the concepts of computer supported cooperative work [Kun88, Gre91].

7. Conclusions

GOOSE is a high level tool that allows existing GIS to incorporate modeling, metadata management and cooperative work capabilities. We have discussed its architecture and with an example, shown how it can be used in geographic environments.

The most relevant feature of the system is the ability to provide multiple views of the datasets and models. While view management for data is a known issue within the database community, view management of models is a novel approach that deserves more attention. Nested models that can be combined using the rules of functional composition are a powerful tool for representing research activities. As we have shown with the example, the user is provided with tools to represent the model and reproduce it at different levels of resolution. The system takes care of version control and information updates as the models and objects are manipulated.

Cooperation through time is also supported by GOOSE. In an environment where complex models are built and used may be years later, it is crucial to provide means to interpret the data and reproduce the model from the information stored in the system. The metadata attributes of the objects provide the necessary information to reproduce the lineage of a dataset and retrace the steps of every model without user intervention. This feature will help to alleviate the problem of legacy data.

Furthermore, GOOSE allows structured modeling, a key component in a system that has to support multiple users cooperating in the design of large geographic models. The ability to support multiple views also allows GOOSE to be used in parallel by users working on different parts of a project or on totally unrelated tasks. GOOSE is currently being implemented as a prototype working on top of the UNIX file system. As part of a GIS project, it will be incorporated eventually in a complete system. We are also considering the possibility of installing GOOSE on top of several public domain GIS.

Acknowledgements

We would like to thank Terry Smith, Jianwen Su and the entire MDBL/MDBS research group for valuable discussions related to modeling and GIS in general.

References

[AE92] G. Alonso and A. El Abbadi. A Graph based Model for Concurrency and Version Control in GIS databases. *Technical Report, University Of California at Santa Barbara, Department of Computer Science, TRCS92-17*, 92(17), July 1992.

[AE93] G. Alonso and A. El Abbadi. GOOSE: Geographic Object Oriented Support Environment. *Proceedings of the ACM Workshop on Advances* *in Geographic Information Systems, Arlington, Virginia, USA*, pages 38–49, November 1993.

[Arm88] M.P. Armstrong. Temporality in spatial databases. pages 880–889, November 1988. Proceedings GIS/LIS.

[BCL86] J.A. Blakeley, N. Coburn, and P.A. Larson. Updating Derived Relations: Detecting Irrelevant and Autonomously Computable Updates. *VLDB*, pages 457–466, August 1986. Proceedings of the 12th International Conference on Very Large Databases, Kyoto.

[BHG87] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987.

[BK85] D.S. Batory and W. Kim. Modeling Concepts for VLSI CAD Objects. *ACM Transactions on Database Systems*, 10(3):322–346, September 1985.

[BLT86] J.A. Blakeley, P.A. Larson, and F.W. Tompa. Efficiently Updating Materialized Views. pages 61–71, May 1986. Proc. of the SIGMOD International Conference on Management of Data, Washington, D.C.

[BM85] J.H. Bair and L. Mancuso. *The Office System Cycle.* Hewlett-Packard, Palo Alto, California, 1985.

[Cea93] D.D. Cowan and et al. Managing Environmental Data - An Extension to the GIS Architecture. *Proceedings of the ACM Workshop on Advances in Geographic Information Systems, Arlington, Virginia, USA*, pages 22–27, November 1993.

[Che85] P. Checkland. *Systmes Thinking, Systems Practice.* J. Wiley & Sons, Chichester, 1985.

[CK86] H. Chou and W. Kim. A Unifying Framework for Version Control in a CAD Environment. *VLDB*, pages 336–344, August 1986. Proceedings of the 12th International Conference on Very Large Databases, Kyoto.

[dHvO92] S. de Hoop and P. van Oosterom. Storage and Manipulation of Topology in Postgres. *Proceedings of EGIS'92, Munich, Germany*, pages 1324–1336, March 1992.

[GM93] N.D. Gherson and C.G. Miller. Dealing with the data deluge. *IEEE Spectrum*, 30(7):28–32, July 1993.

[Gre91] S. Greenberg. An annotated bibliography of computer supported cooperative work. *ACM SIGCHI Bulletin*, pages 29–62, July 1991.

[GWJ91] A. Gupta, T.E. Weymouth, and R. Jain. An Extended Object-Oriented Data Model for Large Image Basis. *Proceedings of the Second Symposium on Advances in Spatial Databases, Zurich, Switzerland*, pages 45–61, August 1991.

[Ham90] M. Hammer. Re-engineering work: don't automate, obliterate. *Harvard Business Review*, 4:104–112, July 1990.

[Han87] E.N. Hanson. A Performance Analysis of View Materialization Strategies. *Proc. of the SIGMOD International Conference on Management of Data, San Francisco, CA*, pages 440–453, May 1987.

[Hir81] R.A. Hirschheim. *Office Automation a Social and Organizational Perspective*. J. Wiley & Sons, Chichester, 1981.

[Kea93] R.H. Katz and et al. Design of a Large Object Server Supporting Earth System Science Researchers. *AAAS Workshop on Adavances in data Management for the Scientist and Engineer, Boston, Massachussetts, USA*, pages 77–83, February 1993.

[KRW93] T. Kampke, F.J. Radermacher, and P. Wolf. Supporting preference elicitation: the FAW preference elicitation tool. *Decision Support Systems*, 9(4):381–391, June 1993.

[Kun88] A.T. Kundig. Future Computer and Communication Supported Working Environments. *R. Speth (ed.) Springer Verlag, Brussels 1988*, 1988.

[Lan88] G. Langram. Temporal GIS design tradeoffs. pages 890–899, November 1988. Proceedings GIS/LIS.

[Lea92] D.D.E. Long and et al. REINAS: Real Time Environmental Information Network and Analysis System: Concept Statement. *Technical Report, Baskin Center for Computer and Information Sciences, University of California at Santa Cruz, UCSC-CRL-93-05*, January 1992.

[LHM86] B. Lindsay, L. Haas, and C. Mohan. A snapshot Differential Refresh Algorithm. *Proc. of the SIGMOD International Conference on Management of Data, Washington, D.C.*, pages 53–60, May 1986.

[LV90] D.P. Lanter and H. Veregin. A Lineage Meta-Database program for propagating error in Geographic Information Systems. *Proceedings GIS/LIS*, pages 144–153, November 1990.

[MC90] T. Malone and K. Crowston. What is coordination theory and how can it help design cooperative work systems? *Proceedings of the Iternational Conference on Computer Supported Cooperative Work*, pages 357–370, 1990.

[Moe88] H. Moellering. Fundamental concepts that form the basis of the proposed american cartographic data exchange standard. *Cartography*, 17(2):9–14, December 1988.

[Mor88] J.L. Morrison. Digital Cartographic Data Standards: the U.S. experience. *Cartography*, 17(2):1–7, December 1988.

[Rad91] F.J. Radermacher. The Importance of Metaknowledge for Environmental Information Systems. *Proceedings of the 2nd Symposium on the Design and Implementation of Large Spatial Databases, Springer Verlag*, 1(1):35–44, August 1991.

[SDT92] SDTS. Spatial Data Transfer Standard. *Spatial Data Transfer Standard proposal by the SWGFICCDC and CDSTF*, 1992.

[SS93] T.R. Smith and J.W. Su. Modeling and Database Systems for Sientific Applications. *Proceedings of the Second Intl. Conf. on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado, USA*, September 1993.

[SSAE93] T.R. Smith, J. Su, D. Agrawal, and A. El Abbadi. Database and Modeling Systems for the Earth Sciences. *IEEE Bulletin of the Technical Committee on Data Engineering*, 16(1):33–37, March 1993.

[Tob79] W.R. Tobler. A transformational view of cartography. *The American Cartographer*, 6(2):101–106, 1979.

[vOV91] P. van Oosterom and T. Vijlbrief. Building a GIS on top of the open DBMS Postgres. *Proceedings of EGIS'91, Brussels, Belgium*, pages 775–787, April 1991.

[Xia89] Q. Xiao. A temporal/spatial database structure for remotely sensed image data management within GIS. pages 116–122, November 1989. Proceedings GIS/LIS.

[ZG91] Q. Zhou and B.J. Garner. On the integration of GIS and Remotely sensed data: towards an integrated system to handle the large volume of spatial data. *Proceedings of the Second Symposium on Advances in Spatial Databases, Zurich, Switzerland*, pages 63–72, August 1991.