An Environment for Developing Individual-Based Ecological Simulations

by
Heather May

A THESIS

Presented to the Department of
Computer and Information Science
at the University of Oregon
in partial fulfillment of the requirements
for the Departmental Honors Program

June 2001

Acknowledgements

I would like to thank my advisor, Dr. John Conery for his patience and assistance throughout the duration of this project. In addition, I would like to thank Professor Daniel Udovic for permitting the use of EcoModeler to displaying the results of the project. Special thanks go to Jim Beard for his input on the model design, and to Jackie Carpentier for her extensive assistance with organizing and editing the paper.

Abstract

Computational modeling of ecological processes can provide a method for performing controlled experiments and evaluating the effects of disturbance on ecosystem health. However, the large number of biological and computational challenges associated with the development of realistic ecological models has prevented their use as tools for environmental management decisions. The creation of an ecosystem modeling environment could facilitate the widespread development of ecological models by helping researchers manage the large number of significant challenges in the formulation and implementation of this type of model. This paper describes an environment for developing forest simulation models (named FOREST), which implements a subset of the features desired in a complete ecological modeling environment. FOREST was able to simplify the process of developing forest simulation models by providing the main time advance algorithm and reusable components to define behavior of the objects in the model. If FOREST was extended into a full ecological modeling environment, modelers could focus on implementing the biological components and determining the appropriate level of detail for their simulations.

Table of Contents

Chapter 1: Introduction to Ecosystem Modeling

1.1 Introduction

Although human activities are altering ecological processes on a global scale, the consequences of these disturbances on ecosystem health are largely unknown. The effects of disturbance are difficult to predict since there are a large number of components (populations, resources, climate, topography, etc.) and interactions between components (predation, competition, mutualism, etc.). The consequences of specific interactions between components may be impossible to isolate in nature, due to the infeasibility of conducting controlled experiments. However, computational modeling allows one to perform controlled experiments on a model ecosystem, and to make long-term predictions of the effects of different types of disturbances (Deutschman, 2000; Levin et al., 1997). As a result, there has been a growing interest in using computational models to guide ecosystem management decisions. Computational models can potentially provide a way for environmental managers to evaluate alternative management strategies, by comparing the predicted short and long-term effects of each strategy on the ecosystem (Shea, 1998).

Despite the potential benefits of computational modeling, the natural complexity of an ecosystem presents a number of challenges to creating computer models designed to make predictions about the behavior of a particular ecosystem (Levin et al., 1997; AEL, STMP). This paper describes a system to facilitate the development and calibration of computational models for guiding environmental management decisions. The first section presents a brief history of the evolution of population and ecosystem models, and evaluates the different approaches to these models. After evaluating the advantages and disadvantages of each type of model, it became clear that the nature of the problem requires an individual-based, discrete-event approach. Chapter 2 addresses the biological and computational challenges associated with the building of individual-based, discrete-event models. The creation of an ecosystem modeling environment could facilitate the widespread development of ecological models by helping researchers manage the large number of significant challenges in the formulation and implementation of this type of model. Chapter 3 describes the desired features of an ecological modeling environment, and chapter 4 describes FOREST, a prototype which implements a subset of these features to create an environment for modeling forest dynamics. Chapter 4 also discusses some possible ways to extend FOREST to include features that will be necessary in a complete ecological modeling environment.

1.2 Approaches to Ecological Modeling

Biologists have historically recognized the value that can be gained through the use of mathematical and numerical models to investigate questions about the underlying causes of observed behavior of a biological system (Banks et al., 2001; Levin, 1997). Models in general can facilitate learning by revealing how interactions between components of the system affect its dynamics. This knowledge can then be used to predict the future state and behavior of the system based on its current state (Banks et al., 2001). A central challenge in ecology is understanding how large-scale community and ecosystem properties emerge from interactions between individuals (AEL, 2001). Ecologists have tried to gain insight into the mechanisms that determine ecosystem complexity through the development and analysis of mathematical models based on theories in population and community ecology. These models have traditionally been used to test the validity of a theory by comparing the results predicted by the model with empirical observations. Competing theories in ecology have been compared on the basis of their ability to accurately predict the future state of the ecosystem. As a result, mathematical models have contributed to the development of many central theories in ecology, including competition theory and the paradox of enrichment (DeAngelis and Rose, 1992).

Two general approaches exist to population-based ecosystem modeling, which differ in the amount of aggregation used to represent the organisms in the ecosystem. The first approach uses a high level of aggregation, where individual organisms in the system are not represented explicitly, but the states of the system are defined by the collective properties of the entire population (Matsimos et al., 2000). For example, in the classic Lotka-Volterra two-species model of interspecific competition, each population is represented by two variables corresponding to its size and competition coefficient (Begon et al., 1996). A population's competition coefficient is a relative measure of the ability of its individuals to compete with individuals of the other species. The behavior and stability of the system is completely determined by the initial size of the populations and the value of their competition coefficients (Begon et al., 1996). In an object-oriented model, the objects in the model are the populations, and changes in state of the model through time are applied to an entire population. In an object-oriented implementation of the Lotka-Volterra model, the size and competition coefficient would be stored as attributes of the populations. These models try to capture the essential dynamics that account for the observed behavior of the system, without identifying which details are directly responsible for that behavior.

In an individual-based object-oriented model, the objects are the individuals and the behavior of each individual organism is modeled explicitly. The changes in state to a population are computed by summing over the changes in state of the individuals. An individual-based model of interspecific competition would represent the populations as collections of individual objects. Each individual can be assigned its own coefficient of competition to account for variation in competitive ability between individuals of the same species. The behavior of the model results from the outcome of interactions between specific individuals, and therefore depends not only on the model's input but also on which individuals interact with one another. These models try to explain emergent behavior of a population as a result of the collective behaviors of its individuals (Levin et al., 1997). "The essence of the individual-based approach is the derivation of ecological systems from the properties of the individuals constituting these systems" (Lomnicki, 1992). Despite the differences in how the populations are represented, the final goal of both types of model is to summarize the state of the populations in the system at some future time. The distinction between the two types is in the method of derivation of the population state from the state of the individuals.

Since the first approach represents a population with a set of state variables, it has traditionally been referred to as the state-variable approach. Likewise, since the second approach explicitly represents each individual in the population, it has been termed the individual-based approach. Unfortunately, this terminology is misleading since all demographic models contain state variables and are based in some way on information about individuals. As a consequence, Metz and Diekmann proposed the use of the terms 'i-state distribution' and 'i-state configuration' to refer to state-variable and individual-based models, respectively (Caswell and John, 1992). An individual's i-state encapsulates the information necessary to predict the individual's behavior, and is used to specify the response of an individual to its environment. This terminology illustrates that the difference between the two types is in the way the state of the population is obtained from the state of the individuals, either by including the complete individual configuration or by reducing that configuration to a distribution function (Caswell and John, 1992). However, this terminology does not intuitively convey how the populations are represented in the model. Therefore, this paper refers to state-variable models as aggregated models and maintains the use of the term individual-based.

1.3 Aggregated Models

The traditional approach to population modeling has been to use aggregated models to describe the attributes of the population(s) the model is intended to represent (Matsimos et al., 2000). The model updates the state of each population at a fixed time interval by computing the changes in its attributes, which usually entails solving a set of first order differential equations. Individuals within a population are often grouped into classes according to some characteristic such as age, stage or size. Different parameters can be applied to the equations for each class to reflect variation in the behavior of individuals in different stages of the life cycle. Empirical observations about the organisms in the model are typically used to determine the interactions that take place during a time step for the individuals in each class. This approach assumes a well-mixed population, where every individual within a class experiences exactly the same environment and responds in the same way. These models have traditionally been deterministic, although some have incorporated environmental stochasticity by varying the conditions that individuals within a class experience over time.

The simplicity and ease of understanding associated with aggregated models has resulted in their widespread use within the field of theoretical population ecology. Since many of the details of the individuals in the population are abstracted away, the results from these models achieve a high level of generality. Well-developed methods exist for deriving population properties from the collective properties of each class of individuals through the use of differential equations, matrix algebra, or stability theory (Caswell and John, 1992). Even though natural populations rarely satisfy the assumptions of an aggregated model, these models can be used to predict the state of the system in the absence of complicating factors. Specifically, these models show what to expect in the absence of certain sources of variation including demographic stochasticity and a spatially or temporally heterogenous environment. These models form the basis for demographic theory, which says that in the absence of controlling factors (limited resources, predation, etc.) populations tend to grow exponentially (Caswell and John, 1992). The observation that natural populations do not grow exponentially for extended periods of time has prompted research into the factors that limit population growth.

Although aggregated models have made significant contributions to ecological theory, there are a number of disadvantages to using this approach for creating models designed to represent specific ecosystems. As previously mentioned, natural populations rarely satisfy the conditions assumed in aggregated models. Therefore, the predictions of the model cannot be applied to the real system unless it can be shown that the model provides a sufficient representation of the essential elements that determine the behavior of the system. If no significant loss of information results from the use of aggregation in the model, then its predictions are valid for the actual ecosystem.

However, many ecologists argue that the loss of information is significant because the assumptions inherent to these models violate two fundamental principles in biology (Caswell and John, 1992). The first principle states that individual organisms display physiological and behavioral differences due to their specific genetic and environmental influences. The nature of aggregated models prevents the inclusion of genetic variation between individuals within the same class, since the changes in state which occur in the model are applied to the entire class of individuals. The second principle is based on the locality of interactions, and states that an individual is affected more by organisms within a close proximity than others that are located farther away. Although it is possible to include the effects of local interactions and a heterogenous environment in an aggregated model, the incorporation of space considerably complicates both the formulation and analysis of these models (DeAngelis and Rose, 1992). Unless all of the individuals within a class experience the same environment, the model must maintain a list of the number of individuals in each class for each location. The maintenance of these lists not only complicates the solution, but also increases execution time due to the extra time needed to update the lists at each iteration of the model.

The complications that arose from trying to narrow the generality of aggregated models to represent specific populations has prompted the development of a different type of model that is better suited for this purpose. Since the individual-based approach simplifies the inclusion of biological detail about individuals, this approach can facilitate the development of more realistic ecological models. The next section describes the structure of individual-based ecological models and the two methods for implementing these models.

1.4 Individual-Based Models

In individual-based ecological models, the state of each individual organism is represented by a set of attributes (e.g. age, size, sex). Individual-based models are conceptually simple, and consist of the following components (Matsinos et al., 2000):

- a set of interacting individuals
- a set of rules that describe the implementation of local interactions between individuals and between an individual and its environment
- a physical landscape with resources

The rules that define state transitions can depend solely on the state of the individual or on the combination of the state of the individual and its environment. These rules are typically based on empirical data or proposed theories about the behavioral and physiological characteristics of the organisms represented in the model (Matsinos et al, 2000). Individual-based models can store more details about the individuals and the environment than aggregated models and do not impose the assumption of a well-mixed population. Therefore, the effects of variation between individuals and between specific locations in the environment can be incorporated into these models. The following sections describe two approaches to implementing individual-based ecological simulations, which differ in the way the model is updated through time.

1.4.1 Discrete-Time Implementation

The most common approach to implementing discrete-time individual-based models in ecology has been the development of cellular automata, a type of model originally proposed by John von Neumann (Ellison and Bedford, 1995). Cellular automata divide the environment into cells within a larger grid, and the changes in the state of a cell during a time interval are determined by the state of the cell, the state of neighboring cells, and a set of transition rules (Ellison and Bedford, 1995). The transition rules may use information about the characteristics of the organisms in the model to determine the next state of the each cell. Although the individuals in the model are the cells rather than organisms, the models are often made to be individual-based at the level of the organism by restricting each cell to hold at most one organism at any given point in time. A well-known cellular automata population model is J.H. Conway's "Game of Life", which uses a simple set of transition rules and each cell has only two possible states.

Cellular automata update all of the cells in the model at each time step, even though some cells may not change in state. This approach results in inefficient use of the processor since a significant amount of time must be spent checking each individual cell. Additionally, since changes in state to all individuals occur at fixed time intervals, all of the events that would occur at different times within an interval are processed at the end of the interval. In general, the use of a discrete-time model to represent a system with discrete changes in state introduces a source of error related to the length of the time between updates. The time step must be small enough so that differences in the timing of events within an interval does not significantly affect the validity of the results of the model. However, the time interval needed to achieve an acceptable level of accuracy may be too small to execute the model in a reasonable amount of time. Therefore, recent research has been directed towards using discrete-event simulations in order to eliminate the error introduced in continuous models (Bolte et al., 1993). The next section describes the discrete-event approach to ecological modeling, following a brief introduction to discrete-event simulations in general.

1.4.2 Discrete-Event Implementation

Discrete-event simulations are used to model systems in which the state variables do not change continuously, but the changes occur at discrete points in time (Banks et al., 2001). The changes in state are defined as events, which may be bound (i.e., the rate of occurrence is predictable) or conditional (i.e., the event occurs when a set of conditions are satisfied). The simulation consists of scheduling and processing events for each entity in the model. This approach requires managing and event list to store the future events for the all of the entities. The main control loop removes the first event from the event list, updates the simulation's current time to the timestamp of this event and updates the state of the entity to reflect changes caused by the event. The simulation iteratively processes events in the event list until the state of the system reaches some predefined terminating conditions.

Contrary to discrete-time simulations, the time interval between updates varies and each object in the model is updated at different times. Although the simulation does not have to update every object at a fixed time step, the management of an event list makes each individual step more complicated. Discrete-event simulations must manage an event list to store the future events for all of the objects in the model. However, the effects of this tradeoff are minimal since there are many methods for implementing sorted data structures with log(n) access times (n = number of objects in the list). In many cases, discrete-event simulations are computationally less intense than continuous individual-based models. The speed of execution increases since the simulation only updates the state of an individual when processing an event for this particular individual. The complexity involved with the implementation of a discrete-event simulation corresponds directly to the complexity of the system. Complexity tends to increase with the number of entities and interactions in a system, especially in systems which have a high occurrence of conditional events.

The discrete-event approach to ecological modeling requires defining events that correspond to the activities of the individuals in the model. Reproduction, competition, and predation are examples of events that are commonly included in these simulations. In addition to scheduling events for individuals, events may also be scheduled for the environment or specific locations within the environment to reflect changes in the availability or distribution of resources. The biggest advantage of discrete-event ecological models is that they can accurately model the discrete changes that take place to individuals in an ecosystem. Therefore, the effects of temporal variation in individual actions will be reflected in the model's results, and the significance of this variation may be evaluated. However, many events in an ecological simulation are conditional since they involve interactions between individuals. The simulation must provide a way to trigger the occurrence of these events, increasing the complexity of implementation.

1.5 Conclusion

There are two major advantages to using an individual-based approach instead of an aggregated approach to ecological modeling. The first advantage is that they allow for differences in the behavior of individuals in the model due to genetic and learned variation. Genetics can be incorporated in the model by adding a representation of genes to the set of attributes of the individuals. The second advantage is the relative ease in which additional details about the individuals and the environment can be added to the model. Each individual in the model is capable of being updated in a way that may or may not depend on temporally and spatially varying factors (Matsinos, 2000). Since the sources of natural variation can be modeled explicitly, individual-based models can potentially provide a more accurate representation of ecological processes. By examining the consequences of the addition and removal of elements in the model, ecologists can perform controlled experiments in order to investigate the mechanisms which account for observed behavior (Shea, 1995).

Since individuals in an ecosystem experience both discrete and continuous changes in state, a discrete-event approach can provide a more accurate representation than a discrete-time approach. However, there are several biological and computational challenges to applying a discrete-event, individual-based approach to ecosystem modeling. Biological challenges result from the large amount of information needed to formulate and parameterize individual-based models (Levin, 1997; Lomnicki, 1992). Implementation of these models requires knowledge about the characteristics of each individual and the factors affecting its behavior, which may not be possible to collect through observation. The computational difficulties stem from the high level of detail and numerous interactions included in the representation of individual-based models. These challenges include providing efficient methods for storage and retrieval of large quantities of data, multiple techniques for analysis and communication of results, and minimizing the execution time (Deutschman, 2000). The next chapter addresses the biological and computational challenges to applying an individual-based approach to ecosystem modeling.

Chapter 2: Challenges to Individual-based Modeling in Ecology

2.1 Overview

The computational challenges to developing individual-based ecological models can be attributed to the higher level of detail and large number of interactions between model components. The challenges to creating highly detailed, individual-based models result from the large data sets, complexity of analysis and communication of results, and the overall difficulty involved in the formulation, implementation and calibration of these models (STMP, 1995). In fact, the creation of large-scale ecological models has only recently become feasible, as a result of the vast improvements in computer hardware and software and the development of parallel and distributed computing systems (STMP, 1995). The following sections of this chapter describe the biological sources of complexity, and the resulting computational challenges.

2.2 Large Data Sets

The development of realistic individual-based ecosystem models has been hindered in the past by the large amounts of memory required to store the attributes for all of the individuals in the model (Gross, 1995; STMP, 1995). The amount of main memory needed for the simulation to run in a reasonable amount of time often exceeded the amount available for a standard desktop computer. As recent developments have increased the maximum amount of RAM in desktop computers and decreased its cost, many individual-based simulations can be executed on a standard machine. However, the memory requirements of large-scale models that include a high level of detail, many thousands of individuals and numerous interactions between individuals can still exceed the amount available on a standard computer today (Gross, 1995; STMP, 1995). Fortunately, improvements in hardware have reduced the cost of large mainframe computers and have correspondingly increased their availability to modelers. In addition to requiring large amounts of RAM, individual-based models often place heavy demands on the processor. Although the calculations used to determine the effects of interactions between individuals tend to be relatively simple, the number of interactions in a system can be quite large (Matsinos, 2000). The resulting computational requirements of these calculations can easily exceed the power of a single-processor machine.

2.3 Data Input and Initialization

Individual-based models are considerably more difficult to initialize than aggregated models, since the values for every attribute for each individual in the simulation must be determined. The two methods for initialization are to use data collected about each individual, or to use statistical distributions based on aggregated field data. To parameterize the model from actual observations requires an extensive amount of information to be collected about every individual represented in the simulation (Slothower, et al., 1996). However, it may not be possible to collect all of the data needed to fully parameterize the model due to biological difficulties in making the observations and the amount of time required for data collection. Some attributes may have to be estimated using randomly generated variates from statistical distributions if collecting the necessary information about each individual is not feasible. For example, in simulations of a specific wildlife reserve, the exact location of every individual at the beginning of the simulation may not be known from field data. The initialization routine must provide a method to disperse individuals throughout the model environment in patterns that match their natural dispersal patterns.

2.4 Analysis and Communication of Results

The large number of interacting factors makes the results from individual-based models more difficult to analyze and interpret than results from state variable models (Caswell et al, 1992; DeAngelis and Rose, 1992). Also, since most individual-based models include some stochasticity, the simulation needs to be run multiple times with the same parameters and initial conditions. Statistical analysis becomes necessary in order to assess the probabilities of various outcomes resulting from each parameter set and to determine the robustness of the model in terms of its sensitivity to initial conditions. Second, due to the large number of variables and the possible estimation of parameters, there is no way to test the validity of a highly detailed ecological model (Swartzman and Kaluzny, 1987). Instead, these models must be evaluated in terms of their level of corroboration, which is a measure of how well a model meets its objectives. The level of corroboration is determined through the comparison of model output with field data over the range of conditions for which the model is to be used (Swartzman and Kaluzny, 1987).

Due to the large data sets and number of simulations required, effective communication of the results often requires graphical displays. Graphical visualization of the data can facilitate understanding of complex relationships, but only if the data are presented in a such a way that patterns are easily recognizable to the human eye. Recently, there has been an increased awawreness of the crucial role that visualization plays in the analysis and interpretation of results from ecological simulations (Deutschman, 2000).

2.5 Model Formulation and Calibration

Resulting from the inherent complexity of ecological systems, an important step in building ecosystem models is determining which details to include. As a general guideline, a model should include only those aspects of the system that affect the problem under investigation, but should contain enough detail to permit valid conclusions to be drawn about the real system (Caswell et al, 1992). However, no complete methodology exists for determining which details in a complex system control particular aspects of the larger system (AEL, 2001). As stated by Levin et al.,

> The problem becomes one of the central problems in science: determining what is signal and what is noise by understanding what detail at the level of the individual units is essential to understanding more macroscopic regularities (1997).

Levin et al. warn against creating models which include more detail than can be measured or parameterized. The output from such models may appear realistic, but does not represent any real system. Many modelers suggest developing simple individual-based models with as few parameters and assumptions as possible (DeAngelis and Rose, 1992). Additional details can be added and removed in an attempt to identify which local interactions affect the broader scale patterns and which are noise (Levin et al., 1997). This approach requires extensive simulations that vary the parameters and level of detail in the model. However, this approach is necessary to improve the realism of the representation of the system and the corresponding accuracy of the model's predictions.

2.6 Overall Complexity

The absence of the widespread development of individual-based ecosystem models as tools for environmental management decisions can be attributed to the computational complexities associated with building and analyzing these models and to the lack of communication and collaboration between modelers (STMP, 1995). Development has been limited by the ability of any single team of researchers to deal with the conceptual complexities involved in the formulation, implementation, calibration and debugging of ecological models (STMP, 1995). Some models are so complicated that they are comprehensible only to the developers, making it virtually impossible to communicate the structure of the model to others. The inability to explain the model structure has prevented the use of these models as decision making tools for ecosystem management, since policy makers (and the public) are unlikely to trust a model they do not understand (STMP, 1995). Furthermore, these models have typically taken teams of experts 2-5 years to develop, which is too long to wait to make a management decision. If the goal is endangered species protection, creating a model to test management strategies is impractical since the population may be irreparably harmed in the time it takes to develop the model.

Additionally, the lack of communication and collaboration between modelers has resulted in different groups of researchers encountering similar challenges in the design and implementation of ecological models. The similarities between ecosystems and hierarchical nature of ecosystems suggest the possibility of creating models with reusable and extendable components, to reduce the amount of redundancy in model creation. One way to facilitate collaboration between modelers and to reduce the complexity associated with building ecosystem models would be to provide an integrated development environment (IDE) designed specifically for developing and evaluating individual-based ecological simulations. The next chapter describes the essential features of such an environment.

Chapter 3: Discrete-Event Ecological Modeling Environment

Although there are a considerable number of general-purpose simulation modeling environments (SME) currently in existence, the most successful SME's have been designed for relatively narrow domains (Banks, et al., 2001). In addition to standard discrete-event SME features, an ecological modeling environment should also provide tools to assist with the formulation, implementation and analysis of results of ecological simulations. The next two sections describe the components common to all object-oriented, discrete-event simulation environments and the additional features specific to ecological modeling.

3.1 Components of a Discrete-Event Simulation Environment

The following list describes the standard features of discrete-event simulation modeling environments (Banks et al., 2001).

- **Clock:** The clock is simply a variable to keep track of simulation time.
- **Event:** Events are used to update the state of the model entities. The event object typically contains the time of occurrence, type of event, and the entities involved.
- **Future Event List (FEL):** The event list is a data structure (sorted by event time) used to hold all of the future events until their time of execution.
- **Random Number Generators:** These generators provide random variates from statistical distributions.
- **Initialization Subroutine:** The state of the system at the beginning of the simulation is created in the initialization subroutine. This routine includes creating all of the objects in the simulation and setting the initial values for all attributes of the objects.
- **Main Control Thread:** The main program advances the simulation through time by processing events in order from the future event list.
- **Methods for Analysis and Display of Results:** These methods gather statistics from cumulative results and print a report at the end of the simulation. Most of the newer SME's also provide graphs and other tools for visualization of the results.

The amount of programming required by the user varies considerably between environments, depending on the level of generality of the environment (Banks et al, 2001). Since the SME proposed in this paper would be targeted specifically at simplifying the process of building ecological models, it would have a relatively low level of generality for a simulation environment. However, the environment needs to have a high level of generality within this specific purpose, in order to accommodate many different types of ecosystems and questions of investigation.

3.2 Additional Features for Developing Ecological Simulations

In addition to the components of a discrete-event SME, an ecological modeling environment should also contain the features described in the following sections.

3.2.1 Simulation Time

In most SME's, time is stored as an integer or floating point number and is computed relative to the initial time of the simulation (Banks et al., 2001). However, some events in an ecological simulation should be restricted to occur during certain time periods. For instance, many organisms reproduce only once a year in the spring and may mate only during a specific time of day. It may be difficult or impossible to accommodate this level of detail in a simulation which only stores time as a number relative to the starting time. Some models have broken up the year into seasons and specify which events occur during each season (Ellison and Bedford, 1995). One disadvantage to this approach is that it requires that the event scheduler is aware of the current season, in order to know what events to schedule. Another disadvantage is that it does not allow for variation in the time ranges of events for different species in the simulation. This approach is acceptable in situations where all of the species represented in the model have the same temporal patterns.

However, even though many species have similar temporal patterns, slight variation in these patterns may have significant effects on community dynamics. For instance, although most species mate in the spring, the actual time of reproduction varies considerably between species. Since these differences are not included in the model, the effects of variation are excluded from the model's results. An ecological SME should provide methods to simplify the addition of time range restrictions to certain events for individuals in the model. The user should be able to specify the exact time range for all events for each species, so that the results may include the effects of temporal variation between species.

3.2.2 Flexibility in Event Scheduling

The SME should allow maximum flexibility in the scheduling of events during the simulation. Rather than restricting events to be scheduled in only one way, events should be allowed to be scheduled using different methods. For example, movement of an organism in the model may be triggered by the need to eat, mate, or escape predation. However, some models may also include random movement, where the time of movement may follow a statistical distribution. The simulation should allow the same event to be scheduled both ways; in response to the occurrence of some other event or according to a statistically determined rate of occurrence.

3.2.3 Data Input Subroutines

Since ecological models can be parameterized by data collected about individuals or by applying statistical analysis to data collected about the populations, the SME should provide methods for both type of input. It is rarely possible to get all values for all of the attributes for every individual from field data, so the user should be able to choose which method to use for each attribute in the simulation. Large amounts of data may need to be entered to parameterize the model, so techniques need to be available to simplify the process of initialization (Slothower et al., 1996).

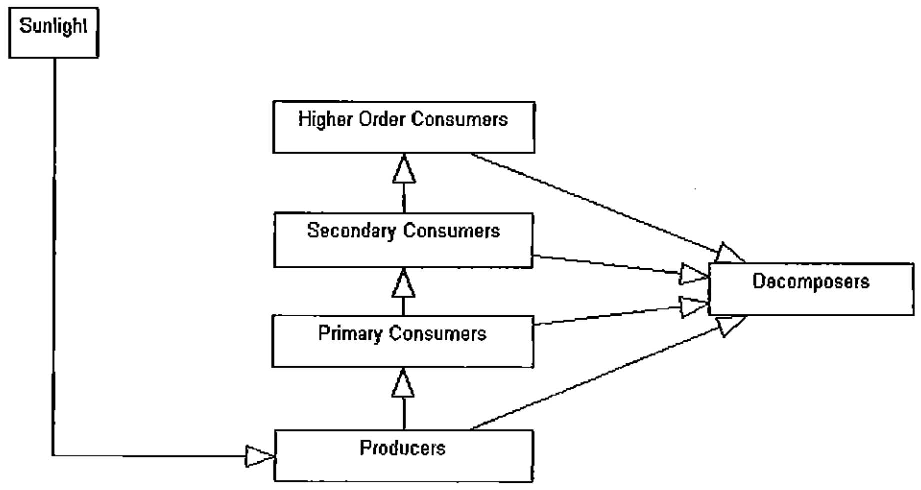
3.2.4 Extended Visualization Package

Visualization of the results of an ecological simulation must usually be implemented at various scales to allow the user to extrapolate all of the desired information. This package should include methods for displaying static and dynamic information about the state of the model environment throughout the duration of the simulation. In addition to graphs that summarize the results from multiple simulations, a graphical display may also be used to illustrate the results of one particular simulation run. The large number of individuals present in a simulation may prevent the explicit visual representation of each individual within the entire environment. The environment should allow the user to view the distributions for various subgroups of the population, such as a particular species or subgroups within a species including males, females, or individuals within a certain age group. In a view of a subset of locations, the distribution of individuals could be shown explicitly, which may reveal patterns not recognizable in the condensed view of relative densities. It may not be possible or desirable to show all attributes of the individuals in the same view, but instead implement multiple views to show different subsets of attributes.

3.2.5 Reusable and Extendable Model Components

Rather than creating all models from scratch, the components and behaviors implemented in one model should be resuable so they can be extended in other models. The simulation environment could provide abstract base classes for the components of an ecosystem, since ecosystems have a natural hierarchical structure and commonalities exist between groups of organisms in different ecosystems. The species present in an ecosystem are often broken down into groups according to functional roles of producer, primary consumer (i.e., herbivores), secondary and higher order consumer (i.e., carnivores), and decomposers (Begon et al., 1996). Figure 1 shows the flow of energy through the functional groups in an ecosystem.

Figure 1: Energy Flow in an Ecosystem



Since organisms within a group have similar attributes and behavior, these organisms would extend the same base class in the model hierarchy. Programmers could extend these base classes to implement functionality specific to their models.

3.2.6 Flexible Implementation

The user should be allowed to decide the level of abstraction used in the implementation of each event in the simulation. The causes of some events may not need to be modeled explicitly, if the details about the event are inconsequential to the results. These events may be scheduled based on an empirically estimated rate of occurrence. The details necessary to include in a model depend not only on the organisms being represented, but also on the particular question(s) being investigated. For instance, a forest model designed to investigate the effects of variation in seed dispersal ranges of different species may not need to specify the causes of tree death. However, a model for investigating the effects of herbivory on tree health would want to identify the cause when an individual tree dies. Ideally, the simulation environment should allow the user to easily change which organisms and behaviors are included in the model and how each behavior is implemented. This would allow modelers to perform extensive sensitivity analysis to explore the underlying determinants of system behavior and to change which details are included in a model when research yields new information about the behavior of the system.

3.3 Summary

An environment targeted at the development of ecosystem models could reduce the time for model development and calibration, by providing the main time advance algorithm, an event scheduler and library routines for components common to all ecological simulations. This type of SME would allow users to focus on the biological challenges of simulation modeling, rather than on the computational challenges of implementing a simulation. The next chapter describes FOREST, an environment for developing and evaluating computational models to simulate forest dynamics. FOREST implements a subset of the features described above, and could be extended to include all of the components of a full ecological modeling environment.

Chapter 4: A Simulation Environment for Building Forest Models

FOREST

(Forest Object-oriented Reusable Ecosystem Simulation Template)

4.1 Description

FOREST is a discrete-event ecological simulation environment written in Java, designed to reduce the time and complexity involved in the development and calibration of individual-based forest simulation models. FOREST is based on the premise that a fundamental step in the process of building an ecological model is determining which details to include. Therefore, FOREST's class heirarchy was designed to allow the user to dynamically determine the entities and events in the model, as well as the implementation of each event. FOREST was also designed to accommodate users with different amounts of programming experience. In addition to the abstract class heirarchy, FOREST could promote reuse of model components by providing a library of concrete implementations of the entities and events commonly included in a forest simulation. Therefore, users with minimal programming experience could create models with the existing components, while experienced programmers could extend these components to create more detailed models. The library would be a collaborative effort, where programmers could add the components they implemented so that other modelers may use these components in their models.

Since FOREST's main goal is to assist the modeler in identifying the appropriate level of detail, the focus was on building the simulation engine and a class hierarchy which would support the development of reusable model components. Much less emphasis was placed on developing techniques for initialization from user input, analysis and visualization of the results, and communication between entities in the model. As a result, FOREST would need to be extended to include these features to provide a complete environment for developing ecological simulations. Section 4.2 describes the implementation of the components of the simulation engine and section 4.3 describes the class hierarchy and the implementation of of the organisms. An example model implemented in FOREST is described in section 4.4.

4.2 Discrete-Event Simulation Implementation

FOREST provides all of the features common to discrete-event simulation modeling environments, and extends some of these features to assist modelers in developing ecological models. The following sections describe how each feature was implemented in FOREST.

4.2.1 Simulation Time

Java's `GregorianCalendar` class keeps track of simulation time. This implementation provides a straightforward way to restrict the occurrence of events to certain time periods during the simulation. Temporal differences in behavior between species can easily be included in the model, by assigning a range of times when the event may occur.

4.2.2 Events

The `Event` class is used to store all of the data necessary to process an event. This data includes the object receiving the event, the time of occurrence, and the name of the event. In order to accommodate continuous changes in state, events for which the time of last occurrence needs to be known must implement the interface `TimeDependentEvent`. This interface is just used as a tag to tell the object receiving the event to record the time that this event last occurred, so no methods need to be implemented in `Event`. If the first occurrence of an event should be scheduled when the model is initialized, it must implement `InitialEvent`. Again, no methods need to be implemented in `Event` since this interface is used to signal the simulation environment to schedule this event when the model is initialized.

4.2.3 Event List

FOREST uses Java's `TreeMap` class to store the future events for the simulation. The event list is sorted by time of occurrence, and ensures log(n) time for insertion and deletion of events (n = number of events in the list). Choosing the next event to process can be done in constant time since the event at the head of the list always has the smallest timestamp.

4.2.4 Event Scheduler

A scheduler is provided that will determine when to schedule the next occurrence of an event based on the parameters passed to it, and then place the event in the event list at this time. If the scheduler is passed an integer, it will schedule the next occurrence of the event for the number of time units in the future specified by the value of the integer (in FOREST, time units correspond to days). The scheduler may also be passed a random number generator and a list of parameters, in which case it will get the next random variate from the given generator and schedule the event accordingly. Although some simulation environments provide automatic scheduling, FOREST lets the user schedule the events in order to achieve maximum flexibility in when and how events are scheduled.

4.2.5 Random Number Generators

FOREST provides methods for the generation of random variates from uniform, normal, binomial and exponential distributions. The parameters for the distribution are set at the time the generator is called for the next random variate. This implementation was chosen since the parameters do not need to be known when the generator is instantiated and the same generators can be used by multiple objects.

4.2.6 Initialization Subroutine and Main Control Thread

At the beginning of the simulation, the initialization subroutine randomly disperses the individuals throughout the model environment and initializes the event list by scheduling the first occurrence of each `InitialEvent` in the simulation. The main control thread then executes events from the event list in chronological order until there are no more events to process or the user terminates the simulation.

4.2.7 Visualization Methods

FOREST uses EcoModeler, a set of visualization tools for population models, to display the results of the simulation. EcoModeler provides methods for viewing aggregate population data (e.g. total size) and individual data (e.g. location) as well as static (e.g. age distribution) and dynamic data (e.g. population size vs. time). However, since EcoModeler was originally designed to display data from aggregate models, it does not provide all of the desired features for displaying the results from individual-based models.

4.3 Model Design

The following objects are provided in FOREST's base class heirarchy.

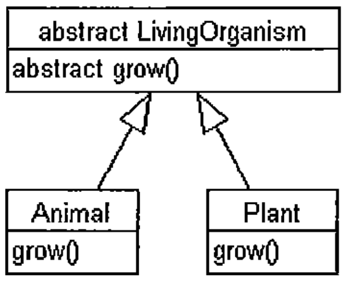
- **`LivingOrganism`** - This is the abstract base class for all living entities in the model. The attributes in this class are the individual's birthdate and species.
- **`Species`** - This class is just a data structure to contain all of the species-specific information for `LivingOrganisms`.
- **`Population`** - This class contains a collection of `LivingOrganisms`, and methods that return aggregate information about the individuals (e.g. total size, average age).
- **`Environment`** - The environment is implemented as a two-dimensional array of cells, where each cell can hold at most one `LivingOrganism` at a time.

In order to assist the modeler in identifying the appropriate level of detail, FOREST's model heirarchy was designed to allow the user to dynamically change the implementation of certain events and examine the effects of such changes. The general technique for increased flexibility of implementation and reuse of components is to separate the behavior of an object from the data used to characterize the object's internal state. This technique allows the user to add, remove, and change the implementation of certain behaviors of an object, without making changes to the object itself. The design is a modification of visitor patterns, a design technique outlined by Gamma et al. (1995). Section 4.3.1 describes the motivation for and limitations of visitor patterns, while section 4.3.2 discusses the implications for FOREST's variation on visitor patterns.

4.3.1 Visitor Patterns

In a traditional object-oriented heirarchy, adding new functionality to the objects requires adding a method to every class in the heirarchy. Figure 2 shows a simple example of a class heirarchy implemented using the traditional approach to object-oriented programming.

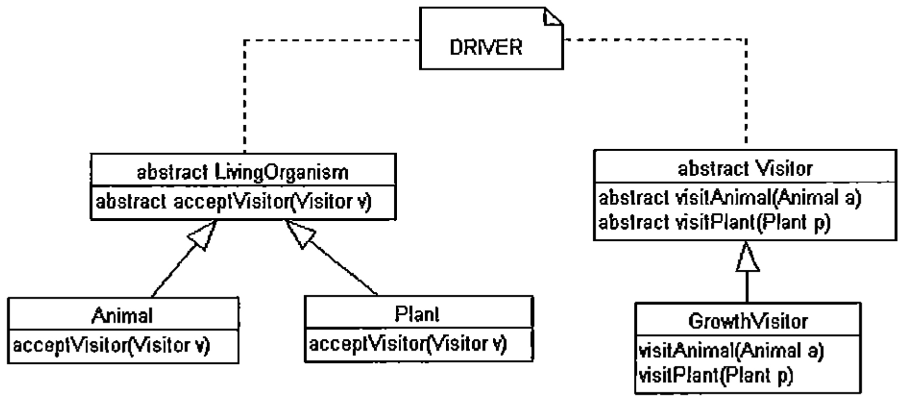
Figure 2: Traditional Object-oriented Heirarchy



The arrows in the figure signify that `Animal` and `Plant` extend `LivingOrganism`. In order to add reproduction as a behavior of a `LivingOrganism`, each class in the heirarchy must include the method `reproduce()`. This requires modifying and recompiling all the existing classes in the heirarchy, a time-consuming process in large systems.

Visitor patterns were created to allow additional functionality to be added to a system at a later date without the need to modify the existing components of the system (Gamma et al., 1995). This is accomplished by implementing additional methods outside of the original class hierarchy, in `Visitor` classes. The Visitor pattern requires implementing two class heirarchies: one for the elements in the system and one for the visitors that define operations on the elements. New operations can be added by creating a new subclass in the `Visitor` heirarchy. Visitor patterns use a technique known as double-dispatch, where the method that gets invoked depends on the kind of request and two types of receivers. "This is the key to the Visitor pattern: The operation that gets executed depends on both the type of Visitor and the type of Element it visits." (Gamma et al., 1995). Figure 3 shows how the heirarchy in Figure 2 would be structured using Visitor patterns.

Figure 3: Visitor Pattern Heirarchy



All classes in the hierarchy are prepared to accept a visitor object via an `accept(Visitor v)` method. Inside the accept method, the visitor's visit method for this object invoked and the object being visited passes itself as a parameter. Invoking this method gives the visitor a reference to the visited object and access to all of its public methods. Inside the visitor classes, there is a visit method for each object in the hierarchy. The example below illustrates how Visitor patterns could be applied to a simple ecosystem model.

Example 1: Visitor Patterns

```
abstract public class LivingOrganism {
   protected double height;
   public double getHeight() { return height; }
   public void setHeight(double h) { height = h;}
   abstract public void accept(Visitor v);
}

public class Plant extends LivingOrganism{
   // plant size is determined by height and stem diameter
   protected double stemDiameter;
   public void accept(Visitor v) {v.visitPlant(this);}
   public double getDiameter() {return stemDiameter;}
   public void setDiameter(double d) {stemDiameter=d;}
}

public class Animal extends LivingOrganism {
   public void accept(Visitor v) {v.visitAnimal(this);}
}
```

```
abstract public class Visitor {
   abstract public void visitPlant(Plant p);
   abstract public void visitAnimal(Animal a);
}

public class GrowthVisitor extends Visitor {

   // when a plant grows, its height and stem diameter increase
   public void visitPlant(Plant p) {
      p.setHeight(p.getHeight*plantGrowthRate*timestep);
      p.setDiameter(p.getDiameter*plantGrowthRate*timestep);
   }

   // when an animal grows, only its height increases
   public void visitAnimal(Animal a) {
      a.setHeight(a.getHeight*animalGrowthRate*timestep);
   }
}

public class Driver {
   public static void main (String [] args) {
      Animal a = new Animal();
      Plant p = new Plant();
      GrowthVisitor gv = new GrowthVisitor();

      // make the animal grow
      a.accept(gv);

      // make the plant grow
      p.accept(gv);
   }
}
```

In this implementation, adding the behavior 'reproduce' to the objects can be accomplished by simply creating a new `Visitor` subclass, and implementing the `visitAnimal()` and `visitPlant()` methods. The class heirarchy for the elements in the system does not need to be modified or recompiled in this approach.

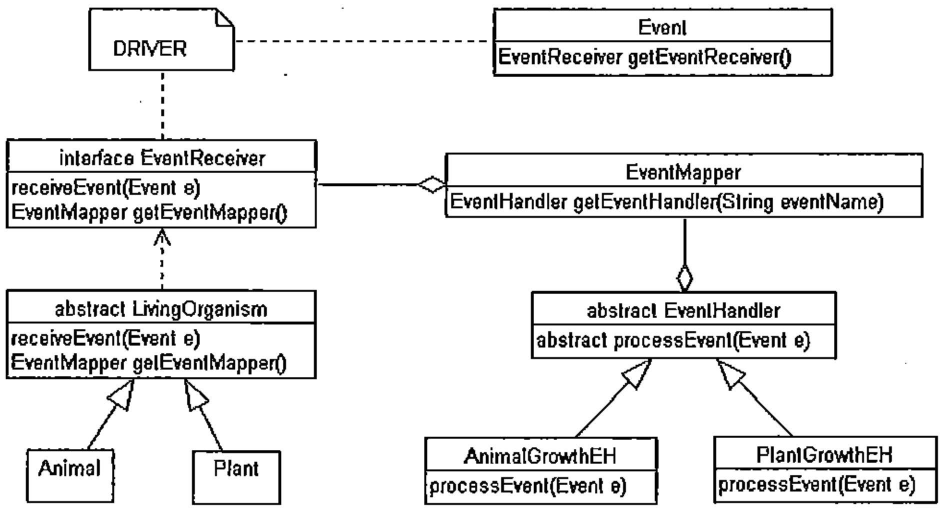
Although Visitor patterns simplify the addition of new functionality to existing classes, there are two drawbacks to using this approach. The first stems from the fact that some of the operations defined on object are implemented outside of the object. "As a result, the pattern often forces you to provide public operations that access an element's internal state, which may compromise encapsulation." (Gamma et al., 1995). Additionally, Visitor patterns actually complicate the addition of new classes into the hierarchy since each `Visitor` class must be modified to visit the new class. For example, if we wanted to add the class `Fungus` to the hierarchy, a `visitFungus` method would have to be added to every visitor class. Even though this example has only one concrete visitor class, there may be many visitors in an actual implementation. For this reason, Visitor patterns have been recommended only for systems whose class hierarchies are unlikely to change.

The appeal of the Visitor pattern results from the fact that the methods of an objects can be added or removed without modifying the model class heirarchy. The implementation of a method can also be easily redefined by removing the current visitor and creating a new visitor for this function. However, two consequences of using the Visitor pattern conflict with the goals of FOREST. The requirement of a static class heirarchy is too strict for an environment designed to facilitate the calibration of ecological models. The users should be able to decide which entities to include in their models, since this will depend on the specific goals of the model and the corresponding appropriate level of detail. Additionally, the implementation of a certain behavior for all objects in the heirarchy must be placed in a single `Visitor` class. It should be noted that the grouping of related behaviors is usually considered one of the advantages to using Visitor patterns. However, this implementation complicates the process of changing the behavior of just one particular entity (or a subset of the entities) in the heirarchy, which may be desired in the process of calibration. Therefore, FOREST uses a modified version of Visitor patterns, as described in the next section.

4.3.2 Event Handlers

The goal in designing FOREST was not only to simplify the process of adding functionality to a hierarchy, but also to allow modelers to add new classes. FOREST modifies Visitor patterns so that the modeler can specify which entities and behaviors are implemented in the model. Accomplishing this goal required a higher level of modularization than commonly used in object-oriented heirarchies. As a result, the entities in the model are used only to encapsulate data, rather than both methods and data. Instead of grouping the implementation of a behavior into one class, FOREST creates a separate class to implement this behavior for each object in the heirarchy. FOREST assigns an `EventHandler` for each behavior that an entity participates in during the simulation. Figure 4 shows the relationships between the entities and EventHandlers in FOREST, using standard (UML) notation.

Figure 4: Event Handler Heirarchy



The `Event`class is simply a data structure to hold all information needed to process the event. The `EventReceiver` is the object for which the event affects. In order to accommodate interactions between organisms, events can contain multiple objects as long as one of the objects is designated as the receiver. Since `EventReceiver` is an interface, any object in the model can receive events. Each receiver must contain an `EventMapper`, which is just a hashtable that maps events to their appropriate `EventHandlers`. In order to process an event, the receiver must be queried to get the appropriate handler for this particular event. The `processEvent(Event e)` method in the handler class is then invoked, and the actual processing of the event takes place in the `EventHandler` class.

In FOREST, each species has its own set of EventHandlers, which implement all of the behaviors for individuals of that species. This implementation allows for species-specific behavior to be incorporated into the model, although it does not require a different EventHandler for each species. The following example shows how the same ecosystem model in the Visitor pattern example would be implemented using EventHandlers.

Example 2: Event Handlers

```
abstract public class LivingOrganism implements EventReceiver{
   protected double height;
   protected Species species;
   protected EventMapper eventMapper;
   public LivingOrganism (Species species) {
      this.species = species;
      eventMapper = species.getEventMapper();
   }

   public double getHeight () { return height; }
   public void setHeight (double h) { height = h; }

   public void receiveEvent (Event e) {
      EventHandler eventHandler = eventMapper.getEventHandler(e);
      eventHandler.processEvent(e);
   }

   public void addEventHandler (String eventName, EventHandler
   eventHandler) {
      // assign this EventHandler to this event
      eventMapper.put(eventName, eventHandler);
   }
}

public class Plant extends LivingOrganism {
   // plant size is determined by height and stem diameter
   protected double stemDiameter;
   public double getDiameter () {return stemDiameter;}
   public void setDiameter (double d) {stemDiameter=d;}
}

public class Animal extends LivingOrganism{ }

abstract public class EventHandler {
   abstract public void processEvent (Event e);
}

public class PlantGrowthEH extends EventHandler {
   // when a plant grows, its height and stem diameter increase
   public void processEvent (Event e) {
      Plant p = (Plant)e.getEventReceiver();
      p.setHeight(p.getHeight*plantGrowthRate*timestep);
      p.setDiameter(p.getDiameter*plantGrowthRate*timestep);
   }
}

public class AnimalGrowthEH extends EventHandler{
   // when an animal grows, only its height increases
   public void processEvent (Event e) {
      Animal a = (Animal)e.getEventReceiver();
     a.setHeight(a.getHeight*animalGrowthRate*timestep);
   }
}
```

```
public class Driver {
   public static void main (String [] args) {
      Animal a = new Animal();
      Plant p = new Plant();
      Event e = new Event("growth");

      // create the associations between events and event
      // handlers
      a.addEventHandler("growth", new AnimalGrowthEH());
      p.addEventHandler("growth", new PlantGrowthEH());

      // make the animal grow
      a.receiveEvent(e);

      // make the plant grow
      p.receiveEvent(e);
   }
}
```

By associating a different `EventHandler` for each species, reusable modules can be developed for the behavior of each species. A new model could be created that uses components from more than one existing model. This design facilitates reuse more than the traditional use of inheritance through class hierarchies, since a new model could reuse or extend parts of an existing model without having to extend the class hierarchy. Another advantage of this design is that it allows the user to choose how events are implemented at runtime. This means that modelers may easily change the implementation of a model and investigate the effects of the changes on the dynamics of the system.

Although FOREST's use of `EventHandlers` alleviates the need for a static class hierarchy in the traditional application of visitor patterns, this implementation results in some new tradeoffs. This design results in a large number of `EventHandler` classes since a new class may have to be created for each (species, behavior) combination. The methods of an object are dispersed throughout the `EventHandler` heirarchy, instead of being encapsulated within the object. Additionally, invoking the appropriate event handler for an object is slightly more complicated since the association is between EventHandler *classes* and objects rather than Visitor *methods* and objects. This require an additional step in processing an event, since the events must be mapped to their appropriate handlers by the `EventReceiver`.

The flexibility in model development comes at the expense of increased complexity in the design and implementation of the base class heirarchies. However, the process of developing and calibrating new models is simplified once the `EventReceiver` and `EventHandler` heirarchies are implemented. The modeler just needs to implement the entities (in `EventReceiver` classes) and events (in `EventHandler` classes), and define the associations between the two types of objects. Once some entities and events have been implemented and added to the library, these components may be used in new models by creating the desired associations. To facilitate the development of realistic ecological simulations, the benefit of modular development outweighs the tradeoff of increased complexity of the base class heirarchy.

4.4 Simple Growth Model

To demonstrate the ability to create forest simulation models in FOREST, a simple model (`GrowthModel`) was developed to simulate tree growth. The model represents an ecosystem with three species of trees which differ in life history characteristics. The behaviors simulated in the model are growth, seed dispersal and death. Although this model was not parameterized from field data, this example shows how to use the environment provided by FOREST when the parameters are known or can be estimated. The model consists of entities and two types of events; state events and scheduling events. State events represent the behaviors in the model and are used update the state of the entities, and scheduling events are used to schedule the occurrence of the state events. The following sections describe how the entities and events were implemented in `GrowthModel`.

4.4.1 Entities

The only entities in `GrowthModel` are the trees, implemented in the class `Tree`.

- **Tree**

The `Tree` class stores the height and diameter of the tree, and provides methods for getting and setting these values. Since `Tree` extends `LivingOrganism`, it implements `EventReceiver` and inherits the `receiveEvent`, `setLastUpdate` and `getLastUpdate` methods. The interface for the `Tree` class is given below.

Attributes:
```
    double height;
    double diameter;
```
Methods:
```
    double getHeight();
    setHeight();
    double getDiameter();
    setDiameter();
```

- **Species**

The species-specific parameters needed to process the events are stored in the `Species` class. Each species contains attributes for the mean and maximum number of seeds produced during a reproductive event. The `Species` class also contains the mean and maximum values of the dispersal range of seeds, as well as the growth rate and death rate for individuals of the species.

Attributes:
```
    double meanOffspring;
    double maxOffspring;
    double meanDispersalDistance;
    double maxDispersalDistance;
    double growthRate;
    double deathRate;
```
Methods:
```
    double getMeanOffspring();
    double getMaxOffspring();
    double getMeanDispersalDistance();
    double getMaxDispersalDistance();
    double getGrowthRate();
    double getDeathRate();
```

4.4.2 State Events

Since the state events correspond to the behaviors in the model, the state events in this model are seed dispersal, growth and death. The following sections describe how each state event was implemented in `GrowthModel`.

- **Seed Dispersal**

When a dispersal event occurs, the number of seeds to disperse is determined by generating a random number from a binomial distribution, using the appropriate parameters for this species. For each seed, the dispersal distance is determined by generating a random number from another binomial distribution. A uniform random variate is then used to determine the direction of dispersal. The seed will survive only if this location is not currently occupied by another tree.

- **Growth**

GrowthModel provides two `EventHandlers` with different implementations of tree growth to show that the implementation of an event can be easily redefined. The first implementation calculates growth using a linear equation with a species-specific slope. The second implementation takes shading into account by decreasing growth for trees that have neighbors in adjacent locations. This version is implemented by querying the environment for the number of trees surrounding a certain tree, and reducing the tree's growth by 5% for each adjacent tree. The user may choose which implementation is applied in the simulation when the model is initialized.

Since `Growth` is a continuous event, it implements `TimeDependentEvent`. Since the size of all of the individuals in a population should be updated at the same time, `Growth` events are scheduled for the entire population. The `EventHandler` first iterates through the population and updates each individual's size, and then schedules the next `Growth` event for this population for the current time plus the time interval between updates. If the time of the next occurrence is outside the growth period for this species, the event is not added to the event list.

- **Death**

When a death event occurs, the `EventHandler` removes the tree from its current population and location.

4.4.3 Scheduling Events

Since FOREST does not provide a mechanism for triggering the occurrence of an event based on the state of the object, the modeler must implement the scheduling of the events in the model. `Growth` events should be scheduled at fixed time intervals throughout the growth period, seed dispersal events need to be scheduled once a year, and death should be scheduled once for each tree. Since `GrowthModel` schedules events based on statistical properties of the population, and the following events are scheduled for the entire population. All of these events implement `InitialEvent`, and are used to initialize the event list at the start of the simulation.

- **Schedule Seed Dispersal**

  `Dispersal` events are scheduled for each population at the beginning of that species' reproductive time of year. This event handler iterates through all of the reproductive individuals in the population and schedules a seed dispersal event for each one. The time of dispersal for each individual in the population is determined by generating a random variate from an exponential distribution. After scheduling the dispersal events for all of the individuals, the next occurrence of this event is scheduled for the beginning of the reproductive season of the next year.

- **Schedule Growth**

  Since Growth events are self-generating (i.e., the next occurrence of the event is scheduled during the processing of the event), this class only needs to schedule the occurrence of the first event of the growth season.

- **Schedule Death**

  The time of death for an individual is determined at birth and follows an exponential distribution with species-specific mean value.

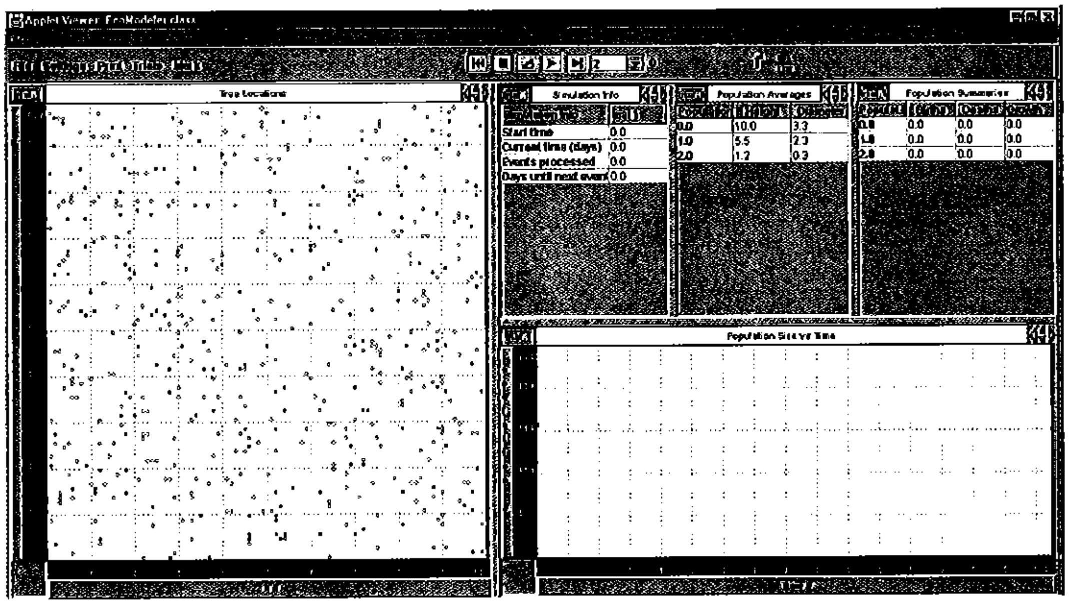
4.4.4 Initialization

The model is initialized from a data file, which creates the populations and sets the initial values for all of the attributes of the trees. The model uses the random dispersal method provided by FOREST to disperse the individuals in all of the populations throughout the model environment.

4.4.5 Results

The output of the simulation consists of information about the populations (e.g. total size vs. time, average height) and about the simulation itself (e.g., current time, number of events processed). The position of each individual tree within the environment is also displayed on a two-dimensional grid. Figure 5 shows the initial state of the model.

Figure 5: Model Initial State



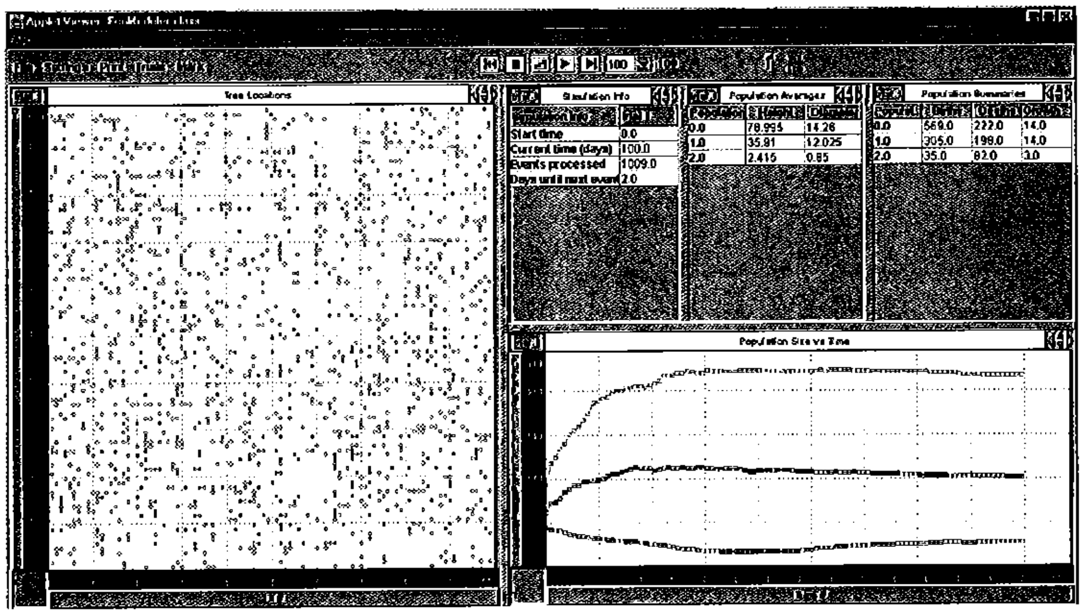
[illegible]	[illegible]
Start time	0.0
Current time (days)	0.0
Events processed	0.0
Days until next event	0.0

[illegible]	[illegible]	[illegible]
0.0	10.0	3.3
1.0	5.5	2.3
2.0	1.2	0.3

[illegible]	[illegible]	[illegible]	[illegible]
0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0
2.0	0.0	0.0	0.0

At the beginning of the simulation, no events have been processed so most of the values shown on the screen are zero. The next two figures show the changes in the results after the simulation has run for 100 days. Figure 6 displays the population vs. time view as a graph to and Figure 7 displays this view as a table so that exact values may be determined.

Figure 6: Output after 100 days (graph display)

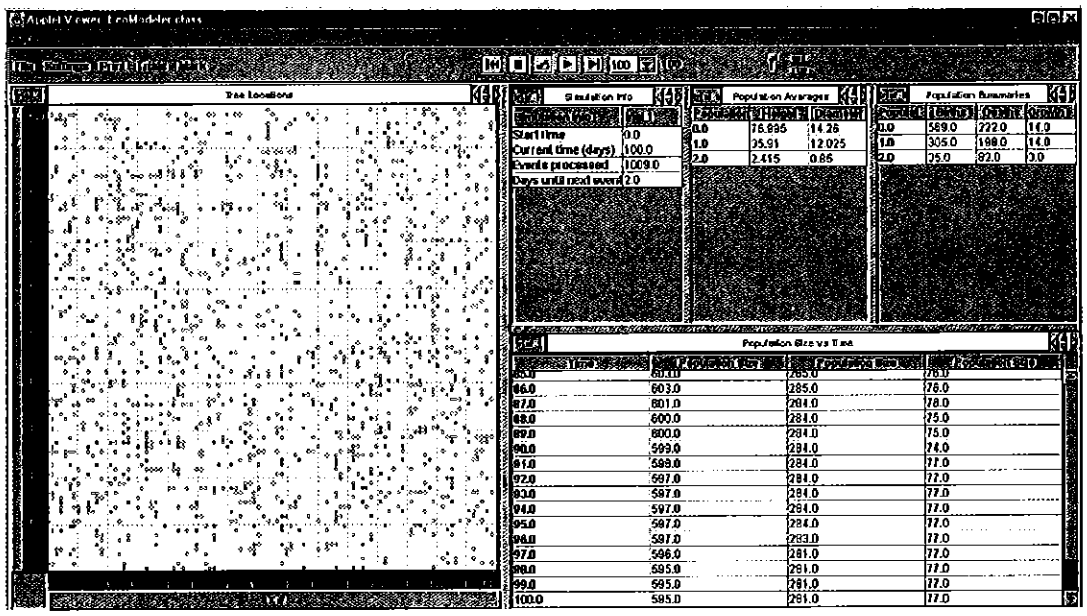


[illegible]	[illegible]
Start time	0.0
Current time (days)	100.0
Events processed	1009.0
Days until next event	2.0

[illegible]	Height	[illegible]
0.0	78.995	14.26
1.0	35.81	12.025
2.0	2.416	0.85

[illegible]	[illegible]	[illegible]	[illegible]
0.0	569.0	222.0	14.0
1.0	305.0	198.0	14.0
2.0	35.0	82.0	3.0

Figure 7: Output after 100 days (table display)



Tree Locations

Simulation Info

[illegible]	[illegible]
Start time	0.0
Current time (days)	100.0
Events processed	1009.0
Days until next event	2.0

Population Averages

[illegible]	[illegible]	[illegible]
0.0	76.895	14.26
1.0	35.91	12.025
2.0	2.415	0.85

Population Summaries

[illegible]	[illegible]	[illegible]	[illegible]
0.0	569.0	272.0	14.0
1.0	305.0	198.0	14.0
2.0	35.0	82.0	3.0

Population Size vs Time

Time	[illegible]	[illegible]	[illegible]
85.0	603.0	285.0	76.0
86.0	603.0	285.0	76.0
87.0	601.0	284.0	78.0
88.0	600.0	284.0	75.0
89.0	600.0	284.0	75.0
90.0	599.0	284.0	74.0
91.0	598.0	284.0	77.0
92.0	597.0	284.0	77.0
93.0	597.0	284.0	77.0
94.0	597.0	284.0	77.0
95.0	597.0	284.0	77.0
96.0	597.0	283.0	77.0
97.0	596.0	281.0	77.0
98.0	595.0	281.0	77.0
99.0	595.0	281.0	77.0
100.0	595.0	281.0	77.0

4.4.6 Summary

Even though the implementations of the behaviors in `GrowthModel` may not be realistic, this model shows that users can define the level of detail in their simulations. This model uses a fairly low level of detail on the individual level, although the amount of detail can be increased by extending the `Tree` class and adding more attributes and by implementing additional `Events` and `EventHandlers`.

4.5 Evaluation of FOREST

FOREST was able to reduce the time needed to implement an individual-based forest simulation (`GrowthModel`), by providing the discrete-event simulation components, extending these components to facilitate the development of ecological models, and implementing a model class heirarchy that supports modular development. The development time was reduced since the model only had to implement the entities and their behaviors, rather than an entire simulation and class heirarchy from scratch. However, the full benefits of using the `EventHandler` implementation would only be realized once a library of component modules had been developed, and techniques implemented for initialization of the model and analysis and visualization of the results.

A new model could be created that uses or extends components from more than one existing model, decreasing the time needed for formulation and implementation of the model. Additionally, modelers could investigate the issue of relevant detail by varying the type and implementation of the events in the simulation and examining the effects of the alterations on the simulation results.

Since FOREST did not implement any extensive techniques for analyzing and visualizing the results, some desired features are missing for communicating the results of the model to the user. In `GrowthModel`, since all three populations are shown in the same color, there is no way to distinguish which population an individual belongs to in the location view. Because EcoModeler was not designed for viewing data about individuals, it does not provide a method to distinguish the populations in the view of locations. However, such a method would be desired if the modeler wanted to examine the spatial patterns of distribution for each species. Another desired feature would be to show the relative sizes of the individual trees in the location view. These methods may be needed to allow the user to extract the desired information from the model's results.

4.6 Conclusion

Although computational models have the potential of becoming useful tools for guiding management decisions, their use has been limited due to the time and complexity involved in developing models to represent specific ecosystems. This complexity may become managable in a modeling environment designed to facilitate collaboration between modelers and to assist in the implementation and calibration of ecological models. The creation of this type of environment would allow modelers to focus on the biological challenges to ecosystem modeling of implementing the behaviors of the entities in the model and determining the appropriate level of detail for the model's purposes.

FOREST was able to simplify the process of developing forest simulation models by providing the main time advance algorithm and reusable components to define behavior of the objects in the model. Since no interactions between individuals were included in FOREST, there was no need for communication between the entities in the simulation. If the model needed to include interactions, the environment would need to provide a mechanism for communication between the objects in the model to trigger the occurrence of conditional events. A full ecological simulation environment would need such a mechanism as well as techniques for initialization and parameterization of the model, and additional techniques for analysis and visualization of the results.

References

1. American Association for the Advancement of Science. 2001. Analyzing Ecological Models (AEL). http://www.sciencemag.org/feature/data/deutschman/eco_model.htm

2. Banks, J., J.S. Carson II, B.L. Nelson, D.M. Nicol. 2001. Discrete-Event System Simulation New Jersey: Prentice-Hall, Inc.

3. Begon, M., J.L. Harper, C.R. Townsend. 1996. Ecology. Oxford: Blackwell Science Ltd.

4. Bolte, J.P., J.A. Fisher, D.H. Ernst. 1993. An object-oriented, message-based environment for integrating continuous, event-driven and knowledge-based simulation. *Proceedings: Application of Advanced Information Technologies: Effective Management of Natural Resources.* ASAE. June 18-19, Spokane, WA.

5. Caswell, H., and A.M. John. 1992. From the Individual to the Population in Demographic Models. In Individual-Based Models and Approaches in Ecology, eds. D.L. DeAngelis and L.J. Gross. New York: Chapman and Hall.

6. DeAngelis, D.L., and K.A. Rose. 1992. Which Individual-Based Approach is Most Appropriate For a Given Problem. In Individual-Based Models and Approaches in Ecology, eds. D.L. DeAngelis and L.J. Gross. New York: Chapman and Hall.

7. Deutschman, D.A. 2000. The Role of Visualization in Understanding a Complex Forest Simulation Model http://siggraph.org/publications/newsletter//v34n1/contributions/Deutschman.html

8. Ellison, A.M. and B.L. Bedford. 1995. Response of a Wetland Vascular Plant Community to Disturbance: A Simulation Study. *Ecological Applications.* 5: 109-123.

9. Folse, J.L., J.M. Packard, and W.E. Grant. 1989. AI Modelling of animal movements in a heterogeneous habitat. *Ecological Modelling.* 46: 57-72

10. Levin, S., B. Grenfell, A. Hastings, and A.S. Perelson. 1997. Mathematical and Computational Challenges in Population Biology and Ecosystems Science. *Science* 275: 334-341.

11. Lomnicki, Adam. 1992. Population Ecology from the Individual Perspective. In Individual-Based Models and Approaches in Ecology, eds. D.L. DeAngelis and L.J. Gross. New York: Chapman and Hall.

12. Matsinos, Y.G., W.F. Wolff, and D.L. DeAngelis. 2000. Can Individual-Based Models Yield a Better Assessment of Population Variability? In Quantitative

Methods for Conservation Biology, eds. S. Ferson and M. Burgman. New York: Springer.

13. Maxwell, T., and R. Costanza. 1994. Spatial Ecosystem Modeling in a Distributed Computational Environment. In Toward Sustainable Development: Concepts, Methods, and Policy, eds. J.C.J.M van denBergh, J van der Straaten. Washington, D.C: Island Press.

14. Slothower, R.L., P. Schwarz, and K.M. Johnston. 1996. Some Guidelines For Implementing Spatially Explicit, Individual-Based Ecological Models Within Location-Based Raster GIS. http://www.sbg.ac.at/geo/idrisi/gis_environmental_modeling/sf_papers/slothower_roger/sf23.html

15. Spatio-Temporal Modeling Page. (STMP) 1995. http://kabir.cbl.umces.edu/SMP/MVD/SMod.html

16. Swartzman, G.L., and S.P. Kaluzny. 1987. Ecological Simulation Primer. New York: Macmillan Publishing Co.