Integrating dynamic environmental models in GIS: the development of a prototype dynamic simulation language

WILLEM VAN DEURSEN
Resource Analysis, Zuiderstraat 110, 2611 SJ Delft, The Netherlands

CEES WESSELING
Department of Physical Geography, Utrecht University, PO Box 80.115, 3508 TC Utrecht, The Netherlands

Abstract A modelling language is presented to run dynamic models in a GIS environment. The focus of the language is on environmental applications. The approach taken was to design and build a GIS which includes a modelling language for the formulation of the dynamic behaviour of environmental systems. The spatial modelling language allows for a component-based modelling approach without the burden of technical implementation of database and algorithm details. Generic components, representing storages and fluxes of mass or energy, can be linked to create hydrological and environmental models.

INTRODUCTION

Current GIS capabilities have become an indispensable tool for feeding spatial models and analysing their results. Especially, the capability to derive spatial attributes from multiple sources, such as remote sensing, sampling, interpolation and digitizing existing maps, and to store these attributes in a geographic database simplifies the collection of input for a model. Analysing model results is helped by the possibility of simultaneous graphic display of multiple spatial attributes.

There are a few approaches to link models to GIS. Relatively simple, static, models can be run in the GIS itself, if all the model operations are part of the GIS functionality. From the database, new data is derived which is added to the database. Dynamic models, with parameters and variables changing over time, are more cumbersome to run in a GIS, because current GISs are focused on querying and maintaining a static database with static phenomena. Current GISs do not explicitly allow for dynamic phenomena to be stored and analysed. A possible solution for the lack of dynamic functionality is to program the dynamic behaviour in scripts or macros, but current practice is to program dynamic models as separate programs and to exchange data between the model and the GIS.

True integration, where a user-defined dynamic model runs entirely in the framework provided by the GIS, is not yet possible. For such an integration a GIS must support all the operations that constitute a dynamic environmental model and have all these operations available in a single modelling language. In this paper we present such a modelling language called DYNAMITE, the associated analytical engine and the spatial database that are all part of the new PCRaster version (Van Deursen, 1995; Van Deursen & Wesseling, 1995).
REQUIREMENTS AND DESIGN ISSUES

Every model is written in some type of language. Whether a model is a FORTRAN program, a sequence of GIS macros or a formula in a spreadsheet, they are all descriptions of a model which is materialized in a particular programming language. Even graphically designed flow charts in which (spatial) operations are chained, (e.g. ERDAS Spatial Modeller or STELLA flow charts) are graphical representations of a programming language. Some of these programming languages are general purpose programming languages while others are specially designed for spatial analysis.

A well-designed programming language is essential in every area of software engineering. Our goal is to develop a special purpose programming language: a modelling language for dynamic models in a GIS environment. The data manipulated by the language is stored in the GIS database. Therefore language design and GIS data model design must be consistent with each other.

Environmental models cover a large variety of applications, ranging from simple erosion risk assessment descriptions based on generalized slope length and soil type characteristics to complex pollution models containing diffusion and advection processes. Thus the language must contain a large set of operators, from which many types of models can be built. However, the set of operators should be chosen carefully, based on a number of conflicting requisites. The set should be as small as possible to keep the language easy to learn and use, but large enough to build all types of dynamic models. The expressive power of an individual operator should enable one to express models as compactly as possible without risking the obscurity which often accompanies compactness. These requisites mainly deal with the needed level of abstraction that hides irrelevant details from the user.

After discussing the structure of the database, this paper will show how the same GIS language is utilized in examples ranging from ad hoc analysis to dynamic models.

THE DATABASE

Computer languages are constructed around data types and data models. The language is the means by which the information in the data can be analysed. The question of how to store data seems inevitable when discussing GIS and data models, as can be concluded from the everlasting raster versus vector discussion. More important however, is the question what is stored in the database, and what the characteristics of the entities stored are.

For a model, the important entities may include soil type maps and land use maps. A cross-tabulation of these entities is possible whether they are stored as rasters or polygons, but a multiplication of soil class numbers with land use numbers will not produce anything useful since both entities are nominal data types. Even a comparison (equality) would be nonsense since the nominal data types represent different entities. To overcome these difficulties, and for a functional support of the operations, type information can be included in the database, and a strong type checking mechanism could be applied to the GIS operations.

The PCRaster database can only hold raster maps and point data. Both data representations have types attached. These types do not only describe the data represent-
Integrating dynamic environmental models in GIS

Integration but also the special properties of the entities modelled. We distinguish the following data types relevant for environmental modelling:

- Boolean;
- nominal, sub-typed by its legend;
- ordinal, sub-typed by its legend;
- scalar field;
- vector field;
- local drain direction (ldd).

Scalar fields are used to describe intensities and potentials of physical fields, such as temperature, population density and precipitation. Vector fields have both a magnitude and direction and can be used to represent (horizontal) fluxes and forces, such as movement of ground water and air.

The local drain direction (ldd) data type is introduced to allow for the definition of the direction of potential flow. It is a data type that maps the major direction of flow of mass for all locations in the database, and can be used in operators that describe lateral flow. In general, this ldd-map is derived from the Digital Elevation Map, and represents the direction of surface flow through this elevation map, but any scalar field can be processed to determine a local drain direction map. For a further discussion on the construction of this data type see Marks et al. (1984), Morris & Heerdegen (1988), Jenson & Domingue (1988), Moore et al. (1993) and Van Deursen (1995).

Operations on the database entities can only be applied if the maps have the right type for the operator. Such a strong type checking scheme assists error detection and helps the user in conceptualizing his ideas about the entities used. An additional advantage of typing the entities is that more intelligence, such as polymorphic behaviour, can be built into the operators. For example, an interpolation operator can automatically choose the nearest neighbour algorithm for ordinal data and bi-linear interpolation for scalar data.

The dynamic behaviour of the database, needed for dynamic modelling, is modelled by time series indexed on time and location and stacks of map layers representing the status of the model at different time steps.

THE SPATIAL MODELLING LANGUAGE

The DYNAMITE language is an extension of the concept of the Map Algebra and Cartographic Modelling Language proposed by Tomlin and Berry (Berry, 1987; Tomlin, 1990). It follows the same approach in the sense that it provides a limited set of generic functions, which can be used as primitives for the models. However, the language appearances are opposites. Tomlin proposes a natural language that is understandable for a large group of users with no former experience in computer programming. In our case, we think the majority of users consists of the developers of dynamic models, who are expected to be familiar with compact mathematical notations. Therefore, the syntax of the language is based on mathematical equations where each equation assigns the value of an expression to a single output.

As an example, the equation for determining excess rainfall and abstractions from storm rainfall known as the Curve Number approach (SCS, 1972) reads:

\[ Pe = \frac{(P - 0.2S)^2}{(P + 0.8S)} \]
with \( Pe = \) excess precipitation or direct runoff (inches); \( P = \) precipitation (inches); and \( S = \) potential maximum retention (inches).

This equation can be applied to find excess precipitation from a 5.0 inch rainstorm on the raster map \( RetentionMap \) using the DYNAMITE command:

\[
Pe\_map = \frac{sqr(5.0 - 0.2 \times RetentionMap)}{(5.0 + 0.8 \times RetentionMap)}
\]

which yields a map \( Pe\_map \) with the excess precipitation distribution.

The command above is a point operation, where a new value for each location, a grid cell, is derived from different attribute values on that same location. Non-point operations, where a new value for each location is derived from attribute values on (possible) different locations, are also modelled as mathematical functions.

Additional to the traditionally available functions for buffer zone creation and the moving windows functions, DYNAMITE incorporates several functions to determine lateral transport over the ldds map. The most simple functions allow for the instantaneous transportation of all the surface water to be accumulated at the outlet point of the catchment (accu-function). More advanced functions allow for the definition of losses and maximum transport capacities along the drainage pattern, including the most general functions for lateral transport: the route functions. They simulate flow through a network, starting with an initial distribution in the network and returning the new distribution (state) and the conveyance of material through the network (flux). The amount that is conveyed is determined by a flux equation. The syntax of these functions is:

\[
ResultMap = routestate(LddMap, FluxEquation, InitialMap)
\]

and

\[
FlowMap = routeflux(LddMap, FluxEquation, InitialMap)
\]

Since both functions operate on the same input maps, they can be combined into one statement:

\[
StateMap, FlowMap = routestate, routeflux(LddMap, S^0.3, S = IniMap)
\]

Any legal operation resulting in a scalar map can be used to describe the flux equations, and the route functions will evaluate the distribution of the material based on the initial distribution and these transport equations.

The previous examples show a single step in the analysis. Entering one or more of these statements on the command line is usually sufficient for answering ad hoc queries. Dynamic models are usually more complex, and are programmed by writing scripts containing series of statements. In addition to the statements used in static scripts and simple one-step analysis, dynamic modelling requires functionality to create and access time series and to define blocks of statements that should be executed iteratively. The time input functions allow for accessing the appropriate data in the input time series database for each individual time step, while the report function allows for the creation and updating of the resultant time series. The language has no explicit structures for iteration, but the script includes a timer section which defines model start time, end time and time step. In the script specialized sections (the dynamic sections) defining the iterative behaviour of the model are controlled by the definition of the timer. The next example shows a simplified surface runoff model.
# timer: start end increment
timer 1 100 1;

initial
# initial status of surface water
SurfaceWater = InitSurfaceWaterMap;
# coverage of meteorological station for the whole area
RainId = RainStationsAreaMap;

dynamic
# add rainfall to surface water
SurfaceWater += timeinputscalar (RainTable, RainId);
# distribute surface water according to drainage pattern
# and return both the state and flux
SurfaceWater, Runoff = routestate, routeflux (Ldd, S^0.3, S = SurfaceWater);
# output runoff at each time step for
# selected locations
report SampleTable = timeoutput (SamplePlaces, Runoff);

The initial section is executed once to initialize the model run. The dynamic section
is run repeatedly for the number of time steps stated in the timer section. In this
example, rainfall is the dynamic input, read from the table RainTable, which lists the
precipitation measured at several meteorological stations. RainId does not represent the
location of the stations, but the area of pixels for which the measurement at that station
is the best estimation of the actual precipitation at each pixel. The RainId map denotes
for each pixel the column number in the time table. At each time step, timeinputscalar
reads the row associated with the current time step from the time series file RainTable,
and returns a map containing the column values as defined by RainId. The last statement
of the example creates a time table sampling certain locations in the Runoff map. These
locations are identified by the SamplePlaces map. Each non zero value in the
SamplePlaces map is responsible for creating and maintaining a column in the time table
SampleTable. Note that the current time step is an implicit argument to all dynamic
functions, such as timeinputscalar and timeoutput.

In the dynamic section, the route functions are used to define lateral flow of
SurfaceWater through the area. At each step the flux function is evaluated that uses the
SurfaceWater values computed in this time step as initial state. For each time step, first
the new amount of SurfaceWater is computed by adding rainfall to SurfaceWater, and
with this amount of SurfaceWater distribution of the water within the flow network is
determined.

COMPONENT-BASED MODELLING

When multiple layers where material can be stored are involved, the sequential approach
of the dynamic section becomes cumbersome. A better structure for these models is to
define components, link them together and describe the characteristics of each individual
component that make up a dynamic system separately. Therefore DYNAMITE offers
a component-based syntax to define the components, storages and their connections,
called transports. The spatial modelling language that results is based on the systems dynamics approach of Forrester (1968) and is a spatial extension of an approach similar to the STELLA modelling environment.

In the next example two storages, Surface and Groundwater are defined. These storages are linked through a connecting transport Infiltration. SurfaceWater is fed by Rainfall, which is also modelled as a transport, connecting an unlimited supply with the storage SurfaceWater. SurfaceWater is redistributed through the network using the approach described above. The component-based script for this model reads:

```
timer 1 100 1;

initial
  InfiltrationRate = lookupscalar (InfiltRateTable, SoilTypes);

dynamic
  report SurfaceSample = timeoutput (SamplePlaces, Surface);
  report GroundWaterSample = timeoutput (SamplePlaces, GroundWater);

storage Surface:
  initial Surface = InitSurface;
  routing route (Ldd, S^0.3, S = Surface);

storage Groundwater:
  initial Groundwater = InitGroundWater;

transport Rain to Surface:
  Rain = timeinputscalar (RainTable, RainId);

transport Infiltration from Surface to Groundwater:
  Infiltration = InfiltrationRate * Surface;
```

Each component definition encapsulates the relevant model code for that component. A storage definition consists of an initialization and an optional routing function. A transport can be a function of any component in the model and can be used for input (e.g. the definition of the transport Rain) and output of an open system. Note that the component-based description of the model does not define the sequence of execution. The execution order is determined by the interdependencies of the storages and transports.

The component-based approach offers a structured method to develop environmental models from scratch, test them, and modify them if necessary. The approach yields a structured, high level simulation language that can be used to implement environmental models in GIS without having to code database access and numerical algorithms to solve process equations.

**DISCUSSION AND CONCLUSIONS**

In this paper we have presented DYNAMITE, a modelling language for dynamic modelling in a GIS environment. Key features are the ability to process dynamic data and a strong type checking mechanism that reveal possible errors in the model.
Integrating dynamic environmental models in GIS

It is important to mention that the language is not a macro language. A DYNAMITE script is not a list of actions that is sent to separate autonomous modules of the GIS. Instead, a single program (CALC) reads a script entirely, checks it for errors and then executes it. This approach yields better error detection facilities and faster execution of the model than macro languages. A disadvantage, compared to macro languages, is that it does not allow a user to add lacking functionality. Therefore, provisions must be made that enable a user to extend the set of functions recognized by the CALC program.

The language discussed here has been developed at the University of Utrecht over a period of five years. In this period, early prototypes of the language were used for numerous environmental models. Published examples are LISEM, a physically-based hydrological and soil erosion model on catchment scale (De Roo et al., 1994), RHINEFLOW, a water balance model for the river Rhine (Van Deursen & Kwadijk, 1993) and Calluna, an ecological model for heath land dynamics (Van Deursen & Heil, 1993). Most of the current applications put a strong emphasis on environmental modelling and especially hydrological surface routing. Therefore, the evolution of the language has resulted in a rich set of global functions involving drainage networks. Demonstrations of the use of vector fields and associated processes, such as diffusion and advection, are not yet available. Further research and development must prove if DYNAMITE is effective in areas such as groundwater modelling, where such processes are used extensively. A demonstration version of the software will be released in 1996 as an invitation to the GIS community for further discussion on GIS and dynamic modelling.

REFERENCES

Van Deursen, W. P. A. & Wesseling, C. G. (1995) PCRaster Software. Contact pcraster@frw.ruu.nl to get version 1 of PCRaster. Version 2, discussed in this paper, will be available at the same ftp site in 1996.