



**US Army Corps
of Engineers**

Construction Engineering
Research Laboratories

**USACERL Technical Report 98/94
July 1998**

Simulating Mobile Objects in Dynamic Processes

by
James D. Westervelt

Geographical Modeling Systems (GMSs) are computer-based, dynamic landscape simulation tools. As computing power continues to become cheaper and faster, GMSs will become increasingly important for the intelligent management of landscapes. However, a number of technical challenges must be met before GMS capabilities are widely accepted, several of which are addressed in this research. First, because landscape processes occur and are modeled at a variety of spatio-temporal scales, it is necessary to support simultaneous simulation of disparate scales. Second, current landscape decision support systems lack the ability to simulate the behavior of individuals. Simulation modeling of populations is unsuitable

for very low population densities where the location of individuals on a large, diverse, and fragmented terrain are important. Finally, in a management setting, techniques for linking distinct landscape models to run simultaneously are necessary. All of these capabilities should run on a common platform with a consistent user interface. More importantly, each submodel should relax its requirement to hold the landscape constant and instead share dynamically varying states between submodels. This effort develops fundamental approaches with prototypes for the simulation of mobile entities within dynamic landscapes.

SF 298

Foreword

This study was conducted for Headquarters, U.S. Army Corps of Engineers, Director of Research and Development (DRD), under Project 40161102AT25. “Environmental Research — Corps of Engineers”; Work Unit EN-IJ5, “Fundamentals of Dynamic Landform and Ecological Monitoring.” The technical monitor was Mr. Tom Hart.

The work was performed by James D. Westervelt of the Natural Resource Assessment and Management Division (LL-N) of the Land Management Laboratory (LL), U.S. Army Construction Engineering Research Laboratories (USACERL). The work was conducted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Regional Planning at the University of Illinois, Urbana-Champaign. Dr. William D. Severinghaus is Operations Chief, CECER-LL. The USACERL technical editor was Gloria J. Wienke, Technical Information Team.

This effort drew from the energies, emotions, and hard work of many individuals. The author wishes to thank his wife, Eileen, and children, Marla and Emily, for permitting him many evening, vacation, and holiday hours and days to work on this project, and for cheering him on through the finish. A specific “thank you” to Eileen for providing one of the final edits. His committee members, Dr. Lewis Hopkins, Dr. Douglas Johnston, Dr. Bruce Hannon, and Dr. Daniel Schneider collectively provided encouragement, insight, pointers to references, and intellectual challenges. Dr. Hopkins was exceptional in the role of major advisor. He saw the makings of a dissertation before anyone else and worked patiently in teaching the author to mold the original set of amorphous ideas into this document. His ability to work simultaneously with the details while never losing sight of the whole was very helpful, as was his consistency in vision throughout many months. Dr. Hannon offered his friendship, spent many evenings discussing the topic, and provided kind pressure to complete the work. This effort involved the kind assistance of several students: Douglas Briggs and Hsuan Chi Lu who provided some key software programming, and Shaun Levi and Steven Harper who helped lead the design and development of the tortoise habitat simulation model adopted by this effort. Finally, the author wishes to recognize that a crucial part of this work relied on the additions made to the Spatial Modeling Environment software by Dr. Thomas Maxwell. Those modifications made it possible to make the link

between the mobile entity software and a dynamic landscape. The combination of professional cooperation, assistance, and friendship from all involved is appreciated.

This report is dedicated to the memory of Dr. Louis R. Shaffer, father and Director of the U.S. Army Construction Engineering Research Laboratories, Champaign, Illinois, until his death in 1994. Without his vision, guidance, and gentle but firm encouragements, this project would not have been accomplished.

COL James A. Walter is Commander, and Dr. Michael J. O'Connor is Technical Director of USACERL.

Contents

SF 298	1
Foreword	2
List of Tables and Figures	6
1 Introduction	8
1.1 Background	8
1.2 Future Scenario	11
1.3 Why Build Ecologically-based Land Simulations?	18
1.4 Ecological Simulation Context of the Study	22
1.5 Organization of the Study	24
2 Perspectives in Ecological Modeling and Simulation	25
2.1 Underlying Theories of Ecology	25
2.2 Ecological Simulation Software	35
2.3 Summary: Ecological Modeling in Time and Space	46
3 Objectives and Framework	48
3.1 I-STEMS Components	48
3.2 Facilitating Discrete Entity Simulations	54
3.3 Linking To Cell-based Landscape Simulations	57
3.4 Summary	58
4 Design	59
4.1 Overall Design	59
4.2 Design and Implementation of General Purpose IMPORT/DOME Classes	62
4.3 Classes Supporting Mobile Entities	66
4.4 Animal-to-Population Interaction Classes	71
4.5 View Classes	74
4.6 Summary	78
5 System Demonstration	80
5.1 Common Development	81
5.2 Static Example	86
5.3 Dynamic Example	87
5.4 Reproduction Example	91
5.5 Predator/Prey Example	92

5.6 Summary	95
6 Evaluation	97
6.1 Overall Approach and Constraints	97
6.2 Variable Time	102
6.3 Variable Space	102
6.4 Scan and Evaluate Surrounding Entities	105
6.5 Interactions with Dynamic Entities	106
6.6 Create New Entities	107
6.7 Interactions with Static and Dynamic Maps	108
6.8 Recommended Improvements	109
6.9 Summary	115
7 Conclusions	116
References	118
Appendix A:GRASS Maps	128
Appendix B:Import/Dome Object Classes	135
Appendix C:Associated Mojave Desert Efforts	152
Distribution	

List of Tables and Figures

Tables

1	Hypothetically available model components	14
---	---	----

Figures

1	Hydrologic simulation model report	15
2	SME process	50
3	Overall design	59
4	Overall IMPORT/DOME design	60
5	Approach to development of demonstrations	61
6	Object classes for simulating mobile entities	63
7	Set of classes to support a mobile entity type	66
8	Animal class hierarchy schema	70
9	Interaction between animal and landscape classes	72
10	Conceptual layer interaction	75
11	Stripchart viewer example	76
12	Bar chart viewer example	77
13	Dynamic map viewer example	78
14	Vegetation submodel developed in Stella	83
15	Precipitation submodel developed in Stella	84

16	Static landscape simulation (day 48)	88
17	Sample SME simulation output frames	89
18	Sample SME simulation output frames	90
19	Dynamic landscape simulation (day 360)	91
20	Reproduction simulation (day 155)	93
21	Predator/prey simulation (day 13)	94
22	Predator/prey simulation (day 285)	95
23	Current overall design	100
24	A future overall design	100
25	The LocationManager role	103
26	Proposed network shared memory interaction	110
27	Sample entity controllers	113
28	Sample entity controllers	113

1 Introduction

1.1 Background

This study investigates new approaches in dynamic landscape simulation techniques that will be helpful in the management of landscapes such as military training lands. In particular, it develops and demonstrates fundamental techniques that allow simulation of the movement of individual entities across a static or dynamic raster (checkerboard) landscape. Raster-based landscape simulation modeling is providing a powerful tool for investigating direct and indirect impacts of land management decisions. Populations distributed across a landscape can readily be simulated with distributed differential equations that capture the dynamics of the population with respect to other populations, resources, and conditions. At very low population densities, however, the meaning of a population becomes lost.

Consider, for example, a density of one individual per hundred hectares being simulated in a landscape array of 10-meter square cells. The individuals, captured as populations in these cells, will result in populations of less than one individual. Perhaps the individuals are birds and in a particular cell a population of .15 birds lays a clutch of .743 eggs. An individual effectively exists across a number of cells. Within that area there are sites that might be either very favorable, or very unfavorable, to the success of eggs turning into hatchlings. With large populations, the distribution of individuals throughout both areas results in an overall success probability with relatively low variability. As the population drops, that variability increases and it therefore becomes important to simulate the movement and behavior of individual animals and plant populations. Various forest models have recognized the importance of modeling individuals (see section 2.3).

This study shows that, using event-driven object-oriented simulation technologies, it is now possible to capture knowledge about the life-history and behavior of individuals in simulation software. Coupled with raster-based landscape simulation modeling software, a powerful tool emerges for exploring the behavior of individuals representing threatened and endangered species, herds, people, vehicles, and collections of vehicles. Research described here focuses on fundamental issues and approaches that (1) facilitate the design and development of mobile

entities within simulated dynamic landscapes, and (2) support interaction of such entities with raster-based landscape simulations.

Over the past 2 years the author gained experience developing simulation models for Fort Irwin, CA, landscapes in conjunction with Professor Bruce Hannon (Department of Geography, University of Illinois at Urbana-Champaign) and three classes in "Advanced Ecological Modeling." Through these classes, we guided an interdisciplinary group of students and faculty in the design and development of a dynamic spatial model of a section of the Mojave Desert that focused on habitat suitability and population densities of the Desert Tortoise. The Stella* (Hannon and Ruth 1994) and Spatial Modeling Environment (SME) software (refer to section 3.1.3) were selected for development of the dynamic model. Landscape maps were prepared with the Geographic Resources Analysis Support System (GRASS) geographical information system software (see Appendix A). For these efforts, a fixed grid of cells 1 kilometer on a side was selected along with a fixed time step of 1 week. This combination ensured that tortoises were allowed to move a full kilometer in the course of a week. A number of shortcomings associated with this approach initiated the design effort of the current study.

Simulation restricted to populations. The Stella programming environment is designed to accommodate state variables, those variables that change over time based on the difference equations provided by the user. It does not easily allow for the movement of individuals. The author did coerce Stella into moving individuals around in a sample model of Sage Grouse behavior on a landscape (Westervelt et al. 1995). To manage this (1) only a single Grouse was allowed to occupy any given cell, and (2) a set of facts about the individual (i.e., existence, age, time since fertilization, and number of eggs and young) were packaged into a single state variable. In any given time-step, this information was teased out of the single variable for use by system equations. When the computations were complete, the updated information was then repackaged for storage in the state variable. The drawbacks to this approach were that simulations (1) were relatively difficult and abstract, (2) were limited with respect to the amount of information that could be stored in a single number (limited by the number of digital bits used to store numbers), and (3) were computationally inefficient.

* Stella is a desktop modeling tool that uses icons and schematics, linked with equations, as the mechanism to build models. It is a product of High Performance Systems, Inc., 400 Lyme Road, Suite 300, 2 Hanover, NH 04755, (800) 332-1202. Mention of this tool does not imply endorsement by the Department of Defense or the U.S. Army.

Forced movement of a full kilometer. In the SME environment, individual cells are treated as homogeneous systems. If an entity moves into the cell, it then has an equal potential for exiting that cell in any of the available directions (north, south, east, and west). This can lead to inaccurate results if that entity has a home range that is much smaller than the cells. For example, the author's tortoise model (Westervelt, et al. 1998) uses 1-kilometer cells, but the tortoise home ranges are 100 meters or less. The 1-kilometer resolution was chosen because that is the maximum anticipated movement of a tortoise in a 1-week time-step.

No inter-entity interactions. Because discrete entities cannot be accommodated, such entities cannot interact with each other. This includes predator-prey interactions, mating, grouping or herding, and maintaining boundaries of home ranges.

The application of modeling and simulating distinct entities on the landscape is widespread. The Endangered Species Act (ESA) established that the United States resolves to take certain steps toward saving identified species. Many threatened and endangered species (TES) on the lists reside across broad regions, but exist at such low densities that every individual within any management area becomes important. All government agencies are required to manage the landscapes within their purview with respect to the ESA. The Army and Department of Defense (DoD), as large landowners, have significant responsibilities with respect to such species because government landscapes often represent dwindling original communities and ecosystems upon which some endangered species depend.

To address the ESA requirement to increase the numbers of such animals (or plants), the scientific community seeks to identify the environmental conditions under which each species will flourish. The relationships between these conditions and the anticipated response by individuals, populations, and the species as a whole form a scientific model. Models are formulated that connect day-to-day environmental conditions with the responses of individuals, the year-to-year conditions with populations, and longer time frames with the genetic status of the species. Computer simulations play a crucial role in this process by (1) providing a framework that facilitates a complete formalization of the model, and (2) providing feedback to the modeler(s) about the logical consequences of the model. Through this process, computer simulations allow the scientific community to further perfect their understandings, which then result in more effective management of threatened and endangered species.

The research conducted here focuses on requirements for supporting the modeling and simulation of threatened or endangered species that exist at very low densities on the landscape. Such species include the Red-cockaded Woodpecker, the Desert Tortoise, and the Sage Grouse, all located on Army installations. These species are at sufficiently low densities that modeling them at the level of the individual may prove to facilitate the design and development of better species management plans. Focusing on individuals requires that the behavior of the individuals be simulated with respect to time resolutions on the order of days. The next section explores, through an imaginary future scenario, what the existence of a powerful, integrated ecological simulation software environment might mean to military installation land management. It is then followed by a more formal exploration of the reasons behind advocating the development of ecological land simulations (section 1.3).

1.2 Future Scenario

To help visualize the utility of a geographic modeling system, this section develops a hypothetical future scenario that involves a simulation challenge at a military installation. Imagine that the year is 2003. Fort Hood, TX has been challenged to expand training areas into adjacent properties. This expansion is desired to accommodate an anticipated expanded tracked vehicle training mission. The environmental office is tasked with generating several annual training scenarios and then evaluating each with respect to the direct and indirect impacts on (1) the ability to train (2) the effects on Golden-cheeked Warbler populations (3) the effects on Black-capped Vireo populations, and (4) the effects on local and regional biodiversity. This effort is part of the environmental assessment (EA) requirements. Management decides that the analysis shall be accomplished by an interdisciplinary group consisting of individuals from the environmental, training, and scheduling offices. They will have at their disposal several workstations that have recently been used to test the latest version of I-STEMS, the Integrated Spatio-Temporal Ecological Modeling System.

Days 1 and 2: Team Assembles

A high-priority meeting is held to assemble and brief the team that will be handling this assignment. They are tasked to develop a dynamic training area simulation model focused on the intended expansion area and adjacent existing Fort Hood properties. The resulting model will be used to evaluate the direct and indirect impacts of a change in mission on a new training area that has been designed for these areas. Known concerns that will need to be addressed before this area can be used for the intended training are: (1) two threatened or endangered species (2)

potentially sensitive ecosystems and habitats (3) water quality requirements for drinking water wells down-stream (4) impact on regional biodiversity initiatives, and (5) timber harvest goals. The team must provide a working simulation model and preliminary results within 20 days to support a briefing to visiting dignitaries. In addition to the workstations at each member's desk, the main server located in the environmental office is available (a \$50K machine containing 256 Mbytes of internal RAM, with four 200-MHz processors, and 20 gigabytes of on-line hard-disk). Fort Hood has been connected to the Internet since the mid 1990s and now has a 10-megabit-per-second connection to the outside world, which provides them with powerful run-time access to several supercomputer centers including the thriving National Center for Supercomputing Applications (NCSA). The team will be using the latest release of I-STEMS.

Following the initial briefing, the team meets and establishes the following subteams:

- Species-specific models
- Weather and climate
- Hydrology
- Communities and ecosystems
- Geographic Information Systems (GIS) and image processing
- Visualization and control
- Training.

Each team is tasked with identifying and evaluating local sources and available model components distributed across the network.

Day 3: Available Component Reports

The simulation team meets to brief each other on the information discovered during a day of exploration. Potential system components are presented in Table 1. All components conform to the I-STEMS standards, which allow them to be readily integrated. Team reports are as follows:

- Species-specific team: Three models of local threatened and/or endangered species are available. Population- and individual-based models are available for the Black-capped Vireo and the Golden-cheeked Warbler. The team recommends adopting the population-based model.
- Weather and climate: Weather and climate models have been identified on the network. Both are identified as standard, accepted models and model outputs.
- Hydrology: The Saghafian model (Saghafian 1993) was located in I-STEMS format. It has now been verified on a wide variety of landscapes. Also, a new

- soil compaction model conforming to I-STEMS has been located at the U.S. Army Engineer Waterways Experiment Station (USAWES) server.
- Communities and ecosystems: The standard plant succession model developed by the Army Corps of Engineers is available in Beta release 4.2 form.
 - GIS and image processing: Extensive historical and current geographical information system and imagery data exists on-site and can be adapted to I-STEMS.
 - Visualization and control: The Internet server at the University of Illinois currently offers a wide variety of visualization and control objects for I-STEMS applications. These include the traditional meters, sliders, menus, feedback panels, dials, and buttons. Several sophisticated new intelligent controllers are now also available at this site to manage tradeoff options, various optimization approaches, and collaborative modeling tools.
 - Training: Two training model sets are available. Fort Hood's training impact tables have been used quite successfully for the past decade and relate training exercises and training areas with degrees of estimated environmental damage. The U.S. Army Construction Engineering Research Laboratories' (USACERL) relatively new set of maps add a spatial dimension to these tables and provide impact information at a resolution of 30 meters. The team decides to adopt these maps and the USACERL approach to developing such maps.

Day 4: Register Available Submodels

A new model is established on the server. This process consists of setting up an information exchange server that will facilitate communications between different processes running on different machines. All participants are told to establish I-STEMS environments on their individual workstations, which attach to this server. Once this is done, all team members can readily query and view any portion of the developing model as well as establish model components on their own machine. Team members then begin to set up the submodels selected from Table 1 on the local machines. By the end of the day, each member is able to view the status of the virtual interconnections between the various submodels. For example, a query on the status of the hydrologic simulation model yields the report shown in Figure 1.

Table 1. Hypothetically available model components.

Potential Component	Source	Description	dT,dS	Inputs Required	Outputs Available
Black-capped Vireo	USACERL	Population model object developed for Ft. Hood (1998)	1 wk, 1 km	Weather, Topology Vegetation (grass, forb, shrub, tree)	Densities in 6 age classes
Black-capped Vireo	U of Texas	Individual based object developed for the State of Texas (2001)	1 day, 100 m	Density of predators Weather Topology Vegetation (5 species)	Location Health indices (5) Age, sex, etc.
Golden-cheeked Warbler	Texas A&M	Population model object developed for Ft. Hood in (1998)	1 wk, 1 km	Weather, Topology Vegetation (grass, forb, shrub, tree)	Densities in 6 age classes
Vegetation density maps	USACERL	Vegetation, grass, shrub, forb, and tree (2002)	N/A, 30 m	N/A	N/A
Vegetation succession model	Colorado State Univ.	20-species succession model (1999)	1 mo, 100 m	Soil type Soil compaction State of starting vegetation	Succession phase
Tracked-vehicle impact model	USAWES	Soil compression model (1997)	N/A, N/A	Tracked-vehicle days per ha, Soil type	Soil compression
Biodiversity model	INHS	10-keystone species model (1998)	1 yr, 10 km	Climate, % land in each of 5 succession states	Densities for each species Genetic variability for each species
Training models	USACERL/ Ft. Hood	Maps created for each exercise and training area combination (2003)	1 day, 30 m	Training exercise Training area	Average tracked-vehicle days per ha.
GIS	Ft. Hood	Extensive 100+ theme digital map database.	N/A, 5 m - 100 m	N/A	100+ themes, some historical data; extensive imagery
Hydrology	USACERL	The Saghafian finite-difference model (Saghafian, 1993)	minutes days, 30 m	Topographic data, land use and cover	Saturation depth, velocity, scouring, and deposition
Weather	National Weather Service	Historical and average weather conditions and probabilities	1 day, 100 m	Day of year	Temperature and rainfall: average, standard dev., and probability

This report begins by indicating that this submodel has been registered with a “main model” called “Ft. Hood Extension Simulation,” which is registered on the machine called env.fthood.army.mil. Connection to this model is accomplished with the code: 175 (which is a port or socket type number). This submodel has registered itself with the main model and will be running on and accessible through hydro.fthood.army.mil. Note that all submodels may run on separate machines. Underlying information brokers facilitate virtually seamless integration of these submodels. Some model metadata is also displayed. Here, that information identifies the version number of the submodel and the latest I-STEMS version under which the model is known to operate. A section on inputs and outputs provides information on how the submodel is currently linked to other submodels. These links were established using user interfaces that probe the model space for available variables and then allow the modelers to establish the desired connections. The CONVERTER column under inputs identifies which, if any, standard

MAIN MODEL INFORMATION				
Name:	Ft. Hood Extension Simulation			
Main Server:	env.fthood.army.mil			
Access Code:	175			
SUBMODEL INFORMATION				
Name:	Hydrologic Simulation			
Model Server:	hydro.fthood.army.mil			
Access Code:	180			
METADATA				
Author:	Bahram Saghafian			
Version:	4.3.1			
I-STEMS version:	2.6			
Resolution:	30 meters			
INPUTS				
NAME	UNITS	INITIATED BY	SUPPLIED BY	CONVERTER
Elevation	meters	GIS	N/A	N/A
Slope	percent	GIS	N/A	N/A
Initial saturation	mm	GIS	N/A	N/A
Soil permeability	mm/day	GIS	N/A	N/A
Manning's K	K	GIS	Vegetation Model	N/A
Water	mm/hr	N/A	Dummy	mm/inch
OUTPUTS				
NAME	UNITS	USED BY SUBMODEL		
Soil saturation	mm	Vegetation		
Water depth	mm	Vegetation		
		Golden-cheeked Warbler		
		Black-capped Vireo		
		Training		
Water velocity		Vegetation		
Soil scour/deposition	mm	Vegetation, Succession		

Figure 1. Hydrologic simulation model report.

unit converters were used to establish the connection. The OUTPUTS section

identifies the submodels that currently use the available outputs. The lists of such submodels will grow and shrink as the different components link themselves with each other.

The input “water” is identified as being supplied by submodel “Dummy.” This is a reserved submodel name that is attached to very simple data generators. Model developers are allowed to create dummy inputs defined by fixed values or graphs that use time (e.g., month) as the independent variable. The purpose is twofold. First, inputs that are not being generated by other submodels can be simply accommodated in this fashion. Second, during debugging and sensitivity analyses, input variables can be set to static values.

Each submodel can be probed in a manner that results in a report similar to the simulation report in Figure 1. Components other than submodels can be established and then viewed through this report. The most important classes are viewers and controllers. Viewers are essentially submodels that only access output from other models; they probe submodels and display information in numerous fashions. Generally this means that they provide run-time views of system states (maps, tables, strip charts, etc.) or dump data to output files for later analysis. Controllers, similarly, are basically submodels that provide input to other submodels. Based on human interactions with graphical user interfaces (GUIs), they supply values to submodels. Such inputs are injected into the associated submodels at the time they are set (typically the receiving submodel controls the data probe).

Of the numerous other interfaces available to the modelers, two require a brief mention here. A main control panel is available for starting and controlling the model, as a whole. This interface allows the user to turn any of the various submodels into ON, OFF, and STATIC modes. OFF makes the submodel appear to be nonexistent. STATIC turns the submodel off, but allows it to generate predefined static information much like the “Dummy” submodel. ON causes the submodel to operate normally during the course of a simulation run. These are used to control the view and controller components as well. The second general type of important interface is the control panel for supporting simple modifications to each of the submodels and view/controller interfaces. For example, a generic population submodel can cover a wide range of populations by simply allowing the modeler to “tweak” such attributes as growth rate, consumption rate, fecundity rate or home-range size. Alternatively, a user interface might allow a wide variety of displays for a given series of data: bar chart, strip-chart, colors, or ranges.

Days 5-10: Research To Develop Missing Components and To Extend or Modify Available Components

The team uses a full week to follow the initial assembly of available components with some development of additional simulation components. In particular, the available visualization tools have to be assembled in a manner that maximizes the match to the current application. The training submodels need to be upgraded to reflect the new training scenarios and weapon systems anticipated for the new landscape. Each submodel is run independently to identify as many potential errors as possible.

It is also decided that two models will be developed to help address the overall goals and objectives. The biodiversity questions require a time-step and resolution sufficiently different from the other questions to warrant a separate model. Outputs from the two models are expected to be used to modify each other as the underlying systems are being driven by the same engines.

Days 11-14: Integrate and Debug

This week's effort involves numerous runs of the simulation model with successively more components turned on. As conditions are discovered to move out of reasonable ranges (negative populations, temperatures over 150 °F, and succession stages out of line with simulated training), errors in the submodels and data are discovered and repaired. Sensitivity analyses are conducted on the more uncertain inputs — some of which are found to be quite important. The developed user interfaces are also tested and improved to remain stable.

Days 15-20: Management Evaluation of Alternatives – Reports Generated

During the final phase, management representatives are invited to participate in the final simulation runs. Some different training schedules are run along with some updates to potential property boundaries and road network possibilities. Output videos are generated for playback at future meetings and are captured for viewing on the Internet. It appears that more of the objectives than first imagined can be met through newly recognized arrangements of the planned training activities. Key locations, thresholds, and leading indicators are identified for particular monitoring as a strategy is implemented. The models are documented and made available to the management team for use in making decisions within the chosen strategy.

This imaginary scenario, based in some fact, suggests that a geographic modeling system will be useful to landscape managers (here military installation training range managers) for the rapid design and development of location specific dynamic simulation models. These models will simulate various components of the landscape, simultaneously using appropriate spatio-temporal scales for each. Long-term and indirect effects and interactions between the various components will be able for managers to explore.

1.3 Why Build Ecologically-based Land Simulations?

Although some land managers consider ecologically-based land simulations an attractive idea, others do not. This research is motivated by the author's perception of the needs of individuals at military installations to adequately manage the installation landscapes. Why might landscape simulation models be useful in this context? To answer that question, we must first ask another: What is a model? Hall and Day offer three definitions (Hall and Day 1977):

1. A model is an abstraction or simplification of a system.
2. A model is a device for predicting the behavior of a complicated, poorly understood entity from the behavior of parts that are well understood.
3. A model may be considered the formalization of our knowledge about a system.

While one may argue that modeling is inappropriate for a variety of reasons, modeling is inescapable because we, as humans, must rely on our concepts of reality to make decisions about how to interact with that reality. These concepts are models by Hall and Day's first and second definitions. Formalizing these, it should therefore be reasonable to create models based on the third definition. That is, specialists can formalize their concepts of the landscape by capturing their internal models in equations, logical rules, and systems of equations.

Turner offers more detailed uses of ecological models (Turner 1989):

1. Understanding and assessing phenomena.
2. Generating hypotheses.
3. Testing the validity of field measurements and assumptions derived from these data.
4. Predicting.
5. Optimizing for environmental decision-making.

This list indirectly indicates that data collection does not simply precede model building. Modeling generates hypotheses (Turner's use 2), which guide the selection of data that need to be collected. Models help us to fold our understand-

ings together into larger wholes that show us the consequences of those understandings. Below we briefly explore the requirements for land management at military installations and how modeling, as defined above, can be useful.

Military land managers are confronted with extraordinarily difficult assignments. They are required to determine how land is used, scheduled, and rehabilitated. Such decisions must be made with respect to multiple objectives that cover broad ranges of time and space scales. Example objectives include:

- Establish satisfactory training of military personnel,
- Sustain the state of the land for support of training over decades and centuries,
- Meet legal requirements related to TES, chemical spills and wastes, air and water pollution, and impacts of noise,
- Maintain the longer-term viability of local ecosystems,
- Manage even longer-term effects on local and regional biodiversity,
- Provide appropriate opportunities for recreation, farming, grazing, timber harvest, and wildlife preservation, and/or
- Maintain aesthetic qualities.

Land managers are confronted by advocates of these different and often conflicting objectives. Because it is virtually impossible to meet the demands of all advocates, the land manager is put in the difficult position of balancing the objectives in coherent short, medium, and long-term management plans. Of course, the strongest local advocates are those assigned to train on the landscape. This results in a stronger impetus to meet the short-term requirements of the trainers. As a result, the requirements for long-term sustainability have resulted in reports demonstrating a general decrease in the ability of military lands to sustain recent training intensities .

Land managers have to consider scientific studies as well as the interests of various advocacy groups. Therefore, they have relatively good understandings of parts of the whole system. Traditionally, the results of scientific inquiries provide (1) the basis for the formal education of land managers, and (2) a permanent record and reference for land managers. Through continued experiences and education, the developing land manager updates and modifies internal landscape models (Hall and Day's definition 2).

In most cases, the only tangible components of the professional manager's conceptual models are the decisions or recommendations provided by that individual. These models meet every one of the definitions offered by Hall and Day and uses suggested by Turner. Unfortunately, these processes are private and

cannot be subjected to formal evaluation, study, and debate. Even when the best professional judgment of a land manager is extraordinarily good, the utility of that individual's models depends on that person continuing to be available. Formal landscape modeling suggests that scientific studies and reports now have a new role to play in the age of fast computer technology. Using formalized computer modeling and simulation, the following inadequacies can be addressed:

- Different models. Even with measurably identical training, every professional views (models) the world differently. The number of available "professional opinions" can always equal the number of professionals.
- Communication difficulties. With different conceptions or models, it can be very difficult for even the "experts" to communicate effectively.
- Different specialties. Every discipline seems to be disaggregated into numerous subdisciplines. Specialists in such disaggregated disciplines can easily develop understandings (models) of the world that are in direct competition with each other. For example, when is it appropriate to view an animal as an individual; as a member of a population; as a member of a species; as a member of a guild; as a part of a community or ecosystem; or as an assemblage of organ systems each responding to the chemistry of the environment?
- Lack of challenge. Internal models are difficult to challenge and difficult to defend. Senior scientists and managers are often presumed to have the best internal models and therefore the most defensible statements based on professional judgments. If the internal model is not formalized, it cannot be inspected.
- Non inter-disciplinary. Internal models are the result of the training and experience of individuals. Few individuals are able to master more than one discipline that provides some knowledge, models, and views on the processes occurring within ecosystems and landscapes. It is therefore virtually impossible for an individual to create internal models that accommodate information derived from a myriad of disciplines including military science, psychology, ecology and biodiversity, medicine, toxicology, and others.
- Difficult dynamics. Internal models or understandings are often very inadequate for realizing anything more than direct cause-effect relationships. Strings of causes and effects resulting in a series of indirect relationships are very difficult for the human mind to comprehend and visualize.
- Difficult to manage spatial relationships. Metapopulations exist in nature at all scales of space. Visualizing simultaneously the ebbs and flows of population densities of different species occurring at different speeds at different spatial scales is extremely difficult.

This report, through an example, advocates the use of computer model simulations to help the land manager translate internalized models of the environment into formalized entities. The objective is to formalize information from an array of disciplines into a software simulation model. Such simulations provide the promise of addressing some of the concerns identified above. Formal models, like formal journal articles, are less ambiguous than internal thought models. They allow individuals to communicate more precisely, more completely, and more efficiently. Simulation models provide an environment for establishing connections between disparate pieces of scientific information, studies, and reports. Numerous authors representing different specialties can participate in the construction of models. As formal models, they can easily be reviewed and challenged. This allows not only the models, but the underlying scientific knowledge to be improved. Perhaps most importantly, dynamic simulation models provide an environment for experimenting with and understanding indirect cause-effect relationships.

Landscape and spatially explicit ecological simulation is not a panacea and should not be oversold. While exploratory models might help scientists to discover some new information, for the most part, such models are no better than the data that underlie their structures. Note however, that errors in such models only reflect gaps in the knowledge upon which the model is based. This is not a problem with modeling, for we make decisions based on our internal thought models of systems and nature. It is the formal capturing of such internal thought models that opens them up for recognizing errors that might otherwise go undetected or ignored.

Formal models, like any formal documentation, provide an education to any who look closely at the model. Any time information flows are altered, those who originally held the information lose some amount of authority and power; those who are recipients of the information, on the other hand, gain. For example, a land manager might for years understand that some activity has a “very positive” impact on some piece of land. In creating a computer simulation, “very positive” must be quantified, and in attempting to do so it might be discovered that there really is no quantification available, or that the quantity is much less than expected. The manager now has better information and the source of the original judgment (e.g., some scientist) has lost a small degree of authority. Knowledge is power and those forced to share or reveal knowledge often do so unwillingly.

1.4 Ecological Simulation Context of the Study

This research effort fits into a broader challenge of creating a software modeling environment to meet the anticipated requirements of the next generation of

military land managers. For the purposes of this study, the future system that will capture the results of this and other efforts will be called I-STEMS, the Integrated Spatio-Temporal Ecological Modeling System. Landscape components include water and air, plant and animal populations and individuals, chemicals, topography, and soils. Modeling environments are required that can capture current best understandings of landscape processes to illuminate the logical consequences of our current knowledge, to improve understandings, and to aid in predicting the impacts of proposed uses, schedules, and changes to the landscape. The modeling environment must allow for specifying interactions between a wide variety of processes operating at different resolutions of time and space. Following is a representative list of landscape components that cover the range of required functionality of the target system:

- Vegetation population. These populations are relatively static over short periods of time and can be modeled as entities fixed in space.
- Animal population. At relatively low spatial resolutions, populations of animals can be accommodated in a manner identical to that for plant populations. At higher spatial resolutions animal populations show a significant amount of flux across the landscape. At even higher spatial resolutions individual animals must be recognized.
- Individual plants. Where individual plants are important, it becomes necessary to separate out plants from their populations as in the JABOWA models (discussed in section 2.2.3).
- Individual animals. Certain endangered or otherwise locally rare animals may require modeling with respect to individuals. The behavior of animals includes decisions related to physical movement.
- Animals with long-range interactions. A wolf or a fast-moving raptor involve changing the state of the system at long distances. When these distances are greater than one step in the spatial resolution, they must be accommodated through other than nearest-neighbor interactions.
- Climate. Climate involves the average patterns of wind, temperature, humidity, and solar insolation. This generally does not vary over shorter time periods (less than centuries), but can vary over relatively short distances (hundreds of meters) in mountainous terrain.
- Weather. The current local rain, temperature, humidity, and solar insolation state comprises the local weather and is variable at very high spatial and temporal resolution.
- Storm events/flooding. As a result of rainfall, the movement of water across the landscape is important in the modeling of some systems.
- Pollen. While plants are generally static in geographical location, their scents and pollen can be dispersed quite rapidly by wind.

- Pheromones. Similar to pollen, pheromones generated by plants or animals can be carried rapidly on the wind to affect the state of surrounding areas.
- Birth, death, and reproduction. Populations and individual animals and plants spawn new individuals through various processes. Entities typically pass from the system as they cease to exist in a particular form. Birth, death, and reproduction are key requirements for an entity-based system.
- Metamorphosis. Some landscape entities are best modeled as one type of entity for part of the simulation time and as another type of entity during other times. For example, some individuals form herds for a particular part of the migration or feeding seasons and are best treated as a single entity, only to be disaggregated into individuals during other seasonal times.
- Regular-time state update. Different components of the landscape change at different frequencies in time and space. Some of these changes can be efficiently captured at fixed time intervals.
- Irregular-time state updates. Some landscape processes remain dormant until some infrequent event triggers them. For example the water content of the ground or streams may change slowly with respect to insolation and evapotranspiration and can be updated at a time step appropriate for the involved vegetation. A storm, however, results in changes to the state of the system at a much more frequent interval. This requires a change in time step, or may require an event-driven time step.
- Food consumption and respiration. Individual entities must be able to recognize food in their environment and then be able to consume and use that food. Where appropriate, entities must be able to modify their environment by leaving digestive waste products behind.
- Predator-Prey. Animals live by consuming other living entities. This predator-prey relationship must be a primary option of an animal simulation environment.
- Disease. Similarly, diseases mediated by microorganisms provide a driving force behind many systems.
- Roads. Paths and roads are created by people and animals and form a key input to and consequence of human and animal behavior.
- Streams. Similarly, streams are created by and form a key input to the dispersion of rainwater. They also provide boundaries to certain regions and communication routes for certain people, animals, and plants.

A system that facilitates a general purpose environment for landscape simulation must accommodate the kinds of entities and processes suggested by this list. This document presents an overall paradigm that will be used to create such a system.

1.5 Organization of the Study

This study is part of an attempt to bridge gaps between computerized modeling and simulation systems that reflect differing ecological viewpoints. Chapter 2 provides a framework for this study through a review of some of the major perspectives in ecology. It attempts to link the various perspectives by presenting them as proceeding from different locations on a time-space continuum. Chapter 2 builds on this framework by reviewing a few modeling and simulation techniques that sharpen these perspectives. Contrasting these approaches suggests various challenges to bridging the gaps between them. Chapter 3 identifies the particular challenges taken on by this study in the form of project objectives. The approach taken to address these objectives is laid out in Chapter 4 followed by an application of the approach to a realistic model based on Army objectives in the ecosystem of the Mojave Desert in Chapter 5. Chapter 6 evaluates the results of the effort followed by general conclusions and future research directions discussed in Chapter 7.

2 Perspectives in Ecological Modeling and Simulation

Underlying the imaginings of the previous chapter are software concepts firmly grounded in modern ecological theories. It is important that such a software environment reflect and explicitly recognize a broad array of current theories in ecology. It will fail if it either focuses on (and therefore facilitates) a small number of ecological theories, or if it seeks to require ecologists to work with and be restricted by new theories created by the software authors. Wu and Loucks (1991) state, "A hierarchical perspective is appropriate and necessary to unify ecological concepts and theories. The unification can be accomplished only by focusing on the multiplicity of scales of ecological phenomena. Such a unifying perspective does not preclude, but builds upon, pluralistic studies at different ecological scales."

A literature review covering theories of ecology and past scientific accomplishments is presented in section 2.1. This provides the historical context within which ecological modeling and simulation should be accomplished. Then, section 2.2 identifies and briefly reviews some ecological modeling systems.

2.1 Underlying Theories of Ecology

This section provides a brief literature overview of current theories of ecology that must be recognized when developing new tools for ecologists and land managers who wish to model and simulate the ecological consequences of land use management decisions. Each section briefly presents a current theory in ecology and then draws guidelines from this theory for the design and development of ecological modeling and simulation software.

2.1.1 Equilibrium Theory

In our hypothetical story (section 1.2), the modeling effort is working on the supposition that the landscape is going to be impacted by a change in land use patterns and schedules. The I-STEMS modeling and simulation software used allows an equilibrium or non-equilibrium approach to landscape simulation.

Equilibrium theory assumes that a system, when perturbed seeks to return to the equilibrium state. Wu reports that Plato and Aristotle provided the first supraorganismic balance-of-nature concept and Carl Linneaus (1707-1778) called the balance *Oeconomia Naturae* (Wu and Loucks 1991). Clements put forth the organismic viewpoint of community ecology and advocated a succession process that leads to a climax state — a state of natural equilibrium. Analytic approaches to modeling nature grew from this philosophy. One of the first was Pierre-Francois Verhulst's logistic equation reviewed in Kingsland (1985). This was joined by others like the Lotka-Volterra equations, Rozenzweig-MacArthur equations, Leslie's predator-prey equations, the Nicholson-Bailey model, and modern derivatives reviewed in DeAngelis and Waterhouse (1987). Mostly analytical, these all have well-defined equilibrium points. Even those that settle into a dynamic equilibrium define a final steady state condition.

If equilibrium states can exist in nature, it can be argued that these states are rarely, if ever, seen in practice because disturbances are continually pulling the system away from such equilibria. Following the disturbance, the system then continues to seek the equilibrium or climax state. Disturbance theory argues that ecological systems are continually in a state of flux. Analytic systems of equations are only useful for modeling systems that are at or near some equilibrium point (Reice 1994). Disturbance is also viewed as a major contributor to diversity, for through disturbance, additional ecological niches are opened for exploitation. Disturbance also occurs at various ecological scales (Pickett, et al. 1989). Systems from individuals to ecosystems can and are disturbed. Disturbance can be described and characterized by spatial distribution, frequency, return interval, rotation period, area, intensity, severity, and synergism. It therefore becomes impossible to adequately characterize ecological systems solely through analytic approaches.

From equilibrium theory and disturbance theory is drawn the following requirement for a general purpose ecological modeling and simulation software environment:

Allow for model components that seek equilibrium points. Examples include carrying-capacity numbers, succession steps, logistic growth, and equations of Nicholson-Bailey, Lotka-Volterra, Rozenzweig-MacArthur, and Leslie's predator-prey.

2.1.2 Non-equilibrium Theory

Wiens et al. (1985) note that studying long-term natural processes is like creating an entire movie from a few frames. At human scales it is natural to view nature at larger scales as systems in equilibrium; or at least systems that move towards equilibrium. However, it is not correct to assume that systems that appear to be stable at human scales are actually at equilibrium. Equilibrium theory has been challenged by non-equilibrium theory. Wu and Loucks (1991) and many others, recognized that equilibrium states require density-dependent population regulation and claim that there is little direct evidence for this. Caswell states, "Equilibrium theories are restricted to behavior at or near an equilibrium point, while non-equilibrium theories explicitly consider the transient behavior of the system" (Caswell 1978). DeAngelis and Waterhouse (1987) write that the "...dynamics of ecological systems at small spatial scales is usually an ephemeral phenomenon with no equilibrium properties ...". It is further argued that ecological systems at any scale demonstrate non-equilibrium processes that appear in equilibrium at larger scales (O'Neill, et al. 1986; Urban, O'Neill, and Shugart 1987). For example, fire is destabilizing at short time intervals, but stabilizing at long time intervals (Loucks 1970).

Non-equilibrium theories are grouped by Chesson and Case (1986) into four distinct types:

1. Those with an "absence of point equilibria." Such systems continually fluctuate in mathematically random and chaotic fashions. These can be argued to be in equilibrium in the sense that the possible states of the system are grouped around chaotic attractors.
2. Those that emphasize "fluctuations in density or environmental variables as dominant processes." (Types 1 & 2 are "enlargements of classical competition theory and its equilibrium extensions.")
3. Those where the mean of climate fluctuations varies over time so that historical factors are important. Here, the scale of centuries and longer provide the driving context within which an ecological system adapts.
4. Those with competitive displacement. "Chance and history may be major factors shaping community structure." This is the expect-the-unexpected viewpoint. Events that appear random at a given scale may be a predictable part of a much larger time scale.

From non-equilibrium theory are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Allow for model components that respond without regard for any preset goal. Such components change over time simply as a function of the state of the external and internal environments. No explicit equilibrium is embodied in the system of equations used to compute future component states. System thresholds, non-differentiable relationships, and logical and quantized responses are all possible.

2.1.3 Hierarchy Theory

The predictability of ecological systems is inherently limited and dependent on scales (Levin 1989; Loucks 1985; May 1986). The degree to which any given ecological study identifies the existence or non-existence of processes that allow a perturbed system to return to some equilibrium state is dependent on the temporal and spatial scales and the level of organization on which the study focuses. “Therefore, there is no single correct scale of investigation and thus no universal law in ecology” (Wu and Loucks 1991).

Wiens et al. (1985) write “Some of the most vociferous disagreements among ecologists arise from differences in their choice of scale.” To illustrate the point, they suggest how differently ecologists studying the relationships between jackrabbits and coyotes at five different scales might view their interactions. These scales were defined as: (1) the location where the entity lives (2) a local patch occupied by many individuals (3) many local populations that interrelate through dispersal (4) a closed system (or approximation thereof), and (5) a biogeographical scale where different climates and different sets of species exist. Depending on the spatio-temporal scale chosen, two species can appear to be highly interrelated or completely independent. Land managers, modelers, and ecologists must always be willing to back away from their particular models and approaches and view the system from perspectives arising from different scales in time and space. This will ensure that the proper scale is chosen with respect to the particular question or set of questions being asked.

Hierarchy theory offers a framework within which to view and integrate different scales. The theory has matured sufficiently to be documented in several books (Allen and Starr 1982; O'Neill, Johnson, and King 1989). There are three dimensions: time, space, and organization. Organization refers to organizational levels of life, which are often viewed as nested. Atoms are organized into molecules, molecules into cells, cells into organs, organs into individuals, individuals into populations, populations into communities, and communities into ecosystems. Natural phenomena, which are represented by a large number of samples at the scale of study (e.g., atoms of an element in a sample or mice in a

county), can be handled very well through statistical approaches and are called large number systems. Phenomena that are represented by very few samples can be handled by careful and thorough study of each sample (low number systems). Landscapes, when studied at the human scale, have too few components to treat statistically and too many components to study each thoroughly. Such “middle number” systems are the focus of hierarchy theory (Allen and Starr 1982).

Hierarchy theory links these levels. Lower organizational levels operate in smaller partitions of space and shorter periods of time. Individuals operate on small scales in time and space, while ecosystems operate on much larger scales in time and space. The apparently neat relationship between these three scales has been discussed and graphically depicted in time-space diagrams. Ocean hydrodynamics (Stommel 1963) and processes in landscape ecology (Urban, O'Neill, and Shugart 1987) have been presented in such diagrams showing a clear and simple relationship (see Johnson 1993). Delcourt and Delcourt (1991) partition time and space into four domains (overall time and space of interest):

Micro-scale (1-500 yr, 1-10⁶ m²): This domain is the most familiar to ecologists. Within it exist population dynamics, productivity, competition, and response to disturbance events

Meso-scale (10⁴ yr, 10¹⁰ m²): Here landscape mosaics and watersheds dominate. Animals and plants develop adaptation to disturbance regimes.

Macro-scale (10⁶ yr, 10¹² m²): This scale involves quaternary studies. Species displacements occur on a subcontinental scale, and rates of spread of species and genetics as well as extinctions define this scale.

Mega-scale (>10⁶ yr, >10¹² m²): At this scale, planetary phenomena like development of biosphere, lithosphere, hydrosphere, and atmosphere and macro evolutionary history of life on earth dominate.

According to hierarchy theory, systems result from evolutionary processes that favor a nested, hierarchical organization. Each level is constructed from identifiable subsystems (Johnson 1993). Hierarchical levels are separated by conceptual surfaces. For ecological modeling, the modeler need normally consider only three levels: the level dealing with the question being asked of the system, the next higher level to provide context (constraints), and the next lower level, which contains the dynamics and structure to be modeled (Johnson 1993). Dynamics of even lower level structures in the hierarchy are, for the most part, sufficiently attenuated to be replaced by average behaviors or even ignored because they are

captured in an attenuated and aggregated fashion through dynamics occurring in intermediate levels (O'Neill et al. 1986). Landscape ecologists, for example, attempt to capture the complexities at smaller than landscape scales into single numbers and indices (Turner 1989).

There is not always a strict hierarchy that proceeds nicely through atoms, molecules, organs, individuals, etc. Examples are ecosystems that exist within individual organisms (intestinal tracts), populations that can exist within individuals or other populations (diseases), or populations that span a number of communities (birds). A general purpose ecological modeling and simulation package must provide modelers with the ability to nest systems within systems without forcing the nesting sequence to be strictly hierarchical. The general purpose I-STEMS simulation package requires open and multiple time and space scales that can be simulated individually or concurrently.

From hierarchy theory are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Allow for the simulation of ecological processes occurring at a wide range of spatio-temporal scales. A fundamental notion of hierarchy theory is that different processes happen at different scales in time and space. Simulation of the processes is most efficiently captured if the largest spatial and temporal scales possible are used.

Ensure that multiple scales in time and space can be simulated simultaneously. Short and long time scales; small and large spatial scales.

Allow any given simulation component to alter its operational time and space scales. In certain life stages or times of the day biotic and abiotic processes are more efficiently modeled and simulated at different scales than at other times. For example, movement of a herd is best simulated at relatively short time intervals over perhaps shorter space during times of migration. Or, the processes involved in moving water across or through a landscape are relatively slower during dry periods than during wet and changing periods.

Provide for the capture of processes within processes. At times it appears prudent to clump or aggregate ecosystem components; at other times (or to other people) it seems wise to separate the simulation into its components. In the latter situation it is important to be able to present the reduced simulation as a whole to other system components. For example, it may be

important to simulate a herd of animals as a herd for certain portions of a simulation, but as individuals for other portions.

Allow for the simultaneous simulation of multiple scales. Modeling at the level of populations, for example, should not preclude the opportunity to simulate the behavior of an individual (as part of a modeled population) as well.

Do not force any particular hierarchy. For example, allow for an ecosystem to exist within an individual as well as an individual within an ecosystem.

2.1.4 Metapopulation and Patch Theory

A key aspect of the I-STEMS concept is the notion of a spatial distribution of resources and processes. The theoretical underpinnings of this approach are found in the theories of island biogeography (MacArthur and Wilson 1967) and early arguments supporting notions of metapopulation theory (Andrewartha and Birch 1954; Hanski and Gilpin 1991). Island biogeography provided a theoretical and mathematical framework for describing the relationships between a set of stable populations and areas of unstable populations (islands). Metapopulation theory extended the concepts of island biogeography by allowing interactions between numerous areas containing unstable populations. It provided a theoretical foundation describing processes by which similar competitors can coexist in a patchy environment (Horn and MacArthur 1972; Levins and Culver 1971; Slatkin 1974).

Simple, but powerful, spatially explicit numerical models developed by R. Levins formed the foundation for a body of literature exploring metapopulation theory and its relationship to metacommunities, landscape ecology, island biogeography, patchy environments, and conservation biology. For a review, see Hanski and Gilpin (1991). Metapopulation theory provides a simple mechanism that explains how it is possible for a landscape to contain a number of direct competitors. In a completely homogeneous environment, the most successful competitors crowd out their inferior competition. Real systems are patchy at all levels of hierarchical organization because of perturbations and disturbances. In such dynamically heterogeneous environments, metapopulation theory predicts the existence of a potentially unlimited number of close competitors. Levins' basic equations have been extended in various different ways. Hanski (1985) added migration to Levins' model (to create a 3-state model). Dynamic complications, caused by immigration, were demonstrated to result in alternative stable equilibria. Gilpin (1990) demonstrated numerical computer models for making predictions of the dynamics

of real systems using metapopulation theory. Finally, Gardner, O'Neill, and Turner (1993) conducted theoretical simulations of competing species with varying disturbance regimes and harvest schemes.

Dispersal is the key activity driving metapopulation dynamics. Hansson (1991) reviews the characteristics of dispersal that influence metapopulation functioning and identifies three categories of factors influencing dispersion: economic thresholds, resource conflicts, and inbreeding avoidance. Economic thresholds for the population involve the relative availability of essential environmental resources such as food, shelter, and water. Resource conflicts often involve these resources as well as potential mates. Even when resources may be plentiful, territoriality behaviors may encourage migration. Some migratory behaviors seemed to be associated solely with drives aimed at avoiding inbreeding. Population size correlates with dispersion rates of different creatures.

With respect to hierarchies of system organization, Hanski and Gilpin (1991), in a historical account of metapopulation theory, define metapopulation scale as part of the following continuum:

Local scale - the scale at which individuals move and interact with each other in the course of their routine feeding and breeding activities. This is the scale immediately beneath the metapopulation scale that provides the dynamics behind metapopulations.

Metapopulation scale - the scale at which individuals infrequently move from one place (population) to another; typically across habitat types that are not suitable for their feeding and breeding activities, and often with substantial risk of failing to locate another suitable patch in which to settle.

Geographic scale - the scale of species' entire geographical range; individuals have typically no possibility of moving to most parts of the range. This scale provides the context within which metapopulation dynamics take place.

While metapopulation theory provides the fundamental basis for accepting that multiple species competing for the same resources can coexist on a landscape, patch theory provides a framework for capturing the dynamics in spatially explicit models. The patchwork of landscapes and habitats has been shown to be very important for allowing local extinctions, but global persistence (DeAngelis and Waterhouse 1987; Kareiva and Anderson 1988; Levin 1988).

Tilman (1994) argues persuasively for the recognition of spatial arrangements as important variables in ecological models. He argues, from the literature, that colonization limitation is important in succession dynamics. Succession theory defines, for any given climate, a succession of vegetative communities. The fundamental basis of succession theory was initially laid out by Clements (1936). The actual trajectory of the succession of plants is not fixed, but is a function of the available seed sources; the local disturbance regimes as defined by fire, disease, flood, and other major perturbations; and the physiological capacity of the plants to respond to the perturbations.

Numerous theoretical demonstrations show that habitat subdivision allows two species to coexist as metapopulations, stabilizes host-parasite and predator-prey interactions, and influences the evolution of cooperative behavior (Tilman 1994). Tilman presents a spatially explicit model based on the work of Levins (1969) and the extensions by Hastings (1980) and Nee and May (1992). This model demonstrates that (1) if a set of species cannot occupy all space in a simulated raster environment (2) there can exist only a single individual representing a single species at any cell (3) there is no spatial advantage given to any species for populating an open cell (species in adjacent cells have no advantage over other species) and (4) that competitively superior species are poorer distributors; any number of species can cohabitate. Using this model Tilman, et al. also demonstrated that habitat destruction has the greatest effect on the best competitor (Tilman, et al. 1994).

Spatially explicit models using fixed grids of patches have been used to explain observations from nature. Reice demonstrated that when openings in a habitat are created, the proportions of the species that recolonize are unpredictable (Reice 1994). Increased levels of disturbance are associated with the increased diversity. Streams have more diversity over ponds because of more disturbances. Using 1-, 2-, and 3-patch simulation models, Wu, Vankat, and Barlas simulated source/sink interactions, persistence, and resilience of populations (Wu, Vankat, and Barlas 1993). Increased patchiness correlated directly with persistence and resilience. Such simulation experiments are reflected in experiences with natural systems. For example, Walde experimented with predator and prey mites on apple trees in groups of 1, 4, and 16 trees. The largest population densities, and most persistent populations were associated with the largest groups of trees (Walde 1991). Similarly, using patches of meadows, Robinson et al. determined that persistence of individuals in patches increased with patch size (Robinson et al. 1992). A scaling result was also discovered: larger bodied mammals did best on larger patches; smaller bodied on smaller patches. Small mammal distribution represents a source-sink pattern with the smaller patches providing the source of individuals.

These small mammals competed directly for resources and were able to coexist simply because of their natural partition of patches by patch size.

From metapopulation and patch theory are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Explicitly recognize the heterogeneous distribution of landscape components. It must be possible to explicitly recognize and be able to simulate the changes to patterns across a landscape. At any given spatio-temporal scale, some processes and landscape components will be best simulated as a single homogeneous process or entity on the landscape (e.g., weather), while others must be captured as heterogeneously distributed (e.g., slope, elevation, streams, and vegetation stands).

Movement between patches. Movement of animals, propagules, and chemicals between patches must be possible.

Movement of patches. Patches must be able to come into existence, disappear, and be able to grow and recede in any given direction.

2.1.5 Landscape Ecology

As discussed earlier in section 2.1.3, different natural processes occur at different temporal and spatial scales. The objective of the larger program to which this research is attached is the development of simulation tools to assist land managers. These professionals typically focus their efforts at temporal resolutions of weeks to decades and spatial resolutions of hectares to thousands of hectares. Landscape ecology is a discipline that shares this focus (Turner 1989; Urban et al. 1987). Delcourt and Delcourt recognize the importance of understanding modern landscapes by reflecting on their changing state over the Quaternary Period (past 1.8 million years) and especially the Holocene Epoch (last 10,000 years). Landscape ecology is especially concerned with the human-nature interface at the landscape scale and tends to take a holistic approach to these processes. This is reflected in a large number of landscape indices that attempt to capture the essence of the entire landscape in a few numbers. Examples are measures of dominance, contagion, and fractal dimension (O'Neill et al. 1988). Several dozen such measures have been developed. Many of these change with changing spatial scale (Turner, et al. 1988)

Another set of indices associated with landscape ecology has come out of percolation theory. This theory concerns itself with the patterns of patches on the landscape

and specifically the probability that patches of certain dimensions and randomness form corridors that span across the landscape. Caswell developed the notion of “neutral models,” simplified computer landscapes that are randomly generated to provide patterns of suitable and unsuitable habitats (Caswell 1976). For example, assume an animal or plant is constrained to live only in the suitable habitat and has no possibility of even crossing unsuitable areas. A number of questions can be posed to such a system. Gardner, et al. demonstrated that below a landscape coverage of 0.6 (60 percent) patches are highly fragmented (Gardner et al. 1991). His simulations demonstrated “... that large differences in species abundance and habitat utilization are produced by small changes in the maximum possible dispersal distance.” Turner, Gardner, and Dale (1989) used percolation models to evaluate disturbance intensity and frequency on various densities of habitat in neutral maps. Disturbance frequency and intensity had variable impact on neutral model landscapes. When the landscape was occupied by less than about 50 percent of the habitat, that habitat was sensitive to frequency, but demonstrated little difference in its response to intensity. Habitats occupying more than 60 percent of the landscape were less sensitive to frequency, but more sensitive to intensity. O’Neill, through random models, showed that hierarchically structured landscapes (vs. random neutral model landscapes) had smaller perimeters, were less clumped on sparse landscapes and more clumped on dense ones. This permits percolation on a broader range of conditions (O’Neill, Gardner, and Turner 1992).

Clearly a modern ecological modeling environment must provide the modeler with opportunities to develop spatially explicit systems. The patchwork found in the matrix of landscapes is an essential component to the processes that determine population densities. This importance of patches is probably important at every spatial scale, although most research has focused at a scale in which individuals or metapopulations are resolved.

From landscape ecology are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Provide the ability to analyze heterogeneous landscapes. The response of an ecological component to a pattern on the landscape might be accomplished either through a brute-force simulation of other entities within the pattern using very short time intervals, or by a single response to knowledge about the landscape pattern as captured in such measures as provided by percolation theory.

2.2 Ecological Simulation Software

Ecological principles, theories, and experimental data have resulted in a large number of computer-based models and modeling environments. This section briefly explores the extent of this development effort.

2.2.1 Animal Simulations

Computer-based simulations of animals can be grouped into three main categories: (1) individual-based simulations for the study of behavior, genetics, and evolution (2) theoretical metapopulation models (see section 2.1.4), and (3) population-based models for landscape and region management. Several individual-based models for exploring fish population responses have been developed by DeAngelis and others (1993a, 1993b). These efforts are also associated with the development of new techniques to improve computational efficiency when dealing with large numbers of individuals in a population (Sheffer et al. 1995). Individual-based modeling has also been used to capture the behavior and energetics of the wood stork on 15-minute time intervals (Fleming et al. 1994). Such models are part of a growing trend in applying current computational capabilities to individual-based models.

Population modeling is more typical of landscape ecologists. One example is the Ecosystem Management Model which integrates ARC/INFO with a FORTRAN-based ecosystem landscape model developed to help manage Elk Island National Park in central Alberta (Buckley et al. 1993). It consists of an integrated set of submodels. There is a spatially explicit process-oriented model of vegetation productivity and growth and ungulate submodels, which take care of population dynamics, predation, parasitism, and animal condition. Cuddy, Davis, and Whigham (1993), reported on a military landscape management program called the Land Management Advice System (LMAS), which simulates the movement of tracked vehicles. In this system, an expert system called ARX (Whigham and Davis 1989) is the primary program. During a simulation run, it makes calls to ARC/INFO, which feeds back landscape information and facilitates landscape analyses. These two examples are typical of how animals have been simulated as populations.

The new discipline of artificial life explores the notions of genetic evolution through experimentation with computer programs that simulate competition between different computer programs. These programs are allowed to reproduce, move in a virtual space, mutate, and develop into processes much different from the original programs. Several popular and powerful environments are currently available to researchers of artificial life. The Tierra simulator is a system for studying

ecological and evolutionary dynamics (Ray 1994a; Ray 1994b). It is a virtual computer and operating system that allows the execution of machine code that can evolve and compete with other programs. The architecture of the virtual computer can be changed and users are provided a wide range of opportunities for developing software programs that operate within this environment. Swarm is a similar environment for developing and exploring the emergent behavior of simulated life forms (Hiebler 1994; Minar 1995). The goal of Swarm is to have a standardized set of tools for exploring complexity. The concept of a swarm is that sets of similar entities can be grouped together and treated as a unit. This unit itself may be combined with similar units to form hierarchies of swarms. Each entity in the swarm operates independently; behavior is based on the internal and external system states. PolyWorld is being developed by Sun computers as a single, powerful artificial life system that “attempts to bring together all the principal components of real living systems into a single artificial living system” (Yaeger 1993). It offers a wide range of behaviors (eating, mating, fighting, moving, turning, focusing, lighting) and provides its simulated life with neural network learning capabilities. Its intended audience is evolutionary ecologists. These and other artificial life programs can be explored and accessed on the Internet. The MIT Artificial Intelligence Laboratory and The Santa Fe Institute are excellent starting points (Bodelson and Butler-Villa 1995; Thau 1995).

From animal simulation software the following requirements are drawn:

Provide for the simulation of individual entities. Although the ability to simulate individual entities is not found in landscape management models, the software advances made by the artificial life community provide a good indication of the latent potential.

Allow system components to learn and evolve. The artificial life systems identified enable model components to learn and evolve. In the context of landscape management, evolution is not a significant factor. However, the evolution of behavior patterns based on learning is important where intelligent animals are concerned.

2.2.2 Storm Simulations

Simulating the hydrologic movement of water across landscapes after storm events is a very active area of research and development motivated by potential losses to property, loss of life, and threats to health due to nonpoint source pollution. To accomplish such simulations on complex landscapes, researchers have chosen recently to integrate geographic information system (GIS) technologies with flow

simulation models. The simplest interconnection affording this combination is to export starting conditions from a GIS into the simulation. Using this type of approach, Battaglin, Kuhn, and Parker (1993) divided the Gunnison River Basin into 20 smaller watersheds and then used a GIS to generate land cover and topography reports for these areas. The results were fed into a precipitation-runoff modeling system. The GIS was used to automate extraction of physical data. Similarly, storm flooding has been simulated for Hong Kong (Brimicombe and Bartlett 1993).

Nonpoint source pollution simulations have involved the linkage of various simulation packages with geographic information systems. For example, ANSWERS (Areal Nonpoint Source Watershed Environment Response System) has been linked with GIS (Krummel et al. 1993; Rewerts and Engel 1991). Hay, Knapp, and Bromberg (1993) have integrated graphical user interfaces, statistical analysis packages, and GIS. For the simulation of the Texas Gulf basin, Srinivasan combined a GIS with the SWAT (Soil and Water Assessment Tool) simulation software. Several different water quality (WQ) models from the Agricultural Research Service, Natural Resources Conservation Service (NRCS) have been captured within the GRASS GIS (Cronshey, Theurer, and Glenn, 1993). DePinto linked the geographically-based WAMS (Watershed Analysis and Modeling System) model with the ARC/INFO GIS to simulate watershed loading, groundwater contaminant transport, and dissolved oxygen in rivers (DePinto et al. 1993).

In many cases, water flow simulation software has been developed and integrated for the purposes of a single project. For example, D'Agnese established a rather complicated mix of several software packages to develop a series of groundwater simulations for an area near Death Valley (D'Agnese, Turner, and Faunt 1993). Frederickson, Westervelt, and Johnston (1994) integrated GIS, HEC1, HEC2 (hydrologic software), and a GUI to develop a flood impact prediction prototype for the U.S. Army Corps of Engineer's Omaha District.

Instead of using the GIS only as a data source, new approaches process raster GIS data in their native cellular format. For example, a general purpose finite-element approach to watershed overland flow simulation has been designed and developed within the GRASS GIS called r.fea (Gaur and Vieux 1992; Vieux and Westervelt 1992; Vieux, Farajalla, and Gaur 1993). A finite-difference approach (CASC2D in GRASS) has been developed by Saghafian (1993).

The Hydrologic Engineering Center's (HEC) series of river flow simulation models are being replaced with object-oriented simulation environments. A recent software design effort using object-oriented approaches for simulating watersheds has been

developed by Bennet, Armstrong, and Weirich (1993). In MHMS (the modular hydrologic modeling system), developed with an object approach, user's pick and choose from hydrological components to develop new simulation models for specific areas (Leavesley, et al. 1991; Leavesley, et al. 1993).

In all cases noted here, hydrologic simulation models have been linked in some manner to GIS databases. Approaches to such linkages vary from exporting GIS data into files and formats read by the models to more fully integrated linkages where GIS data are accessed directly in their native formats. The meaning of linking these environments is covered well by Maidment (1991; 1993).

From existing storm software are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Use existing simulation models. Significant efforts have gone into the research and development that has resulted in a number of very good simulation models. If at all possible, model integration should first use such models intact.

Support the movement of water across landscapes. The movement of storm water across a landscape is very important for establishing location-specific erosion and soil moisture content.

Support the simulation of water movement through stream and river networks. Once water becomes a part of established streams and rivers, the simulation of the behavior of water in these bodies is crucial to the prediction of flood events.

Accommodate the subsurface movement of water. In many environments, the movement of water through aquifers is critical to the prediction of above-ground plant and animal communities.

Support scouring, deposition, and chemical transport. At some spatio-temporal scales, it is the movement of large amounts of material via moving water that is critical to the larger system. In others, the movement of chemicals through different soil types, textures, and chemistries is important.

2.2.3 Landscape Simulations

Simulation of ecological processes at the level of the landscape has also resulted in a significant number of models and modeling approaches. Forest ecologists have

been very active in this area, producing a number of modeling environments. An example is the JABOWA model (Botkin, 1977; Botkin, Janak, and Wallis 1972). It is an individual-based model that tracks the growth of trees and their effects on their neighbors within a small area (about 10 meters square). The loss of large trees within such an area leaves a gap in the forest canopy. More recent versions of gap models simulate a large number of "gaps" that match cells in a raster GIS. One such model is ZELIG; a dynamic simulation environment that divides landscapes into cells that are divided into gap-scale plots (Urban, Bonan, and Smith 1991). The plots are identified with the proportion of total area in different cover types. Another example is LANDIS, a JABOWA/FORET model simultaneously run for each cell in a large raster matrix, developed by Mladenhoff, Host, and Broeder (1993). Individual trees are modeled as part of cells that consider the size, location, type, and state of all member trees. Models in neighboring cells are allowed to dynamically affect each other using this approach.

A large number of modeling approaches based on patch theory (section 2.1.4) are represented by the following examples. PatchMod is (1) a spatially explicit age- and size-structured patch demographic model and (2) a multiple species plant population dynamic model used to model the Jasper Ridge serpentine grassland. Gopher mounds provide the primary patch-generating disturbance (Wu and Levin 1994). The ARC/INFO GIS and a FORTRAN-based ecosystem landscape model were combined through an ecological modeling interface to address vegetation and ungulate management objectives. The natural system is divided for model development purposes, into 12 primary submodels (Buckley et al. 1993).

Numerous estuarine and river basin models have been developed through a cellular modeling approach to assist land managers and politicians in selecting the best long-term land use decisions (Costanza and Maxwell 1991; Costanza, Sklar, and White 1990; Costanza, Sklar, and Day, Jr. 1986). More recently, the approach has been used to develop a Patuxent landscape model (Costanza et al. 1993), and an Everglades landscape model (Costanza et al. 1992). The software supporting these models is called the Spatial Modeling Environment (SME) and is currently under development by its author (Maxwell 1995).

Another application for cellular-based simulation models is forest fire modeling (Clarke, Olsen, and Brass 1993; Kessell 1993; Rothermel 1972). Such models are typically cell-based and employ relevant physics-based governing equations and appropriate stochastic functions to capture uncertainty in such things as the lobbing of embers from exploding logs and shifts in swirling winds. Fire enters a cell where it burns available fuels, expands to neighboring cells based on fire

intensity, local slope and elevation, and the current wind, temperature, and humidity conditions.

From existing landscape simulation software are drawn the following requirements for a general purpose ecological modeling and simulation software environment:

Provide for simulation of individual plants and trees. JABOWA and FORET type models focus on the response of individual plants to the existence and state of surrounding individuals. Such models have been well developed and similar capabilities must be available in any general purpose landscape simulation capability.

Support the simulation of landscape patches. As discussed in Section 2.1.4.

Accommodate the simulation of fire. This could be at scales that capture the minute-to-minute behavior of a fire as well as scales that capture the development of fuel buildup over years, the stochastic occurrence of fire, and the effects of fire on the patchwork established on the landscape.

Allow modeling of individuals. While the JABOWA and FORET families of forest models provide for simulating forest dynamics at the level of the individual tree, few significant discrete mobile entity simulation capabilities have been identified in the context of ecological simulation. One notable exception is efforts to anticipate Army training impacts on a landscape by simulating the anticipated movement of vehicles (Cuddy, Davis, and Wigham 1993). Facilitating the capture of the behavior of individual animals is the principal focus of the research documented here.

2.2.4 General Purpose Simulation Packages

A number of general purpose simulation packages have been developed by the research community and many are currently available via anonymous ftp access. Several of these systems are reviewed briefly below with respect to their capabilities in the context of this effort. They are explored as two distinct sets: nonspatially and spatially oriented. The common thread in both sets is their explicit focus on simulating processes through time.

While any basic programming language is suitable for developing simulation software, the overhead of managing time and for updating state variables in a variety of methods (Euler, Runge-Kutta, etc.), is significant. Researchers began to identify a standard set of software routines that could be used for any number of

dynamic simulations. One such package was SIMPACK: a SIMulation PACKAge. It is a set of C and C++ libraries that can be used by a programmer for developing continuous and discrete event simulation software (Fishwick 1992). The available routines take care of the timing of execution for system components. Some programming staffs found that even with such libraries, the amount of code necessary to put together dynamic simulations could be dramatically reduced by specifying new languages. For example, MODSIM, a MODeling and SIMulation package, is a programming language that supports discrete-event simulation (Belanger, Donovan, et al. 1989; Belanger, Mullarney, et al. 1989). Software programmers write discrete-event simulation software using the MODSIM language, which is translated by the MODSIM software into C code, which is then compiled into executable programs. The advantage of the MODSIM language over C is that the programmer (1) need not worry about the management of event timing, and (2) can focus on the development of objects without the burden of keeping track of C++ syntax overheads. During the early 1990s, the U.S. Army Construction Engineering Research Laboratories enhanced the MODSIM language. This enhancement was called “persistent MODSIM” because it provided simulation persistence. That is, objects and other data would persist between runs in a manner invisible to the simulation programmer (Herring, Kalathil, and Teo 1993). Experiences with enhancing MODSIM led toward the design and development of an intended replacement: IMPORT/DOME (Integrated Modular Persistent Object Representation Translator/Declarative Object Manipulation Engine) (Morrison 1995). This language is further discussed in section 3.1.4.

Through the past decade, spatially explicit modeling and simulation packages have been developed and explored. These add an extra dimension to the simulation capabilities explored above. A general purpose cellular automaton program developed in the late 1980s was called CELLSIM (Langton and Hiebeler 1990). It supported a GUI that allowed users to specify cellular automaton rules, specify initial conditions, and graphically explore the evolution of simple 2-D landscapes. During this same time period, Ronald Shattuck designed and developed CUTM, the Computer Understandable Terrain Model (Shattuck 1993). This research, conducted at George Mason University, combined traditional GIS technology with knowledge about landscapes that resulted in the development of “smart maps.” With simple controls, landscapes could be viewed as they might appear in different seasons or under different land-use and weather conditions. A primary focus of the CUTM research was the creation of user interfaces that allowed an individual to easily explore the system’s capabilities. Another system under development at that time (and whose development is continuing) was the Spatial Modeling Program which has recently been renamed the Spatial Modeling Environment (SME) (Costanza and Maxwell 1991; Maxwell 1995; Maxwell and Costanza 1993). SME

focuses on the flexibility in specifying landscape rules and on the ability to rapidly simulate large landscapes using parallel processing technologies. While the user interface of CUTM is very attractive and easy to use, it is the design flexibility that attracted the author to SME as a foundation for the research of this paper.

The current research effort pulls together the advantages of the IMPORT/DOME language and the SME approach to simulating landscape processes.

2.2.5 Additional Software Challenges

The author has briefly reviewed some of the underlying theories and concepts of modern ecology along with a number of simulations captured in software. From these were drawn a number of guidelines and goals to be addressed by the I-STEMS research. Following, however, is a list of some additional requirements demanded by general software development objectives.

Use existing code. The software development process is very complex and requires a hierarchical approach. While it is arguably true that almost any piece of software could be improved by rewriting it, reformulating key data structures, and using modern software development paradigms (e.g., “object-oriented programming”), the constraints of time and resources generally will not allow wholesale code rewriting and systems integration. As with any hierarchical systems, there will be components that persist because, though perhaps not especially efficient or elegant, they work very well and are reliable. They continue to show up in future generations of the software because the cost of redevelopment outweighs the benefits even though individual developers and designers may feel overly constrained because of the continued existence of such components. This argument can hold true for small and very large system components. System integration efforts must allow, for example, individual systems to be adopted as unchanged whole and complete units. Modifying working code can result in a decreased ability of the original authors to help debug future problems, errors in other parts of the system that relied on the exact behavior of the code fragments before modification, and loss of the ability to easily capture future versions of the integrated code without similar upgrades. For further arguments refer to Frysinger (1993) and Frysinger, Copperman, and Levantino (1995).

Minimize the number of authors of any given module. Software tends to be developed as interconnecting parts. The individual parts may be at different stages of “health.” That is, some parts may be very solidly written, while others have not been fully thought through. Also, a good piece of software is often asked to accomplish more by its users than intended by its writers. In commercial settings

that support large user communities, it becomes very important that the thoughts, concepts, expectations, algorithms, and assumptions behind software be thoroughly documented to improve the efficiency of a person who is new to the code and assigned to fix or improve that software. In research settings however, the lack of a user community makes it possible to experiment more and to actually develop more software in a given amount of time. Experimentation and development are more important than having solid pieces of software in these settings. To compensate for the lack of support documentation, it becomes important that the original authors of software components retain control of those components for as long as possible. That is, if fixes or upgrades are required, the original author (who possesses the understandings of the concepts, algorithms, and assumptions associated with the code to be fixed or upgraded) is afforded the opportunity to make the necessary changes. If a second author makes changes without the benefit of this knowledge shared through good documentation or close collaboration, the changes may result in a product not fully understood by either the original or the second author. It would not be unusual for a third author, assigned to make changes or improvements, to simply decide to completely rewrite the entire module as that could be judged to be more efficient than to take the time to understand the thinking of the original and secondary authors. Hence, in a research environment, it becomes important for the original author to “retain ownership” of whatever software they develop until that software is fully documented. If full documentation is to occur, it often is not developed until near the end of the associated project.

The design and development of a modeling environment must also pay attention to certain principles, objectives, and approaches to modeling in addition to considering the theoretical foundation of the associated discipline. The opportunities provided to a modeler in a software environment are important in the process of model construction. Overton (1977), for example, lists the following as required steps:

5. List the model objectives.
6. Identify submodels and subobjectives.
7. Construct and validate submodels.
8. Assemble the submodels into the complete model and validate.
9. Attempt to address the questions identified in step 1.
10. Examine the general behavior of the model: identify behaviors of interest.
11. Conduct sensitivity analyses; identify the structure and parameters that are causal for the behaviors of interest and validate those causal structures and parameters.

The author has found (independently, through classroom experiences with students participating in the construction of large dynamic, spatial ecological simulation models) that a superset of these steps is necessary. Specifically:

1. Identify available resources. This includes equipment, expertise, participants and their availability, existing and available models and model components, and costs involved in utilizing all resources.
2. Identify the reasons for constructing a model. Explicitly list questions that the model will be designed to address. The process for identifying questions typically involves first creating an extensive wish list of questions from which the objectives will be chosen. Participants must understand that the final simulation model should not be expected to address questions that did not make the final list. Much of the long-term success of a project rests on establishing reasonable expectations at the outset.
3. Identify potential components and interactions. With respect to steps 1 and 2, identify the likely pieces or components of a final model. Identify the required inputs, outputs, and development efforts associated with each.
4. Make fundamental time and space resolution decisions. What will be the model's actual or potential resolutions in time and space? What will be the extent of time and space? The answers to these questions must be based on the results of 1 through 3 and will constrain the possibilities in the following steps.
5. Develop a conceptual model. This step is similar to creating an outline and provides the same function. The entire team should participate in this step and the end result should be a clear and agreed on superstructure for the modeling effort. In the course of this effort, the development of several competing overall models should be encouraged. The group should collectively evaluate the ideas and pull together a single overall structure.
6. Divide the conceptual model into distinct subcomponents. The result may be a hierarchy of submodels. At the lowest level are components that can be completed through individual or very small group (3 or fewer people) efforts.
7. Identify submodel requirements. The input requirements and output possibilities for each submodel must be identified and then matched against the rest of the submodels. For each input and output, estimates should be made regarding the anticipated errors associated with each. Unresolved input requirements must be addressed. Similarly, unused outputs might not require development. The team(s) then rework the design of the submodels accordingly and repeat the process until all data requirements are covered. Also, any state variables that need to be initialized must have associated input data and input data paths for which someone is responsible.
8. Develop full-model proxy. In the next step, submodels will be developed. These must be developed with respect to the anticipated combinations of system states that will be defined collectively by the outputs of the multiple submodels. Each submodel team must therefore create simple anticipated

- time-series samples of anticipated output. It is against this anticipated information that submodels will be developed.
9. Develop submodels. At this point, submodel groups can develop and test their submodels independently. During this period the only interaction that will be required between submodel development groups is when groups identify new inputs or significant difficulties in being able to generate outputs within the promised error constraints
 10. Develop full model. The full model is developed by piecing together and testing submodels. This should be accomplished one submodel at a time until the entire model is assembled.
 11. Conduct sensitivity analyses. Important components should be analyzed with respect to their contribution to simulation model output error. These include initializing data as well as parameters. With respect to the known error in the model components, the overall reliability of the full model must be evaluated.
 12. Address the objectives identified in step 2.

The author has found these steps to be critical in the design and development of large simulation models designed by relatively large teams for assisting managers in making land management decisions.

Hence, for general purpose modeling and simulation software:

Set up user interfaces that facilitate and encourage the outlined steps. The software environment must work with the modeler(s) to address the needs of these steps in an order that minimizes development time and maximizes the utility of the final model.

Standardize systems integration. A general purpose systems integration environment must be designed for rapidly linking existing and disparate simulation systems. The primary effort in this study links two major simulation environments.

Sensitivity analysis of large simulations. The ability to analyze the sensitivity of a simulation or of components of a simulation is crucial in simulation modeling. The complexity of conducting full analyses increases exponentially as the number of coefficients in the model increases. Traditional methods employing Monte Carlo simulations, signal infusion, statistics, and exhaustive model runs become impractical. New approaches need to be researched and developed.

2.3 Summary: Ecological Modeling in Time and Space

This project is a component of a large research and development effort that is intended to result in the development of a next-generation general purpose geographic modeling and simulation environment. This future system is being called I-STEMS, the Integrated Spatio-Temporal Ecological Modeling System. Sections 2.1 and 2.2 explored theories and systems (respectively) that must guide efforts designed to help reach this larger objective. The focus of the current research is on (1) the design and development of a mobile animal simulation environment and (2) the integration of that environment with SME, a raster-based landscape simulation environment. Following is a subset of the general guidelines developed in this chapter that apply to the particular focus of the current effort:

- Provide for the simulation of individual entities. This is a main objective of the research project.
- Allow system components to learn and evolve. This will not be addressed in the current research, but will become a key component of future development.
- Allow for model components that respond without regard for any predefined equilibrium. This is fundamental to the capturing of animal ethologies. At the level of the individual there is no equilibrium for an animal is born, lives, and dies with the possibility of only transient equilibria.
- Allow for the simulation of ecological processes occurring at a wide range of spatio-temporal scales and ensure that multiple scales in time and space can be simulated simultaneously. For example, the raster-based landscape simulation might occur with a spatial resolution of 100 meters and a time-step of a week while the individual animals are simulated at spatio-temporal scales more appropriate for the behavior being modeled.
- Allow any given simulation component to alter its operational time and space scales. The SME simulation approach fixes its spatio-temporal scale, but the design of the animal-based simulation environment should offer the opportunity to dynamically alter scales.
- Provide for the capture of processes within processes.
- Do not force any particular hierarchy. The general purpose simulation environment must allow for the development of a wide variety of hierarchies. It must be possible to develop, for example, an ecosystem within an animal as well as an animal within an ecosystem.
- Explicitly recognize the heterogeneous distribution of landscape components. This is inherent to SME and to the notion of simulating individual entities.
- Allow movement between patches. Animals must be able to move across the landscape and encounter the differences associated with different locations.
- Allow modeling of individuals. This is central to the research effort.

- Use existing code. Clearly, the use of SME supports this requirement, but Chapter 3 will identify the use of other existing software as well.
- Standardize systems integration.

These overall design goals must be addressed in the design and development of software to support mobile entities and with the integration of that capability with SME. The next chapter will then build upon these guidelines with more specific objectives.

3 Objectives and Framework

The primary objectives of this work are to design and demonstrate concepts that will be able to facilitate future development of entity-based spatially explicit ecological simulation models (developed below in section 3.2) and to then explore requirements and techniques for linking such entities with population-based spatially explicit ecological simulation models (described in section 3.3). Requirements and objectives for these efforts must recognize and respond to the overall objectives developed in Chapter 2. Finally, the effort must recognize and work within the constraints (and opportunities) provided through software already selected to be part of the exploration of the larger effort. These are described in section 3.1.

3.1 I-STEMS Components

This section discusses the software already chosen for the I-STEMS design and prototype development. These now form a set of constraints for the subprojects, of which the current study is one. In total, they provide a very fertile environment, offering few real constraints to the imagination and the research objectives.

3.1.1 UNIX

The UNIX operating system has been chosen for the design and development of software within the larger program. It is a solid commercial product that commands a significant amount of attention in the research and development communities at major universities. Most existing significant ecological simulation software has been designed and developed for the UNIX environment. The UNIX operating system provides the backbone for networking across the Internet and running the majority of platforms that serve information to that network. All of the other software components listed below as choices for the design and development program run under UNIX and most were developed for the UNIX environment. Some developers anticipate that the operating system that is likely to be more available to potential markets for the software products that emerge from this program over the next 5 to 10 years will be Windows NT. Migration to

that environment should be relatively easy because it borrowed heavily from the UNIX experience.

3.1.2 GRASS

The Geographic Resources Analysis Support System (GRASS) is a general purpose geographical information system supporting both vector and raster import/export, analysis, and graphical cartographic output (Westervelt, et al. 1992). Recent additions to GRASS as well as ongoing research make this software environment very attractive to landscape modelers. Such additions, existing and planned, include support of real numbers in raster maps, thin-plate spline surface interpolations, perturbation of data fields for performing sensitivity analyses, establishment of a NULL value associated with maps, and development of multi-dimensional raster data to support 3-D and time-series data.

GRASS is an extensive public domain GIS that is enhanced with commercial extensions. Drawing from the framework of the UNIX environment, GRASS consists of a large number of command-line driven spatial analysis programs. This document cannot adequately describe the GRASS programs (or any other public domain or commercial GIS package), but a few key defining capabilities make this software very interesting to the modeler. These are discussed individually below.

GRASS, like UNIX, forms a programming language. The GRASS programs, in this respect, provide an extensive library of spatial analysis subroutines. Traditional GIS operations (Burrough 1986; Star and Estes 1990; Tomlin 1990) can be mixed and matched to form sophisticated terrain analyses. When performed iteratively, the GIS turns into a dynamic spatial modeling system. GRASS is especially well adapted to this type of modeling because it merges seamlessly with the UNIX shell-script approach to programming. This approach is much more accessible to land analysts than the much more difficult FORTRAN or "C" programming.

r.mapcalc is a particularly useful GRASS program as it provides a very flexible environment for performing a huge variety of GIS analyses. This program accepts a series of equations that use map names as variables. Each map contains a single value for each cell in a raster map. When the equations are executed by **r.mapcalc**, the program operates on one cell at a time. Map names in the equations are replaced with values corresponding to the current cell in that map. The results of the computation (or series of computations) are then placed into a result map. This map becomes part of the database and may be used as input to a GIS analysis program. In this manner, **r.mapcalc** (as well as other GRASS programs), when used iteratively, can generate a time-series of modeling output maps.

3.1.3 Spatial Modeling Environment (SME)

The Spatial Modeling Environment was introduced in sections 2.2.4 and 2.2.3. It has been under continual design and development for at least 5 years by Dr. Thomas Maxwell (Maxwell 1995; Maxwell and Costanza 1993). It is a raster-based landscape simulation environment that facilitates the simultaneous operation of cellular models in any number of cells. SME combines intuitive model building environments, parallel processing software capabilities, and state-of-the-art hardware environments to create a powerful spatial modeling environment (Maxwell and Costanza 1993). Figure 2 provides a graphical overview of the SME processes.

The basic characteristics of this modeling environment are as follows:

1. The modeling paradigm is cellular.
2. Time steps are of fixed length (user definable).
3. Models are constructed graphically.
4. Software programming is required for inter-cell movement of information.
5. A single cellular model is run on all cells in the spatial system simultaneously.

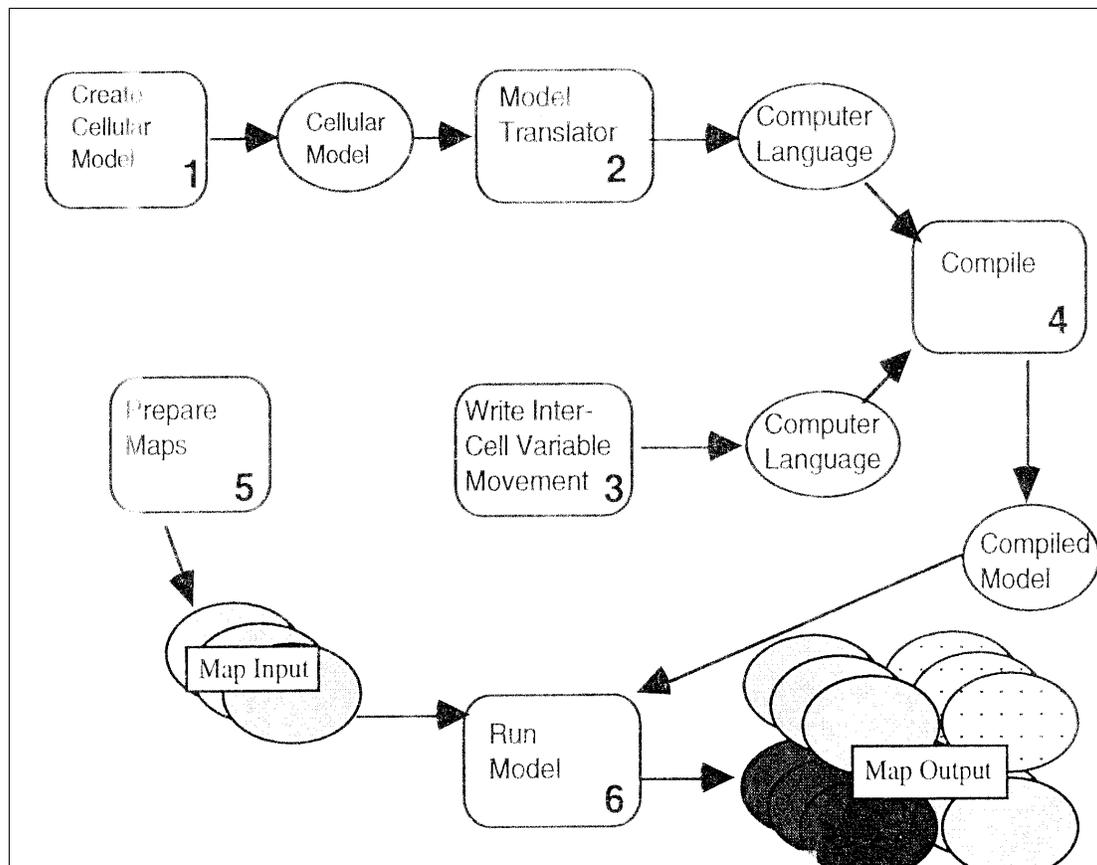


Figure 2. SME process.

The SME process steps are performed on a combination of Macintosh and UNIX software and hardware environments. When run completely in a Macintosh environment, a Hypertext interface guides the user through the model development and execution process. Work is underway to completely migrate the entire process to the UNIX and X-windows environment. The final model can be run in a variety of parallel processing environments. Each step of the process is described below:

Step 1: Create cellular model.

Step 1 in Figure 2 is where the basic cellular model is prepared by the modeler. At this stage the process is owned by the person who understands the landscape interactions that will be modeled. The modeling can be completed by a team, but the interface does little to facilitate serious collaborative efforts. The interface is provided through two different dynamic simulation software packages that run exclusively in the Macintosh hardware environment: Stella and Extend. Parts of previous models can be recovered to form the basis of new models, especially with the Extend software, which facilitates the development and maintenance of objects. Neither Stella nor Extend provide mechanisms for facilitating spatially explicit models. This requires the user to resort to more tedious code development to facilitate inter-cell exchange of information.

Step 2: Model translator.

Once the cellular model is developed, the model must be translated into a language that can be compiled in several different hardware/software environments (step 2, Figure 2). Different versions of SME provide interfaces to FORTRAN, C, and C++. Generally the cellular models are completely developed and mostly debugged in their native software system environments.

Step 3: Inter-cell movement.

Inter-cell movement is not facilitated by the software available in step 1. Step 3 (Figure 2) is accomplished through the traditional approach of writing computer code (in FORTRAN, C, or C++ depending on the version of SME in use). The modelers must communicate their requirements to someone familiar with software programming. The resulting code is then ready for the next step.

Step 4: Compile.

Once the cellular model and the inter-cell movement software have been generated, they are compiled (step 4, Figure 2). This is accomplished with some more

programmer-generated code and the standard process of compiling the various pieces and loading them all together into a single executable. The compilation process can be modified slightly depending on the targeted parallel processing environment. Potential targets include:

- A transputer array attached to a Macintosh.
- An array of Sun or IBM 6000 UNIX machines.
- A CM-5 Connection Machine.
- A CM-3 Connection Machine.
- A single UNIX workstation.

Step 5: Prepare maps.

A series of maps must be generated (step 5, Figure 2) to provide system state initialization for individual cells. Of course, the modeler must be cautious to ensure that the input is meaningful and properly matched with the model requirements.

Step 6: Run model.

Finally, the model can be run (step 6, Figure 2). Because the combination of the various submodels, inter-cell information exchange, and map input creates something distinct from any of the parts, a significant amount of full system debugging must be expected by any modeling team. The operation of the model can generate a tremendous amount of output in the form of system state maps. Generally only some of the available state information will be selected for some time steps. For example, water salinity might be selected from several dozen state variables for sampling every tenth time step. The output provides the principal window into the operation of the model and may be perused with any number of different scientific data set visualization tools.

3.1.4 IMPORT/DOME (I/D)

IMPORT/DOME, the Integrated Modular Persistent Object Representation Translator/Declarative Object Manipulation Engine (Morrison 1995) has been under development at USACERL and the University of Illinois at Urbana-Champaign. It was designed to be a powerful, flexible, and easy-to-use simulation language that is object-oriented, persistent, and event driven. IMPORT is a procedural language that allows programmers to specify object classes and the behavior of those classes. All actions are specified in the usual precise manner required by procedural coding approaches (used traditionally in Basic, FORTRAN, C, and C++). DOME provides a declarative side to the IMPORT/DOME language. It is based on concepts captured originally in the PROLOG language (Bratko 1990).

Here, the programmer does not identify sequences of actions to be taken, but instead provides statements of fact and statements of logical connections between and inferences from those facts. Expert systems are often built with such programming approaches. The combination of both procedural and declarative programming capability into a single language results in a very powerful and efficient environment for supporting dynamic simulations.

A fundamental aspect of IMPORT/DOME is its ability to simulate the passing of time by allowing objects to schedule the activities of objects in the system. An asynchronous "TELL METHOD" is used to facilitate this important capability. Effectively, during the operation of a compiled IMPORT/DOME program, a calendar of events is maintained. Activities are scheduled. During the operation of any given activity, the associated object may schedule other activities to take place at other times.

All variables used in IMPORT are strongly typed. That is, every variable must be declared where it is instantiated and wherever it is shared with other program components. It is virtually impossible to compile code that identifies something as one type and then attempts to use it as another type elsewhere. Multiple uses of a variable name are permitted; the context of the use of the variable establishes which meaning of the variable name is used.

3.1.5 C++

The C++ programming language provides the fundamental environment for all software development within this project. The SME package operates completely with C++. IMPORT/DOME code is translated into C++ code before it is compiled into operational code. This makes it possible to build SME and IMPORT/DOME capabilities into the same operational program.

3.1.6 Related I-STEMS Research and Development

The research effort discussed in this document focuses on the facilitation of discrete entity simulations in ecological modeling (see section 3.2). A number of associated efforts are planned or underway. Each addresses a subset of the goals developed in section 2.1 and 2.2.

Multi-processor and multi-computer - A major goal of the larger program under which this project is taking place is the ability to easily integrate a number of separately running simulation programs. At present, separate programs exist to simulate or model (1) landscape state, through GIS (2) overland water flow (3) flow

of water through stream networks (4) movement of subsurface water (5) forest growth (6) fire spread in natural systems (7) spread of diseases or pests, and other activities. Each simulation artificially holds constant state variables simulated by the other programs. Being able to link a number of simulations would help remove this requirement. Running such simulations will require the use of multiple processors and multiple computers. From this requirement comes the need for software that can move information efficiently between the simultaneously running simulations as well as keep the simulations running synchronously.

User Interface - Existing software simulations (listed above) are accompanied by their own user interfaces. Linking such programs into a common framework requires the construction of a common user interface, which helps to tie the various capabilities together for an end user. Software developed with a model-view-controller (MVC) approach is easily adapted as the view and control software are already developed as replaceable modular components. Other simulation models may require extensive efforts to turn off the existing user interface and replace it with the ability to look into and affect the simulation from programs running separately.

3.2 Facilitating Discrete Entity Simulations

As presented at the outset of Chapter 1, there exists a need in landscape-based ecological modeling and simulation to be able to deal with populations at low densities as individuals. SME (section 3.1.3) provides an excellent environment for capturing the simulation of populations represented as separate metapopulations associated with each cell in a regular grid-cell array. The author's experience with trying to capture the movement of individual entities (Westervelt et al. 1994) provided the impetus for searching beyond the Stella/SME software for a better and more complete approach for the simulation of distinct entities (individuals and spatially connected sets of individuals). Following are a set of defined objectives associated with this need. It is these objectives that are addressed by the current research.

Entity Ownership Retention - When designs become too large for a single individual to author, the project must be split into smaller components. These parts are assigned to individuals or to small groups. To maximize the potential success of the project, the authors (owners) of the individual components must retain ownership throughout the completion of the design and implementation. With any software development it is often necessary that the design be turned over to a programmer. The programmer is not a specialist in the application area and cannot be expected

to thoroughly understand the design. Too often ownership is lost by the original designer to the programmer. During debugging and modification phases it is the programmer who must not only change the software implementation, but must change the design. The resulting product may not be recognizable by the original designer — even if that person could decipher the software.

It is important that the designer retain direct ownership of the design and its implementation as far into the overall system development as possible. No amount of documentation can make up for the training, experience, and background upon which the designer bases decisions.

Variable Time Resolution - Mobile entities must be able to operate at variable time steps. The reasons for this objective are two-fold. First, limited computational powers require the smart use of CPU cycles. While it may be important to update an entity's position on an hourly basis during the day, a several-hour time step may be appropriate at night; a several-day or several-week time step might be adequate during times of hibernation. Secondly, different activities and actions occur at different time scales. For example, over the course of hours an animal might move around with respect to local food sources; over the course of days an animal may respond to changes in temperature; over the course of months, movement may include migrations or other large-scale movements; and over the course of decades change in spatial patterns of genetic material may reflect climate changes.

Variable Space Resolution - Similarly, space must be accessible at a variety of spatial resolutions. This requirement is linked with the variable time resolution requirement; in general, as the resolution of time increases, there is a corresponding increase in the resolution of space. For example, an animal that traverses X amount of space in time Y can traverse $2X$ space in $2Y$ time. Hence, doubling the time resolution is associated with doubling the spatial resolution. As any given entity changes its rate of movement across a landscape, the simulation environment should be able to compensate by adaptively changing the spatial resolution. Also, at any given time during a simulation, entities must be able to view and interact with varying resolutions of space as a function of their particular ethology. This requirement primarily addresses the need for computational efficiency. It also, however, reflects actual differences in the way different entities actually interact with their environment.

Mobility - It is imperative that entities have the ability to move across a landscape. Movement speeds and directions must be unique to each simulated entity. Movement is controlled by decision-making software algorithms developed as a part of each entity and is based on some combination of current and past internal and

external states. External states include the perceived state of surrounding entities and the physical characteristics of the surrounding space. It is anticipated that the DOME component of the IMPORT/DOME language will be used to capture expert-system information and analysis capabilities to facilitate movement decisions.

Scan and Evaluate Surrounding Entities - As entities move around their simulated landscapes, they must be able to detect and probe other entities in their immediate area of influence. To avoid reprogramming existing entities, each entity must come equipped with personalized rules for evaluating other entities. It is anticipated that interesting working simulations will attract the addition of new entities. Such additions should not require the return of all working system components to their authors for updates to deal with the new entity.

Entity Maintenance of Knowledge - As entities move about and interact with the landscape and other entities, they should develop and maintain knowledge about their environment. Knowledge provides an important role in computation efficiency (caching), and is important to the behavior of an animal. A home range, for example, can be thought of as an area that not only contains the requirements for sustaining an entity, but is known sufficiently well by that entity to allow for efficient collection of required resources (i.e., food and water).

Create New Entities - During the course of a simulation, entities must be able to "create" or instantiate other entities. This operation may come into play with the generation of propagules or offspring. It also occurs when an animal transforms from one type of entity into another. The dynamic creation of new entities during a simulation is critical.

Interactions With Other Dynamic Entities - Individual dynamic entities should have virtually unlimited potential for interaction with other entities. All of the different interactions found in nature should be possible within the fundamental design of the simulation environment. These include, but are not limited to predator-prey, mating, communications of all types, symbiosis, pollination, and movement of material and information.

User Control - Once a system is designed and created by an interdisciplinary group of scientists, the end user must have numerous opportunities to change the course of the simulation. This includes, but is not limited to initial system configurations, land-use scheduling, and run-time adjustments of various system components, including individual entities.

Sensitivity Analysis - Developers and users at all levels are continually responsible for participating in the analysis of the most important components of a simulation. Users must be able to experiment with various system components to determine future data requirements and system performance and utility improvements.

Tracking and Saving Entity States - During the operation of a simulation, various methods of tracking and saving of entity states must be available to the user.

Model Design via User Interfaces - This effort will not result in the development of sophisticated user interfaces, but such interfaces must be considered in the design of the system capabilities.

The approach used to address these objectives is presented in section 4.3.

3.3 Linking To Cell-based Landscape Simulations

In the future, mobile entities will be interacting with other simulated landscape components including vegetation, human activities, weather, storms and floods, and plumes of gasses, dusts, and pollens. A related effort in the design and development of I-STEMS is the creation of specifications for a software fabric that will allow disparate simulation software to run simultaneously. As that fabric does not yet exist, the current effort must rely on the creation and use of special purpose capabilities to link the dynamic and mobile entities with a dynamic SME-created landscape.

It is important that the mobile entities be developed and tested within a static landscape context. This reduces the number of variables to a level that allows efficient software debugging. Hence, the approach taken in this effort is to develop both a method for simulating the entities on a static as well as dynamic landscape. In both situations, however, the entities must be able to access information about and the status of the landscape. In particular, entities will want to receive a snapshot of the state of a current location. This information, combined with the entities' knowledge of other local entities and the internal state of the entity, will then affect the behavior and future state of the entity. Section 4.4 discusses the design used to link the dynamic entities with a dynamic landscape.

3.4 Summary

This chapter has drawn from the general objectives identified in Chapter 2 to develop specific requirements for the current research. For the support of discrete animal entity design and development, the research must support variable space and time resolutions, animal mobility, interaction with other dynamic entities, the creation of new entities, and the abilities to find and evaluate neighboring animals, to maintain information about other entities, and to track and save entity states. Consideration must also be given to sensitivity analyses support, the capturing of entity states, and the ability to view and control the simulation. Chapter 4 addresses these objectives, while the completed system is evaluated with respect to the objectives in Chapter 6.

4 Design

The objectives of this research as described in Chapter 3, and couched in a broader perspective presented in Chapter 2, are addressed through the approaches described in this chapter. These approaches recognize the design constraints and opportunities established with the current selection of certain software environments for I-STEMS described in section 3.1. The concepts of the overall design are developed in section 4.1. Sections 4.2 through 4.5 describe the approach to the development of key portions of software used in support of demonstrating the concepts.

4.1 Overall Design

The overall design is based on a merging and integration of two disparate simulation environments (see Figure 3). The well developed Thomas Maxwell product called SME (the Spatial Modeling Environment) provides an excellent environment for designing dynamic spatially explicit population models (section 3.1.3). IMPORT/ DOME offers a programming language perfect for capturing behaviors of discrete entities (section 3.1.4). These two environments must then be intermingled in a manner that ensures a standard simulation time frame and sufficient inter-system exchanges of information.

The primary focus of this effort is on the development of design specifications that

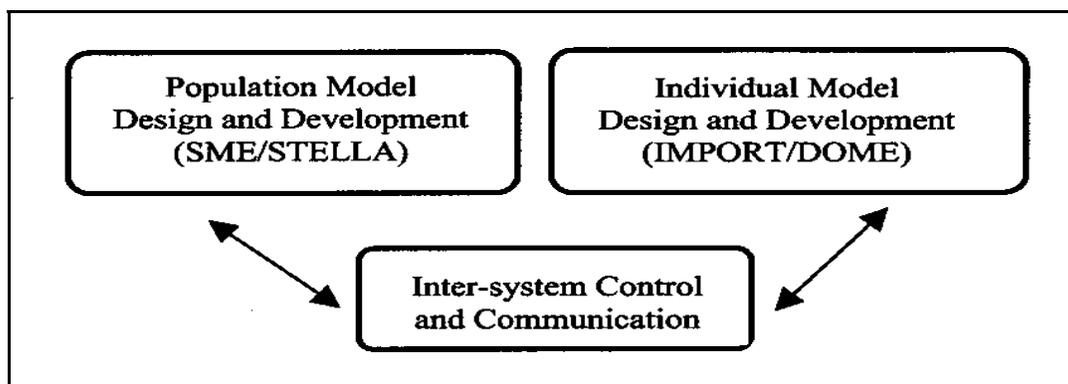


Figure 3. Overall design.

allow for the efficient development of mobile entities that can interact with static GIS maps as well as dynamic SME population models. Dynamic landscape modeling in SME is already available; the present work focuses on the other two boxes in Figure 3. Efforts involved within the top-right box in that figure are expanded in Figure 4, which are then developed in the following subsections. Note the main headings: Model Classes, View Classes, and Controller Classes. The classes are not only grouped easily under these categories, but this approach to software development provides easy upgrade paths to those components of the software that are most volatile: view and control. Separating the model from details involved with user input and output affords the opportunity to extend the life of the model when new input and output capabilities become available. Further, any given model may act as a controller for any other model. To facilitate this programming philosophy, each model object is developed with explicit calls (IMPORT methods) that provide for the input and output of information. Input calls are connected to controllers and output calls to viewers.

Classes involved with intra- and inter-animal communications are covered in

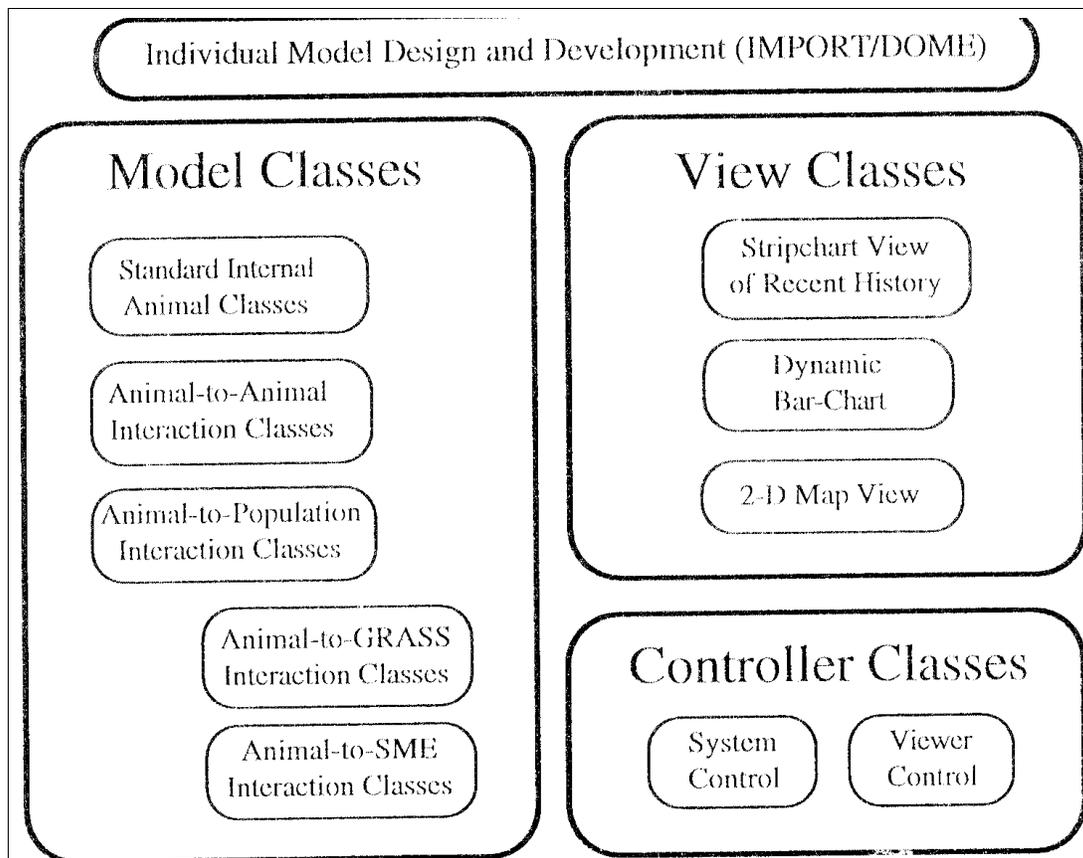


Figure 4. Overall IMPORT/DOME design.

section 4.3. Animal to landscape interaction classes are presented in section 4.4. View classes are discussed completely in section 4.5. In addition, a number of classes that provide more primitive capabilities required for this effort are presented in section 4.2. Figure 5 provides a framework presented from a model developer's perspective. Boxes are arranged in three rows and three columns. The top row captures capabilities that a modeler uses to develop models and are assumed to remain fixed for the complete community of system users. "GRASS Interface Libraries" and "SME's Stella to C++ Translation Code" were developed outside of the current effort and are adopted unchanged. The middle box in the top row represents all of the IMPORT/DOME classes depicted in Figure 4. Users of the combined capabilities (as demonstrated through sample models in Chapter 5) develop model components represented in the second row. These involve the development of static landscape maps in GRASS, optional development of Stella models to capture landscape dynamics using Stella and SME, and the development of animal classes and the main models using IMPORT/DOME represented by the middle box. The result is a set of programs that simulate the behavior of the designed entities within static and, optionally, dynamic landscapes (box in the last row of Figure 5).

Figure 6 expands the IMPORT/DOME section of Figure 5 into an object relation-

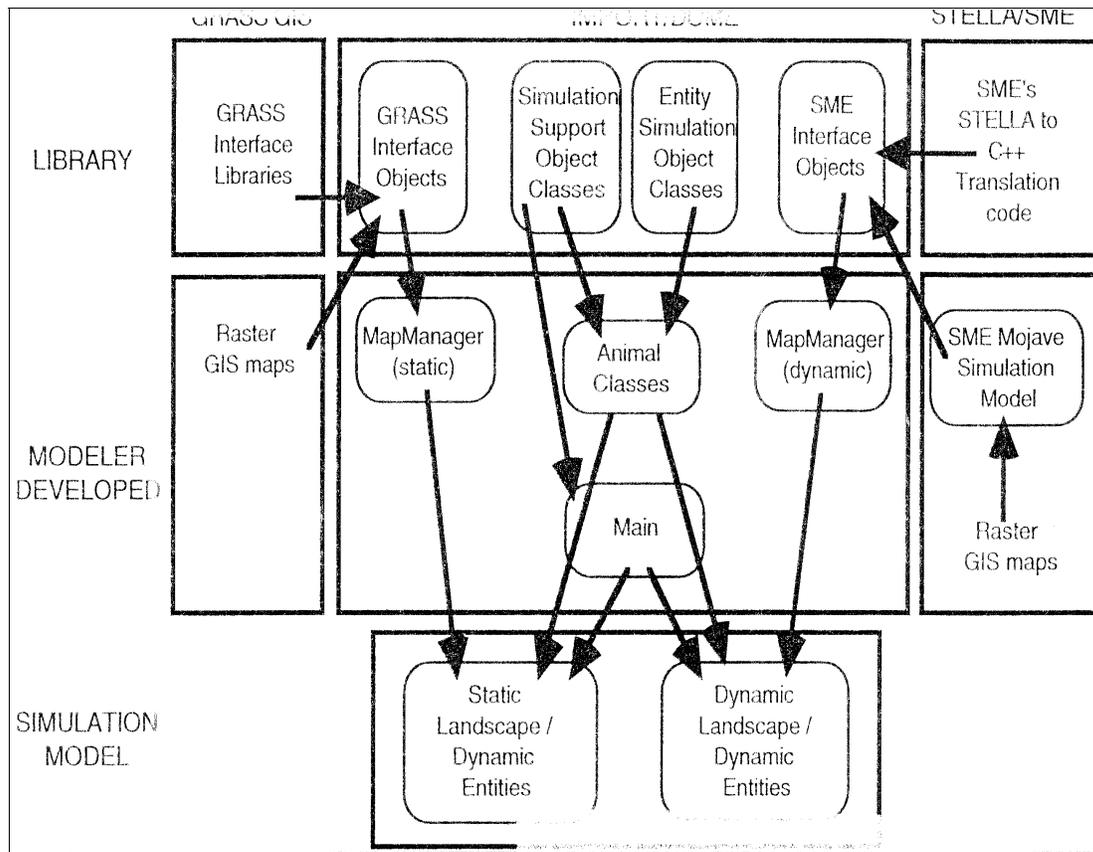


Figure 5. Approach to development of demonstrations.

ship diagram. Readers should continually refer to this figure as the individual parts are discussed in the remainder of this chapter. A large number of IMPORT/DOME support object classes developed for this project, like GraphFunction, LinkedList, and GaussRandomGraph, and various viewer classes, are not shown in Figure 6. Readers unfamiliar with object-oriented programming approaches need only be aware of a few concepts. First, classes are descriptions of objects. When the program is running, an object is instantiated (comes into existence) when memory is dynamically allocated based on a class. A class can be used to instantiate or create any number of objects of that class type. Classes can inherit other classes. For example, a class called animal may be inherited by a class called dog and also by another class called cat. The animal class provides the code that represents common processes between all animals. The dog and cat classes are subclasses of animal (which is a superclass of both). A dog is a (ISA in object-oriented terms) animal. Classes can also instantiate objects based on other defined classes for their own use. If it is important to simulate the behavior of the dog's heart, a heart class could be defined. Using this class, a heart object could be instantiated each time a dog class is instantiated and then be associated with that dog. In this case the dog has a (HASA) heart. If, for some reason, the cat needs to directly affect the dog, it is possible for the cat to know about the heart class in general and a dog's heart object in particular. Read the Figure 6 legend to distinguish among these different relationships that exist among the various object classes.

4.2 Design and Implementation of General Purpose IMPORT/DOME Classes

IMPORT/DOME (I/D), being a new programming language, offered few objects in support of simulation design and development. These included libraries of rudimentary graphics, random number generators, string manipulations, file input and output, list operations, and basic math and trigonometric operations. For this research a number of additional I/D classes were developed to support general purpose simulation. Appendix B provides a programmer reference for the following IMPORT/DOME support classes developed for this research:

- GaussRandom - A random number generator whose output distribution fits a gaussian distribution.
- GRASS - IMPORT/DOME code that encapsulates the C++ based GRASS_interface and GRASSMap code listed below.

Figure 6. Object classes for simulating mobile entities.

- GRASS_interface - C++ code that in-turn encapsulates the GRASS C subroutines for availability to IMPORT/DOME.
- GRASSMap - Associated closely with the GRASS_interface class, this class, written in C++, provides access to individual GRASS maps. Each map is treated as a separate object.
- GraphFunction - Class that allows model developers to express functions of a single variable through a series of line segments.
- SME - Encapsulation of complete SME-generated spatial landscape simulations based on Stella models. Forms the basis of the dynamic version of the MapManager, which is directly interchangeable with MapManagers developed and based on the GRASS classes (above).
- LINKEDLIST - Class for the creation, management, and access of linked lists.
- LINK - Link class used by LINKEDLIST objects.
- StripChart - Object that generates pen plot type graphical output based on data dynamically sampled from running simulations. Up to ten pen plots can be supported on any one strip chart. The horizontal axis is time and the vertical axes are user defined. (Described in section 4.5.1.)
- PenPlot - Up to ten of these objects are managed and displayed by a single StripChart object.
- RandNormGraph - Provides a one-step approach to a graph-type function. A RandNormGraph object is instantiated with any number of graph coordinates pairing an independent variable with a mean and a standard deviation of the dependent output variable. The independent variables must be monotonically increasing. Calls to the ReturnValue method returns a random number chosen with respect to the normal and standard deviation associated with the input value. This is valuable for generating stochastic temperature, rainfall, and other environmental data.
- Region - Class that provides an efficient means for keeping track of the whereabouts of all mobile entities. Individual entities can query the region for a list of entities within a range (provided by the querying entity).
- RasterManager - Class that provides the standard link between instantiations of entities and the static and dynamic versions of MapManager (below).
- Register - Object class that feeds unique numbers used to distinguish between individual entities in the simulation environment.
- TimeManager - Object class that provides information regarding the current time (day, month, year) and associated simulation time steps for each of these units.
- Precip - Object class that provides the means to generate stochastic precipitation amounts for any time of the year. Uses the RandNormGraph class to generate variable storms based on information about storm frequency and

intensity as well as average rainfall and associated standard deviations. A single precip object is used to generate rainfall for an entire region.

- Temperature - Object class that generates temperature based on knowledge of local averages and standard deviations.
- ButtonXGraph - Subclass based on the IMPORT/DOME standard (but primitive) XGRAPH. XGRAPH provides a handful of METHODS for generating graphics through UNIX X-windows. ButtonXGraph essentially adds the ability to place very simple buttons on the screen providing primitive GUI control. Additions to the IMPORT/DOME standard C++ routines underlying the XGraph object were made to support the operation of these buttons. Hsuan Chi Lu contributed to the development of this code and has developed newer capabilities that are now eclipsing these primitive efforts with much more complete and more flexible GUI capabilities in support of IMPORT/DOME developed simulations.

In addition, the following simulation support objects, though not sufficiently standard for inclusion in a general purpose library, are easily modified to support a wide variety of simulations.

- MapManager - IMPORT/DOME object class that provides interface to simulation-specific landscape information. For the demonstrations documented in Chapter 5, the MapManager is a superclass of the more general purpose RasterManager. All access and modification of maps are facilitated through the MapManager. Two versions of the MapManager (they provide identical METHODS) can be created: one for access to static raster maps and another for access to SME generated dynamic maps. For the demonstrations in Chapter 5, the initial version of the MapManager developed to support interface with the dynamic SME maps was written by Douglas Briggs.
- MapView - An object class that displays the current location and, optionally, states of the dynamic entities. Entities are displayed on a static picture backdrop derived from a raster GIS image. Slightly different versions are used in the demonstrations presented in Chapter 5. (Described in section 4.5.3.)
- AnimalView - A viewport into the current state and current environment of mobile entities. It probes AnimalInfo objects as well as the simulation's RasterManager for information to display in bar chart format. (Described in section 4.5.2.)
- WeatherView - A viewport using the StripChart object to view the weather (precipitation and temperature).
- ViewControl - A simple interface that allows the user to generate AnimalViews and WeatherViews during a simulation. The ButtonXGraph is the superclass supporting the graphical display.

- Control - Similarly simple, it provides a simple pause/start capability for rudimentary simulation control.

4.3 Classes Supporting Mobile Entities

This work focuses on facilitating the design, development, and operation of mobile entity simulations. At the center is the specification of the software components that, together, allow the instantiation, control, and simulation of entities. Figure 7 (a subset of Figure 6) graphically depicts the basic classes developed to support and demonstrate the mobile entities. Each animal class is developed as a subclass of an AnimalInfo and a LandInfo class. Its power is discussed in section 4.3.1. The rest of the object classes are developed individually for each animal, although developed classes can be used as templates for the development of classes supporting other animal classes. The LandInfo class is required if the animal interacts with the static or dynamic landscape information captured in GIS maps or Stella/SME cellular models. Animals developed as part of this work incorporated the LandInfo as a superclass. The remaining classes identified in Figure 7 are optional, and indeed suggestive of a wide variety of objects used by the animal class. These are discussed in section 4.3.2.

Figure 7. Set of classes to support a mobile entity type.

4.3.1 The *AnimalInfo* Class

A main goal of the effort was to facilitate the rapid development of animal entities. The approach used in this study was to divide the animal into a set of classes, each of which becomes either a superclass of the animal class through multiple-inheritance or an associated class either instantiated by or simply known by the animal class. The advantage of this approach is that any given animal may be constructed with any number of different parts, or building blocks. There may be more than one version of any given building block, thereby providing the model developer with a rich set of components from which to choose. There is one main superclass, *AnimalInfo*, developed for this project. It provides two fundamental capabilities intended to be used by all animals. First it is a single location for accessing and changing certain standard facts associated with a given animal type. This cache of facts provides a common area for the animal support classes discussed in 4.3.2 to communicate with one another. Second, it provides the window into the facts associated with all other animal entities in a simulation. Associated with this window is a primitive query capability.

The following facts are currently maintained by the *AnimalInfo* class:

- **Region** - This is a pointer to the simulation's region manager that accepts the registration and re-registration of the location of individual entities. It is used by *AnimalInfo* to register the location of the associated animal and is used to query for a list of animals within a certain range of the animal.
- **RegionKey** - A pointer to the Region's stored information about the animal which is used to unregister the animal as it moves or dies.
- **NorthVel** - The current velocity in meters per day of the animal in the north-south axis.
- **EastVel** - The current velocity in meters per day of the animal in the east-west axis.
- **ID** - A unique number assigned to the animal for individual labeling and identification.
- **Prey** - The Easting location (meters) of the last prey item (or 0).
- **PreyN** - The Northing location (meters) of the last prey item (or 0).
- **E** - The Easting location (meters) of the animal (or 0).
- **N** - The Northing location (meters) of the animal (or 0).
- **Sex** - The sex of the animal.
- **Mass** - The animal's mass in grams.
- **Range** - The radius in meters of the smallest circle bounding the animal's home range.
- **Age** - Age in days.

- Length - Length in mm.
- Carniv - A 0-100 index; 0 is completely vegetarian and 100 is completely carnivorous.
- DT - Time between animal updates in IMPORT/DOME simulation time units.
- Stress - A 0-100 index; 100 is very stressed (as determine by animal-specific software).
- dStress - The most recent change in stress per day.
- Satis - A 0-100 index; 100 is very satisfied (as determine by animal-specific software).
- dSatis - The most recent change in satis per day.
- Dead - Set to TRUE if the animal is dead and FALSE if alive.
- Name - A string representing the type of animal. This can be used to recognize others of the same type.
- Activity - A string describing the current activity of the animal. No standard activities have been chosen. Currently used for labels and for animals recognizing their current activity state and modifying behaviors accordingly.
- Neighbors - A list of other animals within a specified range returned on demand from the Region manager.
- Fertile - The simulation day on which the animal was fertilized (or 0).
- QueryResult - List of animals (via AnimalInfo information) meeting the query specifications identified.
- Color - Color used for displaying animal. The UNIX X-windows program "showrgb" lists some optional colors.
- QueryLengthHi - The highest length accepted in a query.
- QueryLengthLo - The shortest length accepted in a query.
- QueryMassHi - The heaviest mass accepted in a query.
- QueryMassLo - The lightest mass accepted in a query.
- QueryFemale - Whether or not females are desired.
- QueryMale - Whether or not males are desired.
- QueryAgeHi - The oldest age accepted in query.
- QueryAgeLo - The youngest age accepted in query.
- QueryCarnivHi - The highest carnivorous index sought in query.
- QueryCarnivLo - The lowest carnivorous index sought in query.
- QueryDeath - Whether or not the query is seeking live or dead animals.

Another one of the design goals of this project was to facilitate the ability to add new entities to an environment without explicitly revisiting all existing simulation entities for the purpose of "teaching" each about the new entity. It is possible to develop entities that respond to other entities based on their name. For example, dog entities could be developed to react to cats because they are called cats. Using this approach, dog entities must be explicitly reprogrammed if another animal (e.g.,

a raccoon) is added to the simulation. Alternatively, if the dog entity was developed to respond to the characteristics of creatures it encounters, then the new raccoon objects might not require a reworking of the dog object. The raccoon, like the cat, would be asked to reveal certain features about itself (captured in the AnimalInfo fact list) and the dog would respond accordingly. The AnimalInfo class provides all animals with a consistent set of facts through which animals can evaluate each other. The management, updating, and use of that information will be peculiar to each individual animal.

A second way that animals can communicate with one another is to act on each other. There are two approaches currently available to facilitate this kind of communication. First, the information maintained by the AnimalInfo class can be modified by any animal. This is not recommended for general practice. However, a common approach used in the demonstrations (Chapter 5) is for one animal to kill another by setting the other's "Dead" variable in AnimalInfo to TRUE. Another way is to provide METHODS in AnimalInfo that allow an animal to respond to some action initiated by another animal. For example, the AnimalInfo class provides a method called BeAttackedBy, which takes as an argument the AnimalInfo object of the attacking animal. By default, the METHOD simply returns "TRUE" to indicate that the attack has resulted in death. IMPORT/DOME allows the animal classes (as a subclass of AnimalInfo) to override this method and provide a BeAttackedBy, METHOD that might evaluate the current terrain potential for hiding or evasion, the motivation of the attacker, along with its size, weight, and carnivorous rating before deciding the outcome of the attack. The results of unsuccessful attacks might be saved in a memory database that provides a persistent knowledge associated with the type of animal that attacked. In addition to attacks, a whole series of other actions of one animal upon another will certainly be developed. Such actions will include feeding, protecting, communicating, and intraspecific fighting.

4.3.2 Optional Animal Classes

The previous section covered the important and required AnimalInfo class. Figure 7 shows this class along with an assortment of optional classes that are used by, but not superclasses of, an animal class. Figure 8 shows these classes as optional building blocks for the construction of various animal classes. These optional "ANIMAL CLASS BUILDING-BLOCKS" operate as stand-alone objects which, because they all communicate with other animal building-blocks through the shared AnimalInfo class, are interchangeable. For example, if a newly developed cat class uses a "health" class that is better than the equivalent class developed for dogs, then the dog class can be readily switched to use the cat-developed health

Figure 8. Animal class hierarchy schema.

object class. Ideas for the functionality of some of these optional classes are developed below.

- **Eating and Drinking** - Eating and drinking are fundamental activities for all animals. Such activities are handled here. The desire to eat and drink may be managed within this object class and may be further based on food opportunities locally available as a function of the current location.
- **Health** - The health class will manage animal stress levels based on food and water intake, current and historical temperatures, and stressful interactions with other entities. Stress levels may be captured by any number of simulated indicators ranging from fat reserves, blood chemistry, or even psychological measures.
- **Learn Surroundings** - Animal movement is undertaken in response to a picture or view of the surroundings. Knowledge of the surroundings is based on both the information available to be known and the time invested in learning. This class manages how much is known about the surroundings; the results are then available to other classes for making decisions. In particular, the movement classes will rely on what is known about the surroundings.
- **Movement** - Movement is a key component of animal simulations. A movement class captures the rules for movement which are, in turn, a function of a potentially large number of facts about both external (environmental) and

internal states. These facts can be processed through an expert system defined through DOME-captured rules and facts, or through procedural algorithms expressed in IMPORT.

- **Reproduction** - The reproduction class captures the generation of propagules, the laying of eggs or bearing of young, and the change from one life stage into another. It also may capture behaviors, feedings, or other energy expenditures involved in the rearing of young or maintenance of social bonds.
- **Activity** - This class can be construed as a helper class to “movement.” It provides the analysis to identify whether an animal is active or resting through hibernation or aestivation.

Notice in Figure 8 the explicit opportunity for developed animal classes to be used as superclasses. The design approach and object-oriented programming principles provide the opportunity for model developers to create generic animal classes that are then slightly modified through inheritance to capture peculiar details associated with different animals.

4.4 Animal-to-Population Interaction Classes

The previous section dealt with the abilities provided for entity-to-entity interactions. These entities, in a landscape setting, must be able to recognize differences across a heterogeneous surface. Figure 9 (a portion of Figure 6) diagrammatically shows the relationship between the classes involved in this process. The main IMPORT/DOME program instantiates a single RasterManager and any number of Animals (and possibly any number of different kinds of animals). The RasterManager uses either the static or the dynamic MapManager as a SuperClass that provides information associated with the landscape. Remember that the static version is preferred for the initial design, development, and testing of animal classes because of significantly increased simulation speeds and a decrease in the number of variables changing simultaneously. Each instantiated animal has its own LandInfo as a superclass that provides the animal’s link to the map information through the single instantiated RasterManager. Through this link, land information can be queried and could be modified (e.g., a herbivore decreasing the cover of vegetation). Each of these key classes is discussed individually below.

4.4.1 LandInfo Class

The LandInfo class is developed specifically for each particular animal class. It may simply fetch information directly from the RasterManager as required by the animal. However, it exists to provide a key information preprocessing step for

Figure 9. Interaction between animal and landscape classes.

converting the mapped information before providing the information to the animal. For example, an animal may simply want to ask of the landscape how much herbivorous food is available, but the RasterManager (through its MapManager superclass) is simply able to provide percent land cover for grass, forbs, and trees. The LandInfo may then fetch this information and through an algorithm developed specifically for that animal convert this information into available food. If appropriate, an index on food quality could also be generated.

4.4.2 RasterManager Class

Information about the landscape is received by the LandInfo class through the RasterManager class. A single RasterManager object is instantiated at simulation start-up time. Its address is provided to each instantiated animal (of any and every type) and then associated with the LandInfo superclass. The RasterManager is a library class and is used unchanged in any simulation. The actual access of raw landscape information is facilitated through either a dynamic or static MapManager superclass described below. The METHODS provided by the RasterManager are generic and allow such things as map coordinate conversions, map extent information, and the averaging of raw data across the circular ranges associated with animals. Details on these METHODS are found in Appendix B.

4.4.3 MapManager Class

The class MapManager provides the single interface to static or dynamic landscape information. Currently these classes must be developed specifically for each simulation. Versions can of course be cloned from working simulations and modified to access the particular landscape information that will be supplied to the simulated entities. In the development of landscape simulations with entities, typically a static and a dynamic version of the MapManager will be created. These versions must appear identical to the rest of the system with respect to MODULE, OBJECT, and METHOD names. Being identical in this manner, they can be interchanged with one another without any modifications to any other part of the simulation software. METHODS associated with the MapManager class provide (1) access to information about certain data associated with the landscape (2) optionally, metadata associated with this information, and (3) whether or not a given coordinate is within the area being simulated.

For the demonstrations developed in Chapter 5, both static and dynamic MapManagers were developed. The static version opens, at run-time, the following GRASS maps, which are then randomly accessed during a simulation:

- Elevation
- Study area
- Percent brown vegetation
- Percent green vegetation.

The dynamic version opens a Stella/SME simulation, which in-turn uses the following GRASS maps taken from the same region as the maps accessed through the static version of RasterManager (some maps are reproduced in Appendix A):

- Vegetation density
- Elevation
- Slope
- Aspect
- Average water content
- Study area.

Both versions then return the same information to the rest of the simulation: temperature, precipitation, slope, aspect, elevation, percent green cover, and percent brown cover. However, they may provide that information through different processes. Elevation is a constant for both RasterManagers and is merely pulled from the elevation map for the static version and from static information maintained by the dynamic version. Slope and aspect are also static information in both and are accessed in the same manner as elevation in the dynamic version.

The static version of MapManager uses an IMPORT/DOME function to convert elevation information to slope and aspect data. This could easily be changed to access information directly from GIS-based slope and aspect maps. Percent green and brown vegetation is retrieved by the static version directly from GIS maps and, in the dynamic version, from dynamic stock variables maintained by the Stella/SME model. Finally, temperature and precipitation are dynamically computed by Stella/SME for the dynamic MapManager. Because of the key role that can be played by temperature and precipitation in entity simulations, the static MapManager actually uses specially developed temperature and precipitation objects to compute precipitation and temperature regionally. The regional temperature is further adjusted to take into account latitude, slope, and aspect of any given location.

4.4.4 Synchronizing SME and IMPORT/DOME Simulations

Besides facilitating the exchange of information between Stella/SME and IMPORT/DOME, the dynamic version of the MapManager provides another key requirement associated with the merging of these two software environments. Figure 10 diagrams the two fundamental needs associated with merging concurrently running simulations: information exchange and timing control. This figure suggests four different simulations running concurrently: a raster-based simulation (like Stella/SME), a mobile object simulation (like that developed here with IMPORT/DOME), traditional GIS analyses, and perhaps some other simulation (overland water flow simulation, economic simulation, or some geologic process). The user and the individual models communicate with one another through a master calendar and one or more information brokers.

Individual models can be designed and developed using the Stella/SME environment and the IMPORT/DOME language. Each simulates the passage of time for the model using software control techniques. When combined, a single time manager must synchronize time for both simulation components. In collaboration with this project, Dr. Thomas Maxwell modified SME in such a manner that the main controlling (timer) routine could be separated from the system update code. This main program is relatively simple because time, for SME code, passes in fixed time steps. A very short main routine simply initializes the simulation and then loops through a user-specified number of time steps. The C++ routines used to accomplish these tasks were encapsulated in the SME object class (details in Appendix B). This then allows the "Master Calendar and Control" requirement (Figure 10) to be accommodated through IMPORT/DOME.

Figure 10. Conceptual layer interaction.

4.5 View Classes

The Model-View-Controller system development paradigm being used in this effort requires that model objects not communicate directly with user peripherals. Instead, each class offers external objects the opportunity for various inputs and outputs. This provides the opportunity for developed “view” classes to probe objects and then provide the returned information to a human operator through appropriate visualizations. It is anticipated that these classes will be the most volatile and short-lived classes of the system. As distinct objects they can be removed and replaced without any requirements to modify the model objects being visualized.

View classes can be numerous because of the multi-dimensional nature of dynamic simulations. The system generally works with 2-D landscapes, but may involve the third dimension of elevation. Time provides a key dimension. Every state variable is associated, through a simulation, with a time-series of values. Each object is associated with a number of state variables, equation constants, and interim computations. Further, each object may be composed of a hierarchy of other objects; and each object may be associated with other objects in a variety of ways.

Allowing the user to visualize key components of the system in an efficient manner is the challenge that results in a number of view classes.

The following classes were developed. Each is discussed with respect to its intended utility and its particular “slicing” of the information space.

4.5.1 Stripchart

Visualizing change over time is very important and is facilitated here with a stripchart simulation object. Up to ten variables using a variety of colors can be plotted over time. A series of data can be plotted as continuous lines or as points. In a typical simulation run, a number of stripcharts may be displayed — each rendering information about a number of key system parameters. Figure 11 shows an example stripchart viewport actually generated by a StripChart object. In this case, temperature in units “degrees C” are plotted along with rainfall in units “mm”. Any number of these objects can be instantiated.

4.5.2 Bar Chart

For visualizing the state of a given entity in the system at any given time, a bar chart object is provided. Any number of values associated with the internal and/or external state of an entity are displayed as changing values printed in text juxtaposed with a dynamic bar providing a quick visual rendering of the current magnitude of the value. A sample bar chart visual is shown in Figure 12.

4.5.3 Map

Perhaps the first visualization that landscape simulation developers consider is a

Figure 12. Bar chart viewer example.

dynamic representation of the landscape in a map format. Within this effort, a relatively simple map version allows overlaying the current positions of selected objects on a picture of the landscape. Entities are represented by preprepared small images. For this effort, those images are simple renderings of different colored marbles. When it is time to update the image, the backdrop is laid down followed by the placement of the marble images centered on the location of the various entities. Different colors may be used to represent different information about the entities (e.g., type, age, health). Figure 13 shows an example snapshot of the position of a single entity displayed on a shaded-relief picture of the associated terrain. Any image representing some set of landscape characteristics could be displayed as a backdrop.

4.5.4 Desired Viewers

The viewers identified above can be augmented in the future with additional capabilities that might include the following:

- Charts - Like the stripchart graphics, this viewer would provide time series information about a number of system variables, but would display the information in tabular form.

- Dynamic maps - The current map object uses a preprepared picture of the landscape as a backdrop. Rendering the landscape by dynamically displaying variable landscape information will be important to most system users.
- Time-series maps for playback - All current viewers provide snapshot views of the state of the system. None of the information is stored (except for some amount of recent historical information associated with the stripcharts). Mapped information can be saved by SME and viewed at the end of a simulation. Because simulations can take a very long time to complete, users will want to peruse time-series output during the simulation.
- Time-series state-variables for playback - Similarly, time-series information associated with the state of system objects is important to review during and after a simulation.

4.6 Summary

The design components of the system described here address the requirements specified in Chapter 3. As designed, animal objects can move around a landscape, update their positions at dynamically set time intervals, and interact with as much surrounding terrain as is appropriate at any given time step. In this design, animals are able to use dynamically varying circular home ranges to query the landscape and the surrounding entity space. Interaction with other entities is

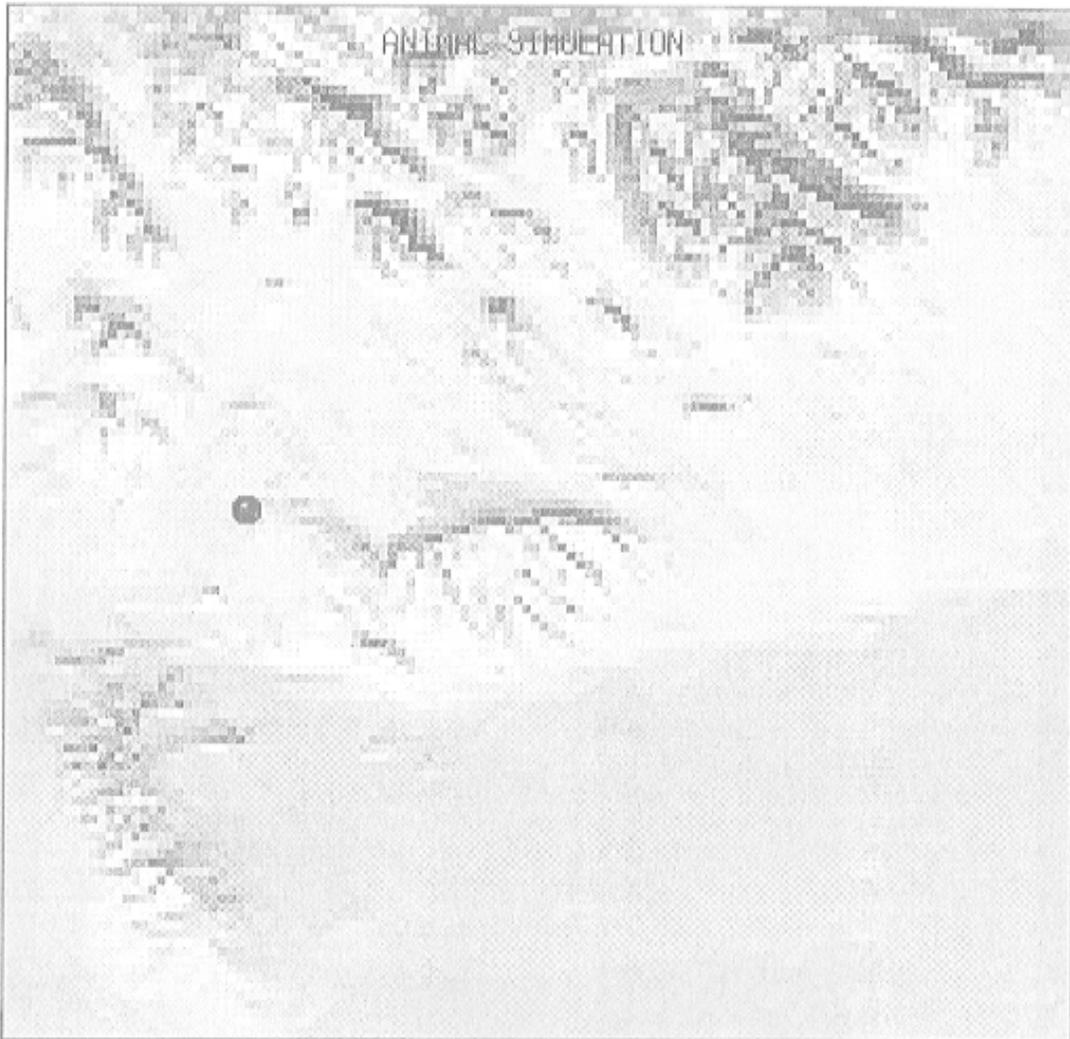


Figure 13. Dynamic map viewer example.

accomplished through sharing, between entities, of a fixed set of information about the animals. Any animal encountering a second animal is able to probe (and when appropriate change) this data set. Entities are permitted to instantiate (give birth to) new entities. User interfaces are provided to give examples of how the dynamically changing entities can be viewed in simulation time. Other viewers can be created to easily capture system state in files for post-simulation viewing and analysis. The following chapter demonstrates some of these capabilities through

proof-of-concept demonstrations. Evaluation of the functionality of the design is then discussed.

5 System Demonstration

Through a series of “proof-of-concept” demonstrations, this chapter discusses the capabilities and flexibilities of the concepts designed and developed in Chapter 4. These demonstrations were developed to be realistic for the purpose of readily communicating the potentials of the designs. The reader is cautioned not to view these examples as ecologically defensible or as having any substantive utility in making land management decisions. It is anticipated, of course, that future collaborations between the author, ecologists, and land use managers using the design concepts developed here will result in simulation capabilities that are useful in making future land-use management decisions.

To test and evaluate the design approaches, proof-of-concept demonstrations were developed. They were inspired by recent work on the Mojave Desert being conducted by Dr. Anthony Krzysik, USACERL, Dr. Bruce Hannon, University of Illinois, and myself (refer to Appendix C). A sample landscape was selected from an area near the southern border of Fort Irwin, CA, in the Mojave Desert. The region occupies the south-central portion of the installation with the following Universal Transverse Mercator (UTM) coordinates in the zone covering California:

UTM north	3898600
UTM south	3886600
UTM west	533000
UTM east	545000
Cell resolution	60 meters
Rows	200
Columns	200

The region was chosen because of its combination of available data, diverse terrain, relatively high densities of desert tortoises, and related local efforts focused on modeling and simulating the tortoise habitats in this area. While a related effort to simulate tortoise populations (Westervelt et al., 1998) worked with 1-kilometer cells covering the entire installation, a 60-meter resolution was adopted for this effort.

The efforts required to create these demonstrations are captured in the middle row of boxes in Figure 5. This involved the development of static raster maps (left box of that row), the development of a dynamic landscape simulation model using Stella and SME (right box), and the development of animal entities and simulation control software (middle box). The activities accomplished with each of these efforts are discussed in the following sections, which led to a number of working demonstration models. Development of each of the models followed a common approach outlined in section 5.1. Sections 5.2 through 5.5 discuss the unique development and output associated with a number of demonstrations. Finally, section 5.6 provides comments on these proof-of-concept demonstrations.

5.1 Common Development

The proof-of-concept demonstrations developed for this project are presented individually beginning with the next section. They were all developed within a common framework that begins with the specification of digital landscape maps for a subsection of Fort Irwin as described above. The common elements of these demonstrations include GIS maps (section 5.1.1), a dynamic landscape model developed with Stella (section 5.1.2), static and dynamic RasterManager classes (section 5.1.3) and simulation specific viewers (section 5.1.4).

5.1.1 GIS Maps

A set of raster GIS maps provides both a picture of the landscape used in the static demonstration and a starting point for the dynamic demonstration's landscape. The maps, their uses, and their sources are indicated below. Many of these digital maps are reproduced as images in Appendix C.

- Elevation, slope, and aspect. These maps were derived from U.S. Geological Survey (USGS) digital elevation models (DEM) from 7.5-minute quadrangles. Douglas Youngs, working for USACERL in 1992, processed the raw data to combine maps delivered in both feet and meters into a single DEM representing all of Fort Irwin at a post spacing of 30 meters. The GRASS 4.1 program `r.surf.sor` was also used to interpolate across all data dropout areas resulting from original raw data errors, and any incomplete data where edges of raw maps were sewn together. Slope and aspect maps were created using the GRASS 4.1 `r.slope.aspect` program.
- Vegetation density maps. Vegetation maps did not exist or were not available at the outset of this project. Access was available, however, to LCTA (Land Condition Trend Analysis) program data. LCTA is an Army standard program that collects detailed landscape information on randomly located 100-meter by

6-meter transects at military installations (Tazik et al. 1992). Field teams visit these sites annually to record such details as densities of plant species, amount of disturbance, and some animal information. Shrub, grass, forb, and overall vegetation density data derived from LCTA data were available for 168 of the 176 transects. To full-coverage maps of the study area, these ground measurements were correlated to Thematic Mapper (TM) satellite imagery, elevation, shade, distance from roads, and total upstream area using a back propagation neural network simulation program (Westervelt 1990; Westervelt 1994). To meet the objectives of the Stella/SME-based landscape simulation models, these maps were converted to “brown” and “green” vegetation maps (see Figure 14) required by the tortoise habitat simulation model developed by Westervelt et. al. (1998).

- Tortoise density map. Development of the tortoise population model used by the Stella/SME simulation model, which provided the precursor to the landscape simulation component of the demonstration models, required initialization with estimates of desert tortoise densities. Such maps did not exist, but like the requirement for vegetation density maps, density measurements from random transects did exist. Anthony Krzysik led field teams during the 1980s to collect tortoise density estimates at Fort Irwin (Krzysik 1994; Krzysik and Woodman 1991). Using the neural network technique, tortoise densities measured in 1989 were converted to full-coverage maps (Wu and Westervelt 1994). These maps guided the placement of tortoises in the simulation models developed and described below.

5.1.2 Stella/SME Model Review

To provide a demonstration of replacing a static landscape with one that is dynamic, the vegetation and weather components of the static map are replaced with dynamic models running in the SME environment. These were borrowed from a study I oversaw with Bruce Hannon, and conducted at the University of Illinois (Westervelt et al. 1998; Westervelt et al. 1995). A large raster-based cellular model was developed as a number of submodels. These included general precipitation (Figure 15), temperature (based on time of year, slope, aspect, elevation, and latitude), surface water, soil moisture, vegetation community composition, and brown/green vegetation (Figure 14). Each figure depicts a Stella sector that has been used to capture the dynamics of a different submodel. Solid rectangles are the stock, or state, variables, which are carried over from one time-step to the next. Stocks are changed with flows (in and/or out) represented by lines with arrows connected to circles, which represent “valves.” The opening of the valve is controlled by an equation associated with each valve (not seen in these diagrams). Each equation is a function of a number of stocks, constants, or other equation

results represented by arcs flowing from the source of the information to the circles. Some circles are not associated directly with flows and are called converters. The results of their equations provide input to other equations. In Stella, the equations are accessed by a double-click on the associated icon. They are typed in by the modeler for any single convertor or “valve” after it is selected. Stella is effective in multidisciplinary efforts as it provides an easy-to-learn language that can be mastered by people with a wide variety of professional backgrounds.

5.1.3 MapManager, Static and Dynamic

The static MapManager provides the dynamic entities with a landscape framework within which to behave. It is designed to provide a static landsurface within which to design, develop, and test entities that will eventually interact with a dynamic landscape simulation. Once animal objects are performing properly on the static landscape, the static MapManager class is replaced with a dynamic MapManager class that provides a set of class METHODS with identical names and calling syntax. The modeler provides the following capabilities in the MapManager:

- Access to landscape conditions. These conditions are mostly static and are provided through direct and indirect access of GRASS raster maps, which are read during the simulation. For the demonstration, percent land cover maps for shrubs, grasses, and forbs are read directly from a native GRASS database.

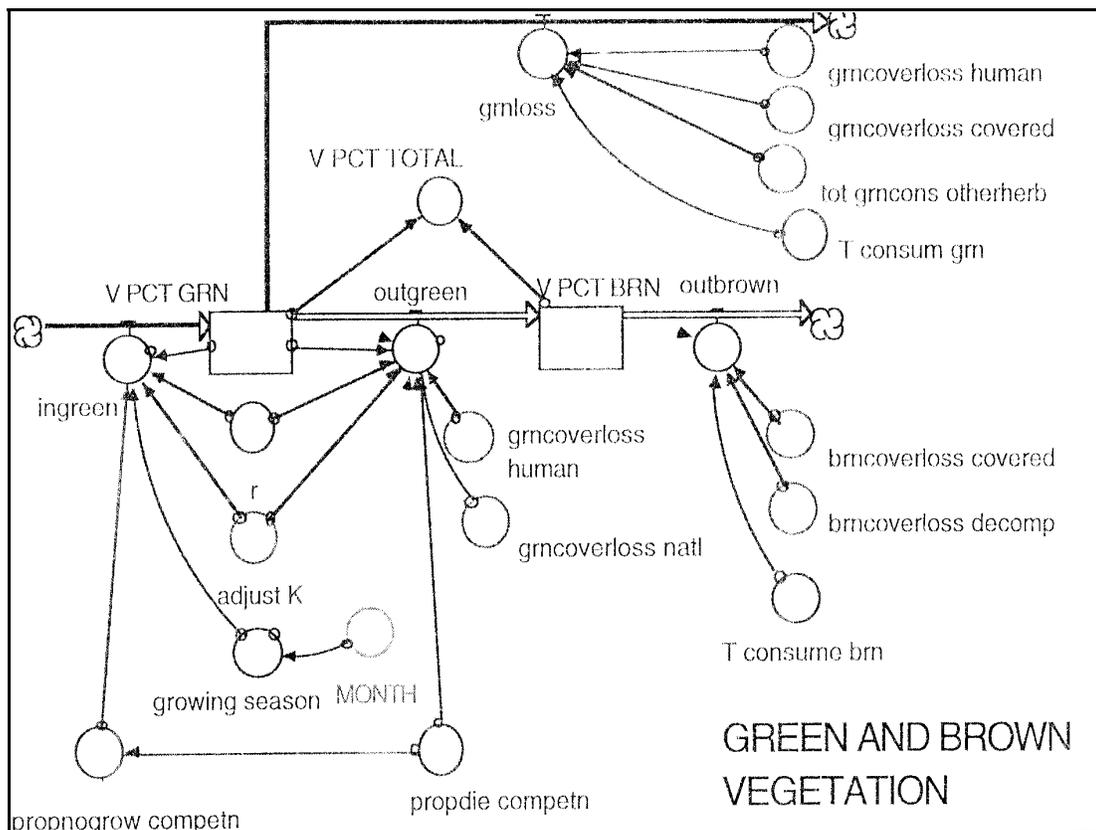


Figure 14. Vegetation submodel developed in Stella.

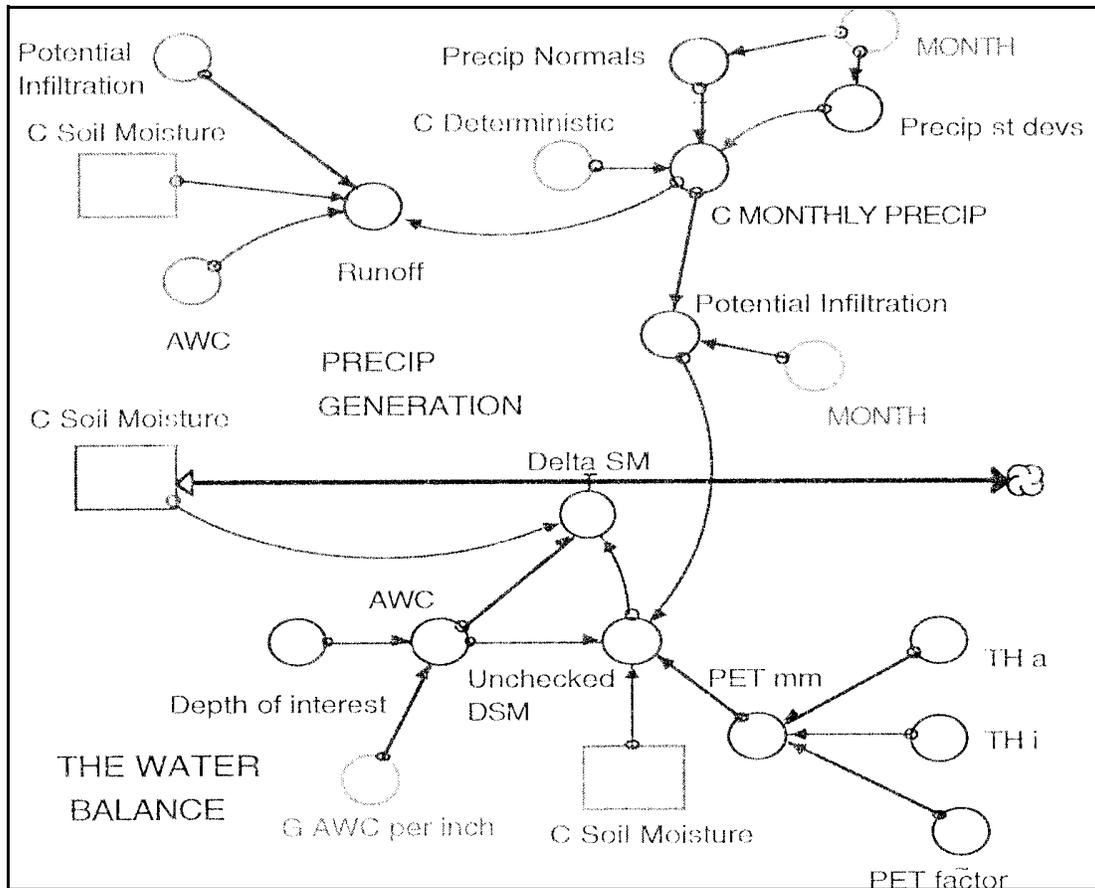


Figure 15. Precipitation submodel developed in Stella.

DEM data is read in this simulation as feet above sea level and converted to meters above sea level to be consistent with requirements elsewhere in the model. Slope and aspect could be accessed directly from GRASS maps, but for this demonstration, IMPORT/DOME routines convert elevation information to slope and aspect data as required. The MapManager also reads a GRASS map that identifies where, within a rectangular area on the landscape, the current model should actually operate, and then provides this information to the simulated entities upon request.

- Access to weather information. This takes the form of temperature and precipitation and is provided through a set of superclasses (Temperature and Precip) that form part of the “Simulation Support Object Classes” (see Figure 4). Details on these superclasses may be located in Appendix B. Such weather simulation capabilities will probably be useful to any modeler developing mobile entities that behave differently under different temperature and precipitation conditions. Having weather simulated in this “static” environment should reduce the time necessary to debug entity simulation software operating later in a dynamic landscape environment.

- **Coordinate information.** It is often necessary to locate objects within the simulated space with row and column coordinates and with UTM coordinates. The MapManager provides METHODS that allow for converting between these two systems as well as keeping track of the extent of the study area.

All of the information and processes found in the MapManager are specific to the simulation model and therefore must remain the responsibility and burden of the modeler.

To facilitate the switch from a static landscape to a dynamic landscape, the MapManager that supports the static data must be swapped out and replaced with one that accesses the SME-driven dynamic data. To the rest of the IMPORT/DOME code, the dynamic MapManager must look identical to the static version. That is, the METHODS that define the MapManager module must be identically named and require the identical data types in the method calling arguments. All references to GRASS format data are simply replaced with references to SME dynamic data as seen in Figure 4. While access to GRASS data, at this point, requires linking the executable code only with GRASS libraries, access to SME data requires linking with Stella models translated by SME into executable libraries. Each such library is simulation dependent. At run-time, the dynamic version of the MapManager is responsible for initializing the SME-managed dynamic landscape code and for then updating the dynamic landscape at appropriate simulation intervals.

5.1.4 Modeler Developed

Once the MapManager and the entity classes are developed, they need to be pulled together with each other and also with chosen controller and viewer classes. For this demonstration the following visualization classes were created: MapView, WeatherView, and AnimalView. The MapView viewport provides a running snapshot of the state of the system in a 2-D map format. The one developed for these demonstrations displays a fixed picture derived from a digitally generated shaded relief map of the region (refer to the top-left viewport in Figure 16). It provides the viewer with a sense of location. Against this backdrop the location and state of individual mobile entities can be displayed during the course of a simulation. The meaning of the particular symbolism is discussed as part of each of the demonstrations. The WeatherView class is based on the library StripChart class (section 4.5.1) developed for the project. It provides a historical view in a stripchart format of recent temperature and precipitation. See the lower right viewport in Figures 16, and 17. Note in Figure 17 that sufficient time has passed to allow the display of 1 year of temperature (jagged upper plot) and precipitation

(plot with spikes indicating intensity of rain events). Finally, the AnimalView class was created to view the changing internal condition of individual entities along with the state of their surrounding environment. The upper right viewport in Figure 16 is an example. These classes are used throughout all of the demonstrations described below.

The “Main” class is finally constructed to pull all of the components together into a well-orchestrated whole. It instantiates the various viewers and entities and then allows the simulation to unfold.

5.2 Static Example

The purpose of this static example is to demonstrate the ability to generate a dynamic entity that interacts with static GRASS maps accessed directly from native GRASS databases. This demonstration, like the rest, uses the GRASS data and IMPORT/DOME general purpose support classes described earlier. The animal class for this demonstration was inspired by the Desert Tortoises found in the region represented by the raster maps. This class uses optional support classes like those described in section 4.3.2, including health, movement, activity, and “LearnSurround” objects.

The key focus of this simulation is the behavior of the 50 instantiated animal objects with respect to the weather and the landscape. The simulation begins by opening up the associated GRASS maps and initializing objects that generate the various viewports. It also instantiates support objects including a Region manager, a TimeManager, and an animal Register. Each of the viewports is associated with an internally defined update frequency; these objects schedule their first update. The simulation then instantiates a number (here 50) of animals. Associated with each animal are a number of support objects, instantiated for each individual animal, that includes movement, health, and activity. Each animal schedules itself for updating.

When an animal updates itself it goes through the following steps. First it checks to see if it is still alive. If not, the animal does nothing and does not schedule itself for any further updates. It then collects information about its surroundings including the other animal entities that fall within its defined home range radius. This range is represented by a circle surrounding each animal in the MapManager generated viewport (top left window in Figures 16 and 17). Other information includes the average slope, elevation, temperature, and precipitation in the home range area. Then, based on temperature, the activity of the animal is determined

by the Activity object. If too hot, the animal aestivates and if too cold it hibernates. In either case the animal dynamically sets its update interval to a week; otherwise it is set to a day. It then, through its associated Health object, eats, drinks, and determines changes to its age, weight, length, stress index, and satisfaction index. The Health object maintains information about the current state of the animal's hunger and blood concentration. If the stress level of the animal gets too low it can die based on a death potential that rises with increasing stress. A LearnSurround object then adjusts the animal's satisfaction with the surroundings by allowing it to "learn" a little more about its surroundings. This object simulates the ability of an animal to come to "know" its local terrain over time. Finally the "Move" object then facilitates movement of the animal based on its home range and motivation to move, which is a function of stress and satisfaction. After all this is accomplished, the animal schedules itself for another update at a later time in the simulation. The delay time is a function of the animal's activity as determined by the Activity object.

Look now at the two snapshots of the dynamic simulation that result from the above code representing day 48 (Figure 16) and day 407 (Figure 17), a span of about a year. This demonstration shows that individual entities can be instantiated, placed on the landscape, and then interact with the landscape as defined by static GRASS maps accessed directly out of the GRASS database structure. Some of animals have died and are indicated by locations in the second figure that have lost their "home range" circle. Some of the animals have moved very little and others quite a bit reflecting different satisfactions of the animals with their initial positions.

5.3 Dynamic Example

The next demonstration is identical to the previous, except that the static landscape maps are replaced with the dynamic landscape simulation defined by the model reviewed in section 5.1.2. This model was converted into C++ code using the SME software (section 3.1.3). Access to the model was facilitated by a dynamic MapManager class (see sections 4.3.5 and 5.1.3).

Figure 18 provides two snapshots of the landscape simulation running with SME-generated output. The output options are quite flexible. This figure shows how three user-selected variables are displayed in map form and in time-series graphs for a single cell in the map. Several other display possibilities are available, including tabular and time-series movie formats.

stripchart viewports. While weather is recalculated by IMPORT/DOME weather objects in the static landscape example, it is generated by the Stella/SME code in the dynamic landscape example. Because the Stella/SME portion of the program updates every month (30 days), the temperature stays constant for 30 days at a time.

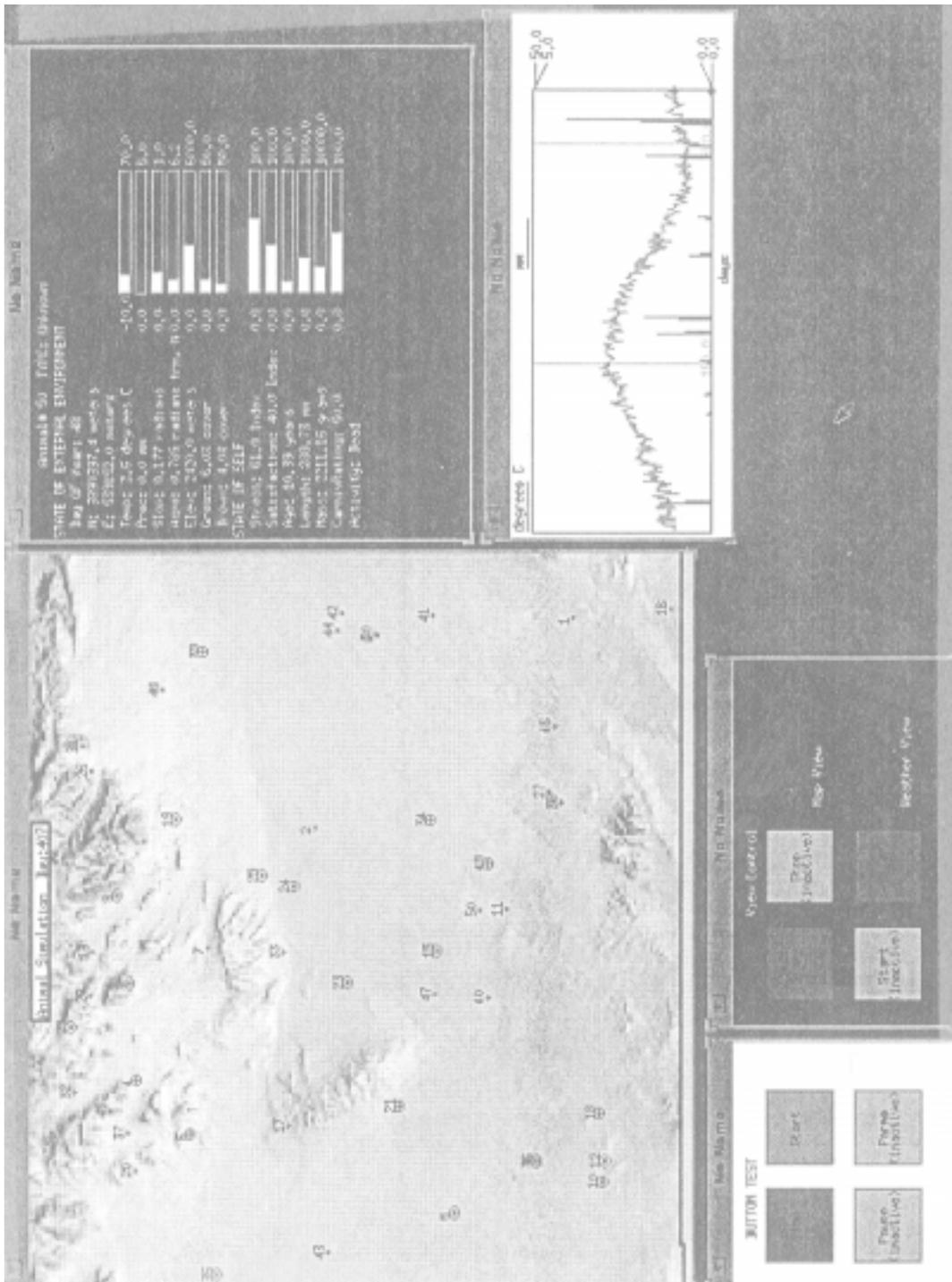


Figure 17. Sample SME simulation output frames.

Figure 18. Sample SME simulation output frames.

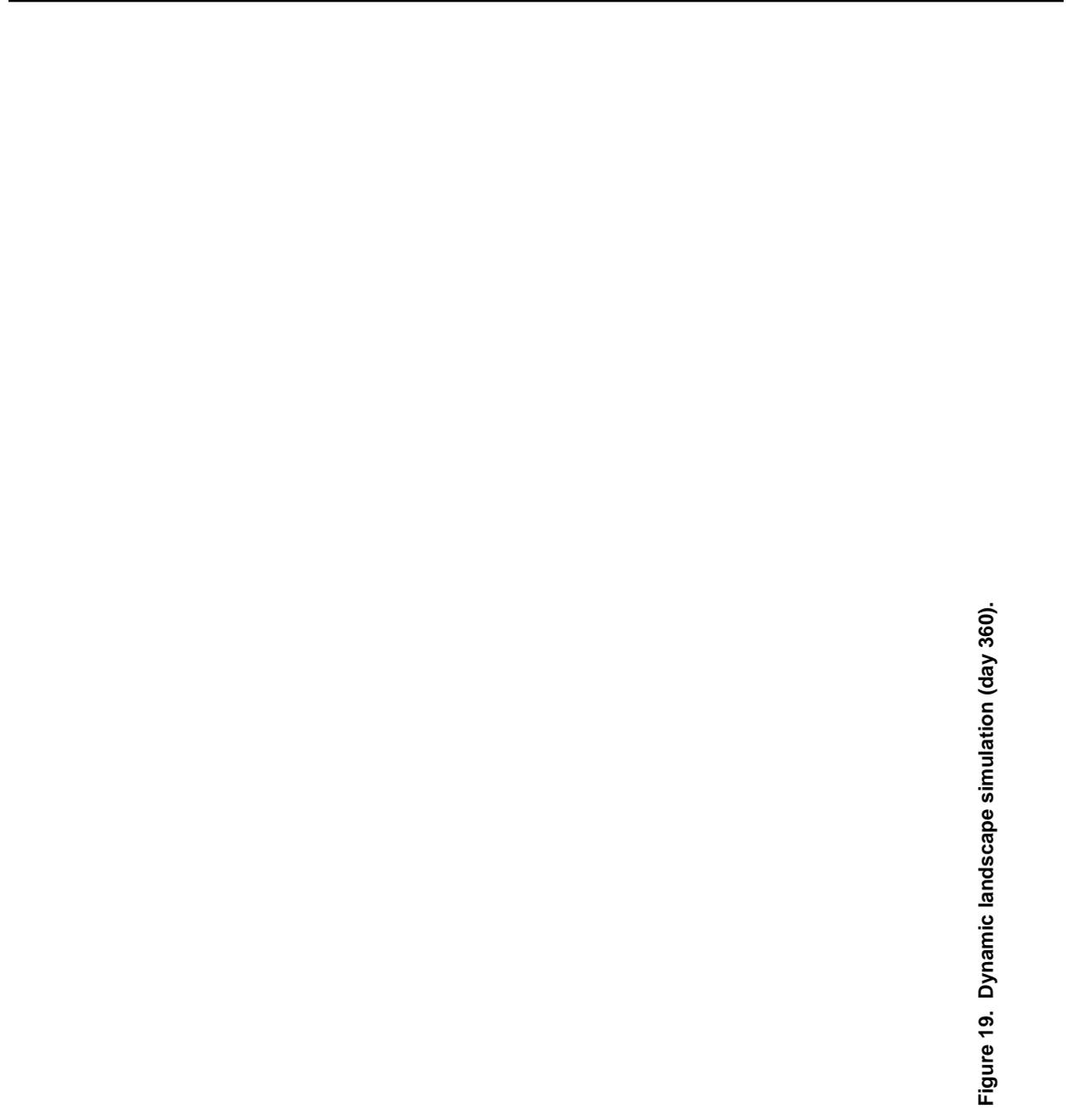
The figure area is mostly blank, suggesting the simulation output was not rendered or is obscured. The caption indicates it is a dynamic landscape simulation at day 360.

Figure 19. Dynamic landscape simulation (day 360).

Animal behavior is also slightly different. Fewer animals have died in the dynamic landscape version. Also, different animals have been motivated to move different distances. This is probably due to different changes in vegetation densities as generated by the simulation over time.

5.4 Reproduction Example

This demonstration adds reproductive behavior to the example of section 5.2. Animal behavior is facilitated by the identical activities described in that section. In addition, animals have been distinguished as male or female and have been assigned different behaviors accordingly. This was facilitated by a new animal

support object called Reproduction. During the course of an animal update, if it is of appropriate age, it engages in reproductive activity. For males this means fertilizing females of age who are not already fertilized and exist within his “home range.” The probability of fertilizing a female is based on the distance between the female and the center of the male’s range and the age of the male. More distant females are less likely to be encountered by the male and therefore less likely to be fertilized. Males who have just entered their reproductive years are less likely to succeed in fertilizing a female than males in the middle of those years. Similarly, fertilization success decreases as a male gets very old. Fertilized females then “give birth” after a preset number of days required for “gestation.” In the Desert Tortoises the babies are provided no parental care after birth.

Inspect the snapshot of simulation day 155 in Figure 20. A significant amount of variability and information can be seen in this image. Male animals are given unique numbers preceded by the letter “M”; females by “F.” As before, deceased animals are represented by animals with no “home range” circles. Note a number of animals with very small home range circles representing the “baby” tortoises. By day 155 (about May) a number of the animals have died (represented by the dots that are red in color images) as in the previous demonstrations. However, females have been fertilized and have “given birth” to animals represented by very small home range circles. At this point there is only one fertilized female, number F-15, who is about a quarter of the image up from the bottom and a third of the way across from the left. (Her range circle is slightly lighter in grey-scale images and green in color images.)

5.5 Predator/Prey Example

Like the previous example, the predator/prey example is also based on the first demonstration (described in section 5.2). The animals of that demonstration have been turned into prey animals. They have relatively short lives and have been given asexual reproductive ability. They still must live off the vegetation represented by the static GRASS green and brown vegetation maps and can die of starvation as well as old age. A new predator animal has also been developed for this demonstration. It is larger, has a bigger range, and its hunger, stress, and satisfaction indices are adjusted to represent a predator. Attacks always result in death to the prey in this demonstration, but that is a decision made by the prey itself. Future demonstrations could easily allow prey to avoid being killed in an

Figure 20. Reproduction simulation (day 155).

attack based on their local terrain, the weather, the predator itself, or other factors peculiar to that animal.

Figures 21 (day 13) and 22 (day 285) are taken from the dynamic landscape simulation based on the predator and prey animals developed. Note on day 13 that the single predator (F-51) has a much larger home range compared to the prey animals. The prey have varying sizes of home ranges based upon their ages which are initialized at random. One prey animal, F-38, located just south of the predator has been killed by the predator and is associated with a pink dot (very light grey in gray-scale images), indicating a predator death. By day 285 the predator has killed a good number of prey and quite a few other prey have died (either of

starvation or old age). The prey items have increased in density in some areas of the simulation area, and are completely absent from others. This makes it difficult for the predator to locate prey, but once located it can remain nearby and feed for quite a while. Patterns of prey patches are developing based on a combination of food availability for the prey and predation patterns.

Figure 21. Predator/prey simulation (day 13).

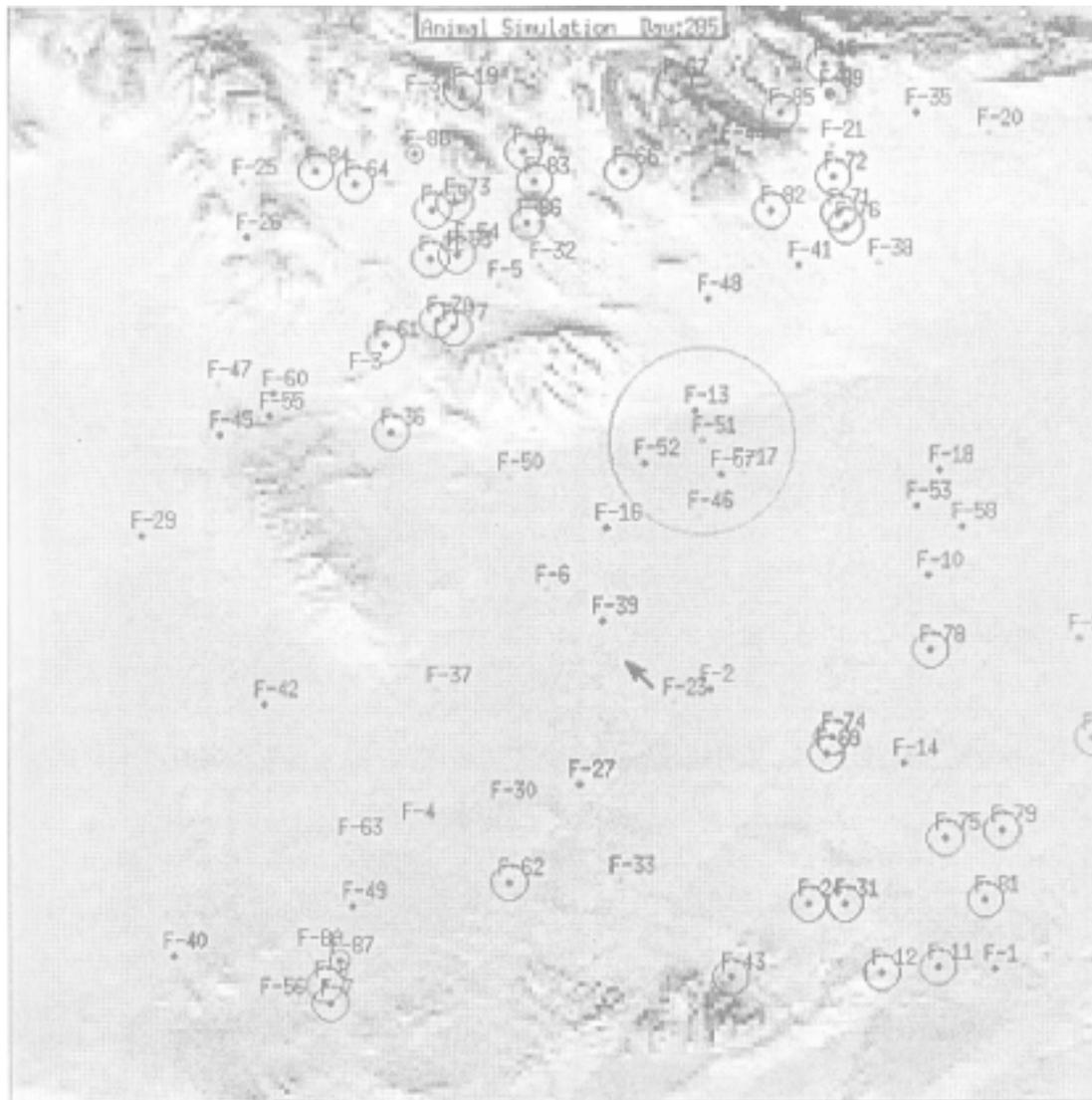


Figure 22. Predator/prey simulation (day 285).

5.6 Summary

This chapter discussed an existing dynamic landscape model developed with Stella and converted to a spatial simulation using the SME package. Then, using the components designed to support dynamic animal entities and interface with the SME model, four demonstrations were reported. The first demonstrated interactions between a very simple mobile animal entity that interacted with a static landscape. Demonstrated were the fundamental capabilities of animal movement, interaction with the landscape (through eating available vegetation),

dynamically changing time steps, and management of internal animal information shared by different object classes, which facilitated animal health, movement, and activity. The demonstration also introduced the use of several runtime viewports, which provide the user with a visualization of the behavior of the model during a simulation. In the second demonstration, the static landscape was replaced with the SME dynamic landscape. Here, the dynamic landscape model provided information about the landscape and updated that landscape at a different time step and spatial resolution than the interacting animals.

The final two demonstrations showed the ability for animals to interact with each other through reproduction and multispecies interactions respectively. Through the reproduction demonstration animals were identified as male and female, each of which behaved differently with respect to reproductive behavior. Males encountered females and, assuming they were the correct age and the encountered female was also of a reproductive age and was not already pregnant, fertilized the female. The probability of mating was a function of several factors including distance between the pair, their level of reproductive “interest” based on age and time of year, and the health of each. Pregnant females gave “birth” to a single offspring after a set “gestation” period. The baby animals were instantiated with defined internal variables, one of which was a relatively small “home range.” Finally, a predator-prey demonstration established the ability to design multiple animal types (species) that interact with one another through evaluation of each other’s characteristics. A predator species killed (and ate) a prey species. Probability of a successful kill was based on proximity of the prey to the predator and predator health and hunger. The single predator helped establish patches of prey, which eventually made the prey hard enough to find that the predator starved and died. These demonstrations establish a basis for further design and development of multispecies simulations in which the participating animals are provided with a very wide range of interaction capabilities. The following chapter evaluates these demonstrations with respect to the design goals established in Chapter 3.

6 Evaluation

This effort has addressed objectives laid out in Chapter 3 within the constraints described in section 3.1. Both the general and the detailed approaches used are critically evaluated in this chapter. We begin with a look at the overall approach followed by particular solutions used to address specific objectives.

6.1 Overall Approach and Constraints

From the broadest perspective, the objective of this study was to explore design components that could be useful in the design and development of individual-based simulation models linked to raster-based landscape ecological simulation models. This section evaluates the object-oriented programming, IMPORT/DOME, and SME constraints. This is followed by a review of the overall design approach.

6.1.1 Object-oriented Programming

Object-oriented programming was chosen as the programming paradigm used to capture the simulation of individuals. This approach to programming has become popular over the past decade because it provides a hierarchical programming framework composed of distinct self-contained components. These components can be easily designed, developed, and tested as distinct pieces separate from other pieces. This allows software development efforts to match typically human hierarchical conceptualizations of the real world, thereby providing a direct mapping between the requirements of scientists and the tools of programmers. Scientists seek generalizations with the idea that any particular instance of the object of study is simply a more specialized version of the generalization. In object-oriented programming, general objects are inherited by more specialized objects. Dogs and cats could be generalized to a four-legged animal object, which then forms the shared basis of more specialized dog and cat objects. This is called inheritance. It is also possible that the cat object simultaneously inherits from a class called night-predators. This semantically matches the way things are typically defined. A cat is a (ISA, in object-oriented jargon) four-legged night predator. Assuming the reader understood the definition, the concept of cat would begin to emerge. Similarly, assuming that objects existed for each of the pieces of

the definition, a cat object (inheriting from each of the definition objects) could be created.

Object-oriented programming allowed the mobile entities to be developed in a manner that facilitates efficient upgrades, modifications, and improvements. Because the approach mirrors the natural human conceptualization process, building interfaces to the various object classes will be straightforward. User interfaces will present system options that are associated with object classes, which are easy for a system user to grasp. User interface screens will be able to have a one-to-one match with the underlying system classes. The system paradigm presented to the end user is virtually identical to the organizational paradigm used by the software developer. This results in a clean system design.

6.1.2 IMPORT/DOME

The IMPORT/DOME language was chosen as the implementation language for the mobile objects because of its powerful integration of objected-oriented programming with procedural and declarative coding possibilities, its fundamental event-driven dynamic simulation core, and its ability to integrate C++ code. All of these features have been exploited within this enterprise. The language is still under development and has not been tested in the commercial marketplace yet. It is also not yet stable enough to guarantee that the code developed here will compile under future releases of IMPORT/DOME.

6.1.3 GRASS

The GRASS geographical information system has proven to be a good choice for testing the design concepts. It is a completely wide-open software environment allowing for even fundamental changes to the source code. For use in IMPORT/DOME, fundamental GRASS subroutines written in C were encapsulated into C++ objects, which in turn were encapsulated as IMPORT/DOME objects. As a result, the interface to GRASS format raster maps from the IMPORT/DOME environment is very easy. The encapsulation process can be duplicated with virtually any raster-based GIS as long as there is access to the subroutines that control map access.

6.1.4 Spatial Modeling Environment

The Spatial Modeling Environment is a powerful cell-based landscape simulation package which, like IMPORT/DOME is under continual development and, as yet, is unstable. It is extremely powerful from the standpoint of model design

flexibility, distributed parallel processing capabilities, and extendibility. The most significant limitations involve fixed time steps and fixed spatial resolution. Newer versions promise to offer multiple time steps wherein each portion of a model (a sector) may be attached to a different, yet fixed time step. The underlying spatial resolution currently must be fixed to a single non-varying value.

6.1.5 Overall Design

Several overall system design decisions were made and need to be addressed here. Some of these flow directly from the constraints identified above, while others do not. First, this project generated modeling capabilities and dynamic landscape simulations that were captured in a single program that ran on a single CPU on a single machine. Current hardware platforms combined with experimental compilers may provide the possibility to relax all of these restrictions — each one opening possibilities not explored in this effort. Future design and development efforts in support of dynamic, spatial, ecological modeling and simulation will probably adopt a client-server approach for supporting some ecological system component interactions. Additionally, using shared-memory approaches on single machines will allow some components to run as separate programs, and yet remain tightly connected. Figure 23 shows the current design of the capability as developed in this project. Users enter simulation model specifications via Stella for landscape unit models and IMPORT/DOME code for mobile entities. These are processed into C++ code through the SME Translator and IMPORT/DOME “Compiler” respectively. The resulting C++ code is compiled together into a single program, which is then executed as a whole.

It is recommended that the approach depicted in Figure 24 be evaluated as a superior alternative to the current approach. In this approach, individual entities, groups of entities, landscape simulations through SME, and other simulations can be run as separate programs that query and interact with common information through shared memory. That is, each program works simultaneously with the same computer memory to allow for rapid communication and efficient sharing of important information. This approach will help facilitate the addition of new system components with little or no modification of an existing system. An important result of this approach is that full ownership of the simulation model components is maintained, which is important for maintaining the integrity of a large software product.

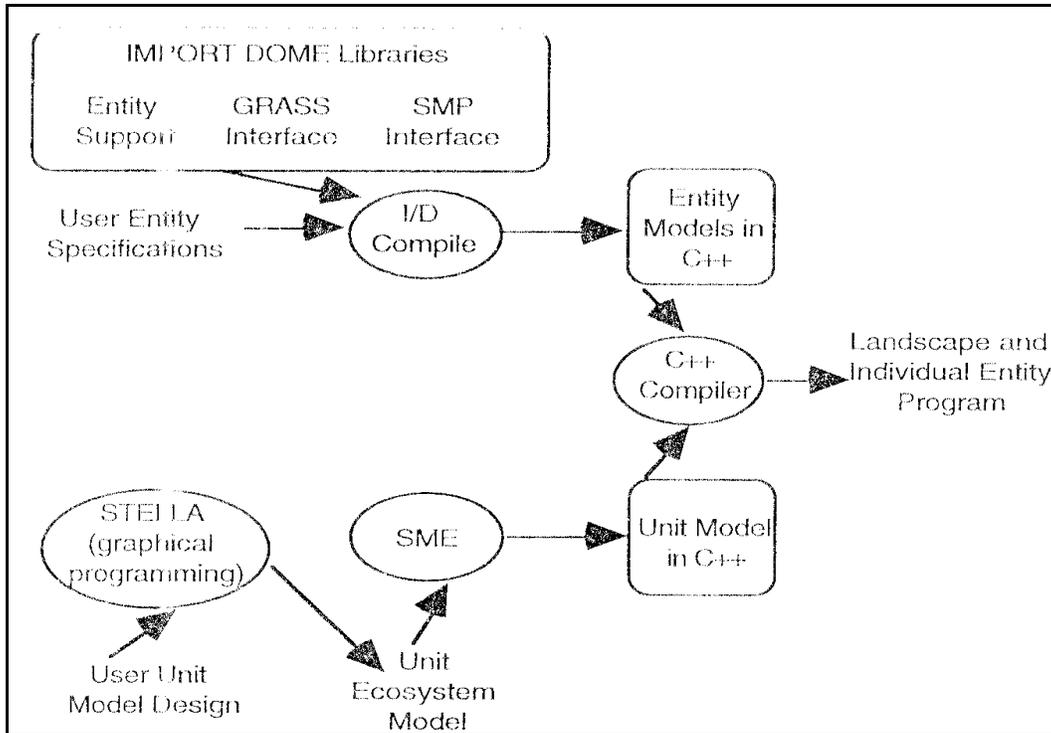


Figure 23. Current overall design.

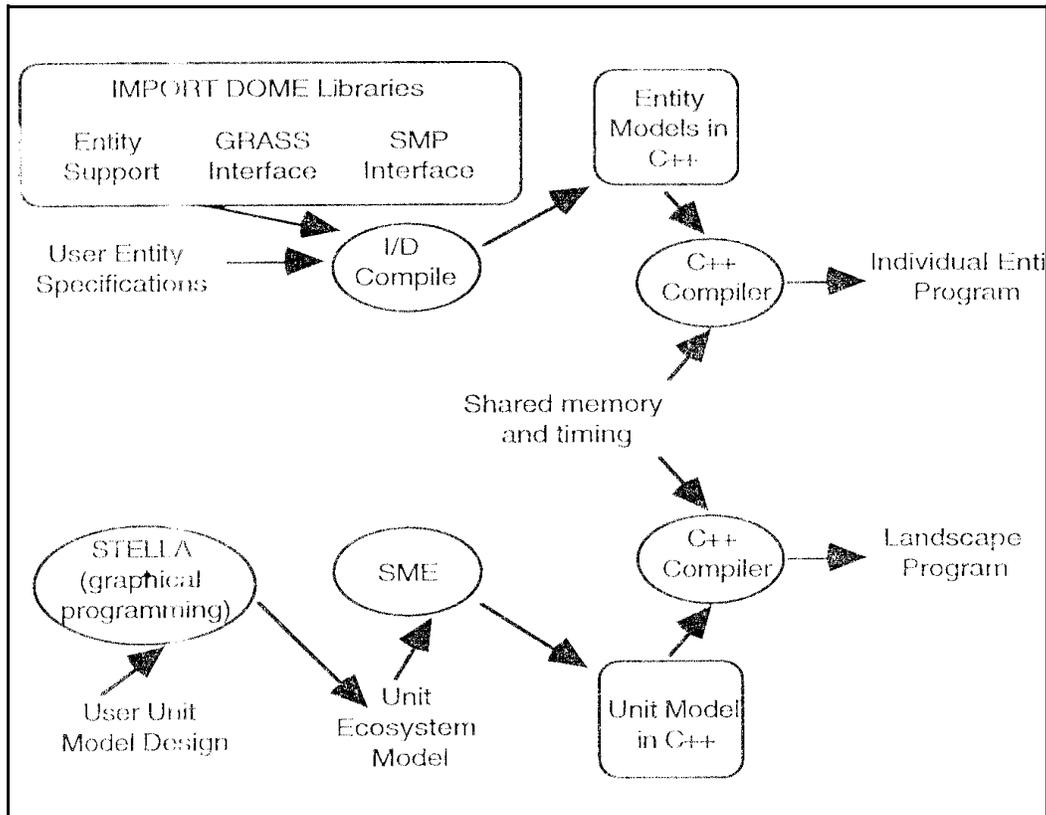


Figure 24. A future overall design.

SME and IMPORT/DOME are two separate, stand-alone capabilities. Each is a dynamic simulation environment that contains a unique approach to time management. As discussed immediately above, the current effort combined the two capabilities into a single software program. The IMPORT/DOME management of time superseded the internal SME time manager. IMPORT/DOME deals with time through an event-driven approach, making it possible to work with virtually any timing scenarios. SME requires that time steps be defined initially and remain fixed throughout a simulation run. IMPORT/DOME therefore has the more flexible timing capabilities and was selected to perform the role of master timekeeper.

As additional dynamic simulation capabilities are merged, it will be necessary to formalize more completely the way in which time is managed. For example, the proposal depicted in Figure 24 may result in potentially many different programs running simultaneously with other programs. The current working approach in this effort is sufficient for connecting disparate simulation programs and therefore will need to be rethought.

6.2 Variable Time

6.2.1 Synchronizing Time Between I/D and SME

IMPORT/DOME (I/D) was used to manage the passage of time for all of its objects and for SME components of the model. An I/D encapsulation of several key SME functions was developed to provide an interface between the dynamic mobile entities and dynamic landscape models. This took the form of an IMPORT/DOME object that allowed for (1) the initialization of an SME model (2) the execution of a number of time steps of the model, and (3) the querying of the state of the SME model at any location within the geographical extent of the model. This approach worked quite well. SME models could be developed outside the combined environment and, when working properly, could be compiled into an IMPORT/DOME program that controlled and used that SME model as a backdrop to the simulation of the mobile entities. Section 6.1.2 discusses the limitations associated with extending this approach to encompass still other simulations that may be distributed across a number of CPUs or computers.

6.2.2 Event-driven Time In I/D Objects

The design of this system must accommodate variable time steps. This was accomplished using IMPORT/DOME's fundamental event-driven time management scheme. Any given object in a simulation updates its internal state and position at

intervals dynamically determined by the simulation as a whole. In the sample system, entities modified their update frequency based on whether or not they were in “hibernation” or “aestivation.” During these times, the time step was set to longer intervals to improve simulation performance. Different objects could update at different frequencies. The objects were not tied to the fixed time step associated with the SME part of the simulation.

6.3 Variable Space

Variable space has been accommodated in several different ways in support of dynamic entities. Because different ecological components interact with their surrounding space at different resolutions, it is important to provide for varying ranges of resolutions. If a resolution is too small, computational efficiency may suffer; if it is too gross, the probability of some important component in the space being overlooked during a simulation is increased. Sometimes it is important for some particular landscape component to be available at a number of different resolutions.

Spatial resolution and temporal resolution are tightly intertwined. Any objects that move should avoid skipping across portions of the landscape without interacting with those portions. For example, an animal that traverses the landscape at 100 meters per hour should not be modeled on a landscape with a time resolution of 1 hour and a spatial resolution of 50 meters. This will require that the animal skip over cells when moving at full speed. The spatial resolution could be decreased to 100 meters or more. Alternately, the time resolution could be decreased to 30 minutes. Decreasing the spatial resolution by 100 percent will decrease the computation requirements by 25 percent, but this runs the risk of losing important landscape simulation details. Increasing the temporal resolution by 100 percent will double the computational requirements, but this runs the risk of slowing the simulation with no significant increase in functional utility. Another animal moving at a top speed of 10 meters per hour has yet a different pair of optimal spatial and temporal resolutions with which to interact. SME offers, as of this writing, a single spatial resolution and a single time step. These must be chosen so that the most rapid landscape process can be appropriately captured. Future releases of SME will relax this restriction enough to allow different portions of the simulation to run at unique, but still fixed, time steps.

The remainder of this section reviews the various ways the current study accommodated notions of variable space and suggests possible improvements.

6.3.1 LocationManager Object

It is important for entities to be able to scan those entities in their immediate vicinity. This provides, along with raster-based simulation data, a picture of the environment to which the animal must respond. As the environment gets larger and more complex, it becomes increasingly less reasonable for each entity to consider every other entity in the simulation. A LocationManager class was developed to facilitate the ability to address this potential information overload. As entities move around the simulation space, they register and reregister their current location with the LocationManager (Figure 25). This manager provides query capabilities that will return a list of objects that exist within a certain distance of any given location. An entity may want to consider its response to other entities within 10 meters of its location, while one of those entities will consider all others within 100 meters. The LocationManager assists in narrowing down the list of all entities in the simulation to those that might be within range. While excellent in reducing the search and entity analysis time that any given entity must perform to evaluate its surroundings, it does carry the burden that each entity must responsibly register its location on a regular basis.

A LocationManager can easily store and retrieve information at a wide variety of resolutions. Information is indexed internally to the LocationManager through a quad-tree indexing approach. Although only a single LocationManager was used in all of the test models, multiple LocationManagers can be used in future models if there are clear sets of objects that never need to know about other objects. It was designed for and is most useful when used with objects that are located as points. It was not designed to work with extended objects that cover regular or irregular shapes on the landscape. For example, the programming approach for individual entities described here is suitable for modeling herds of animals if the spatial extent of the herd is a fraction of the spatial resolution of the simulation. Once the extent spills across space, say two or more times as large as the simulation resolution, the current software environment will be inadequate.

6.3.2 Potential Extensions for Supporting Variable Space Resolution

The current software environment interacts with a fixed raster resolution through the GIS and SME interfaces. This betrays a fundamental restriction associated with raster-based geographical information systems. Spatial data are generally retrievable only at the resolution at which they were stored. The GRASS GIS compromises this limitation to some extent by resampling the stored data at run-time using a nearest-neighbor resampling approach (Shapiro et al. 1993). This always works well when the resampling is at the same or higher resolution than the stored data. If, however, the resampling is at a lower spatial resolution, much of the stored data is lost. This can be a problem if there are important features that exist at or below the resampling resolution. The process of bringing out small features when sampling at a lower resolution is often case dependent. In some cases the smallest feature should be identified. In other cases, the user may wish to prioritize the importance of various features that should be stored in a particular map layer. This could ensure that when a patch of digital landscape is sampled at a lower resolution than the stored data, the most important information in that location is returned. Alternately, any given map might be opened a number of times, each time ignoring all information except for one particular landscape feature. For each map, the particular feature will be identified as occurring within a referenced cell if, in the stored map, that feature exists anywhere within that cell.

6.4 Scan and Evaluate Surrounding Entities

6.4.1 Scanning Entities

Once a given entity has acquired a list of nearby entities from the LocationManager, the entity must evaluate these neighbors and determine how to act with respect to this evaluation. The first step is scanning these entities for information about them. There are two facets of this operation. First, actions that any entity takes on any other entity can be remembered by the receiving party. For example, if entity A attacks B, B may maintain a memory of that attack, which may persist for some amount of simulation time. Second, an entity, after receiving a list of nearby entities from the LocationManager can ask each of those entities to provide a predetermined set of basic entity information. This set includes, length, mass, and a "carnivorous rating." The collected information may be retained in entity memory for some period of time. Unique entity identification codes, attached to each entity, can allow for the scanning of an entity to occur only occasionally. It

may also allow information to be “forgotten” if that entity is not part of the local environment for some period of time.

6.4.2 Evaluate Entities

Once the information about the surrounding entities is scanned, that information must be evaluated. Objects have access to a simple generic querying capability for creating lists of entities that meet specified criteria. For example, if an entity needs to respond to potential predators, it might scan its surrounding entities to identify all that are larger than a certain size and have a carnivorous rating above a certain index.

The entity knowledge available is quite limited; hence the evaluation potential of entity objects is also limited. Extending the amount of available information, though straightforward, would conflict with other design objectives. These objectives are (1) maintain the ability to add new objects to a simulation without modifying existing objects, and (2) allow for objects not mutually recognizing each other's class type. In response to both of these objectives, a set of characteristics has been created that any given object can access about any other given object. Currently, this is the only guaranteed information that can be shared between any two objects and the only mechanism for doing so. If some future object requires access to different information, the current design makes it possible to add this information to the generic list. This will result in the need for every existing object class to be modified to both provide and deal with this information. Here lies the conflict between two opposing objectives. If there is no reasonable set of minimal characteristics that every entity must provide, then the requirement that we be able to add new objects to a working simulation will be in jeopardy. If the latter requirement cannot be relaxed, then we will need to accept limitations in mutually available information.

6.5 Interactions with Dynamic Entities

Once a given entity scans and evaluates surrounding entities, it applies its own unique rules to decide how to respond to those entities. Interactions can take several forms within this system: change of positions, change of internal state, and attacks on and potential death of other entities. Actions on other entities generally require two steps: (1) notice of action on another, and (2) reaction by that other.

6.5.1 Actions on Others

In this limited demonstration, actions on others can be passive or active. Passive actions occur when a single entity responds to the perceivable location and state of another entity. For example, a female entity may consider itself fertilized if the season and her health are appropriate and if any male of her own species of mating age is present. The male need not actually do anything within the model except be available. Indeed, whatever he might do is likely to be associated with a time scale that is not being directly simulated. Similarly, if shelter is present for a given class of objects, it may be presumed that objects of that class use the shelter. The shelter, if simulated as an entity, need not act upon the animal objects. Active actions in the current prototype can only take the form of an “attack” message. This is an index value between 0 and 100 that indicates the intensity of the “attack” and is processed by a generic superclass, which must be associated with each animal object. Information of the “attack” is immediately processed and a kill (true) or no-kill (false) message is returned to the attacker. The information is also stored for processing by the recipient of the attack at a later time.

6.5.2 Reaction to Others

Reaction to “attacks” currently takes two different forms. First, the attack is evaluated by the recipient in a synchronous (immediate) manner to (1) identify if the attack was sufficient to kill, and (2) to store information about the attack for processing at a later time. Second, the next time the receiving entity is scheduled to update its internal state and location, the stored attack information may be used to modify health, movement motivation, and other parameters. Note that the time scale of the current demonstrations is on the order of days to weeks. It is impossible, therefore, for the simulation to capture rapid movements and other interplays between the predator and the prey. Instead, the sum-total of interactions over the course of the time between entity updates must be considered as a day-long (or week-long) set of actions.

6.6 Create New Entities

Object-oriented approaches to software development provide a fundamental capability that allows the objects to be created and destroyed. Every IMPORT/DOME object is associated with creation and destruction instructions that are used as explained in this section.

6.6.1 Entity Death

Mobile objects are associated with classes that allow them to be destroyed, in a software sense. That is, all memory allocated for a given instantiation of a class can be erased and deallocated. Problems however can arise if this is done too hastily. If, for example, an object is queued, through a TELL method, to do something at time step 20, but is deallocated at time step 10, the entire simulation will die abruptly at time step 20. For the purposes of this study, objects are never deallocated; instead, they are marked as inactive or “dead.” When an object is asked to respond through a TELL method, the object does nothing when in this state. Future improvements will require that the object more cleverly deal with this situation so that memory can be freed.

6.6.2 Generating New Entities

Entities must be able to create new entities. This is important in reproduction, and sometimes important in development if the object changes form significantly (e.g., caterpillar to butterfly). IMPORT/DOME provides a simple approach, which involves using a “NEW” call. Any class can be instantiated into a new object in this manner. The object that is instantiating another object must “know” about the class type of the new object. This means that during compilation, the second class must have been processed by the compiler either before or during the processing of the first class. In some cases an object is part of a life-cycle which may involve (at a minimum) adults and eggs. If the adults are to create eggs and the eggs to create adults, the egg and adult classes must be developed as part of the same IMPORT/DOME module. Basically, all life phases being simulated for a single animal must be developed within the same module. This can be a significant drawback if the different objects are developed separately from each other, for example, by different research teams.

6.7 Interactions with Static and Dynamic Maps

Interactions with information captured in a GIS or dynamically simulated with the SME environment provide needed links between population and mobile entity simulations, a primary goal of this effort. Interactions were facilitated by compiling the different software components into a single program. Subroutines (METHODS) were developed that allowed the request for and exchange of information.

6.7.1 Interactions with Static Maps

Mobile landscape objects are able to retrieve information from static GIS maps through an IMPORT/DOME object class called MapManager. This object is instantiated once for a simulation and is made known to all mobile entities. It is responsible for opening the GIS maps and for providing both raw GIS data, and information derived from these data. In the case of the model developed for this effort, a single METHOD provides the main viewport into the GIS data: ReturnCellData. Input covers the current day of the year and the current location. Output covers raw GIS data (elevation and densities of shrubs, forbs, and grasses), derived GIS data (slope and aspect), and derived weather data (precipitation and temperature that has been adjusted for slope, aspect, and elevation). MapManager, in this design, must be modified by the simulation builder because the information it retrieves is very specific to the simulation at hand. Future design updates should break out the components that must be modified by the builder of a particular simulation from the generic code required for all simulations.

Matching dynamic entity simulation to static GIS maps requires two key steps. First, the compilation of the simulation must include fundamental GRASS software libraries. Second, at run-time, the user must be running GRASS. This requires that a number of environment variables be set and that the maps that will be used exist within the GRASS database selected. The north, south, east, and west edges of the area within which the mobile entities are allowed to roam must be identical to the edges of the working area selected through the GRASS programs. All of these conditions must be met by the user and are not yet automated. Future upgrades of this environment can make this a more automatic and foolproof procedure.

6.7.2 Interactions with Dynamic Maps

Once dynamic landscape objects are functioning properly with the static GIS maps, they are ready to be attached to the dynamic maps managed by the SME software. This is accomplished by simply swapping out the GRASS-specific objects with SME-specific objects. Both objects use identical object names (MapManager) and provide identical IMPORT/DOME METHODS, thus providing a simple swapping capability. Completely different activities are performed by radically different code underneath these names however. The end result is two “drivers,” which are interchangeable at system compile time.

Matching dynamic entity simulation to dynamic SME landscape requires certain steps that are very different from those required for the static GIS maps. The SME

simulation must (1) be working properly as a stand-alone SME process (2) provide the maps that IMPORT/DOME code references (3) use the proper units within the maps (4) have north, south, east, and west boundaries that precisely match those in the IMPORT/DOME simulation, and (5) be compiled as a C++ library. This library is then compiled with IMPORT/DOME wrappers, which have calls that map precisely to the SME maps. Finally, the IMPORT/DOME mobile objects and SME timing software are compiled into an integrated IMPORT/DOME and SME simulation. This procedure may, in the future, be matched with software that checks for incompatibilities and completely automates this process.

6.8 Recommended Improvements

6.8.1 Interaction Between Simulation Software

Currently the GIS, SME, and IMPORT/DOME environments are carefully attached through software that provides weak links among them. Each environment is currently developed with different funding and is driven by different objectives. Although more tightly intertwined research and development efforts are desirable from the standpoint of some objectives, the degree of coordination required can be too costly. A common approach to facilitating interoperability is to identify a set of standards that define what is minimally required to coordinate efforts.

An approach to facilitating multiprogram and multicomputer-based dynamic simulations is graphically depicted in Figure 26. Static or dynamic maps are managed by one or more programs running on one or more machines. Simulations of individual entities across the landscape are managed by one or more other programs running on one or more computers. These may be the same computers running the static or dynamic maps, or they may not. Here the goal is to be flexible with respect to which processes are managed by which programs. These programs, in turn, may run on a variety of machines. Interactions between model components are carried out at the lowest level possible. For example, components running within the same program may simply use the same memory allocated to that program. When the components are running within different programs on the same machine, memory that is shared between the two programs is used. An associated information “QueryManager” facilitates requests for information. If the information being requested is associated with a process running on another machine, the query manager fetches that information from locations identified by the “Master Memory Information” manager running in a master control program (Machine B in Figure 26) and stores it in local shared memory (acting as a data cache) and lets the process that requested the data know where and how this

information can be accessed. A “Master Timer/Calendar” provides the timing control (under run-time user control) which keeps the entire simulation synchronized. This approach will require significant design efforts upfront to ensure the efficient exchange of information between a wide variety of processes managing a wider variety of information.

6.8.2 Improve Object Ability to Learn

A significant challenge to the future utility of simulating individual entities is the need to improve the simulation of the learning process. Mobile entities may be endowed with intelligence — the power to learn and adapt. If the time and space scale is small enough, learning becomes important in the simulation. The examples demonstrated in this effort involve animals that are being simulated with a time resolution on the order of a day and a space resolution on the order of tens of meters. The process of learning and forgetting is important with such entities at this scale. The current software is very weak in its ability to capture this process. Generic entities do scan their surroundings at some prescribed distance and can store information about the noted objects. While this may be construed as learning and forgetting, it is simply a programming approach to speed up the system by caching information. Learning is simulated by simply allowing an object to perceive an increasing percentage of the available information as it remains in an area. A viewer of the system may notice that an animal may move quite rapidly through a perfectly suitable habitat when it is initially displaced from an area. This is because the animal is not allowed to become familiar enough with the area to “know” that it would be a good area to remain in. Mathematically this is similar to a gradient descent algorithm with momentum.

The complexities involved with perception, learning, and forgetting and the physiological impacts on these processes are difficult. The practical application of simulating learning behavior will continue to remain very limited in landscape simulations. Not only is the software expected to be relatively demanding, but required data will not be available. Instead, learning models will rely on the compilation of huge amounts of field work that statistically measure how behaviors follow one another and are triggered by external and internal variables. Such data is also currently very limited.

6.8.3 Sensitivity Analysis

An important component of dynamic simulations is the ability to perform sensitivity analyses. This takes many forms starting with the selection of simulation components to Monte Carlo (and other) simulations to determine the importance of different system variables. As created, the current capabilities provide only the ability to run a simulation with one set of preset variables. Only by changing these variables and rerunning the simulation can one begin to evaluate the overall system behavior. For each run, of a set of runs, the system can be (1) fundamentally changed by adding or subtracting different components (2) started

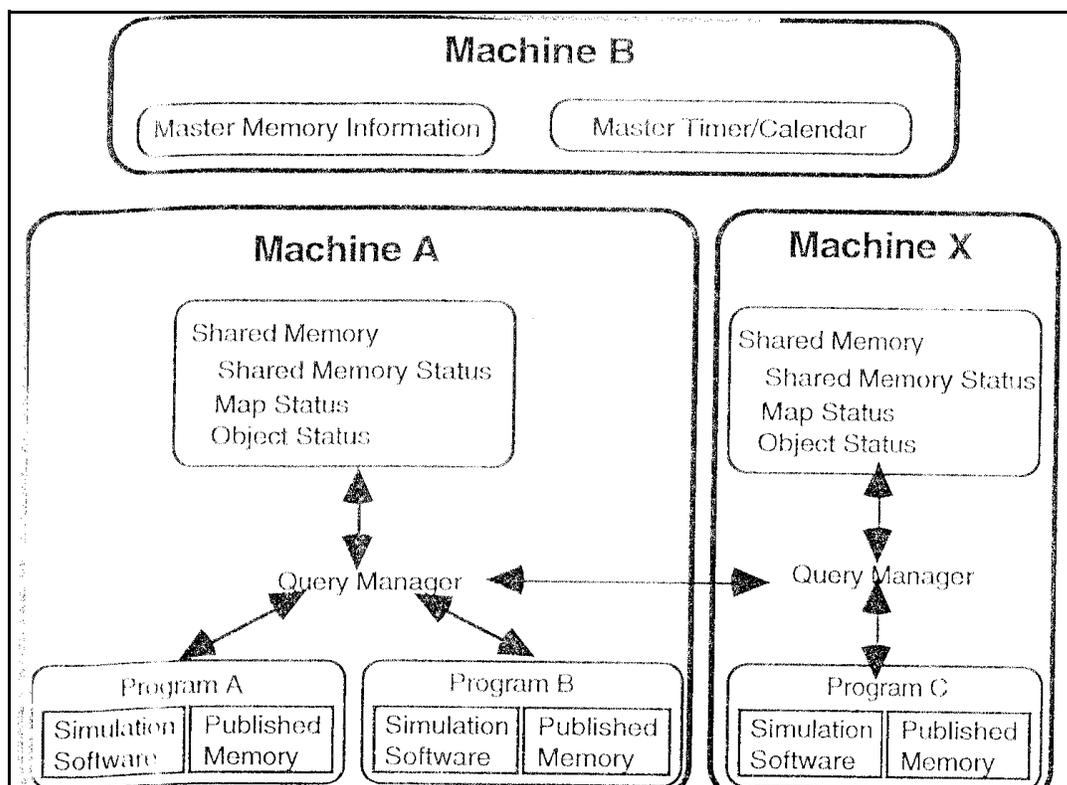


Figure 26. Proposed network shared memory interaction.

with slightly different states, and (3) run with different sets of parameters that control behavior of individual components.

6.8.4 Tracking and Saving Entity States

Dynamic simulations involve rapidly changing system states with very large numbers of state variables. For example, a system that contains 300 by 300 cells with 100 state variables per cell attached to 100 mobile entities each containing 100 variables, has approximately 9.1×10^6 variables at any given time. If the system ran with a fixed time step of 1 week and completed 100 years, we are left with 4.7×10^{10} data points. Assuming these are all 32-bit floating point values, we have something on the order of 1.9×10^{11} bytes of data. That is approximately 200 gigabytes of data per run. Clearly, only a very limited amount of the information can be reasonably stored for later viewing and analysis. More commonly, the user identifies a set of information to be displayed visually during a simulation run and other information stored on disk.

The demonstrations developed for this effort allow for various pieces of information to be visualized during a run. Additional simple “viewers” in the form of IMPORT/DOME objects can be simply generated to probe portions of the simulation and store this information in files for later analysis.

6.8.5 Graphical User Interface

The need for graphical user interfaces has been discussed in the objectives and evaluation chapters (3 and 6). Graphical user interfaces must be developed to remove the need for simulation developers to edit text files containing software code, to configure entities, and to configure simulation and run-time behaviors. User interfaces must be provided to allow powerful run-time probing and analysis of system behaviors during simulation. Finally, post-processing analyses of system parameters after simulation runs are essential. Such interfaces may take many forms and will depend on target users and the extent to which the capabilities are merged with other simulation capabilities.

Controller classes connect the simulation model classes to user input devices. Limitations with GUI possibilities provided throughout the IMPORT/DOME language prevented the design and development of more sophisticated user interfaces. Some simple beginnings to the development of general purpose GUIs for the management and control of a running simulation and its entities are presented here.

Model classes provide methods that accept input from outside the object. These may be accessed based on user intervention during the course of a simulation. Access is not direct because model objects are not being allowed to communicate directly with computer input peripherals. The controller classes suggested here demonstrate basic approaches to user run-time command and control of a simulation. These controllers are also view classes in that they do provide feedback directly to the user through visualizations.

GUI control of simulations in the current demonstrations is intentionally limited and weak. Some very limited controls (Figures 27 and 28) are suggested for the purpose of providing rudimentary control and helping to visualize what might be possible with the underlying simulation capabilities. User interfaces must be a function not only of the underlying software, but more importantly of the capabilities and intentions of the user. If other capabilities are being connected, it is the domain of the GUI builder to provide an apparently seamless integration of the software engines behind the interface.

Interfaces are probably the most volatile component of any software. To ensure longer-term viability of software, it is important to separate the code that drives user interface peripherals (most notably keyboards and monitors) from the

Entity list for: <u>animals</u>		Constants			
	ID number	Default	Change	Name	Units
<input type="checkbox"/>	2	<u>20</u>	<u> </u>	Max Age	years
<input type="checkbox"/>	3	<u>1000</u>	<u>800</u>	Max Speed	meters/day
<input checked="" type="checkbox"/>	4	<u>10</u>	<u> </u>	Birth Length	mm
		<u>10</u>	<u> </u>	Birth Mass	gram
		<u>60</u>	<u>80</u>	Max Stress	0-100 index
		Variables			
		New Value		Name	Units
		<u> </u>		Age	years
		<u>1000</u>		Mass	grams
		<u> </u>		Length	mm
		<u> </u>		Carniv Rating	0-100 index
		<u>100</u>		Stress	0-100 index
		<u> </u>		Fecundity Rate	eggs/month

Figure 27. Sample entity controllers.

Entity list for: <u>animals</u>			Displayable Variables			
BC	SC	ID number	Range		Name	Units
			Lo	Hi		
<input type="checkbox"/>	<input type="checkbox"/>	2	<u> </u>	<u> </u>	<input type="checkbox"/> Age	years
<input checked="" type="checkbox"/>	<input type="checkbox"/>	3	<u>0</u>	<u>1000</u>	<input checked="" type="checkbox"/> Mass	grams
<input type="checkbox"/>	<input checked="" type="checkbox"/>	4	<u> </u>	<u> </u>	<input type="checkbox"/> Length	mm
			<u> </u>	<u> </u>	<input type="checkbox"/> Carniv Rating	0-100 index
			<u>0</u>	<u>100</u>	<input checked="" type="checkbox"/> Stress	0-100 index
			<u> </u>	<u> </u>	<input type="checkbox"/> Fecundity Rate	eggs/month

Figure 28. Sample entity controllers.

underlying models. In general, a software development approach called Model-View-Controller (MVC) has been utilized in the current design. For the purposes of this effort, this effectively means that no model objects communicate

directly with any peripherals. METHODS are used to accept input and provide output through arguments. Other user interface specific objects mediate between the model objects and the peripherals. These can easily be replaced as user interfaces develop further or must be replaced to meet the needs of a different user community or different application. Because model objects provide only input and output capabilities through METHODS, other model objects can play the role of viewer or controller.

Control of entities during a simulation is important for experimenting with system reaction to small changes in the state of entities. For this purpose, each entity class should be associated with an entity control class. Figure 27 is an example interface that allows an entity to be chosen from a list of like entities and then allows key constants and variables (stocks) to be modified during the course of a simulation.

Each entity class should be associated with a view control class. A single object representing this class would be instantiated for each simulation run and could then be used to manage which entities are being viewed through associated StripChart and BarChart objects. Using the mouse, the user could scan through a list of currently running objects and select those for visualization in this manner. A viewport for this control is depicted in Figure 28. This represents an entity list of class animals. It shows the unique identification (ID) numbers assigned to each animal object and which objects are currently being displayed using BarChart objects (BC) and StripChart objects (SC). The check mark next to ID 4 indicates that the right side of the viewport shows which variables are selected for stripchart display and what the anticipated range of values is for each variable.

6.9 Summary

There are three main objectives associated with this study: (1) simultaneous simulation of multiple scales (2) enabling mobile entity simulation, and (3) integrating disparate landscape simulations. The demonstrations show that a fundamental ability exists here to support simultaneous simulation of multiple scales. Dynamic entities exist within a dynamic home range and also interact with their environment at dynamic time scales. The size of the home range is a function of current activity, health, size, and type of animal. Dynamic temporal scales are established based upon the animal's activity level. In these demonstrations, aestivating or hibernating animals updated themselves on a weekly basis while active animals performed daily updates. While the animals interact, the landscape

itself updates at a fixed 1-month interval using a fixed spatial scale. Finally, integration of disparate landscape simulations was demonstrated by linking the Stella/SME simulation environment with the IMPORT/DOME programming language environment. Dynamic entities developed in the latter were able to identify the state of their surroundings as established with static GIS maps read in their native GRASS format or through probing the simultaneously running Stella/SME simulation of the dynamic landscape. The interaction was facilitated by combining the two environments into a single software program. This approach should be replaced with one that allows the programs to remain separate, but retain the ability to communicate with each other at run-time.

7 Conclusions

This project has developed a fundamental approach to facilitate modeling of dynamic mobile entities in combination with either static or dynamic, raster-based landscape simulations. The approach provides significant flexibility in specifying a wide variety of entities captured as classes in the IMPORT/DOME object-oriented dynamic simulation language. At a more general level, an ecological landscape modeling paradigm has been demonstrated that not only allows, but encourages, the capturing of natural processes within simulation environments based on different software. This is the fundamental advantage that must be expected from future geographic modeling systems (GMS). The various simulation components operate separately, but communicate with each other during simulations.

This approach differs from the gap-based forest models wherein individual trees are simulated within square landscape patches, which themselves are arrayed upon the landscape. In those models a neighboring tree is treated differently based on whether or not it falls within the same patch. The square patches are imposed upon the fundamental data structure of the system that captures the location of and relationships between trees. In contrast, the approach to landscape simulation used in this project allows two disparate software simulation environments to be used, each for what it is best designed to do, yet communicate with each other as appropriate. A powerful raster-based landscape simulation environment was used to capture the dynamics of weather, soil saturation, and vegetation densities, while an equally powerful object-oriented simulation language was used to capture the dynamics associated with individual animals acting within and upon the landscape. Neither environment is suitable for both applications, although a force fit is certainly possible. Neither is captured within the context of the other; each is provided equal status. Future GMS will provide a wide variety of interacting, yet distinct landscape simulation capabilities. Modelers and landscape managers will be able to match landscape processes with the most appropriate simulation module.

The two simulation environments joined were SME, the Spatial Modeling Environment, and the new system developed here to support the simulation of the behavior of individual animals. The latter was developed with the IMPORT/DOME dynamic simulation language. Specific capabilities included:

- Dynamic, object specific time steps
- Animal reproduction

- Interaction between animals, including predator-prey and reproduction
- An efficient search mechanism for finding neighbors
- Generic means for evaluating neighbors
- Recognition through evaluation of characteristics
- Ability to add new animals without reprogramming others
- Interaction with static or dynamic landscapes.

Nature, as presented in Chapter 2 operates at a variety of different spatial and temporal scales, organized at a number of hierarchical levels. Using the integrated combination of SME and IMPORT/DOME, systems can be developed that explicitly simulate the behavior of organ systems through communities. The simulation focus here is the landscape, a “middle-number” system, which is not analyzed well with statistics, because it lacks a large enough sample size and is not easily simulated as a small set because there are too many distinct parts. The approach developed in this work embraces the hierarchy theory approach to dealing with middle number systems. System components at multiple hierarchy levels can be simultaneously simulated using software approaches that reflect the spatio-temporal scale of each level. No one level dominates the simulation. Instead, each is allowed to communicate with other simulations being conducted at different scales. For example, a simulated individual animal belonging to species A may interact with a population model of species A, an individual model of species B, or a population model of species B.

The power and potential of this combination of simulation approaches opens the door to completely new classes of dynamic, spatial, ecological models. Further, this new simulation environment opens the options for integrating knowledge and results gained through disparate environmental studies. It provides an interdisciplinary forum for formally capturing this information in a manner that increases its potential for affecting land management practices.

References

- Allen, T. F. H., and Starr, T. B. (1982). *Hierarchy*. Chicago: University of Chicago Press.
- Andrewartha, H. G., and Birch, L. C. (1954). *The Distribution and Abundance of Animals*. Chicago: University of Chicago Press.
- Battaglin, W. A., Kuhn, G., and Parker, R. (1993). "Using a GIS to link digital spatial data and the precipitation-runoff modeling system, Gunnison River Basin, Colorado." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Belanger, R., Donovan, B., Morse, K., Rice, S., and Rockower, D. (1989). *ModSim: a Language for Object-Oriented Simulation; Reference Manual*. 3344 North Torrey Pines Ct., La Jolla, California 92037: CACI Products Company.
- Belanger, R., Mullarney, A., Rice, S., and West, J. (1989). *ModSim: a Language for Object-Oriented Simulation; Tutorial*. 3344 North Torrey Pines Ct., La Jolla, California 92037: CACI Products Company.
- Bennett, D. A., Armstrong, M. P., and Weirich, F. (1993). "An object-oriented model base management system for environmental simulation." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Bodelson, K., and Butler-Villa, E. (1995). Santa Fe Institute.
<http://www.santafe.edu>: Internet URL.
- Botkin, D. B. (Ed.). (1977). *Life and Death in a Forest: The Computer as an Aid to Understanding*. Wiley and Sons.
- Botkin, D. B., Janak, J. F., and Wallis, J. R. (1972). "Some ecological consequences of a computer based model of forest growth." *Journal of Ecology*, 60, 849-872.
- Bratko, I. (1990). *PROLOG: Programming for Artificial Intelligence*. Worthingham, England: Addison-Wesley Publishing Company.
- Brimicombe, A. J., and Bartlett, J. M. (1993). "Linking Geographical Information Systems with Hydraulic Simulation Modeling for Flood Assessment: The Hong Kong Approach." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.

- Buckley, D. J., Coughenour, M., Blyth, C., O'Leary, D., and Bentz, J. (1993). "The Ecosystem Management Model Project: Integrating Ecosystem Simulation Modelling and ARC/INFO in the Canadian Parks Service." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Burrough, P. A. (1986). *Principles of Geographical Information Systems for Land Resource Assessment*. Oxford: Clarendon Press.
- Caswell, H. (1976). "Community structure: a neutral model analysis." *Ecological Monographs*, 46, 327-354.
- Caswell, H. (1978). "Predator-mediated coexistence: a nonequilibrium model." *American Naturalist*, 112(983), 127-154.
- Chesson, P. L., and Case, T. J. (1986). "Overview: Nonequilibrium Community Theories: Chance, variability, History, and Coexistence." In J. Diamond and T. J. Case (Eds.), *Community Ecology*. New York: Harper and Row.
- Clarke, K. C., Olsen, G., and Brass, J. A. (1993). "Refining a cellular automaton model of wildfire propagation and extinction." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Clements, F. E. (1936). "Nature and structure of the climax." *Journal of Ecology*, 24(252-284).
- Costanza, R., DeBellevue, E. B., Maxwell, T., and Jacobsen, M. (1993). "Development of the Patuxent Landscape Model (PLM)." [UMCEES] CBL Ref. No. 190: University of Maryland, Center for Environmental and Estuarine Studies, Chesapeake Biological Laboratory.
- Costanza, R., Fitz, H. C., Bartholomew, J. A., and DeBellevue, E. (1992). "The Everglades Landscape Model (ELM): Summary Report of Task 1, Model Feasibility Assessment." [UMCEES] CBL Ref. No. 190: University of Maryland, Center for Environmental and Estuarine Studies, Chesapeake Biological Laboratory.
- Costanza, R., and Maxwell, T. (1991). "Spatial ecosystem modelling using parallel processors." *Ecological Modeling*, 58, 159-183.
- Costanza, R., Sklar, F. H., and White, M. L. (1990). "Modeling Coastal Landscape Dynamics." *Bioscience*, 40(2), 91-107.
- Costanza, R., Sklar, R. F., Day, J. W. Jr. (1986). Modeling spatial and temporal succession in the Atchafalaya/Terrebonne marsh/estuarine complex in South Louisiana, *Estuarine Variability*, (pp. 387-404). New York: Academic.
- Cronshey, R. G., Theurer, F. D., and Glenn, R. L. (1993). "GIS - Water Quality Computer Model Interface: A Prototype." Paper presented at the Second International Conference/Workshop on

Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.

Cuddy, S. M., Davis, J. R., and Whigham, P. A. (1993). "An Examination of Integrating Time and Space in an Environmental Modelling System." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.

D'Agnese, F. A., Turner, A. K., and Faunt, C. C. (1993). "Using Geoscientific Information Systems For Three-Dimensional Regional Ground-water Flow Modeling in the Death Valley Region, Nevada and California." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.

DeAngelis, D. L., Shuter, B. J., Ridgeway, M. S. and Sheffer, M. (1993a). "Modeling growth and survival in an age-0 fish cohort." *Transactions of the American Fisheries Society*, 122(5), 927-941.

DeAngelis, D. L., Rose, K. A., Crowder, L. B. and Marschall, E. A. (1993b). "Fish cohort dynamics - applications of complementary modeling approaches." *American Naturalist*, 142(4), 604-622.

DeAngelis, D. L., and Waterhouse, J. C. (1987). "Equilibrium and Non Equilibrium Concepts in Ecological Models." *Ecological Monographs*, 57(1), 1-21.

Delcourt, H. R., and Delcourt, P. A. (1991). *Quaternary Ecology: A Paleoecological Perspective*. London: Chapman and Hall.

DePinto, J. V., Atkinson, J. F., Calkins, H. W., Densham, P. J., Guan, W., Lin, H., Xia, F., Rodgers, P. W., Slawewski, T., and Richardson, W. L. (1993). "Development of GEO-WAMS: A modeling support system for integrating GIS with watershed analysis models". Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.

Fishwick, P. A. (1992). "SIMPACT: Getting Started with Simulation Programming in C and C++." Technical Report TR92-022: Dept of Computer and Information Science; University of Florida; Bldg. CSE, Room 301; Gainesville, FL 32611.

Fleming, D. M., Wolff, W. F. and DeAngelis, D. L. (1994). "Importance of landscape heterogeneity to wood storks in Florida Everglades." *Environmental Management*, 18(5), 743-757.

Frederickson, K. E., Westervelt, J. D., and Johnston, D. M. (1994). *A Geographic Information System/Hydrologic Modeling Graphical User Interface for Flood Prediction and Assessment*. Technical Report EC-95/03/ADA 290202: U.S. Army Construction Engineering Research Laboratories.

Frysjinger, S. P. (1993). "An open architecture for environmental decision support." *International Journal of Microcomputers in Civil Engineering*, 10(2), 123.

- Frysjer, S. P., Copperman, D. A., and Levantino, J. P. (1995). "Environmental decision support systems: an open architecture integrating modeling and GIS." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Gardner, R. H., O'Neill, R. V., and Turner, M. G. (1993). "Ecological implications of landscape fragmentation." In S. T. A. Pickett and M. J. McDonnell (Eds.), *Humans as Components of Ecosystems: the Ecology of Subtle Human Effects and Populated Areas*. New York, NY: Springer-Verlag.
- Gardner, R. H., Turner, M. G., O'Neill, R. V., and Lavorel, S. (Eds.). (1991). *Simulation of the scale-dependent effects of landscape boundaries on species persistence and dispersal*. N. Y.: Chapman and Hall.
- Gaur, N., and Vieux, B. (1992). r.fea (Version GRASS 4.1). Oklahoma City: U.S. Army Construction Engineering Research Laboratory.
- Gilpin, M. E. (Ed.). (1990). *Extinction of finite metapopulations in correlated environments*. Oxford: Oxford Science Publications.
- Hall, C. A. S., and Day, J. W., Sr. (Eds.). (1977). *Systems and Models: Terms and Basic Principles*. Wiley and Sons.
- Hannon, B., and Ruth, M. (1994). *Dynamic Modeling*. New York: Springer-Verlag.
- Hanski, I. (1985). "Single-species spatial dynamics may contribute to long-term rarity and commonness." *Ecology*, 66, 335-343.
- Hanski, I., and Gilpin, M. (1991). "Metapopulation dynamics: brief history and conceptual domain." *Biological Journal of the Linnean Society*, 42, 3-16.
- Hansson, L. (1991). "Dispersal and connectivity in metapopulations." *Biological Journal of the Linnean Society*, 42, 89-103.
- Hastings, A. (1980). "Disturbance, coexistence, history and competition for space." *Theoretical Population Biology*, 18, 363-373.
- Hay, L., Knapp, L., and Bromberg, J. (1993). "Integrating geographic information systems, scientific visualization systems, statistics, and an orographic precipitation model for a hydro-climatic study of the Gunnison River Basin, Southwestern Colorado." Paper presented at the Proceedings of the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, CO.
- Herring, C., Kalathil, B., and Teo, J. (1993). *Research in Persistent Simulation: Development of the Persistent ModSim Object-Oriented Programming Language*. USACERL Interim Report FF-93/07/ADA268568. Champaign, Illinois: U.S. Army Construction Engineering Research Laboratories.

- Hiebeler, D. (1994). "The Swarm Simulation System and Individual-based Modeling." Paper presented at the Decision Support 2001: Advanced Technology for Natural Resource Management, Toronto, Canada.
- Horn, H. S., and MacArthur, R. H. (1972). "Competition among fugitive species in a harlequin environment." *Ecology*, 53, 749-752.
- Johnson, A. R. (1993). "Spatiotemporal hierarchies in ecological theory and modeling." Paper presented at the Second international Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Kareiva, P., and Anderson, M. (1988). "Spatial aspects of species interactions: the wedding of models and experiments." In A. Hastings (Ed.), *Community Ecology*, (pp. 38-54). New York, New York: Springer-Verlag.
- Kessell, S. R. (1993). "The integration of empirical modeling, dynamic process modeling, visualization and GIS for bushfire decision support in Australia." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Kingsland, S. E. (1985). *Modeling Nature*. Chicago: The University of Chicago Press.
- Krummel, J. R., Dunn, C. P., Eckert, T. C., and Ayers, A. J. (1993). "A technology to analyze spatiotemporal landscape dynamics: application to Cadiz Township (Wisconsin)." Paper presented at the Second International Conference/ Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Krzysik, A. (1994). The Desert Tortoise at Fort Irwin, California: A Federal threatened species: United States Army Corps' Construction Engineering Research Laboratories.
- Krzysik, A., and Woodman, P. (1991). "Six Years of Army Training Activities and the Desert Tortoise." The Desert Council, *Proceedings of 1987-1991 Symposia*, 337-368.
- Langton, C., and Hiebeler, D. (1990). *Cellsim* (Version 2.5). Los Alamos, NM: Center for Nonlinear Studies; Los Alamos National Laboratory.
- Leavesley, G. H., Restrepo, P. J., Stannard, L. G., and Dixon, M. (1991). "The Modular Hydrologic Modeling System." Paper presented at the First International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Boulder, Colorado.
- Leavesley, G. H., Restrepo, P. J., Stannard, L. G., Frankoski, L. A., and Sautins, A. M. (1993). "The Modular Modeling System." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Boulder, Colorado.
- Levin, S. (1988). "Pattern, scale, and variability: an ecological perspective." In A. Hastings (Ed.), *Community Ecology*, (pp. 1-13). New York, New York: Springer-Verlag.

- Levin, S. A. (1989). "Ecology in Theory and Application." In S. A. Levin, R. G. Hallam, and L. J. Gross (Eds.), *Applied Mathematical Ecology*, (pp. 3-8). Berlin: Springer-Verlag.
- Levins, R. (1969). "Some demographic and genetic consequences of environmental heterogeneity for biological control." *Bulletin of the Entomological Society of America*, 15, 237-240.
- Levins, R., and Culver, D. (1971). "Regional coexistence of species and competition between rare species." *Proceedings of the National Academy of Sciences*, 68, 1246-1248.
- Loucks, O. L. (1970). "Evolution of diversity, efficiency, and community stability." *American Zoologist*, 10, 17-25.
- Loucks, O. L. (1985). "Looking for surprise in managed stressed ecosystems." *BioScience*, 35(7), 428-432.
- MacArthur, R. H., and Wilson, E. O. (1967). *The Theory of Island Biogeography*. Princeton: Princeton University Press.
- Maidment, D. R. (1991). "GIS and hydrologic modeling." Paper presented at the First International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling.
- Maidment, D. R. (1993). "Environmental modeling within GIS." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Maxwell, T. (1995). "Distributed Modular Spatial Ecosystem Modeling."
<http://kabir.umd.edu/SMP/MVD/CO.html>: University of Maryland.
- Maxwell, T., and Costanza, R. (1993). "Spatial Ecosystem Modeling in a Distributed Computational Environment." In J. van den Berg and J. van der Straaten (Eds.), *Concepts, Methods, and Policy for Sustainable Development*; Island Press.
- May, R. M. (1986). "The search for patterns in the balance of nature: advances and retreats." *Ecology*, 67(5), 1115-1126.
- Minar, N. (1995). Swarm Web Pages. <http://www.santafe.edu/projects/swarm/>: The Santa Fe Institute.
- Mladenhoff, D. J., Host, G. E., and Broeder, J. (1993). "LANDIS: A spatial model of forest landscape disturbance, succession, and management." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Morrison, V. P. (1995). "Import/Dome Language Reference Manual" (Unofficial software document).
ftp_pike.cecer.army.mil:/pub/asset/import: U.S. Army Construction Engineering Research Laboratories.

- Nee, S., and May, R. M. (1992). "Dynamics of metapopulations: habitat destruction and competitive coexistence." *Journal of Animal Ecology*, 61, 37-40.
- O'Neill, R. V., DeAngelis, D. L., Waide, J. B., and Allen, T. F. H. (1986). *A Hierarchical Concept of Ecosystems*. Princeton: Princeton University Press.
- O'Neill, R. V., Gardner, R. H., and Turner, M. G. (1992). "A hierarchical neutral model for landscape analysis." *Landscape Ecology*, 7(1), 55-61.
- O'Neill, R. V., Johnson, A. R., and King, A. W. (1989). "A hierarchical framework for the analysis of scale." *Landscape Ecology*, 3, 193-206.
- O'Neill, R. V., Krummel, J. B., Gardner, R. H., Sugihara, G., Jackson, B., DeAngelis, D. L., Milne, B. T., Turner, M. G., Zygmunt, B., Christensen, S. W., Dale, V. H., and Graham, R. L. (1988). "Indices of landscape pattern." *Landscape Ecology*, 1(3), 153-162.
- Overton, W. S. (Ed.). (1977). *A Strategy for Model Construction*. Wiley and Sons.
- Pickett, S. T. A., Kolasa, J., Armesto, J. J., and Collins, S. L. (1989). "The ecological concept of disturbance and its expression at various hierarchical levels." *Oikos*, 54, 129-136.
- Ray, T. (1994a). Tierra Simulator V4.1,
<http://alife.santafe.edu/pub/SOFTWARE/Tierra>: Santa Fe Institute.
- Ray, T. S. (1994b). "An evolutionary approach to synthetic biology: Zen and the art of creating life." *Artificial Life*, 1(1/2), 195-226.
- Reice, S. R. (1994). "Nonequilibrium determinants of biological community structure." *American Scientist*, 82, 424-435.
- Rewerts, C. C., and Engel, B. A. (1991). "ANSWERS on GRASS: Integrating a watershed simulation with a GIS." ASAE Paper 91-2621: ASAE, St. Joseph, MI.
- Robinson, G. R., Holt, R. D., Gaines, M. S., Hamburg, S. P., Johnson, M. L., Fitch, H. S., and Martinko, E. A. (1992). "Diverse and contrasting effects of habitat fragmentation." *Science*, 257, 524-525.
- Rothermel, R. C. (1972). "A mathematical model for predicting fire spread rate and intensity in wildland fuels (INT-115)." USDA Forest Service Research Paper.
- Saghafian, B. (1993). "Implementation of a distributed hydrological model within Geographical Resources Analysis Support System (GRASS)." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Shattuck, R. (1993). "Smart-Pixel Technology" (Product Report): Titan Corporation and George Mason University.

- Shapiro, M., Westervelt, J. D., Gerdes, D. P., Higgins, M. E., Larson, M. J. (1993). *GRASS 3.0 Programmer's Manual* (USACERL Technical Report N-89/14): U.S. Army Construction Engineering Research Laboratories.
- Sheffer, M., Baveco, J. M., DeAngelis, D. L., Rose, K. A., and Vannes, E. H. (1995). "Super-Individuals and simple solution for modelling large populations on an individual basis." *Ecological Modelling*, 80(2-3), 161-170.
- Slatkin, M. (1974). "Competition and regional coexistence." *Ecology*, 55, 128-134.
- Star, J., and Estes, J. (1990). *Geographic Information Systems*. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.
- Stommel, H. (1963). "Varieties of oceanographic experience." *Science*, 139, 572-576.
- Tazik, D., Warren, S. D., Diersing, V. E., Shaw, R. B., Brozka, R. J., Bagley, C. F., and Whitworth, W. R. (1992). U. S. Army Land Condition-Trend Analysis (LCTA) Plot Inventory Field Methods (Technical Report N-92/03/ADA 247931): U. S. Army Construction Engineering Research Laboratories.
- Thau, R. S. (1995). MIT Artificial Intelligence Laboratory.
<http://www.ai.mit.edu>: Massachusetts Institute of Technology.
- Tilman, D. (1994). "Competition and biodiversity in spatially structured habitats." *Ecology*, 75(1), 2-16.
- Tilman, D., May, R. M., Lehman, C. L., and Nowak, M. A. (1994). "Habitat destruction and the extinction debt." *Nature*, 371, 65-66.
- Tomlin, C. D. (1990). *Geographical Information Systems and Cartographic Modeling*. Englewood Cliffs, N. J.: Prentice Hall.
- Turner, M. G. (1989). "Landscape Ecology: The effect of pattern on process". *Annual Review of Ecological Systems*, 20, 171-197.
- Turner, M. G., Gardner, R. H., and Dale, V. H. (1989). "Predicting the spread of disturbance across heterogeneous landscapes." *Oikos*, 55, 121-129.
- Turner, M. G., O'Neill, R. V., Gardner, R. H., and Milne, B. T. (1988). "Effects of changing spatial scale on the analysis of landscape pattern." *Landscape Ecology*, 3(3/4), 153-162.
- Urban, D. L., Bonan, G. B., and Smith, T. M. (1991). "Spatial applications of gap models." *Forest Ecology and Management*, 42, 95-110.
- Urban, D. L., O'Neill, R. V., and Shugart, H. H. (1987). "Landscape ecology." *BioScience*, 37, 119-127.

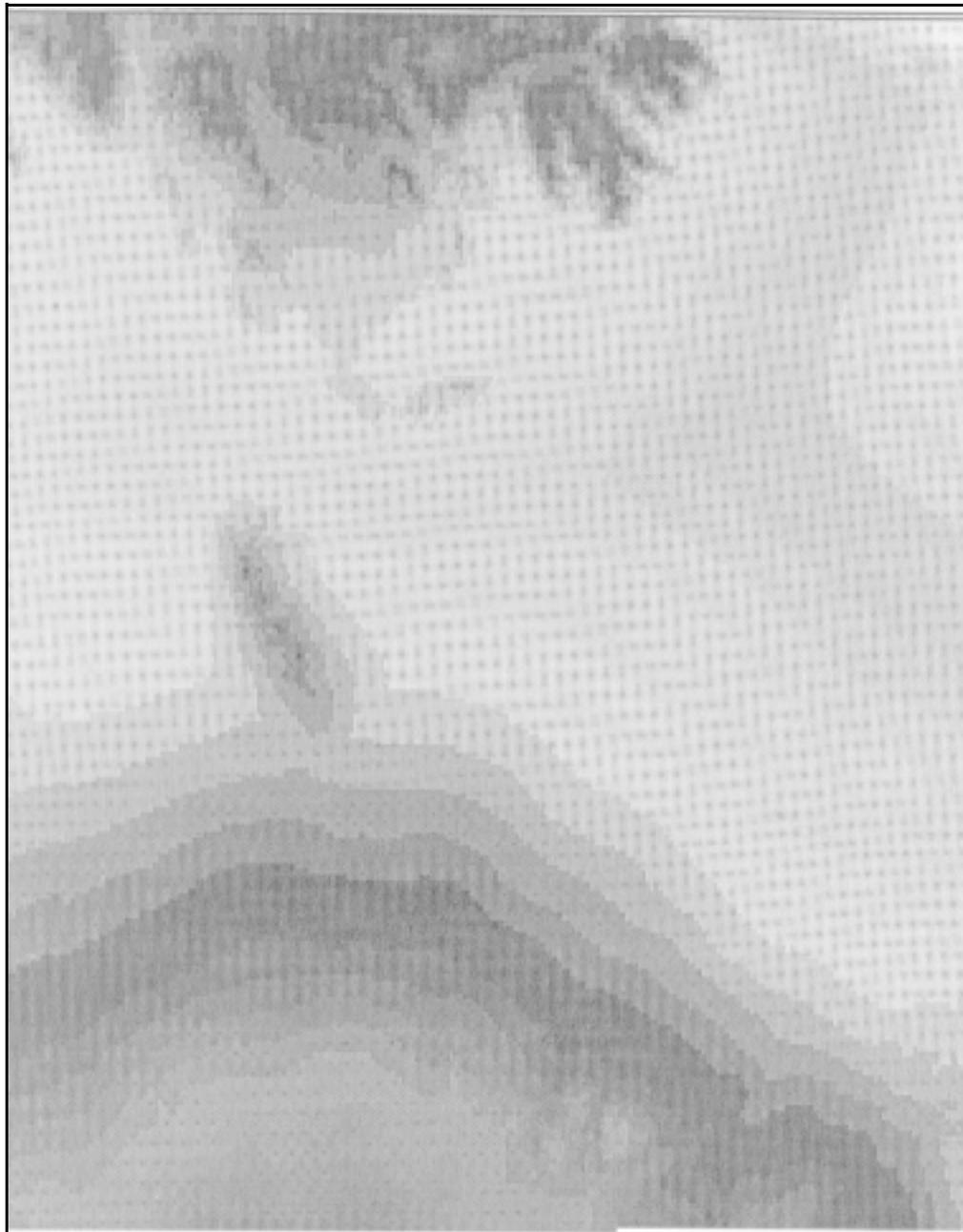
- Vieux, B., and Westervelt, J. (1992). "Finite element modeling of storm water runoff using GRASS GIS." Paper presented at the Computing in Civil Engineering and Geographic Information Systems Symposium, Dallas, Texas.
- Vieux, B. E., Farajalla, N. S., and Gaur, N. (1993). "Integrated GIS and distributed stormwater runoff modeling." Paper presented at the Second International Conference/Workshop on Integrating Geographic Information Systems and Environmental Modeling, Breckenridge, Colorado.
- Walde, S. J. (1991). "Patch dynamics of a phytophagous mite population: effect of number of subpopulations." *Ecology*, 72(5), 1591-1598.
- Westervelt, J. D. (1978). "Development and Demonstration of LAGRID: A Grid-Cell Data Base Management and Analysis Package." Unpublished Masters Thesis, University of Illinois.
- Westervelt, J. D. (1990). BP: A Back Propagation Neural Network Simulator (USACERL Software): Construction Engineering Research Laboratories.
- Westervelt, J. D., Hannon, B., Levi, S., and Harper, S. (1998). A Landscape Simulation Demonstration of the Desert Tortoise Habitat at Fort Irwin, California (USACERL Technical Report 98/76): US Army Corps of Engineers' Construction Engineering Research Laboratories.
- Westervelt, J. D., Hannon, B., Sheikh, P., Cheng, A., Rubin, C., Dvorak, B., Nugteren, A., Gildensoph, L., Mitsova, H., Lambert, E., Iverson, L., Pabich, K., and Shapiro, M. (1995). *Dynamic, Spatial, Ecological Modeling: A Demonstrated Simulation of the Sage Grouse Habitat at the Yakima Training Center, Washington* (USACERL Technical Report 95/16 ADA 299720): U.S. Army Construction Engineering Research Laboratories.
- Westervelt, J. D., Shapiro, M., Goran, W. D., and Gerdes, D. P. (1992). *Geographic Resources Analysis Support System (GRASS) Version 4.0 User's Reference Manual* USACERL Technical Report N-87/22 Rev./ADA 255218: U.S. Army Construction Engineering Research Laboratories.
- Whigham, P. A., and Davis, J. R. (1989). "Modelling with an integrated GIS/expert system". Paper presented at the ESRI User Conference, Palm Springs, CA.
- Wiens, J. A., Addicott, J. F., Case, T. J., and Diamond, J. (1985). "Overview: the importance of spatial and temporal scale in ecological investigations." In J. Diamond and T. J. Case (Eds.), *Community Ecology*, (pp. 169-193). New York, New York: Harper and Row.
- Wu, J., and Levin, S. A. (1994). "A spatial patch dynamic modeling approach to pattern and process in an annual grassland." *Ecological Monographs*, 64(4), 447-464.
- Wu, J., and Loucks, O. L. (1991). "Balance-of-nature and modern ecological theory: a shift in ecological thinking." In Sino-ECO (Ed.), *Development and Trends in Modern Ecology*. Hefei: Univ. Sci. and Tech. Press.
- Wu, J., Vankat, J. L., and Barlas, Y. (1993). "Effects of patch connectivity and arrangement on animal metapopulation dynamics: a simulation study." *Ecological Modeling*, 65, 221-254.

Wu, X., and Westervelt, J. D. (1994). *Using Neural Networks To Correlate Satellite Imagery and Ground-truth Data* (USACERL Special Report EC-94/28/ADA 285486): U.S. Army Construction Engineering Research Laboratories.

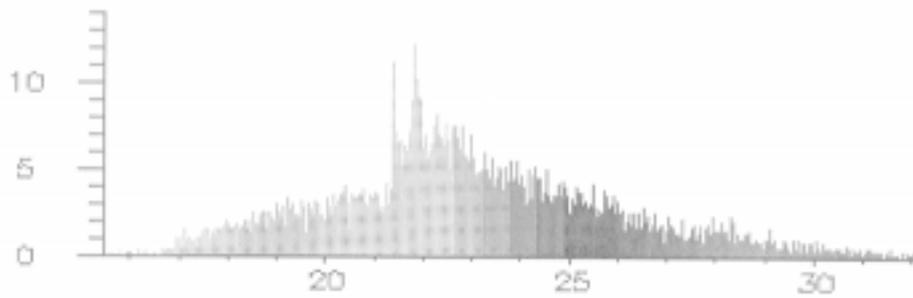
Yaeger, L. (1993). "Computational Genetics, Physiology, Metabolism, Neural Systems, Learning, Vision, and Behavior or PolyWorld: Life in a New Context" (report): Apple Computer, Inc.

Appendix A: GRASS Maps

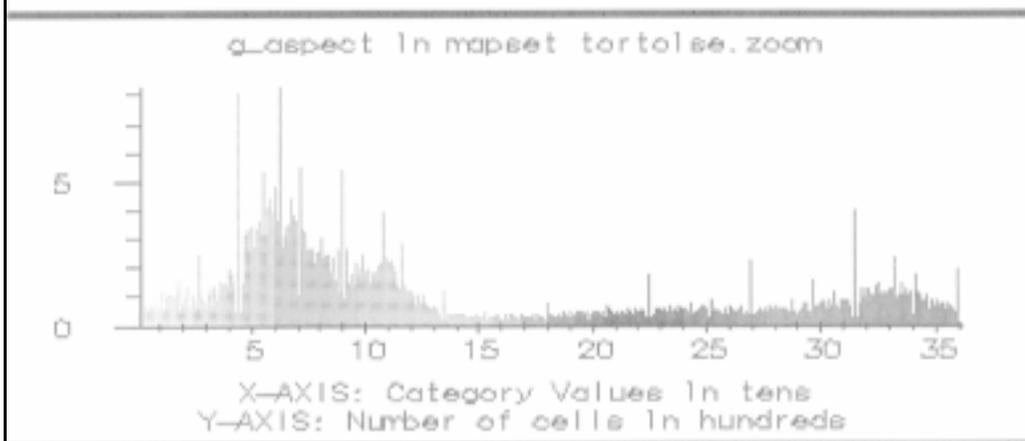
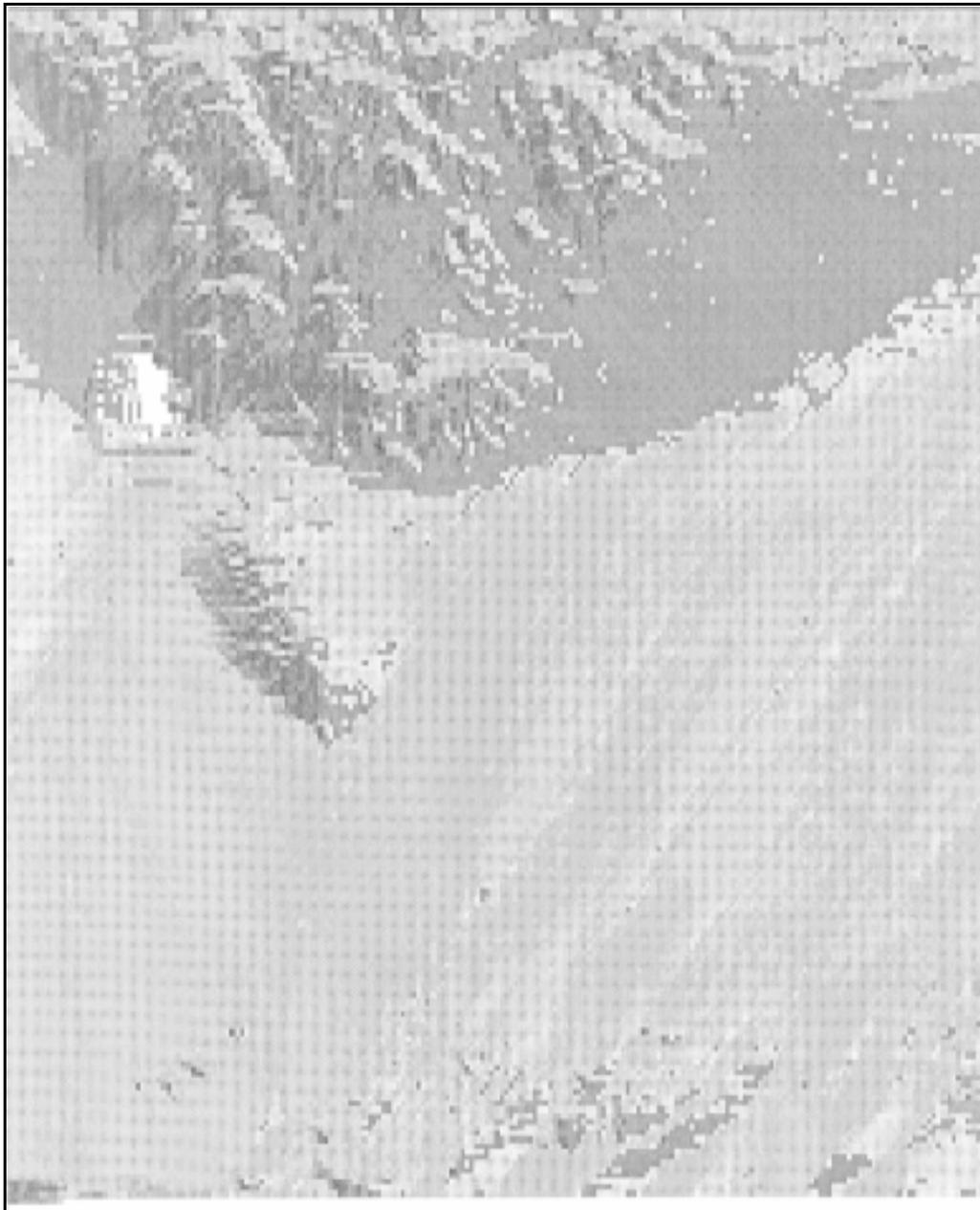
The following pages contain renderings of the GRASS maps used either directly by the static MapManager IMPORT/DOME object class, or indirectly by the dynamic MapManager object class. In the later case, the GRASS maps are used to initialize the dynamic landscape simulation designed through Stella and translated into a raster-based landscape simulation C++ program by the Spatial Modeling Environment (SME). All maps have a resolution of 60 meters per cell with 200 cells in both axes yielding a full 40,000 cells. The area covers a central southern portion of Fort Irwin's National Training Center that is about 7.5 miles square.

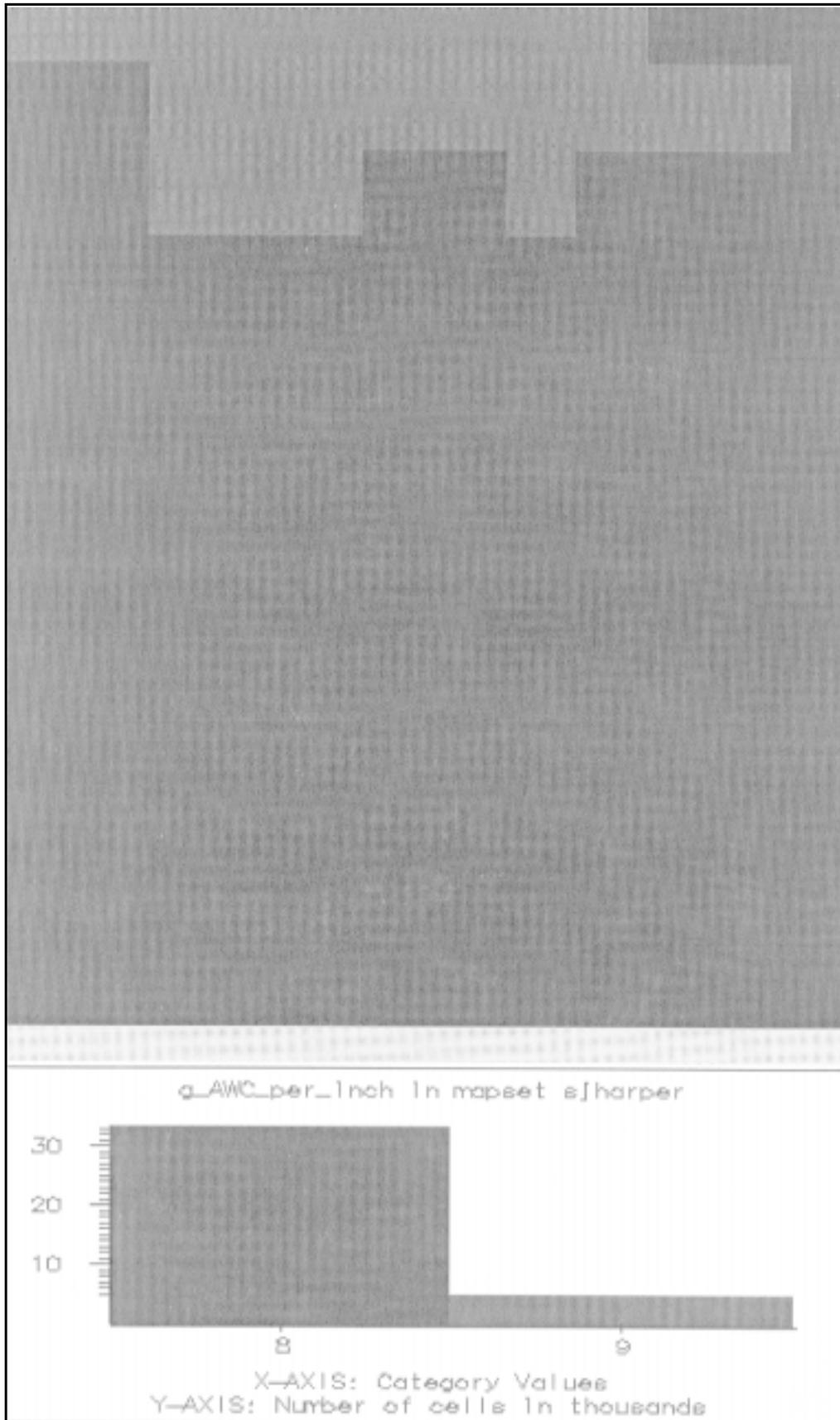


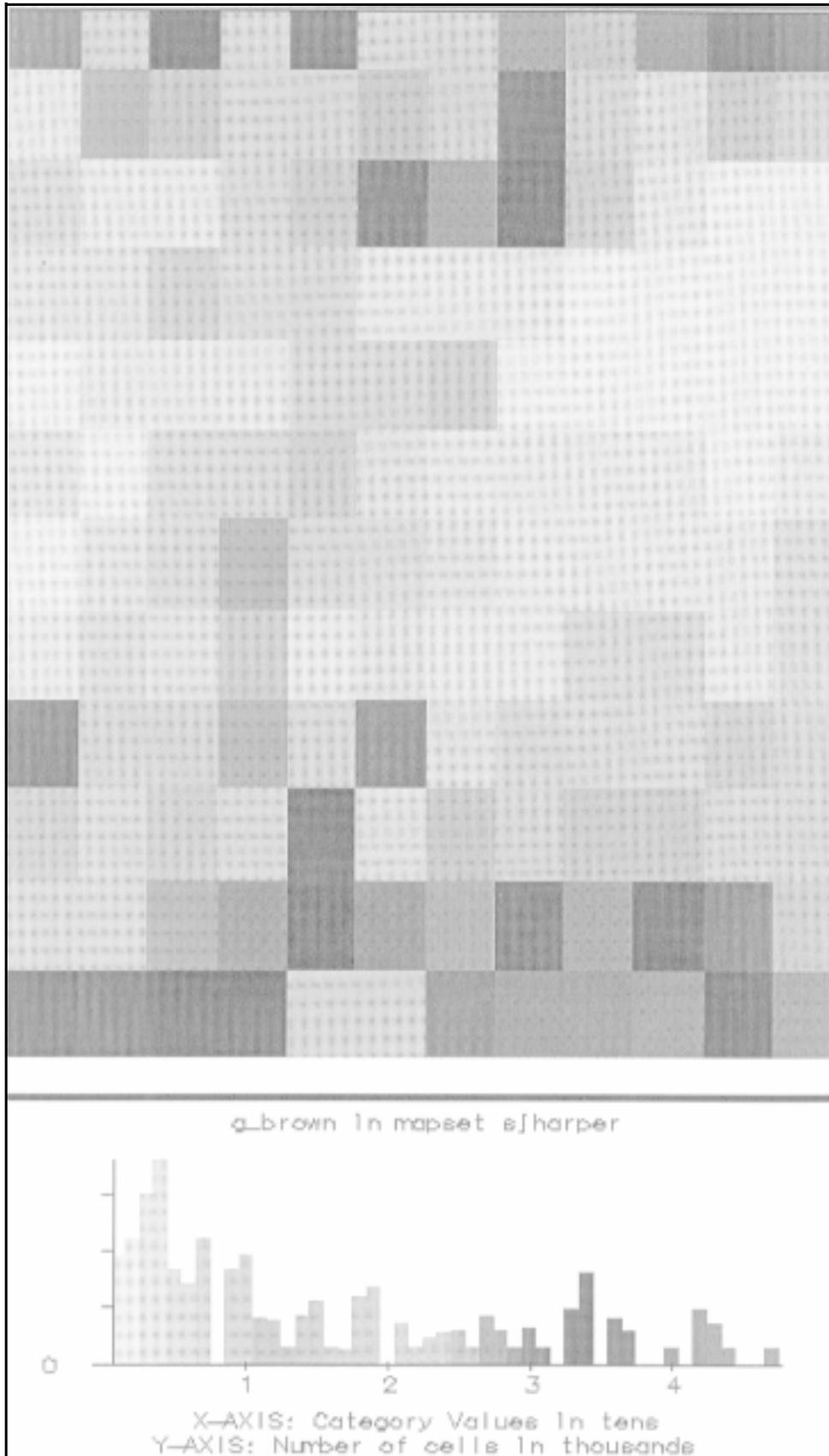
g_elev In mapset tortoise.zoom



X-AXIS: Category Values In hundrede
Y-AXIS: Number of cells In tens

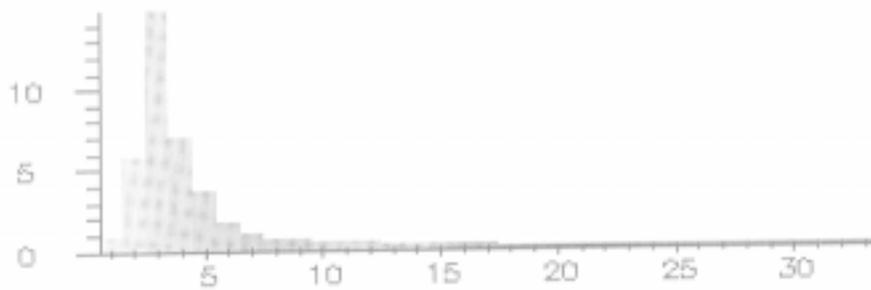








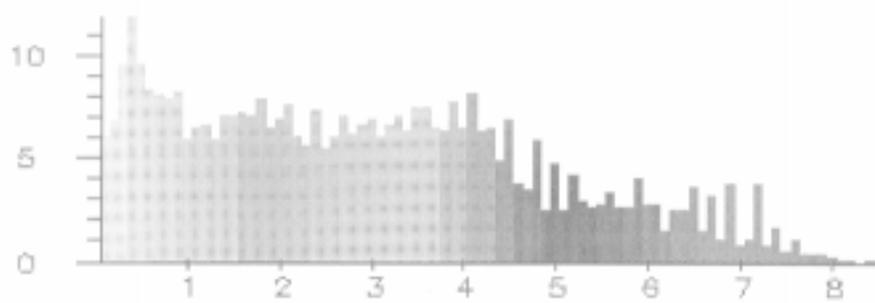
g_slope in mapset tortoise.zoom



X-AXIS: Category Values
Y-AXIS: Number of cells in thousands



a_vegetation in mapset tortoise.zoom



X-AXIS: Category Values in tens
Y-AXIS: Number of cells in hundreds

Appendix B: Import/Dome Object Classes

B.1 General Purpose Objects

The objects described here provide general purpose support for dynamic spatial modeling and simulation. They can be viewed as extensions to the Import/Dome standard runtime libraries.

GaussRandom = OBJECT(Random)

MODULE GAUSSRANDOM

IMPORTED MODULES:

RANDOM

DESCRIPTION

Generates random numbers that collectively form a gaussian distribution with a normal of 0 and a standard deviation of 1. This is a subclass of the IMPORT/DOME Random class.

METHODS

CONSTRUCTOR METHOD GaussRandom()

Instantiates an instance of the gaussian random number generator

ASK METHOD ReturnValue() : REAL

Returns a random number that is part of a distribution with a normal of 0 and a standard deviation of 1.

GraphFunction = OBJECT

MODULE GRAPHFUNCTION

IMPORTED MODULES:

None

DESCRIPTION

Used in situations where a dependent variable is derived from a single independent variable using a series of x,y coordinates that represent a graph. For values between explicitly provided dependent variables, a linear interpolation between the next highest and next lowest associated independent variables is used to return the dependent value.

METHODS

CONSTRUCTOR METHOD GraphFunction(IN Independent, Dependent:
LIST)

Inputs are IMPORT lists that contain REAL values. The length of the two lists must be greater than two and the Independent list must be monotonically increasing.

ASK METHOD ReturnValue(IN X : REAL) : REAL

Any X value will be converted to a corresponding dependent variable value. This is accomplished with a linear interpolation between the pair numbers in the input list (Independent) that bracket the input value. Input values that are lower than the lowest value in the In list will cause this lowest value to be returned. Similarly, an input that is higher than the highest number on the theIn list returns that highest number.

GRASS_interface = OBJECT INTERFACE(pointers to C++ code)

MODULE GRASS_INTERFACE

IMPORTED MODULES:

None

DESCRIPTION

Encapsulates the C++ code, which in-turn encapsulates the GRASS C subroutines for availability to IMPORT. The GRASSMap object is defined in the same module. These provide access to GRASS and can readily be extended to provide support for other GRASS capabilities. It is not recommended that model programmers use these objects directly, but rather access the capabilities through the GRASS object.

METHODS

CONSTRUCTOR METHOD GRASS_interface();

Initializes GRASS through the GRASS G_gisinit() C subroutine.

ASK METHOD SetWindow(IN N, S, E, W, NS_res, EW_res : REAL) ;

Sets the GRASS region, through which maps will be accessed, to these values. Note that this affects the current process only and is remembered.

ASK METHOD GetWindow(OUT N, S, E, W, NS_res, EW_res : REAL) ;

Returns the current GRASS region.

GRASS = OBJECT(GRASS_interface)

MODULE: GRASS

IMPORTED MODULES:

GRASS_INTERFACE

LINKEDLIST

DESCRIPTION

Provides the primary IMPORT interface to GRASS. It is used to initialize GRASS and then provides the ability to instantiate GRASSMap objects that provide IMPORT access to raster maps. Underlying C++ code

encapsulates access to GRASS data through standard GRASS libraries written in C. Use of GRASS in this, and any other, environment requires that certain GRASS environment variables and files in the users home directory be properly established. One way to ensure this is to run the compiled IMPORT/DOME code that uses this object from within GRASS.

METHODS

CONSTRUCTOR METHOD GRASS()

Initializes the GRASS C subroutines.

ASK METHOD OpenMap(IN MapName: STRING): GRASSMap

With map names in the standard GRASS map name format, the associated

map is opened and initialized. GRASS maps are opened as "segmented" maps. This process requires that a GRASS map be reformatted at run-time into a form that provides an efficient method of paging sections of the map into core memory. There is an up-front reformatting cost that is indicated at run-time with a running percentage completion output while each map is opened. This METHOD should be modified to provide the opportunity to open maps in this manner as well as (1) reading the map into core memory and (2) not reformatting the map at all, but relying on standard GRASS map reads for each data request.

ASK METHOD CloseMap(IN Map: GRASSMap)

Maps are closed and allocated memory is deallocated.

ASK METHOD CheckForMap(IN MapName: STRING): GRASSMap

Queries an internal list of currently opened maps, returning the GRASSMap object if the map is found to be opened and available.

Returns NULL otherwise.

ASK METHOD ListMaps()

Lists to standard error all of the currently opened maps.

GRASSMap = OBJECT INTERFACE(Pointers to C++ code)

MODULE GRASS_INTERFACE

IMPORTED MODULES:

None

DESCRIPTION

Low-level IMPORT encapsulation of C++ code which in-turn encapsulates the GRASS C routines that open, close, read and write GRASS maps. GRASS maps should be opened through the GRASS object, which returns

a GRASSMap object, but may then be manipulated through the METHODS

available to GRASSMap objects.

METHODS

CONSTRUCTOR METHOD GRASSMap(IN NapName: STRING) ;

Opens a GRASS map and prepares the data for rapid random access. Subsequent map reads and writes are with respect to the random access segment map.

DESTRUCTOR METHOD GRASSMap() ;

Closes the associated GRASS map and deallocates memory.

ASK METHOD GiveName() : STRING ;

Returns the name of the maps used to open it initially.

ASK METHOD GetValue(IN E, N : REAL) : REAL ;

Returns a value at the Easting and Northing value specified. E and N are in the native units of the GRASS database currently being used. Value is in units of the associated GRASS map, but always cast as a REAL.

ASK METHOD PutValue(IN E, N : REAL; IN Value : REAL): INTEGER ;

Changes the value in the GRASS map at the given coordinates to the given value.

ASK METHOD WriteOut(IN MapName : STRING): INTEGER ;

Writes the map back out under the specified name.

SME = OBJECT INTERFACE(Pointers to C++ code)

MODULE SME

IMPORTED MODULES:

None

DESCRIPTION

Low-level IMPORT encapsulation of C++ code that open, run, and close SME simulations and read and write SME maintained simulation maps.

METHODS

CONSTRUCTOR METHOD SME() ;

Opens the SME simulation associated with the compiled IMPORT/DOME program.

DESTRUCTOR METHOD SME() ;

Closes the associated SME simulation.

ASK METHOD Update(IN steps : INTEGER);

Runs the SME simulation through "steps" dT steps.

ASK METHOD GetValue(IN Index, Row, Col : INTEGER) : REAL ;

Returns a value of map associated with index value "Index" at the row and column coordinate specified. Row and Col are provided in the SME

coordinate system that places the 0,0 position at the northwest corner of the study area.

ASK METHOD SetValue(IN Index, Row, Col : INTEGER; IN Value : REAL):
IINTEGER ;

Changes the value in the SME map at the given coordinates to the given value. Note that coordinates are in row and column values.

LINKEDLIST = OBJECT

MODULE LINKEDLIST

IMPORTED MODULES:

None

DESCRIPTION

Together with object LINK, LINKEDLIST provides a utility for storing and retrieving data in linked lists. The user instantiates the list and adds/removes links with LINKEDLIST methods. The LINK object is used for traversing the list and for retrieving stored objects associate with each link.

METHODS

CONSTRUCTOR METHOD LINKEDLIST()

Instantiates a new, empty list.

ASK METHOD AddLink(IN Item: TAGGED) : LINK

Adds a new link to the end of the list and returns that link.

ASK METHOD GetFirstLink() : LINK

Returns the first link in the list. This provides the first step in traversing a list.

ASK METHOD RemoveLink(IN Link: LINK)

Deletes a link from the list.

LINK = OBJECT

MODULE LINKEDLIST

IMPORTED MODULES:

None

DESCRIPTION

Together with object LINKEDLIST, LINK provides a utility for storing and retrieving data in linked lists. The user instantiates the list and adds/removes links with LINKEDLIST methods. The LINK object is used for traversing the list and for retrieving stored objects associate with each link.

METHODS

CONSTRUCTOR METHOD LINK(INNewItem : TAGGED; IN theList : LINKEDLIST)

This is intended to be used exclusively by the LINKEDLIST object when it is requested to add a link.

ASK METHOD SetPreviousLink(IN Link : LINK)

Used by LINKEDLIST to establish the link within a list.

ASK METHOD SetNextLink(IN Link : LINK)

Used by LINKEDLIST to establish the link within a list.

ASK METHOD ReturnItem() : TAGGED

Returns the item associated with the link.

ASK METHOD ReturnPreviousLink() : LINK

Identifies the previous link in the linked list; NULL if there is no link.

ASK METHOD ReturnNextLink() : LINK

Identifies the previous link in the linked list; NULL if there is no link.

ASK METHOD ReturnList() : LINKEDLIST

Returns the linked list with which this link is associated.

PenPlot = OBJECT

MODULE STRIPCHART

IMPORTED MODULES:

Xgraph

ASSOCIATED OBJECTS:

StripChart

DESCRIPTION

PenPlot objects are instantiated with calls to StripChart objects. They are the “pen objects” associated with a stripchart. It is recommended that only the AddPoint METHOD be used directly.

METHODS

CONSTRUCTOR METHOD PenPlot(IN low, high: REAL; IN Units, Color: STRING; IN Connect: BOOLEAN): PenPlot

This instantiates a new “pen” on the stripchart. Establish its low and high range to be “low” and “high” ; set the pen color to “Color”, and identify through the Boolean “Connect” whether data points shall be connected to form a continuous line or be displayed as dots. This METHOD should not normally be used; instead, access it indirectly through the StripChart object. This is because the StripChart class does the actually drawing and does so by querying each PenPlot object. By using StripChart to instantiate a PenPlot, StripChart maintains a list of all associated PenPlots.

ASK METHOD AddPoint(IN data: REAL) ;

The value of “data” is associated with the current time which the object accesses through the SIMTIME() method. Only the last 256 data points are retained. Only those that fit within the time-range originally established with the creation of the associated StripChart are actually displayed.

ASK METHOD Plot() ;

Causes all of the data in the current plot to be updated on the display.

This normally not called except via the Plot method associated with the associated StripChart object.

RandNormGraph = OBJECT(GaussRandom)

MODULE RAND_NORM_GRAPH

IMPORTED MODULES:

GRAPHFUNCTION

GAUSSRANDOM

DESCRIPTION

Provides a one-step approach to a graph-type function which is associated with a running series of normals and standard deviations paired with a dependant value. Calls to the ReturnValue method returns a random number chosen with respect to the normal and standard deviation associated with the input value. This is valueable for generating stochastic temperature, rainfall, and other environmental data.

METHODS

CONSTRUCTOR METHOD RandNormGraph(IN Independent, Dependent, StDev: LIST)

Each of these lists of REAL values must be of the same length; the nth values of each are associated with each other. The “Independent” values must be monotonically increasing and might typically represent the day (or month) of a year. The “Dependent” LIST values provide the normal values of the output variable. Actual output values are perturbed in a gaussian fashion with the “StDev” LIST values that are the “one standard-deviation” values associated with the associated average value.

ASK METHOD ReturnValue(IN Independent: REAL) : REAL

A dependent value associated with the input “Independent” value is established and then perturbed based on the standard-deviation value also associated with the “Independent” value.

RasterManager = OBJECT(MapManager)

MODULE RASTERMANAGER**IMPORTED MODULES:**

MAPMANAGER

DESCRIPTION

All access to landscape maps (and some derived information) is provided to animal objects through the RasterManager. It, in-turn, accesses raw map information and region data through a static or dynamic MapManager.

METHODS

CONSTRUCTOR METHOD RasterManager(OUT N, S, E, W, NS, EW: REAL)

Instantiates the associated MapManager superclass.

ASK METHOD ReturnAreaData(IIN mapname: STRING; IN DayOfYear: INTEGER; IN CenterE, CenterN, range: REAL) : REAL

Returns the average value for the cells within a circle of given location and radius for the identified map.

ASK METHOD ConvertEtoCol(IN east: REAL) : REAL

Converts an Easting UTM value to a raster column value.

ASK METHOD ConvertNtoRow(IN north: REAL) : REAL

Converts a Northing value to a raster row value.

ASK METHOD ReturnN() : REAL

Returns the value of the northernmost edge of the simulation area.

ASK METHOD ReturnS() : REAL

Returns the value of the southernmost edge of the simulation area.

ASK METHOD ReturnE() : REAL

Returns the value of the easternmost edge of the simulation area.

ASK METHOD ReturnW() : REAL

Returns the value of the westernmost edge of the simulation area.

ASK METHOD ReturnNS() : REAL

Returns the value of the north-south cell resolution of the simulation area.

ASK METHOD ReturnEW() : REAL

Returns the value of the east-west cell resolution of the simulation area.

REGION = OBJECT**MODULE REGION****IMPORTED MODULES:**

unknown

DESCRIPTION

This class provides an efficient means for retrieving lists of entities that exist within an arbitrary rectangular landscape space. The principal methods allow landscape objects to register themselves and query for a list of nearby objects. It uses a quad-tree approach internally to register

objects. The number of levels in the quad-tree is specified in the CONSTRUCTOR METHOD. The total number of cells at the lowest level is 4 raised to the number of levels. 5 levels results in 1024 (45) cells at the lowest level (level 0).

METHODS

CONSTRUCTOR METHOD REGION(IN levels: INTEGER; IN N, S, E, W : REAL)

If the level is not 0, four sub-quads are instantiated, otherwise a leaf node is established. This method is used recursively to create a multi-level tree.

DESTRUCTOR METHOD REGION()

Deallocates all quads associated with the region.

ASK METHOD AssignEntity(IN entity: TAGGED; IN E, N : REAL) : LINK

Sends the TAGGED object to the first-level quad, which continues to pass it on until a level 0 quad finally adds the object to its list.

ASK METHOD RemoveEntity(IN LinkID TAGGED)

Entity is identified in the quad-tree and removed. The LinkID is the link address returned by the call to AssignEntity, which originally stored the location.

ASK METHOD GetEntityList() : LIST

Returns a LIST of all the entities currently stored in the region.

ASK METHOD GetEntityListInRectangle(IN N, S, E, W: REAL) : LIST

Returns a LIST of all the landscape objects associated with all 0 level quads that overlap the rectangular area. This LIST is guaranteed to contain all the desired objects but may contain others as well. This is because all objects in a 0 level quad are added to the list if the quad overlaps the desired rectangle at all.

ASK METHOD ConvertCoorsToQuad(IN E, N : REAL) : INTEGER

Identifies the 0 level quad that contains the coordinate.

ASK METHOD ShowStatus()

A debugging METHOD that lists the contents of all 0 level quads.

StripChart = OBJECT

MODULE STRIPCHART

IMPORTED MODULES:

Xgraph

ASSOCIATED OBJECTS:

PenPlot

DESCRIPTION

This object provides the ability to view, in stripchart form, the recent historical values of up to ten variables. Each variable is associated with

its own color and upper/lower limits. Each can be plotted as a continuous line or as discrete dots. Its purpose is to allow for the observation of time-series values at run-time. The PenPlot object works in conjunction with the StripChart object. PenPlot objects accept data to be plotted. StripCharts are intended to be superclasses of view objects that probe models for system status information and then render that information via the stripchart.

METHODS

CONSTRUCTOR METHOD StripChart(IN UnitSteps, SimstepsPerUnit: INTEGER; IN Units: STRING);

Instantiates a stripchart and sets the total amount of time units, the number of simulation steps per unit, and the name of the unit.

ASK METHOD OpenNewPlot(IN low, high: REAL; IN Units, Color: STRING; IN Connect: BOOLEAN): PenPlot

This call tells the penplot to instantiate a new "pen" on the stripchart. Establish its low and high range to be "low" and "high"; set the pen color to "Color", and identify through the boolean "Connect" whether data points shall be connected to form a continuous line or be displayed as dots. This METHOD returns a PenPlot object that is subsequently used to feed in data.

ASK METHOD SetColors(IN Background, Text, Graph: STRING);

Each of these are strings recognizable by X-windows as colors.

Background defines the color of the main background of the graphic object; Text is the color of text; and Graph is the color of the "paper" portion of the stripchart.

ASK METHOD SetMarkings(IN frequency: REAL; IN Color: STRING);

By default, there are no markings on the "paper." This method established markings. "Frequency" sets up how far apart markings will be in the units established in the Constructor method. "Color" identifies the color used for these markings.

ASK METHOD Plot();

Causes all of the data in the current plots to be updated on the display.

ASK METHOD ClosePlots();

DISPOSE's all of the currently active PenPlots.

Register = OBJECT()

MODULE REGISTER

IMPORTED MODULES:

None

DESCRIPTION

A single instantiation of this object is made available to all objects that generate landscape objects. Its purpose is to provide every landscape entity with a unique identification number.

METHODS

CONSTRUCTOR METHOD Register()

Instantiates the generator and sets the next number to 1.

ASK METHOD ReturnNum() : INTEGER

Returns the next available integer.

TimeManager = OBJECT

MODULE TIMEMANAGER

IMPORTED MODULES:

None

DESCRIPTION

The time manager provides a query point for landscape objects to access the information that associates model simulation time with Simtime() time steps. The later is an IMPORT/DOME METHOD which returns an integer representing simulation time. The CONSTRUCTOR METHOD provides information to allow conversion of this time to days, months, and years.

METHODS

CONSTRUCTOR METHOD TimeManager(IN StepsPerYear, StartDay, StartMonth, StartYear: INTEGER)

This object works with 360 day years. The self-defined input values allow for the computation of the day, month, and/or year at any given time.

ASK METHOD ReturnSimStepsPerYear() : INTEGER

Returns the number of SIMTIME() simulations steps associated with a full year as provided through the CONSTRUCTOR method.

ASK METHOD ReturnSimStepsPerMonth() : INTEGER

Returns the number of SIMTIME() simulations steps associated with a 30-day month.

ASK METHOD ReturnSimStepsPerDay() : INTEGER

Returns the number of SIMTIME() simulations steps associated with a day.

ASK METHOD ReturnDay() : INTEGER

Uses SIMTIME() and returns the current day in the year (1-360).

ASK METHOD ReturnYear() : INTEGER

Uses SIMTIME() and returns the current year.

B.2 Animal SuperClass Object

The objects listed here provide the fundamental building blocks for constructing animal objects. These provide the functionality to support eating, health, prey activity, general facts, and entity knowledge. It is expected that multiple versions of each of these will develop over time in support of an increasingly broad spectrum of animals. The knowledge object is of particular interest as it provides the animal with its view of surrounding animals. This object addresses one of the key design goals: that it be possible to add new entities and entity types to an ecosystem simulation without reworking each existing entity to recognize the new entity type.

AnimalInfo = OBJECT
IMPORTED MODULES

ANIMALINFO
 REGION
 IMEMANAGER
 RASTERMANAGER
 LINKEDLIST
 REGISTER

DESCRIPTION

Common object known to all animals and auxiliary animal support objects.
 It provides a location for sharing information about and between animals.

METHODS

CONSTRUCTOR METHOD(IN Register: Register ; IN theRegion: REGION;
 IN Time: TimeManager; IN atE, atN: REAL; IN IsMale: BOOLEAN ; IN
 AgeIn: INTEGER ; IN LengthIn : REAL ; IN MassIn : REAL ; IN RangeIn:
 REAL ; IN StressIn : INTEGER ; IN SatisIn: INTEGER ; IN DTIn:
 INTEGER)

Internal variables are set based on the supplied initializing information.

ASK METHOD ReturnLocation(OUT e, n: REAL)
ASK METHOD ReturnPreyLoc(OUT e, n: REAL)
ASK METHOD ReturnNVel(): REAL
ASK METHOD ReturnEVel(): REAL
ASK METHOD ReturnName() : REAL
ASK METHOD ReturnColor(): REAL
ASK METHOD ReturnID(): INTEGER
ASK METHOD ReturnIsMale() :BOOLEAN
EXPORT ASK METHOD ReturnAge(): REAL
EXPORT ASK METHOD ReturnLength(): REAL
EXPORT ASK METHOD ReturnMass(): REAL
EXPORT ASK METHOD ReturnRange(): REAL
EXPORT ASK METHOD ReturnCarniv(): INTEGER
EXPORT ASK METHOD ReturnDT(): INTEGER
EXPORT ASK METHOD ReturnStress(): INTEGER
EXPORT ASK METHOD ReturnSatis(): INTEGER
EXPORT ASK METHOD ReturndStress(): INTEGER
EXPORT ASK METHOD ReturndSatis(): INTEGER
EXPORT ASK METHOD ReturnSelf(): TAGGED
EXPORT ASK METHOD ReturnActivity(): STRING
EXPORT ASK METHOD ReturnNeighbor(): LIST

```

EXPORT ASK METHOD ReturnQuery(): LIST
EXPORT ASK METHOD ReturnDead(): BOOLEAN
EXPORT ASK METHOD ReturnFertile(): INTEGER
EXPORT ASK METHOD SetLocation(IN atE, atN: REAL)
EXPORT ASK METHOD SetPreyLoc(IN atE, atN: REAL)
EXPORT ASK METHOD SetNVel (IN x:REAL )
EXPORT ASK METHOD SetEVel (IN x:REAL )
EXPORT ASK METHOD SetName(IN x:STRING )
EXPORT ASK METHOD SetColor (IN x:STRING )
EXPORT ASK METHOD SetID(IN x:INTEGER)
EXPORT ASK METHOD SetIsMale (IN x:BOOLEAN)
EXPORT ASK METHOD SetAge (IN x:REAL )
EXPORT ASK METHOD SetLength(IN x:REAL )
EXPORT ASK METHOD SetMass(IN x:REAL )
EXPORT ASK METHOD SetRange(IN x:REAL )
EXPORT ASK METHOD SetCarniv (IN x:INTEGER)
EXPORT ASK METHOD SetDT (IN x:INTEGER)
EXPORT ASK METHOD SetActivity(IN x:STRING )
EXPORT ASK METHOD SetDead (IN x:BOOLEAN)
EXPORT ASK METHOD SetStress(IN x:INTEGER)
EXPORT ASK METHOD SetSatis(IN x:INTEGER)
EXPORT ASK METHOD SetFertile (IN x:INTEGER)
EXPORT ASK METHOD UpdateNeighbors()
EXPORT ASK METHOD ResetQuery()
    Resets the query parameters (set by the SetQuery methods) to a state that
    is intended completely unrestricted.
EXPORT ASK METHOD SetQueryDeath(IN death: BOOLEAN)
EXPORT ASK METHOD SetQuerySex(IN doMales: BOOLEAN)
EXPORT ASK METHOD SetQueryLength(IN lo, hi: REAL)
EXPORT ASK METHOD SetQueryAge(IN lo, hi: INTEGER)
EXPORT ASK METHOD SetQueryMass(IN lo, hi: REAL)
EXPORT ASK METHOD SetQueryCarniv(IN lo, hi: INTEGER)
EXPORT ASK METHOD RunQuery() : INTEGER
    Searches through the list of nearby entities and creates a new list of those
    that meet the restrictions established by the SetQuery methods above.
EXPORT ASK METHOD BeAttackedBy(IN animal: AnimalInfo) : BOOLEAN
    Returns TRUE, to indicate that the associated animal has been killed in
    the attack.
Set up to be overridden by an animal SUBCLASS method that
allows the individual animal to respond to the attack in its own way.

```

B.3 Weather Support Objects

Precip = OBJECT(RandNormGraph)

MODULE PRECIP

IMPORTED MODULES:

RAND_NORM_GRAPH

DESCRIPTION

Provides the means to generate stochastic precipitation amounts for any time of the year.

METHODS

CONSTRUCTOR METHOD Precip(IN Days, Precip, PrecipDev: LIST)

Days, Precip, and PrecipDev are Import/Dome lists containing equal numbers of list items of type REAL. Days is considered to be the

independent variable; Precip the dependent value perturbed in a gaussian random form with respect to the PrecipDev value. "Days" is monotonically increasing and generally covers the range 0-360 as there are 360 days in the simulated year.

ASK METHOD GetPrecip(IN DayofYear: INTEGER) : REAL

Returns a precipitation value given the DayOfYear independent variable.

Temperature = OBJECT(RandNormGraph)

MODULE TEMPERATURE

IMPORTED MODULES:

RAND_NORM_GRAPH

DESCRIPTION

Provides the means to generate stochastic temperature values for any time of the year.

METHODS

CONSTRUCTOR METHOD Temperature(IN Days, Temp, TempDev: LIST)

Days, Temp, and TempDev are Import/Dome lists containing equal numbers of list items of type REAL. Days is considered to be the independent variable; Temp the dependent value perturbed in a gaussian random form with respect to the TempDev value. "Days" is monotonically increasing and generally covers the range 0-360.

ASK METHOD UpdateTemp(IN DayofYear: INTEGER)

Computes the base temperature for the simulation area for the given day. This can be adjusted to a local temperature with ReturnAdjustedTemp.

ASK METHOD ReturnTemp(): REAL

Returns the last base temperature computed with the UpdateTemp method.

ASK METHOD ReturnAdjustedTemp(IN DayofYear: INTEGER; IN latitude, tau, LapseRate, elevation, slope, aspect, BaseElevation: REAL) : REAL

Returns a local temperature adjusted for solar insolation differences caused by the inputs. Latitude, slope and aspect are in radians. Elevation and BaseElevation are in feet. "tau" is the rate of temperature change with solar azimuth angle. "LapseRate" is the increase in temperature (Fahrenheit) with every 100 ft of descent.

B.4 Sample Main Simulation Objects

The following objects provide an example of how the objects presented above can be used to assemble a working dynamic simulation environment.

Main = OBJECT

MODULE MAIN

IMPORTED MODULES:

MAPMANAGER

REGION

REGISTER

TIMEMANAGER

MAPVIEW

ANIMALVIEW

ANIMAL (Each animal module that identifies animals that shall be instantiated at the beginning of the simulation)

DESCRIPTION

The main object, as expected, establishes the initial state of the system and then begins running the objects. All initial user oriented viewers and controllers are also initialized.

METHODS

CONSTRUCTOR METHOD Main()

This is the only method and provides all of the instructions required to start the simulation.

MapView = OBJECT

MODULE MAPVIEW

IMPORTED MODULES:

XGRAPH
 ANIMALINFO
 RASTERMANAGER
 REGION

DESCRIPTION

Provides a viewport of the spatial layout of the landscape. An xpm graphic file derived from a GIS database provides a backdrop. Small icons move around this backdrop to provide a visualization of the location of the entities on that landscape.

METHODS

CONSTRUCTOR METHOD MapView(IN TM: TimeManager; IN RM: RasterManager; INRegion : REGION; IN daysteps: INTEGER; IN scene : STRING)

Initializes a graphic viewport through which the positional location of entities on the landscape appear. The single TELL METHOD provides the loop that keeps the visualization running during the course of the simulation.

ASK METHOD IdentifyState() : BOOLEAN

Returns whether or not the instantiation step completed properly.

TELL METHOD Run()

A self-calling method that renders an image of the current state of the system.

MapManager = OBJECT(GRASS, Temperature, Precip)

MODULE MAPMANAGER**IMPORTED MODULES:**

GRASS
 GRASS_INTERFACE
 TEMPERATURE
 PRECIP

DESCRIPTION

The raster manager s intended to be the single entry point into the raster landscape. This object shall have two versions. One version provides a run-time interface to static GIS (GRASS) data; another provides run-time interface capabilities to an external dynamic landscape simulation. Only one MapManager object is instantiated for all of the entities that might exist on the landscape.

METHODS

CONSTRUCTOR METHOD MapManager(OUT N, S, E, W, NS, EW:
REAL)

Connections to all GIS (or dynamic maps) are established. Regional weather data is prepared for the simulation.

OUT ASK METHOD ReturnCellData(IN DayOfYear: INTEGER; IN E, N: REAL;
temp, precip, slope, aspect, elev, grass, forb, shrub: REAL): REAL

The arguments to this method will be different for each simulation model developed. Essentially, all information associated with a current point in the simulation space is returned. The DayOfYear is used to compute the temperature and precipitation.

ASK METHOD ReturnCellData(IN mapname: STRING; IN DayOfYear:
INTEGER; IN E, N: REAL): REAL

Returns the value of the given map at the given location.

ASK METHOD CheckIfOnMap(IN E, N: REAL): BOOLEAN

Returns TRUE or FALSE depending on whether the given location falls within an active cell.

Appendix C: Associated Mojave Desert Efforts

The demonstration model development effort associated with this research relies heavily on the accomplishments of a number of different projects and individuals. They include the development of a comprehensive geographical information data set for Fort Irwin, the combination of this mapset with field measurements using neural networks to yield density maps for tortoises and vegetation, and sample dynamic landscape simulation models developed for predicting Desert Tortoise habitat.

C.1 GIS Maps

Raster- and vector-based GIS maps have been developed for Fort Irwin over the years using the GRASS geographical information system. The author's experience dates back to the development of early raster-based GIS software that he used to develop perhaps the first GIS database of Fort Irwin, California. Since then much more detailed databases that incorporated satellite imagery, vector maps of roads, ownerships, and boundaries, and many other themes were developed for Fort Irwin. Existing map layers include Landsat and SPOT satellite imagery, vegetation, geology, digital elevation models (DEM), and DEM-derived data such as watershed delineation, slope, and aspect. Vector maps include roads of various different types, ownership boundaries, and edges of training areas. Much of the digital data development was accomplished between 1978 starting with the author's first digital database of the area (Westervelt 1978) and the present at the U.S. Army Construction Engineering Research Laboratories (USACERL). Currently Fort Irwin maintains and develops its own database through on-location efforts, contractors, and government laboratories. A resource that needs to be tapped in the future is a legacy of remote sensing data using military satellites dating back perhaps 50 years and archived at the U.S. Army Corps of Engineers Topographic Engineering Center (TEC). GIS maps that are used in both the current effort and the concurrently developed Desert Tortoise population model effort (Westervelt et al. in press) initialize the raster data using digital elevation models, slope, aspect, tortoise density, vegetation density, soil properties, and geology maps stored in GRASS raster formats.

C.2 Transect Data and Neural Networks

In the development of spatially explicit tortoise population models for the classes taught with Professor Hannon, it was determined that initial tortoise densities and initial vegetation densities were desired for the landscape. The only data available consisted of density approximations derived from transect measurements. Dr. Anthony Krzysik of USACERL provided tortoise densities at approximately 400 transects walked in 1989 (Krzysik and Woodman 1991) scattered across Fort Irwin. Similarly, an Army program called the Land Condition Trend Analysis (LCTA) provided vegetation densities for transects scattered randomly across Fort Irwin. The simulation models required density approximations in a full-coverage map form while the collected data provided data for only sets of transects.

To convert the transect data to full-coverage maps, a back-propagation neural network was used to correlate a selected set of GIS map (and imagery) layers to the ground-based transect observations. Details of the approach used for each set of transects are discussed in Wu and Westervelt 1994; Westervelt, Krzysik, and Seel 1995; and Westervelt 1995.

C.3 Existing Raster Simulation Models

The demonstration models were built upon a pure SME population model designed and developed through University of Illinois Geography 495 classes offered during the Spring semesters of 1994 and 1995 (Westervelt et al. in press; Westervelt et al. 1995). The questions that were being asked of the modeling effort involved predicting short and long-term impacts of military training on populations of Desert Tortoise at Fort Irwin, California. A fixed time step of a week was chosen; this corresponded to a cellular spatial resolution of 1 kilometer (the farthest practical distance that a tortoise might traverse in a week). Students participating in the class provided a multidisciplinary team that covered biology, ecology, geography, regional planning, landscape architecture, civil engineering, and even pre-law and philosophy. In each class students divided into a number of teams that were assigned to different components of the modeling effort. Components included tortoise life stages and accompanying growth and death rates, tortoise migration, hydrology, climate and weather, vegetation, and human impacts. These were developed using the Stella graphical programming language. This simulation model was used as the basis for the demonstration of the I-STEMS software concepts and designs as discussed in section 5.1.2.

USACERL DISTRIBUTION

Chief of Engineers

ATTN: CEHEC-IM-LH (2)

ATTN: CEHEC-IM-LP (2)

ATTN: CECC-R

ATTN: CERD-L

ATTN: CERD-M (2)

Defense Tech Info Center 22304

ATTN: DTIC-O (2)

10

4/98