

Thesis

1

2004

0129310

This is to certify that the
thesis entitled

**DIGITAL ORGANISMS IN A VIRTUAL
ECOSYSTEM: DESIGN AND
IMPLEMENTATION**

presented by

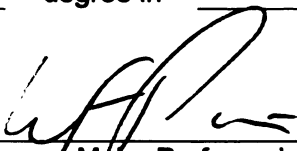
Zhen Lei

has been accepted towards fulfillment
of the requirements for the

Master

degree in

Science



Major Professor's Signature

5/13/04

Date

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.
MAY BE RECALLED with earlier due date if requested.

DATE DUE	DATE DUE	DATE DUE

**DIGITAL ORGANISMS IN A VIRTUAL ECOSYSTEM
DESIGN AND IMPLEMENTATION**

By

Zhen Lei

A THESIS

**Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of**

MASTER OF SCIENCE

Department of Computer Science and Engineering

2004

Abstract

Digital Organisms in a Virtual Ecosystem Design and Implementation

By

Zhen Lei

In this thesis, I discuss the design and implementation of Digital Organisms in a Virtual Ecosystem (DOVE) to study phenotypic plasticity and trait-mediated interactions. We design and implement software that represent populations of model organisms (i.e. predators, omnivores, herbivores) -- denoted digital organisms (DO) -- that evolve and interact in a virtual ecosystem (VE) which is a spatial landscape with heterogeneous resources. DOVE is developed on the SWARM platform using Objective C to fulfill its core functions.

We adopt three principal inter-related computational tools and approaches (i) agent-based models (ABM) for emulation of natural complex systems (ii) computational methods to model the behavior of organisms (e.g. probabilistic rules); and (iii) genetic algorithms for individual adaptation.

We also conduct some experiments to examine competitive species on an aquatic food web by using DOVE. From the experiments, we arrive at some important hypotheses about phenotypic plasticity and trait-mediated interactions.

I present a DOVE user manual and parameter profiles. The profiles describe how to set up the simulation parameters of food webs and import/export information that helps with implementation.

Acknowledgments

It is not easy that I finish this thesis. Hereby I want to thank all the people who have helped me in this work.

First, I am very grateful to my advisor: Dr. Scott Peacor. He helps me finish this very interesting thesis research and always gives me direction when I feel lost in my thesis research. I also acknowledge my academic advisor Dr. William F. Punch and Dr. Eric Torng. Their inspiring comments are very important to my thesis research.

I also want to thank Rick Riolo, Tim Hunter, and Len Kofman at University of Michigan and Great Lakes Environmental Research Laboratory (NOAA). Their advice helps me overcome many obstacles.

At last many thanks to my parents and my girl friend, who are always supportive to my study and career.

Table of Contents

1.	INTRODUCTION.....	1
1.1	MOTIVATION.....	1
1.2	OBJECTIVE.....	3
1.3	ORGANIZATION	3
2.	REVIEW OF TRADITIONAL APPROACH.....	5
2.1	OVERVIEW	5
2.2	AGENT-BASED MODELING	6
2.3	APPLICATIONS OF AGENT-BASED MODELING.....	8
2.4	WHY AGENT-BASED MODELING IN DOVE	9
2.5	GENETIC ALGORITHM.....	10
3.	DIGITAL ORGANISM IN A VIRTUAL ECOSYSTEM.....	13
3.1	GOALS AND PRINCIPLE	13
3.2	AGENT-BASED MODELING(ABM)	15
3.3	MODELING OF ORGANISM BEHAVIOR	16
3.4	GENETIC ALGORITHM	18
3.5	CORE FEATURES AND CONCEPTS	19
3.5.1	<i>HABITATS</i>	19
3.5.2	<i>RESOURCES</i>	20
3.5.3	<i>ORGANIMS</i>	21
3.5.4	<i>INTERACTIONS</i>	22
4.	MODELING OF ORGANISMS AND RESOURCES.....	25
4.1	ORGANISM AND RESOURCE HIERARCHY	25
4.2	AGENT CLASS.....	27
4.3	RESOURCE AND PERIPHYTON CLASSES	28
4.4	SPECIES CLASS.....	30
4.5	ORGANISM CLASS.....	33
4.5.1	<i>CLASS DEFINITION</i>	33
4.5.2	<i>CLASS METHODS</i>	34
4.5.2.1	<i>MOVING ACTIVITY</i>	34
4.5.2.2	<i>EATING ACTIVITY</i>	35

5.	GENETIC ALGORITHM FOR ADAPTATION	41
5.1	PRINCIPLE	41
5.2	IMPLEMENTAION	44
5.2.1	<i>INTELLIGENTCE CLASS</i>	44
5.2.2	<i>GENE CLASS</i>	45
6.	MODELING OF ECOSYSTEM ENVIRONMENT	48
6.1	SYSTEM ARCHITECTURE	48
6.2	DOVEGRID CLASS	49
6.3	MODEL CLASS.....	52
7.	EXPERIMENTS AND ANALYSIS	57
7.1	OVERVIEW	57
7.2	EXPERIMENTS	57
7.2.1	<i>PHENOTYPIC PLASTICITY AND BEHAVIORS EVOLUTION</i>	58
7.2.2	<i>WITHOUT PHENOTYPIC PLASTICITY</i>	60
8.	CONCLUSION AND FUTURE WORK	63
8.1	CONCLUSION.....	63
8.2	FUTURE WORK	64
APPENDIX A.	HOW TO RUN DOVE.....	66
APPENDIX B.	PARAMETERS SETTING.....	69
APPENDIX C.	IMPORT/EXPORT.....	76
APPENDIX D.	USING DRONE TOOL.....	78
APPENDIX E.	EXPERIMENT PROFILES	81
BIBLIOGRAPHY		86

LIST OF FIGURES

Figure 1: Resources levels in different habitats of DOVE	20
Figure 2: Food Web in DOVE	22
Figure 3: Organism Hierarchy	25
Figure 4: Examples of Gene sequence.....	42
Figure 5: Examples of genetic Evolution.....	43
Figure 6: Dove Model Architecture.....	48
Figure 7: Behavioral Evolution.....	59
Figure 8: Experiment without phenotypic plasticity.....	61

1. Introduction

1.1 Motivation

The study of complex systems is a very new research area that interests scientists and researchers in diverse disciplines, which include but are not limited to ecology and environment (DeAngelis and Gross, 1992, etc.), psychology and cognitive science (R. Serra, G. Zanarini, 1990, etc.), social simulation (Epstein and Axtell, 1996, etc.), and industry (Bunn and Larsen, 1997, etc.). A complex system is usually constituted of many elements that interact. The complexity of the system is proportional to the numbers of the elements, the number of interactions in the system, and the complexities of the elements and of the interactions (Bar-Yam, 1997).

Ecological communities are one of the most complex systems that scientists attempt to understand. Many global patterns or properties of the ecological system are not presented in parts of the system. Instead, they emerge from the interactions of the elements, e.g. various resources and organism of different species, of the systems. The complexity in ecological system further arises from the nonlinear interactions of large numbers of individuals and species in dynamic environments (Brown 1994). Clearly, an ecological community is more than the sum of its parts.

In order to better understand the structure and dynamics of ecological communities, ecologists need tools and conceptualizations to capture the global pattern and properties of the communities, and thus can make reliable prediction of the ecosystems. From this perspective, different theoretical approaches of food webs, which define the non-linear

and complex network of many interconnected food chains and feeding relationships, is critical in modeling and emulating the natural ecology communities.

Traditionally, researchers tried to construct the whole food web in ecological system by combining static descriptions of direct species-pair interactions, which describe the relationship between different species and how the densities of species in the species-pair affect each other, to model more complex communities (e.g., Levine 1976, Bender et al. 1984, Holt 1987, Yodzis 1988, Leibold 1989). However, this “building blocks” method neglects a critical property of ecological systems – the existence of phenotypic plasticity (PP): individual organisms adapt to short-term environmental changes by altering their phenotypes, e.g., traits such as behavior, morphology or life history (Lima 1998, Tollrian and Harvell 1999). The change of the traits of species in the species-pair can affect the intensity of the interactions, and potentially the species-pair relationship itself. Moreover, in a complex ecological community, different species in the food web may affect others indirectly through intermediate species. An important property of food web originating from such indirect relationships among species is that the trait changes not only affect interactions with the predator that they are responding to, but also other species in the food web. For example, the reduction in prey activity may reduce the prey’s consumption on a food resource, which not only leads to a decrease in prey growth rate but also an increase in resource abundance. On the other hand, the change of the prey activity will also affect the growth rate and the number of the predators, because predators cannot get enough foods with fewer available preys that response more acutely to the predators.

Thus, predators can affect both prey and base resources even without actually killing prey (e.g., Wootton 1993, Abrams 1995). Such types of indirect interactions caused by

trait changes are denoted trait-mediated interactions (TMI), because they arise from the modification of a trait (e.g., behavior, morphology or life history), rather than the density, of the intermediate species (Abrams et al. 1996).

Dr. Scott Peacor and other researcher have conducted many field experiments on trait-mediated interactions, and proved that TMI and its requisite mechanism, phenotypic plasticity (PP) in traits, are ubiquitous and important in natural systems communities (Werner and Peacor 2003; Lima 1998, Agrawal 2001; Peacor and Werner 2001).

1.2 Objective

We, including Dr. Scott Peacor, Dr. Rick Riolo, myself, and other collaborators, are developing a new computational system, denoted “DOVE” which stands for “Digital Organisms in a Virtual Ecosystem” in order to study ecological communities and in particular examine the origin and role of phenotypic plasticity and of consequent trait-mediated interactions. The principal goal of the proposed computational system is to address the shortcomings of traditional approaches and provide a complementary approach in the research of ecological communities.

1.3 Organization

Chapter 2 provides an overview of previous research and approaches on complex systems. Chapter 3 gives an overview of DOVE and introduces three central computation tools/approaches used in DOVE. Chapter 4 discusses the agent-based modeling in DOVE and the modeling of organisms and resources. Chapter 5 explains genetic (evolutionary) algorithms for individual adaptation. Chapter 6 describes how to model the ecosystem

environment of DOVE. In chapter 7, we conduct experiments to examine different species on aquatic food webs by using DOVE and present the results. Chapter 8 summarizes all the works carried out in this thesis research and suggests future directions of study.

2. Review of Complex System Theory

DOVE is designed to examine ecological communities as complex systems. In this chapter, I first introduce complex system theory and review, focusing primarily on the applications of Agent-Based Modeling (ABM), in this area. As examples, I summarize current applications of ABM in different disciplines, including my previous research on agent-based modeling for land use change. I then provide a brief introduction of Genetic Algorithm (GA), which we use in DOVE.

2.1 Overview of Complex Systems

Defining complex system is a complex task. There are more than seventy definitions of complexity (John Horgan, 1995), used in diverse areas. Because it is inspired in practically all branches of knowledge and is a very new field of study, I will try to describe complex systems in a general way.

A complex system usually has many components that interact at one level of organization to create patterns at higher levels of organization. A system in the life, social, physical or decision sciences are considered a complex system if it has a significant number of characteristics below (Bar-Yam, Y., 1997):

- Agent-base: the fundamental building blocks of the system are the characteristics and behaviors of the individual elements (or called agents) in the environment
- Heterogeneous: agents differ from each other in important characteristics.

- **Dynamic:** The behaviors and characteristics of agents may change over time, as they learn from their experiences, adapt to their environment, or evolve through the natural selection in reproduction process.
- **Feedback:** The agents perceive and response to a dynamic of environment and this causes feedbacks that usually become the driving force for the changes in their characteristics/behaviors.
- **Organization:** Agents can belong to groups or form hierarchies. The infrastructure of the diverse agents changes the properties of whole system. These changes are usually nonlinear, sometimes even chaotic.
- **Emergence:** The global behavior of the complex system arises from the interactions of the elements of the system. In this sense, a complex system is more than the sum of its parts. A complex system has properties not present in its parts. They emerge from the interactions of the components of the system.

Many man-made and natural systems are complex in this way—from economies, societies, cultural norms, and traffic jams through ecosystems, immune systems, weather patterns. The "non-linear" dynamics of these systems are difficult to study using traditional models. Approaches to study such complex systems, including the DOVE ecosystem we propose, are usually constituted of many elements that interact.

2.2 Agent-Based Modeling

Agent-based modeling is the modeling approach that has been developed in the complex systems research community. The term "agent-based modeling"(ABM) refers to a collection of computational techniques in which individual software agents and their

interactions are explicitly simulated, and emergent properties are observed. In traditional differential-equation modeling methods, larger-scale properties of a system are the fundamental elements of the model, e.g. population densities of species, densities of resources. However, agent-based modeling uses the individual agents/individuals as the fundamental elements of the model. The goal of agent-based modeling is to design models that are sufficiently simple that the mechanisms of emergence can be understood and yet elaborate enough to show interesting behavior (Mitchell and Newman, 2002).

Because the very complicated behaviors of real-world complex systems emerge from the relatively simple, local interaction of many different individual components, Agent-Based Modeling (ABM) is a very appealing approach to emulate the complex system such as ecological communities. A principal advantage of ABM is the ability to access the plausibility of the behavior of agents, the way in which the agent interact and the consequences of that behavior and interaction to the structure and dynamics of the ecological community.

In agent-based modeling, a software agent is an autonomous, problem-solving computer program, which carries out goals/tasks and operates in dynamic and open computational environments. Each agent has a specific goal or function, which it will try to complete. It interacts with other agents that have possibly conflicting aims. Agents can be embedded in other agents. From the computer science perspective, a software agent is a component: an object (Booch, 1994) which might communicate with other components. Since the tendency in software systems is to develop them in a distributed way, agent theories are having a great influence on computer science, for example, agent-oriented programming (Shoham, 1993) and agent-based software engineering (Wooldridge, 1997).

2.3 Applications of Agent-Based Modeling

Agent-based systems are constructed to represent and simulate problem-solving situations, where collaborative and conflict behaviors can co-occur as they do in real human and natural environments of our daily life (Murch and Johnson, 1999). Agent based models have been developed to understand human societies (e.g., Epstein, et al., 1996), avian evolution (e.g., Axelrod, 1997), the life histories of animals in dynamic landscapes (DeAngelis, 1992) and the evolution of economic systems (e.g., Holland and Miller, 1991), to name a few. Agent based modeling approaches have been recently applied to simulate land use changes (Alexandridis and Pijanowski, 2002; Parker, 2001). Multi-agent systems are being used to simulate a variety of real-world behavioral scenarios.

As an example of the Agent-Based Modeling, the land use simulation system “Multi Agent-based Behavioral Economic Landscape (MABEL)” (Lei and Pijanowski, 2003; Alexandridis and Lei, 2004) provides a bottom-up approach to allow the analysis of dynamic and complex features and relations in land transformation observations. The agents in MABEL represent the real-world landowners who base their behavior and actions on their perception of maximizing their total marginal utility. Agents communicate, interact, conflict, and collaborate, to their environment and to each other. They generate a complex, dynamical system of changes that leads to the land transformation patterns throughout time. Because of the application of agent-based modeling, MABEL allows the analysis of dynamic and complex features and relations, as well as the need for comprehensive relational schematics in land transformation. In MABEL, agent-based modeling increases the efficiency on assessing a more real-world

observational system, especially in conjunction with land use, environmental issues, and environmental economics. Through the modeling of real-world entities in comprehensive attributions and behaviors, agent-based modeling provides a ‘unified’ approach between living system’s organizational entities in environment and land use, and socio-economic interpretations, which goes beyond the limitations of the traditional statistical and econometric assumptions.

2.4 Why Agent-Base Modeling in DOVE

Compared with traditional equation-based modeling, agent-based modeling is a very appealing approach to emulate the ecological communities and examine the phenotypic plasticity of our interest. I summarized the advantages of Agent-base modeling in DOVE below;

First, the principal advantage of agent-based modeling is that it provides a bottom-up approach to capture the global behavior of ecological communities and examine the complex dynamic arising from the non-linear relationships among organisms. In DOVE, the aggregate measures like food web structure and interaction coefficients, are emergent properties of the system, i.e., the patterns of species densities and interactions. They can be determined from the “bottom up” through the repeated local interactions over time and space of the multiple agents (e.g., Epstein & Axtell 1996, DeAngelis & Gross 1992). Through the ABM approach, we can access the plausibility of the behavior of agents, the way in which the agent interact and the consequents of that behavior and interaction.

Second, agent-based system is code-efficient and easier to build and run (Bar-Yam, 1997), In an agent-based model, such as DOVE and MABEL, the heart of the code is the

agent behavioral methods, and we write these only once; the behavioral module is the same for each agent. Thus, a relatively short program at compile-time is actually a very large program at runtime if we have a large number of agents interacting. Moreover, running such a model is simply instantiating an agent population, letting the agents interact, and monitoring what happens. There is no need to appeal to representative agents. In this sense, executing the model is all that is necessary in order to solve it.

Third, agent-based modeling offers an additional level of validation (Mitchell and Newman, 2002). Both agent-based modeling and equation-based modeling can be validated at the system level, by comparing model output with real system behavior. In addition, ABM can be validated at the individual level, since the behaviors encoded for each agent can be compared with local observations on the actual behavior of the domain individuals. This advantage of ABM allows us to validate the functionalities in DOVE by simply turning on/off some parameters and then comparing the simulation results.

Forth, ABM supports more direct experimentation at individual level (Murch and Johnson, 1999). We can play some “what-if” experiments by turning on/off the parameters of organisms. Such capability provides the flexibility we need for extensive simulations on phenotypic plasticity.

2.5 Genetic Algorithms

A goal of DOVE is to allow the phenotypic plasticity of organisms “emerge” through interactions at the individual level with the environment. To achieve this, we use genetic algorithm to allow behaviors, and hence phenotypic plasticity, to evolve.

John Holland at University of Michigan formally introduced genetic algorithms in the 1970s (Holland, 1975). Genetic algorithms belong to the class of stochastic search methods. Whereas most stochastic search methods operate on a single solution to the problem at hand, genetic algorithms operate on a population of solutions (Goldberg, 1989).

To use a genetic algorithm, a solution to a specific problem should be represented as a genome (or chromosome), which emulates the biological genome with a finite set of character strings or binary values (in DOVE we use double types of numeric values). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s).

In the genetic algorithm, each individual must represent a complete solution as a genome (or chromosome) to the problem one is trying to optimize. The representations for the individual genomes in the genetic algorithm include strings of bits, arrays, trees, lists, or any other object. For a certain representation of genome, we must define corresponding genetic operators (initialization, mutation, crossover, comparison).

Two of the most common genetic algorithm implementations are 'simple' and 'steady state'. In the simple genetic algorithm, the entire population is replaced each generation; while in the 'steady state' genetic algorithms, only a few individuals are replaced each 'generation'. This type of replacement is often referred to as overlapping populations. In Dove, we choose a more natural and flexible approach of the simple genetic algorithm by allowing users to define the maximal age, which can be a very large numeric value, for each species.

Genetic algorithms use various selection criteria so that it picks the best individuals for mating (and subsequent crossover). The objective function determines how 'good' each individual is. If you use a selection method that picks only the best individual, then the population will quickly converge to that individual. So the selector should be biased toward better individuals, but should also pick some that aren't quite as good (but hopefully have some good genetic material in them). In DOVE, we have no prior knowledge about the global level behavioral preference before we run the program. Thus, we choose to emulate the natural and implicit fitness mechanisms by setting up the interaction rules between organisms and the requirements for reproduction. We hypothesize that our implicit fitness mechanisms will favor those who can respond promptly to the changes (e.g. predation risk level) in the environment.

The genetic algorithm is very simple, yet it performs well on many different types of problems. In particular, genetic algorithms work very well on mixed (continuous and discrete), combinatorial problems. Genetic Algorithms have been applied in molecular structure optimization (D. M. Deaven and K. M. Ho, 1995), Medicine (B. Sahiner, H. P. Chan, 1996), and Automated Design Methodology (E. D. Goodman, K. Seo, 2002) etc. Genetic algorithms are less susceptible to get 'stuck' at local optima than gradient search methods. However, Genetic algorithms tend to be computationally expensive.

3. Digital Organisms in a Virtual Ecosystem

3.1 Goals and Principles

In order to study phenotypic plasticity and trait-mediated interactions in ecological communities, we propose to develop a new agent-based computational system - DOVE.

Because of the nature of ecological communities, we adopt agent-based modeling in the design of the DOVE system. The principal idea behind DOVE is to create populations of model organisms (i.e. predators, omnivores, herbivores) as intelligent software agents - denoted digital organisms (DO) – that evolve and interact in a virtual ecosystem (VE) which is a spatial landscape with heterogeneous resources.

With the flexibility and capabilities provided by computational simulation in DOVE, we can construct population of organisms in the different scenarios to examine the role of phenotypic plasticity and trait-mediated interactions at different levels of complexity in extensive environments. For example, we can build simple or complex food web structures with various species to compare the impacts of phenotypic plasticity on the same species under different environments. Moreover, we can turn on/off the properties of population, i.e. evolutionary capability, individual variation, or change the environmental parameters like the number of habitats, to examine what the roles of these properties and how they are related in determining the behaviors/characteristic of organisms in the long term. In addition, DOVE also provides us a way to compare the global behavior of the system caused by phenotypic plasticity with those that obeys assumptions made in traditional equation-based models. Thus, DOVE is an effective tool to study the dynamic of the ecological communities at different complex levels. The

flexibility of DOVE allows the researchers to control species properties, population combination, food web structure, and environmental factors.

In DOVE, various organisms/agents differ in their behaviors as a function of their perception of their “states”. These states include the state of the environment (e.g. density of predators in local vicinity or in the global ecosystem) and their internal state (e.g. hunger level). For a certain combination of internal and external states, an individual organism can take an action, such as feeding, sitting or switching habitat, with a particular probability. An organism’s behavior is described by the probability to take particular actions under a certain situation. These probabilities serve as the gene that evolves.

We model agents/digital organisms based on the behaviors of natural organisms. As organisms in a natural ecological community, the digital organisms that fit well with the changes in environment are less likely to be killed and more likely to reproduce. For example, consumers that are less active when predators are present, and more active when predators are absent, will have less possibility to be captured by predator and be more likely to get enough foods for growth, since activities normally make organisms more recognizable and vulnerable to predators. Thus, they can accumulate sufficient resources to reproduce and pass on their genes, which eventually evolve the fitness towards the environment.

We have designed DOVE such that phenotypic plasticity and trait-mediated interactions are emergent properties arising from local interactions and evolutionary selection for individuals that maximize fitness in sexual reproduction. After individual organisms acquire enough resources to reproduce, they can reproduce sexually and create

offspring, who inherit the behaviors from parents. Because the organisms that fit well with the environment have a better chance to survive and reproduce, the population of new offspring will inherit and evolve the behaviors that optimize fitness in the dynamic virtual ecosystem gradually. On the other hand, the variation among individuals and their behaviors in the sexual reproduction, plus the possible mutation in the gene recombination, also ensure that the evolution of organisms will not get 'stuck' at short-term optima.

In order to achieve our goals, we adopt the three inter-related computational tools and approaches used in general complex systems approaches (Chapter 2) in the design of DOVE, including: (i) agent-based models (ABM) for emulation of nature complex system (ii) probabilistic methods to model the behavior of organisms; and (iii) genetic algorithms for individual adaptation. I then explain how we apply these approaches in DOVE, and what make them different in our system.

3.2 Agent-based Modeling (ABM)

In DOVE, phenotypic plasticity and trait-mediated interactions are the emergent properties arising from simple interactions of digital organisms. The relationships between pairs of species are pre-defined according to their positions in the food web structure. We create and instantiate software agents for individual organisms with some distribution of initial states, and model their behaviors with defined traits and rules of interaction with other agents and the environment. The agents, which have behaviors and traits, provide a natural and fundamental basis for the interaction at individual level.

Agents can perceive the internal and external statuses, react upon to the environment, and adapt to the changes in the environment.

The evolutionary mechanism enables agents to change the trait of species over generations. The driving force behind the trait changes is genetic algorithm that emulates the natural selection in reproduction process through (see details in next section). This selection process ensures that the organisms that react promptly to predation risk and fit well with environment chances have a better chance to survive and propagate their fitness through sexual reproduction.

Thus, modeling agents at the individual level present a natural basis to access the behaviors of organism and examine the role and impact of PP as a response to local interactions. Moreover, it provides a natural platform to incorporate evolution mechanism, by which we can understand the dynamics of ecosystem and study what environmental factors produce PP in the evolution process.

We develop the DOVE computational system in the SWARM simulation environment developed at the Santa Fe Institute. SWARM is an agent-oriented approach for building simulations, which extends software agent theory itself. It was designed with simulation purposes, including artificial intelligence and artificial life.

3.3 Modeling of Organism Behavior

In DOVE, we declare objects of organisms and specify their behaviors. The integration of all the objects into an environment provides the simulation model. Therefore, an object of organism is described by an entity that holds both the descriptive attributions of the objects as well as its behaviors.

Currently, we model the organism traits and capabilities by probabilistic method. The organisms react with actions towards the states that they perceive. These states include:

- Predation risk: how many predators the organism can observe?
- Internal resource level: how close the organism is to starving or reproducing?
(this is analogous to a real organisms size and energy reserves)
- Available resource: how much food is in the organism's feeding range?
- Relative resource level: how high are the resources in the feeding range compared to what the organism has seen recently?
- Number of habitats: how many habitats can the organism inhabit?

For a certain combination of internal and external states, an individual organism may take an action, such as feeding, sitting or switching habitat, with a particular probability. The possible actions and associated probabilities under a situation are organisms' behaviors that we model.

Our experiments in DOVE suggest that the modeled digital organisms can evolve some kinds of phenotypic plasticity analogous to that observed in many natural systems. We plan to study more complex representations of agent trait and behavior rules, e.g., by modeling each agent as a Classifier System in which a set of interacting behavioral rules collectively determines the behavior of the agent (Holland 1986; Lanzi & Riolo 2002) in the future. By exploring different representations in the same scenario, we can avoid the biases that particular representations of behaviors may introduce.

The probability modeling of organism behaviors required a single behavioral probability matrix, or call chromosome with genes, for each agent. Thus, it provides a natural basis for the incorporation of genetic algorithm.

3.4 Genetic Algorithm

We employ genetic algorithms in our computational systems to model the evolution of individual traits, including phenotypic plasticity. The genetic algorithm is well suited to our needs, since it can be used to model evolution in a population of agents with traits encoded as single “chromosomes”, to evolve rules in agents modeled by probabilistic method.

In DOVE, a gene’s instructions produce an outcome (or payoff) that affects the agent’s reproductive fitness relative to other organisms in the computational ecology. Relative fitness determines the probability that each behavior will propagate. Propagation occurs when two organisms mate, while involves the recombination of their genes. Such recombination allows them to create an entirely new organism that may integrate the best (or worst) abilities of each “parent,” making possible the new organism superior to either contributor. Thus, the new behaviors may eventually displace both parent behaviors in the population of organisms. In addition, the new gene may contain random copying errors. These mutations restore the heterogeneity of the population, counteracting selection pressures that tend to reduce it.

An important feature of DOVE is that it has no explicit objective function, or call fitness function, on the system level. Generally, genetic algorithm employs various selection criteria so that it picks the best individuals for mating. The objective function determines how 'good' each individual genome is. In my understanding, such explicit objective functions on system level are more suitable for optimization purpose with definite goals. In DOVE, we have no prior knowledge about the global level behavioral preference before we run the program, and we believe the emergence of global behaviors

arises from the interactions of the elements/organisms of the system. Thus, we choose to emulate the natural and implicit fitness mechanisms in DOVE. Because the interaction rules between organisms and the requirements for reproduction favor those who can response promptly to the changes (e.g. predation risk level) in the environment, such implicit fitness mechanisms allows organisms to evolve towards an optimal solution for the world without the user having to specify an objective function.

3.5 Core Features and Concepts

DOVE is an agent-based simulation environment for ecosystems. Researchers can create different resources, organisms in heterogeneous habitats to construct complex food web structures. Before explaining DOVE design and implementation in the next chapter, I first introduce some key concepts on ecosystem in DOVE.

3.5.1 Habitats

DOVE emulates the natural ecological systems in heterogeneous habitats. As their counterparts in nature, organisms and resources in DOVE are located at particular positions of the world and interact with other organism or resources in the neighborhood. We model the world of DOVE as a 2-D grid with heterogeneous habitats, which is distinct from each other in environmental attributions, such as available resources growth rates and predation risk levels.

In DOVE habitats, there are no edge effects. Each habitat is a torus so that if organism moves left and goes off the world it would appear on the right most portion of the world.

The habitats have many real world analogs. Multiple organisms and resources can live in same location. Different digital organisms can live and move around in different habitats. The heterogeneity in habitats allows organisms to switch habitat to prevent predation risk or get more foods. Currently DOVE can support either one or two habitats in the simulation world.

3.5.2 Resources

The resources are initially distributed evenly within habitats, and then grow at user-defined rates. As mentioned above, multiple resources can grow in the same location.

Currently, resources can grow logistically in DOVE. According to a certain growth function, a resource may grow over time until it reaches the carrying capacity of the resource, which represents the maximum level of resources available per location in a habitat.

$$R(t+1) = rR(t) \left(1 + \frac{(r-1)R(t)}{K}\right)^{-1}$$

Where R is the resource level, r is the intrinsic growth rate, and K is the carrying capacity. $r = 1$ is zero growth.

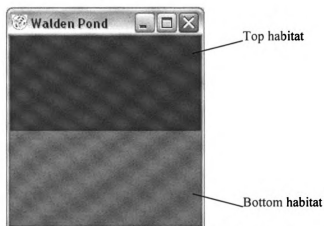


Fig.1. Resources levels in different habitats of DOVE

As illustrated in Fig.1, The various shades of green to black represent different tropic levels of resources.

3.5.3 Organisms

The organisms in DOVE can be either herbivores (creatures that only eat resources) or predators (organisms that eat other organisms). Each organism has an internal resource level. As the organisms in nature, the digital organisms in DOVE consume energy because of metabolism, even if they just sit statically. For each species, users can define the amount of energy lost due to metabolism at every time step. Thus, organisms' internal energy/resource level goes down every time step unless users set it to zero, and increases when the organisms eat. The digital organisms can take one of the three possible actions: feed, sit and switch, according to the "state" it observes. We discuss how organism chooses the action in next section of this chapter.

An organism can die in four ways: being eaten by predator, aging, starvation, or by "background death" (i.e. die for other reasons external to the environment represented). Predators can eat the organisms and get all internal resources of the preys. Each organism is born with a maximum age and each time step its age increases by one. Organisms die if their ages reach the maximum age. The organism starves if it does not get sufficient foods and dies when its internal resource level drops below zero. Organisms may die for other unclear or unpredictable reasons. Users can define a background death probability for this case.

As a critical part of natural organisms, the resource requirement for sexual reproduction ensures that those fitting well with the environment will have better chance

to survive and reproduce. Whenever an organism accumulates sufficient internal resources, it has the ability to reproduce with another organism ready to mate. They will bear an offspring that inherits from their “gene” of behaviors probability matrixes. After reproduction, parents lower their internal resources level to the initial status.

In DOVE, we also introduce a new entity - global predators, to emulate the global predation risk in the ecological environment. In some sense, global predators is the observable and system-level predation risk, which research can manipulate in the simulation to represent the predation pressure for the whole ecological community and examine the behavioral reactions of organisms. They are a special class of predators. Global predators make their appearance by having the number of global predators change to a random number after a random period. They have the ability to eat preys anywhere with a particular probability. However, they do not mode, reproduce or die, and thus have no behavioral evolution among them. Global predators are used in DOVE in order to be able to create predators with a defined density and dynamics that are not subject to feedbacks and changes in the simulated environment.

3.5.4 Interactions

The fundamental interactions between organisms and resources are, consumer→resource and predator→prey, shown in Fig 2.

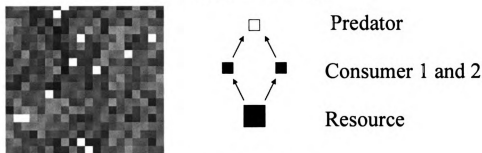


Fig.2 Food Web in DOVE

Consumer→resource interactions are simple and on the bottom of any food web, and provide the energy input of the ecological community. A consumer can eat part of whole amount of the resource and convert it to internal resource. Normally, a consumer cannot eat out all the resource at a spot. There are two parameters putting the limitation: diet refuge and eat percent. Diet refuge represents the percentage of the available resource that organism cannot eat at any cell; eat percent indicates the percentage of the available resource that organism can eat after diet refuge mechanism. Thus, the amount of resources available to be eaten is $K \cdot (1 - \text{DietRefuge})$, where K is the carrying capacity (maximum level) of the resource, and the amount of resource consumed by an organism is $K \cdot (1 - \text{Diet Refuge}) \cdot \text{Eat Percent}$.

Predator→prey is straightforward while critical relationships in the food web. The predator can eat organisms of a particular species, and get all the preys' internal resources when they do. Obviously, a predator in a predator→prey relationship may be the prey in others. A predator can also eat resources and organisms of multiple species.

In DOVE, the predator→prey relationship is more than a who-eat-who issue. Because individual organisms can adapt to changes of the predation risk by altering their behaviors, the change of the organism behaviors in the Predator→prey relationship can significantly affect the intensity of the interactions. Therefore, I then explain how the presence of predator affects the behaviors of preys.

As we mentioned above, organisms have three possible actions, feed, sit and switch, to take for the next time step. The strategy in choosing which action to take represents the

behaviors of organisms. Different organisms, even in the same species, present different behaviors for the same situation/state.

In third section of this chapter, we mentioned that an individual organism may take an action with a particular probability for a certain state. Thus, in the beginning of the simulation, user can initialize the behaviors of organism with the probability matrix for all states. There are two issues that users should consider in defining the possibility that an organism takes a particular action: the risk to be captured by predator, and the need to get foods for surviving and growth. Thus, whether taking an action or not depends on the trade-off between these two issue according to current “state” the organism observe.

We cited some examples on these trade-off issues from the knowledge on natural organisms. A natural organism that feeds can get the maximum resource that it observes, while taking highest risk of being recognized and captured by predators. Organisms taking sit action are less liked to be captured by predator, while they do not get food to reproduce and make up for internal resources loss due to metabolism. Organisms in a habitat of high predation risk may choose to switch to other safe habitats to live, while others who feel hungry can take risks to switch to a fertile but dangerous habitat.

In this chapter, I explain how we adopt the three inter-related approaches, agent-based modeling, genetic algorithm, and probabilistic modeling, in the design of DOVE, and briefly introduce the core concepts and entities of DOVE. In the next three chapters, I will present the implementation details of important system modules.

4. Modeling of Organisms and Resources

The modeling of organisms and resources is fundamental to the design of our virtual ecosystem. In this chapter, I explain how organism and resource objects are implemented in DOVE. First, I will present the hierarchy with a brief introduction of what the classes of organisms and resources do, and then give a more detailed account of each class.

4.1 Organism and Resource Hierarchy

I use the Object Oriented Programming (OOP) diagram to illustrate the hierarchy of organisms. As Figure 3 shows below, Agent is the base class for all organisms and resource classes that exist in the virtual environment.

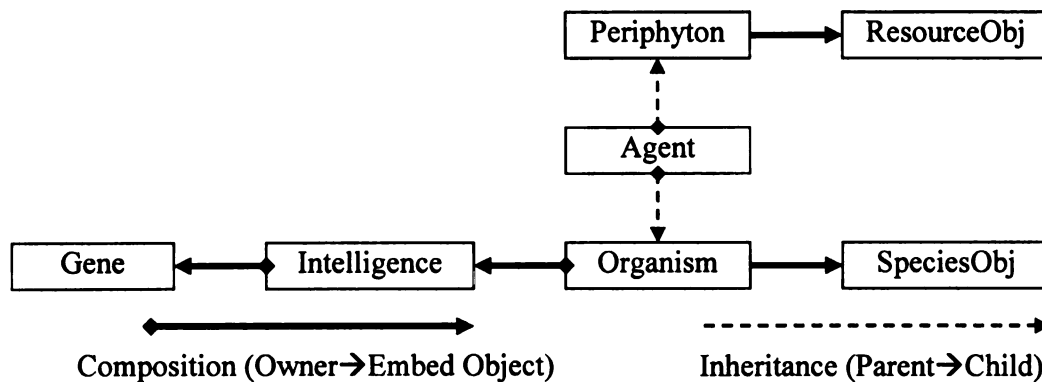


Figure 3. Organism Hierarchy

Every resource belongs to a particular resource species. The attributions of a species such as maximum age are constant. On the other hand, resources objects may be different in physical conditions and growth phases. Thus, we have two classes: Resource class and Periphyton Class. The former includes all species-level attributions, while the other specifies the properties or states that may be different among resource objects. The species-level attributions in ResourceObj class include carrying capacity, growth rates, initial resource level setting, and species-level statistic functions. Users can define these

species-level attributions of a ResourceObj object before the simulations start. This class is also responsible to collect different resource statistics at the species level. The individual resources are all instances of the Periphyton class. This class holds attributions such as periphyton object's current location and resource level. The Periphyton objects can be eaten and grow.

A similar structure exists for the organisms. There is a SpeciesObj that has all the species-level attributions. These include a large number of variables ranging from average death age to anti extinction number. The functions in this class are mainly statistic collection and export/import functions. In parameters files, users can also define the attributions of a species that will be loaded to the corresponding species object in the initialization phase of the simulation.

The individual organisms are all objects of the Organism class. The Organism class is one of the largest and most complex objects in the program. It includes functionality for most of the organisms' activities including moving, eating, reproducing and dieing. There is also a large section in the code to deal with import and export of organism's attributions in population files. In order to make future changes on the intelligence module of organisms easier, we separate the intelligence components from the Organism class. This leads to the creation of two objects, the Gene object and the Intelligence object.

The Gene class implements the genetic algorithm. A state-action behavioral matrix of an organism represents the genome of this organism. The Gene class also provides a gene operator such as crossover and mutation to create a new genome for offspring. Given a set of state variables, the Gene object can select an action from available options. The

Intelligence object is an interface between the Organism class and the Gene class. By having this separate Intelligence class, the genetic algorithm can be more easily replaced with possible other learning systems such as a classifier system or neural networks.

4.2 Agent Class

Agent class defines the basic properties and methods shared by both organisms and resources, so that organisms and resources classes can inherit the same manipulation interface from this class. The definition of Agent class is shown below.

```
@interface Agent : SwarmObject {  
    int ID;  
  
    id myGrid;    // what world/grid do I live in?  
  
    int myX, myY; // my location in my world/grid  
  
    int myHabitat; // which habitat of the world (top=0 or bottom=1) am I in?  
  
    Color myColor; // for drawing me  
  
    int speciesNum; // species number for predation determination  
}
```

The important functions in the class are:

➤ -(id) drawSelfOn: (id <Raster>) r;

Usage - draw agent itself in the world/grid

➤ -(id) getGrid;

Usage - get the pointer to the world/grids

➤ -(void) setGrid: (id) G;

Usage - set the pointer to the world/grids

Note the simulation world is represented in grid/cells format. Each agent can only occupy a single cell at a certain time. I will discuss the world/grid implementation in the sixth chapter.

4.3 Resource and Periphyton Classes

As mentioned before, the Resource class defines the common properties shared by individual resources, such as carry capacity, growth rates etc. The definitions of ResourceObj class are shown below.

```
@interface ResourceObj:SwarmObject {  
    int idNum;  
  
    int numActions;           // currently only 1 action – grow  
  
    //in the following four arrays, array[0] - top habitat, array[1] - bottom habitat  
  
    float growthRate[2];      // growthRates  
  
    float carryCapacity[2];    // maximum amounts of resources in a cell  
  
    float initialMean[2];      // mean value for normal distribution of initial resource  
  
    float initialVar[2];       // variance value for normal distribution of initial resource  
  
    char resourceName[50];     // resource name  
  
    float averageResource;     // statistics variable, average resource in “world”  
  
    float averageResourceTop; // statistics variable, average resource of top habitat  
  
    float averageResourceBottom; //statistics variable, average resource of bottom habitat  
}
```

Currently, we only have the periphyton resource existing in the aquatic environment. It simply includes a specific instance of ResourceObj class, and a variable indicating current resource level.

```
@interface Periphyton : Agent {
    double resource;

    ResourceObj * resourceInfo;
}
```

We explain several important functions in Periphyton class below:

➤ -(double) getResource;

-(void) setResource: (double) r;

Usage - get/set resource level

➤ -(double) eatPeriphyton: (double) amountToEat;

Usage - Eat some portion of this resource. The amount to be the same is specified in amountToEat.

➤ -(void) grow;

Usage - define resource growing function. Each cell has resources that grow according to a fixed equation

$$R(t + 1) = rR(t)\left(1 + \frac{(r - 1)R(t)}{K}\right)^{-1}$$

Where R is the resource level, r is the intrinsic growth rate, and K is the carrying capacity. $r = 1$ is zero growth. This equation is called the logistic equation, and is used frequently in ecological theory to describe density dependent growth.

Resources do not move between cells. There can be more than one species of resource. For each species, there can be one resource agent on each cell.

Unlike animals, resources are not important as individual agents. They serve as a resource for animals, but we do not follow individuals. We are not concerned with their evolution, or variation in traits, etc. However, we do record their mean value, and

possible metrics of their heterogeneous distribution in order to understand the dynamics of the system.

In the schedule of SWARM program, step function will be executed every time step. Thus, we include the grow function in step() function of Periphyton class.

```
-(id) step{  
    [self grow];  
    return self;  
};
```

4.4 Species Class

In the natural ecosystem, every organism belongs to a certain species. The organisms of the same species share most common physical and behavior/trait characteristics, while they may be in different physical conditions or lifetime phases, and demonstrate minor variances in behaviors/traits.

In DOVE, the SpeciesObj class has all species-level information ranging from average death age (a trait of the organism) to anti extinction number(an operation parameter that affects the evolution of traits). These species-level variables can be initialized with the parameters of ANIMAL block (refer to Appendix B). The definition of SpeciesObj class is shown below.

```
@interface SpeciesObj:SwarmObject {  
    id  modelPtr; // pointer to the Model object to access the environment info  
    int idNum;    // species ID  
  
    // below is a long list of species-level variables,
```

```

int  eatRadius; // eating radius of the species

int  moveRadius; // moving radius of the species

int  initialAgeMean; // Mean value for dying age distribution

struct dietType * speciesDiet; // what kinds of preys or resource I can eat
...

// statistics variables, recording all the behavior or activities statistics of all
// organisms belonging to this species

int deadHistory;

int deadStep;

int deadHistoryAge;

};

```

The definition of struct dietType is below:

```

struct dietType {

    int resOrOrg;      // 0 if resource; 1 if organism

    int species;      // the ID of the species that I can eat

    int actions;      // how many actions can the species perform?

    float * chanceEat; // probability to eat the diet item when it is doing action n

    float refugeRate;  // the percentage of the available resource that organism can not
                      // eat at any cell.

    float eatPercent; // the percentage of the available resource that organism can eat
                      // after diet refuge mechanism.

    struct dietType * next; // name of the diet

};

```

In addition to setting or retrieving species properties, the functions in Species object are mainly statistic collection and export/import functions

- `-(id) getModelPtr { return modelPtr; }`
- `-(void) setModelPtr: (id) m { modelPtr = m; }`

Usage - get/set the pointer to the core Model objective to access the environment information

- `-(int) getSpecObjNum {return idNum;}`
- `-(void) setSpecObjNum: (int) id {idNum=id;}`

Usage - define/retrieve the species ID

- `-(int) getEatRadius {return eatRadius;}`
- `-(void) setEatRadius: (int) s {eatRadius=s;}`

Usage - get/set the radius that the species has access to for consumption.

- `-(int) getTotalLiveCount;`

Usage - get the number of organisms of this species that are alive

- `-(double) getSitFraction;`

Usage - get the statistic about the percentage of organisms alive taking sit action

- `-(double) getEatFraction;`

Usage - get the statistic about the percentage of organisms alive taking eat action

- `-(double) getSwitchFraction`

Usage - get the statistic about the percentage of organisms alive taking switch action

- `-(void) writeExportHeaderToFile: (FILE *) f;`

Usage - writes the column headers to file

- `-(void) exportSpeciesInfoToFile: (FILE *) f;`

Usage - writes the actual species related info to export file

Because each organism object embed a pointer to a certain species object, these species-level variables of a single SpeciesObj object can be shared by organisms belonging to the same species.

4.5 Organism Class

4.5.1 Class Definition

The Organism class is the core class of DOVE class hierarchy. The individual organisms are all instances of the Organism object. The Organism object is one of the largest and most complex objects in the program. The members of Organism class are shown as following:

```
@interface Organism: Agent {  
    // variables for current conditions of organism  
  
    int  live;           // 1=alive, 0=dead  
  
    int  resource;       // my internal resource level  
  
    int  lastAction;     // last action organism took  
  
    int  stepCount;      // current time step  
  
    int  eatOn;          // enable global predator to eat  
  
    int  age;            // current age of organism  
  
    int  maxAge;         // max age organism can be  
  
    float deathProb; //death probability caused by other reasons than aging or predation.  
  
    int  globalPredator; // current knowledge on predation risk  
  
    struct dietType * myDiet; // pointer to the embedded speciesDiet  
  
    SpeciesObj * mySpecies; // pointer to embedded species object
```

```

    Intelligence * myBrain;    // pointer to embedded Intelligence object

...

};

```

Besides the variables indicating an organism's status, the Organism class includes two important components: mySpeices object of SpeciesObje class, and myBrainobject of Intelligence. The former defines the common characteristics/traits of organisms, while the later specifies the decision-making component (brain) of the organism, which can evolve using a genetic algorithm. We will discuss the details of genetic algorithm in the next chapter.

The organism class includes functionality for most of the organisms' activities including moving, eating, reproducing and dieing. These are explained in details below.

4.5.2 Class Methods

4.5.2.1 Moving Activity

In the cells/grid of Dove world, each cell (x, y) holds the lists of organisms and resources at this location. An important feature of DOVE is that each habitat, weather there is 1 or 2, is a torus: organisms do not see the edges, but move from one edge to the opposite edge. Since Organism class is the child of Agent Class, it inherits the myGrid member from its parent. This variable points to the "torus" world that the organism lives. We will discuss the details of the myGrid object of DoveGrid2d class in sixth chapter.

An important species-level variable on moving activity is MoveRadius. This variable represents the radius (how many cells) the organisms can move in one time step. A directly diagonal cell is still considered one space away. MoveRadius is specified for each species.

Organisms movement is described by three parameters:

➤ -MoveByX:(int)xd Y:(int)yd;

Usage - Move the specified distance

➤ -MoveRandomUpToX:(int)xd Y:(int)yd;

Usage - Randomly move some distance up to the specified amount in any direction.

The actual amount of x, y, the distance to move from current position, are randomly picked between [-xd, xd] and between [-yd, yd] respectively.

➤ -MoveRandom;

Usage - Randomly move up to 1 cell in any direction.

4.5.2.2 Eating Activity

Similar to MoveRadius for moving activity, EatRadius defines the radius around which an organism can eat. If the radius is set to 0 it means an organism can only eat on its own cell. A directly diagonal cell is still considered one space away. For an organism that eats resources (like an herbivore), it eats a user defined percent of available resources on each cell within the radius. For animals that eat animals, there is a user-defined probability it will eat every organism within the EatRadius. Thus, if an organism MoveRadius is 4, and EatRadius is 2, it will move from 0 to 4 cells away from its present cell, and eat within a radius of 2, totally 25 cells (1+8+16).

When a natural organism forages optimally, it first observes its neighborhood and then chooses the location that it can find maximal foods for eating. After it moves to this best location, it then eats the preys/foods that are in its attacking range. Thus, the eating activity of a digital organism involves two functional operations: determining the cell to feed from, which may involve searching for the “best spot” with the highest potential

resource return, and then consuming resources once the cell is determined. I explain both operations below in detail.

- **Searching Functions:**

➤ `-(void) findBestX: (int*) tX Y: (int*) tY FoodThere: (float*) bestFood;`

Usage – find the best location to eat

An organism can search its neighborhood for best spot to move in the next time step. The user determines the probability that the organism can search for the “best spot”, versus random placement. The criterion is the amount of potential food at this spot. The food refers to eatable resources at that location for consumer, or is preys’ internal resources if the organism is a predator.

The implementation of this function is as following steps:

1. Find the extent of where the organism can move.

2. Allocate and fill the array of reachable cells to store the food values of them. This is a 2-d array that "flattens out" any torus effects so that we end up with a simple grid of values. Each cell will contain the amount of food available to me at that location. Note that it is possible for a single cell in the real grid to show up multiple times in this foodArray grid. For example, given a 10x10 world (torus) and a move radius of 4 and an eat radius of 3. The organism could move left 4 cells and then eat something 3 cells to left of its new location. That is the same cell as if the organism ate something 3 cells to the right of its current location without moving. It simplifies the code greatly to just let that happen. When we compute the foods value from each cell the organism can move to, we will eliminate those duplicates.

3. Search the range of cells the organism can get to and determine how much food the organism can get from each location (x, y) with the eat radius.

4. Now determine which spot was the best. If multiple spots have the same best value then the organism will randomly choose between them. This will often be choosing from just 1 possible value.

- **Real Eating Functions:**

- `-(double) resourceFrom: (id) prey`

Usage - How much resources do the organism gain by eating the specified prey item?

The resource amount depends on:

- 1) How much resource the prey item has.
- 2) What the prey item is doing.
- 3) The probability that the organism could catch it.

- `-(void) eatResource;`

Usage - This is the main eating routine. It eats both resources and organisms

The implementation of eatResource function follows the steps below:

1. Measure the area covered by organism's eat radius. It is possible that its eatRadius will allow it to reach the same cell multiple times (due to the torus shape of the world). That would be invalid, so set the limits of its reach to ensure that it only eat from each cell once.

- 1) If eatRadius is small enough, just set limits to the eatRadius
- 2) if eatRadius is too big, set limits to the world size.

Note that this test is independent for X and Y.

2. Do the eating.

1) If the organism is a consumer, it eats a user defined fraction of all resources in its eating area.

2) If the organism is predator, it eats the preys in its eating area that it captures, as a function of capture probabilities defined by the user.

4.5.2.3 Step Function

Step function is the place that organism behavior is executed. The notion of a "step" method is a nice simplification for basic simulations. As mentioned before, step function will be scheduled to execute every time step.

➤ -step;

Usage - do anything that all Organisms must do each time step. Here is how it works.

1. Update time related variables, e.g. age, stepCounter, current resource level. If the organism is going to die by aging or the reasons other than predation, such as internal resource level reaching zero, the organism mark itself dead and exit the function.

2. If the organism has enough resources to reproduce, add itself to the reproduction list for its species.

3. Find the best location for the organism to move to. How much food would be available to the organism if it moved there?

4. Determine a predation risk. An organism knows about predators in "the whole world", regardless of where they are (top/bottom, x, y), and that is the information used to determine predation risk.

There are two ways to determine predation risk

1) Every organism computes the value in its turn. This will likely be the preferred procedure when the predation is based on characteristics of individuals such as length, spines, speed, etc.

2) Compute it once for all members of a species and use that value for each member of the species. This works ok when predation is "species-based", such as any Fox will eat any Rabbit, regardless of size, etc.

Method 2 is much faster (computation is done once per species versus once per organism). It does, however, have a scheduling issue in that by the time a particular organism executes its step method the number of predators may have changed (they may have died, reproduced, etc.) We are ignoring that issue and assuming that the organisms get their information at the beginning of the time step.

5. Decide what to do this step

0 = sit still ("inactive" state: do nothing at all)

1 = feed = move and eat

2 = switch habitats ($0 \leftarrow 1$ or $0 \rightarrow 1$)

As I discussed in Chapter 3, global predators is the observable and system-level predation risk, which research can manipulate in the simulation to represent the predation pressure for the whole ecological community and examine the behavioral reactions of organisms. Global predators exist so that we can have a predator species with dynamics that are pressed, rather than responding to the dynamics of their prey and associated feedbacks. Global predator is essentially a variable predation rate. Because global predators have distinct properties, we have a specific step function for it.

➤ `-stepGlobalPredator;`

Usage - Do anything that global predators do each time step.

Things done for normal organisms, but not for global predators, are:

- 1) increment and test age (no death by old age)
- 2) decrement and test resource level (no starvation)
- 3) reproduction (no hanky-panky)
- 4) find best location (the whole world is good)
- 5) calculate a risk to me from predators (they are invincible)
- 6) decide an action (they always just sit and eat)

In this chapter, I explained how we modeled the resources, organisms and their activities in DOVE. By separating the species-level properties from the variable individuals' states, we created a flexible hierarchy that facilitates the modularization of DOVE. Moreover, we demonstrate that how to design organisms' activities within a “no edge” world, which is an important feature of DOVE and will be discussed in DoveGrid2D class of Chapter 6.

5. Genetic Algorithm for Adaptation

5.1 Principle

In this version of DOVE, the evolution of the organisms is guided by the genetic algorithm. The interaction rules between organisms and the internal resource requirement for reproduction favor those who can respond adaptively to the changes (e.g. predation risk level) in the environment. This allows them to evolve towards an optimal solution for the world. The genetic algorithm in DOVE differs from the traditional approaches in that there is no user-specified objective function that determines which agents reproduce. Instead, we emulated the natural evolution by incorporating interaction rules among species and resource requirements for reproduction.

In DOVE, the action taken by an organism that defines its behavior depends on organism's perception of its' internal state and the state of the environment. Currently, an organism has five state variables that the user can define.

1. Predator Risk Level: This measures how high the risk is that organism is captured by predator, which is a function of the density level of predators in the system.

2. Hunger Level: this measures how close the organism is to starving or reproducing. It represents the number of different internal resource levels the organism is aware of, e.g. if user defines three levels of hungry and the organism's internal resource level below the lowest level, the organism is very hungry, which may lead to a feeding action. In contrast, if an organism's internal resource level reach the highest level, which indicates that the organism is ready to sexually reproduce.

3. Resource Level: This measures how high the resource is compared to HighAvailResLevel, the highest resource on an absolute scale that user defines.

4. Relative Food Level: This measures how high the available food is relative to what the organism has observed recently.

5. Habitat: This describes which habitat the organism occupies.

Users can also define the number of possible actions, such as feed, sit, switch, that an organism can take. Normally, users define up to three possible actions.

Users can initialize these states with any arbitrary number. Normally, Relative food level is set either higher or lower level, and habitat has as many states as the number of habitats.

The “action matrix” of combination of above five parameters determines the action taken by an individual. For each state that the individual can perceive, a probability distribution determines its action in that state.

Environmental state:

Habitat	1	1	1	2	...
Pred. risk level	1	2	3	1	...

Action probability “gene” for two individuals:

Sit	Feed	Switch	0.6	0.3	0.1	0.5	0.4	0.1	0.4	0.5	0.1	0.7	0.2	0.1	...
Sit	Feed	Switch	0.66	0.24	0.1	0.3	0.5	0.2	0.4	0.4	0.2	0.7	0.2	0.1	...

Figure 4. Examples of Gene sequence

In the example shown in Fig.4, we suppose the state variables other than predation risk and habitat are constant values. If the individual is in Habitat 1, and perceives the predation risk as intermediate, then the probability that it will sit is 50%, feed is 40% and switch habitats is 10%. In contrast, a second individual in the same population has a 30%, 60% and 20% percent probability of these actions.

If two individuals from the same species acquire sufficient resources, they reproduce. The organisms in the system have a simplified form of sexual reproduction. Whenever an organism has sufficient internal resources to reproduce, it is added to the reproduction list. Once each time step the program goes through this list and if there are two organisms from the same species on this list, it will take them off the list, lower their resources, and create a new organism based on their parents' parameters. Offspring genotypes result from the mutation and recombination of parental genes, and are thus able to “evolve”.

Parent 1	0.6	0.3	0.1	0.5	0.4	0.1	0.4	0.5	0.1	0.7	0.2	0.1	...
Parent 2	0.66	0.24	0.1	0.3	0.5	0.2	0.4	0.4	0.2	0.7	0.2	0.1	...
Offspring	0.66	0.24	0.2	0.3	0.5	0.1	0.4	0.5	0.1	0.7	0.2	0.1	...

Mutation ↑ ↑ Crossover point

Figure 5. Examples of genetic Evolution

During sexual reproduction, there is currently one crossover point and a certain mutation rate. In the example of Fig.5 shown, the genes before crossover point are from parent 2, the rest genes are from parent 1, and there is also a mutation at this genome. Notice that the gene operations such as mutation may generate a new probability sequence, in which the sum of the probabilities of possible actions for a “state” may be not equal to 1 anymore, e.g. in Figure 6, after the crossover and mutation operations, in offspring' gene, the probabilities of feeding, sitting, and switching for the first “state” is 0.66, 0.24 and 0.2. In this situation, we have to normalize the probabilities of actions and make the sum of new probabilities to 1. For the case above, the normalized probabilities of actions for first “state” are $0.66/(0.66+0.24+0.2)$, $0.24/(0.66+0.24+0.2)$, $0.2/(0.66+0.24+0.2)$.

One of the goals of the system is to evolve an intelligent population before they go extinct. For this reason, DOVE has a feature called anti-extinction. DOVE monitors the density of a specific species, and if it is less than the anti extinction number, then each creature is cloned a user-defined number of times. This provides more organisms and allows the creatures another chance of evolving. During cloning (for anti extinction) there is no gene mixture or mutation.

Note that there is no prescribed objective or fitness function on system level. Organisms that have strategies to gain enough energy to reproduce before they are killed pass on genes and reproduce. Thus, there is selection for strategies that are the most fit in the dynamic landscape.

5.2 Implementation

To make changes easier in the future, the intelligence for each organism is separate from the actual organism. For this purpose, we have the Intelligence class and the Gene class. The Gene class implements the genetic algorithm, while the Intelligence class provides an interface between Gene class and Organism class.

5.2.1 Intelligence Class

The definition of Intelligence class is low:

```
@interface Intelligence: SwarmObject {  
  
    id myOrganism;           // pointer to organism object  
  
    id myGenes;              // pointer to decision-making brain  
  
    id <List> foodMemory;  
  
    int  foodMemSize, foodMemFirst, foodMemLast, foodMemCount;  
  
    float * foodMemArray;
```

```
}
```

For embedded members, it simply includes a pointer to the organism object and a pointer to the gene object, and some variables about food memory.

The most important function of intelligence class is `getAction`, which is defined below:

```
➤ -(int) getAction: (int) predRisk  internal: (float) internalRL  
    goodFood: (int) foodPotential  habitat: (int) myHab  
    reproductionLevel: (float) reproLevel  absResource: (float) absRL  
    maxPredRisk:(int)mrisk    maxResLevel:(float)mres
```

Usage – inquire about the action to taken. Here is how it works:

1. Compare my current location's food potential to the food potential of my last N locations (averaged).
2. Compute an actionIRL (IRL = Internal Resource Level) that is a truncated integer multiple of the reproduction resource level.
3. Similarly, compute a predation risk level index
4. Calculate an index based on the available resources
5. Inquire gene object what action to take

5.2.2 Gene Class

The Gene object is the actual brain of the organism. The definition of Gene class is shown as following:

```
@interface Gene:SwarmObject {  
    id myOrganism;  
    id <NormalDist> normDist;
```

```

int numPredRisks, numHungryLevels, numFoodLevels, numHabitats, numActions;

int numStates, numResourceLevels;

float * geneSeq;      // the full gene sequence as an array of float

float * actionArray; // sized at creation time for use in deciding action at each
timestep

}

```

Gene class also includes a number functions for gene sequence manipulation:

```

➤ -(int) getStateOffsetFromRisk: (int) risk Hungry: (int) hunger Food: (int) foodQ
    Habitat: (int) hab Resource: (int) res;

```

Usage - Determine the correct section of the gene string, based on the state

```

➤ -(void) mixGenes: (Organism *) parent1 parent: (Organism *) parent2
    withMean: (float) mutationMean withVariance: (float) mutationVariance
    withChance: (float) mutationChance

```

Usage – mix the parents’ gene sequence and make possible mutation.

```

➤ -(int) askAction: (int) predRisk hungry: (int) resLev
    foodLevel: (int) curFoodLevel
    habitat: (int) myHab resourceLevel: (int) absResLev

```

Usage - inquire about the action to take. This is the core function of the class.

1. Determine the correct section of the gene string, based on the state.
2. Build an array that has progressively larger values so we can eventually determine a course of action. For example, suppose that we have three actions (sit, feed, switch). And suppose that the probability for each of these three actions (for the current state, whatever that may be) is:

`sit = 0.20`

`feed = 0.50`

`switch = 0.30`

We get those three values from the gene string. Now, we need to have some mechanism so that 20% of the time we choose to sit, 50% of the time we feed, and 30% of the time we switch. What we do is build this `actionArray` with three values:

`0.00 0.20 0.70 (0.20 + 0.50)`

Then we will generate a random number from 0 to 1. If it is in the range `[0.0, 0.20]` then we will sit. If it is in the range `(0.20, 0.70]`, we will move and if it is in the range `(0.70, 1.00]` we switch. Note: there is a slight added chance of sitting due to the way the test is constructed. A random number value of exactly zero is one extra value for sitting. That is an extremely small difference, though, since we are dealing with floats.

In this chapter, I explained how to model the organisms' behaviors through probabilistic method. Moreover, I present the design and implementation of the genetic evolution with genetic algorithm, from both theory and implementation perspectives. Through the applications of these computational methodologies, plus the interactions rules among organisms and resource requirement for sexual reproduction, the digital organisms in DOVE can evolve towards optimal solutions without users-specified objective functions.

6. Modeling of Ecosystem Environment

6.1 System Architecture

DOVE is built in the Swarm platform with objective C. It was programmed following object-oriented principles with the emphasis on scalability and extensibility. In Figure 6, I illustrate the system architecture of DOVE.

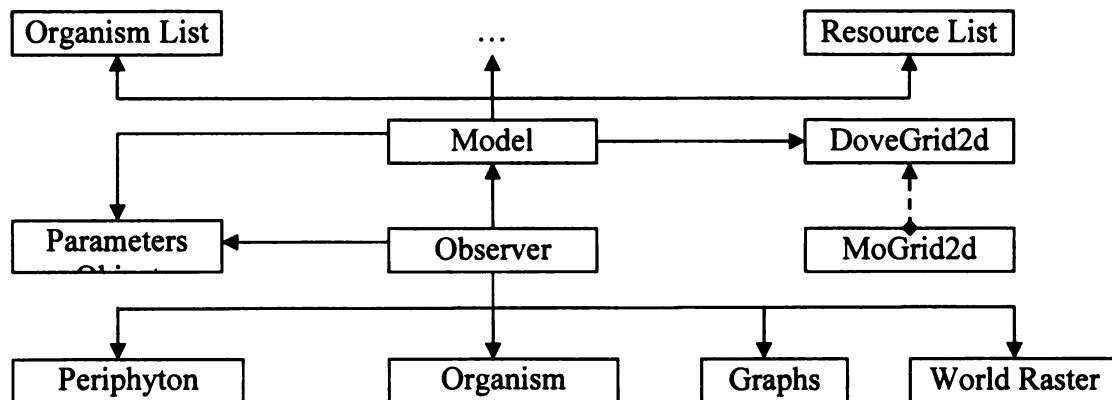


Figure. 6 Dove Model Architecture

The real core of DOVE begins in a class called Model. Model controls the schedule and holds lists of all species, resources and individual organisms. Model also has the functions for all aspects of global operations. These includes importing and exporting populations, managing global predator densities, maximum resource levels, reproduction, anti-extinction, reporting and communication with the graphs. Note that Model is analogous to the setup and implementation of an experiment by an investigator doing experimental ecology.

Model also holds a pointer to an instance of the DoveGrid2d object. DoveGrid2d takes care of coordinating the actual grid on which everything lives. Moreover, DoveGrid2d helps to manage all interactions between organisms and resources. This includes finding

the best location for an organism to go, eating resources and other organisms in the neighborhood, etc.

The Parameters object takes user-defined simulation parameter profiles of different resources, organisms, habitats and food webs structure and import/export information for the initialization and running of the program.

The Observer object is a standard swarm object, which deals with all Graph User Interface (GUI) details. It retrieves information from Model and Parameters objects, and displays world, organisms, periphyton, and graphs for analysis.

Below, I explain the DoveGrid class and Model class in details.

6.2 DoveGrid Class

DoveGrid class holds the list of cells that both resources and organisms inhabit. An important feature on DOVE is that multiple organism and resources can occupy the same cell. The class definition is below:

```
@interface DoveGrid2d : MoGrid2d {  
    id myModel;          // pointer to model object  
    id <List> cellList;   // list of everything in a particular cell  
}
```

There are times when the world object of DoveGrid2d needs to be able to call methods of the Model object. This pointer of myModel gives them that ability.

In the MoGrid2d class, the super class of DoveGrid2d, each cell has an associated list that contains all of the objects residing at the cell. In DOVE, we are extending that definition. The list of the cell no longer has the actual occupants of the cell, but rather contains two lists.

List 0 = resources

List 1 = organisms

Furthermore, lists 0 and 1 are themselves lists of lists. That is, list 0 contains a separate list for each type of resource and list 1 contains a separated list for each species of organism.

There are some important functions in DoveGrid2d:

➤ -(void) moveAgent: (id) agnt toX: (int) newX toY: (int) newY

Usage – move an agent to a new location.

Because each cell has two lists: resourceLists for resource objects, orgLists for organisms objects, we have to figure out if the agent is an organism. If it is, we check which species (spNum) it is. To actually move an organism to a new location, we have to first remove the organism from the appropriate species list of its current location/cell, then add this organism into the appropriate species list at the new X,Y values.

➤ -(double) attemptEat: (id) prey by: (id) predator

Usage - Attempt to eat an organism. “Attempt” refers to the fact that there is only a probability that a predation act will be successful. If successful:

1) Mark the organism as dead, but do not do anything else to the organism. We will handle the actual "killing off" of the organism later.

2) The amount of resources attained by the attacking individual is equal to the internal resource of the attacked organism. This value will be the return value of the function.

➤ -(double) eatOrganism: pred whichSpec: (int) specNum atX: (int) x atY: (int) y

Usage – eat all organisms of a particular species in the cell (x, y). It actually calls `attempEat` function for all of the organisms of a particular species at (X,Y). Before agent eat, it must ensure that

- 1) the prey must be alive

- 2) cannot eat itself

➤ `-(int) calculateRiskForOrganism: (id) anOrganism Global: (int) gR`

Usage - Calculate a perceived risk of being eaten by a predator.

The risk is based on the value of `gR(Global)` switch. If `gR` is true (non-zero), it assume that the prey organism somehow has knowledge of any predators that exist anywhere. All we need to check for is if any global predators exist in the world.

If `gR(Global)` is false (zero), the prey organism will only consider risk from local predators, which refer to the predators other than global predators, within range to eat it. This is based only on:

- 1) prey's current location (x,y)

- 2) local predator's location (x,y)

- 3) local predator's `EatRadius`

The return value is the number of predators that are considered to pose a threat to the prey organism. In the future, we will likely want to consider potential movement as well, since that changes the danger zone around a predator.

➤ `-(int) calculateRiskForSpecies: (id) specObj Habitat: (int) ha`

Usage - Calculate (for a species) a perceived risk of being eaten by a predator.

This is very similar to the `calculateRiskForOrganism` routine above. The difference is that we are just calculating a risk for an entire Species, therefore we have no location

information to use. That means we are always in the "Global=true" mode for the organism risk routine. The return value is the number of predators that are considered to pose a threat to the prey species. We assume risk is the same for each individual of the species, and we ignore variation in the position of local predators.

6.3 Model Class

The Model class is the core module of the ecosystem. The definition of this class is below:

```
@interface Model: Swarm {  
  
    id parameters;           // pointer to parameters object  
  
    int worldSizeX, worldSizeY; // how big the world is  
  
    int yDivide;             // the border that divides two habitats  
  
    int numberSpecies, numberResourceTypes;  
  
    int antiExtinctionSteps;  // the time step threshold of anti-extinction function  
  
    int topBottomMultiplier;  
  
    ...  
  
    int doGlobalPredation;    // 1=yes; 0=no  
  
    int globalPredationSteps; // number of steps left before switching states  
  
    int maxGlobalPredators;  
  
    int activeGlobalPredators; // 0..max. 0 indicates global predation is OFF  
  
    ...  
  
    id modelActions;         // scheduling data structures  
  
    id modelSchedule;  
  
    DoveGrid2d * worldObj;   // This is the object that holds the world
```

```

id <List> orgList, resourceList, reproductionList;

...

}

```

Below I explain several important functions about scheduling and organism activities.

➤ -buildObjects

Usage – Model object builds the embedded objects in the following order:

1. Initial lines of report file
2. Create the various lists of resources and organisms.
3. Create the list shuffler object using for scheduling the activities
4. Get the parameters and build the world using those parameters
5. Set up the global predator
6. Read in organisms' and resources' properties for parameters in files and command line, then build the populations.
7. create the report files for resources and organisms.

➤ -(void) steps

Usage - This is the largest function that gets called every time and actually runs the program. It will first allow the resources (periphyton) to grow, then it will ask all the organisms what they want to do. It will then take all the predator prey relations into a list and randomize the list. It will then execute them and hopefully organisms will eat and be eaten.

Here are the details about how it works.

1. Check whether there are organism from the same species are ready to reproduce.

If it is, make it happen.

2. Run the anti-extinction code if needed.
3. Update the statistic for average resource and organisms.
4. Calculate the predation risk for each species of organism. Note that by calculating it once (at the beginning of the time step) it may be somewhat out-of-date by the time a particular organism executes its own step method. Thus, we may wish to revisit this at some point, especially if scheduling becomes a more important issue.
5. Go through all organisms (in random order) and step with them.
6. Compute some statistics on the regular organisms, then let any active global predators do their thing.
7. Check the status of each organism. If it has died at any point in this time step up until now, add it to the temporary Dead List. Note that when a predator eats an organism, the only thing we do to the organism at that time is change its live status. In this way we ensure that we do not try to kill it more than once. Predators also check that live status before attempting to eat an organism.
8. Go through the temporary dead list and kill all the creatures.
9. Update the global predator status. Global Predator state at each time step is from 0 to maxGlobalPredators active. When we turn on a group of global predators, they will stay on for some specific number of time steps. Similar thing happens when we turn them all off. Once that number of time steps has expired we re-compute the number of global predators to turn on. There are 2 possibilities:
 - 1) $\text{globalPredationTimeOffMean} > 0$

This means that we will have a consistent off period between each on period.

So the number of predators will be something like:

4, 0, 2, 0, 5, 0, 1, 0, 3, 0, 3, 0, 1, 0, etc

2) `globalPredationTimeOffMean = 0`

This means that we never force a zero. We will always use the `globalPredationTimeOnMean` value, and we simply change the number of active global predators each turn. The progression would be something like:

1, 3, 6, 0, 3, 2, 3, 1, 0, 5, etc

➤ `-(void) reproduceWithMom: (Organism *) mom Dad: (Organism *) dad`

Usage - create a new baby object

1. figure out where to be born (random location in mom's habitat) move it there (ask the grid to move it)

3. ask the baby to be born from the given mom and dad.

➤ `-(void) buildPopulation`

Usage – build the populations of various resources and organisms

1. Build the resource types, then build individual resource agents at each cell.

2. If specified, create and initialize the global predators. They are created as ordinary organisms for simplicity. They will, however, be treated a little bit differently by the model. They are not added to all of the model lists that regular organisms are. Note that we will always create a global predator species object and that species number 0 will always be reserved for the global predator. This is true even if the user specifies that there are no global predators. It simplifies life for coding the rest of the model if we can always start at species number 1 for all other organism species.

3. build the "normal" organism species objects and set up the diet for each species.

Note that all organisms of a particular species have the same diet.

4. Add organisms of each species and place them randomly through the world.

In this chapter, I presented how DOVE incorporates various resources and organisms, and organizes their interactions. I explained two main classes in the ecosystem: DoveGrid2D class and Model class, both of which manage the interactions among organisms and resources, which finally lead to the emergence of the global pattern behaviors of the ecosystem.

7. Experiments and Analysis

7.1 Overview

We have conducted some preliminary experiments to study phenotypic plasticity and trait-mediated interactions. In order to validate our approaches in DOVE, including agent-based modeling and genetic algorithm, in the designing of the DOVE, we performed a preliminary experiment, in which we examine a simple system with a resource species, a consumer species that could evolve, and a global predator. . Using this system, we can examine the behavioral responses of a consumer species to changes in predator density and analyze these results caused by phenotypic plasticity.

Before the experiments, we have a hypothesis on the phenotypic plasticity. We believe that phenotypic plasticity will evolve in species faced with trade-offs between predation risk and energy gain in a dynamic environment. Therefore, the species evolving by phenotypic plasticity will have a competitive advantage over the species that cannot modify phenotype.

7.2 Experiments

We will examine the behavior of the consumer with two scenarios, in which the consumer does, and does not, have a gene structure that allows the potential for phenotypic plasticity to evolve. In both cases all other parameters are identical. The resource is initialized and distributed evenly in the world, which is evenly divided into two habitats. We also set up the habitat attributions in the parameter files. In our setting,

the top habitat is fertile but more dangerous for the prey organisms, while the bottom habitat where resources are growing relatively slow provides a safer environment to the prey organisms evading the threats from predators. The presentation or absention of the global predator is scheduled by a random generator, which thus brings in the change of the predation risk that the prey organisms can observe. The detailed parameter setting of the following experiments are in Appendix E.

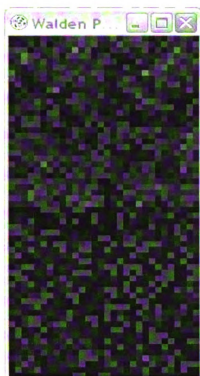
7.2.1 Phenotypic plasticity and Behaviors Evolution

First consider the case in which the behavioral reactions in the prey organisms have the propensity to develop phenotypic plasticity. This is achieved by allowing the consumer (i.e. prey) species to perceive the presence and absence of the predators, and act in different ways respectively. Thus, the gene - “state matrix” of the prey species defines two different probabilities sets of actions for different predation risk levels that the organisms of prey species observe. If the ability to behave differently when the predator is present and absent evolves, this would represent phenotypically plastic behavior. Similar to natural organisms, the prey organisms in this experiment can evolve through genetic evolution.

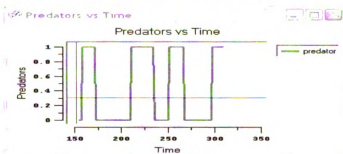
In this scenario, we have one global predator and twenty prey organisms initially, and the resource in both habitats. Figure 7 illustrates the results at two distinct time periods early (300th time step), and later(1200th time step), in the simulation. At both times, an three figures show a “snapshot” of the world, Global predator presence vs. Time graph, and Prey activities vs. Time graph.

In the former graph, when global predator appears, the blue line jumps to 1; while global predator is absent the blue line goes down to 0. In the Prey activities vs. Time graph, we can find four lines: the red, blue and yellow lines represent what percent of

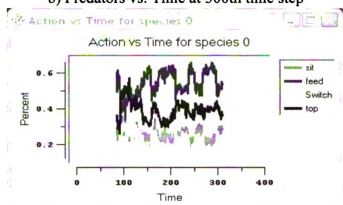
organisms in this species are feeding, sitting, and switching respectively. The green line indicates what percent of organisms in this species inhabit in the top habitat.



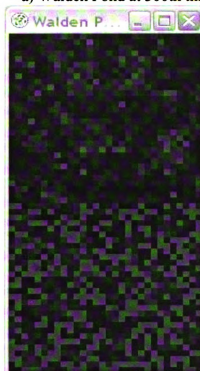
a) Walden Pond at 300th time step



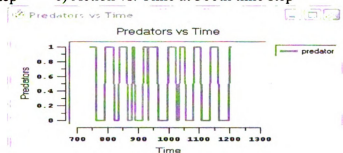
b) Predators vs. Time at 300th time step



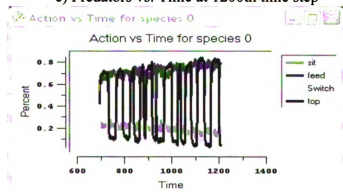
c) Action vs. Time at 300th time step



d) Walden Pond at 1200th time step



e) Predators vs. Time at 1200th time step



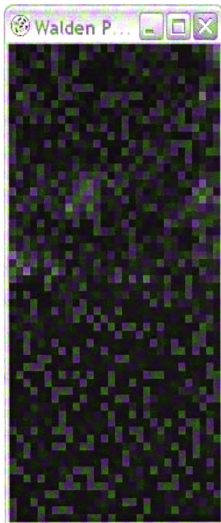
f) Action vs. Time at 1200th time step

Figure 7 Behavioral evolution

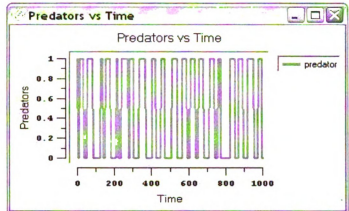
In this experiment, the population of the prey does exhibit phenotypic plasticity through the genetic evolution. Comparing the screen shot (Fig.6. a) and graphs (Fig.6. b and Fig.6. c) at 300th time step with the screen shot (Fig.6. d) and graphs (Fig.6. e and Fig.6. f) at 1200th time step, we can see a clear gene-select process, in which a larger percent of the organisms chooses the lower and safer habitat when the predator presents, and more organisms are back to the fertile habitat to feed while predator is absent. Through the activity history shown in prey activities graphs (Fig.6. c and Fig.6. f), we can see that the evolving of organisms' behaviors gradually selects for the organisms that respond more quickly to the change of the predation risk. This trend indicates that the organisms that display a strong phenotypic plasticity and response to the change of the predation risk more promptly have a competitive advantage over those organisms that do not modify phenotype or have a weak phenotypic plasticity for the change.

7.2.2 Without Phenotypic plasticity

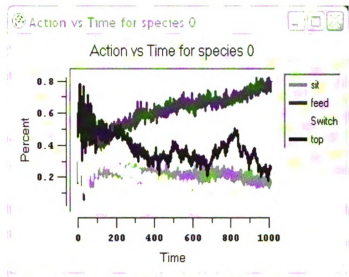
In order to examine the role of phenotypic plasticity, I next conduct another experiment, in which all parameters are the same except that there is no potential for phenotypic plasticity among prey organisms. We “turn off” phenotypic plasticity of prey by setting the number of the predation risk level to 1. Because the gene - “state matrix” of the prey species defines only a single probabilities set of actions for different predation risks (predator is present or absent), the prey organisms lack the ability of phenotypic plasticity to behave differently when the predator is present and absent, even though they can evolve through genetic evolution. The results of the experiment without phenotypic plasticity are shown in Figure 8.



a) Walden pond at 1000th time step



b) Predator vs. Time at 1000th time step



c) Action vs. Time at 1000th time step

Figure 8. Experiment without phenotypic plasticity

Initially, the probability of being on the top is 50%, and the probability of doing one of the 3 actions is 33% (Fig. 8c). But over time, the consumer evolve a strategy the optimizes their behavior given a changing predator density: only in this case, it is a constant behavior. After 1000 steps, the population of the preys without exhibit phenotypic plasticity does not display a behavior adaptation with the fluctuation of the predation risk, because they can not perceive the presence of predators. The activities of the prey population that were randomly initialized evolve under the predation pressure

posted by predator, even though they can not sense the change of the predation risk. Because predators have different probabilities to eat preys taking different actions and feeding actions make preys more vulnerable, not all organisms are doing feeding action even though they can not sense the danger from predators. Also because predators have a much better chance to catch preys in the top habitat, preys that inhabit in the top habitat are generally fewer than those in the bottom habitat.

As we demonstrated above, these experiments demonstrate that digital organisms can evolve phenotypic plasticity with the DOVE platform. Moreover, the success of the experiments validates the interdisciplinary approaches, e.g. agent-based modeling, we adopted in the designing of the DOVE. Next, the ecologists can use this platform to perform experiments examining how phenotypic plasticity will effect population and community structure and dynamics.

8. Conclusion and Future Work

8.1 Conclusion

This thesis discusses the design and implementation of a computational ecosystem - DOVE, to study phenotypic plasticity and trait-mediated interactions among digital organisms. It also covers some experiments on the dynamics of food web, and reaches several hypotheses about the impacts of phenotypic plasticity in feed webs. Later it provides a user manual and parameter profiles explaining parameter settings to construct dynamic and complex food web and conduct large-scale simulations using Drone tool.

The main contributions of my research and this thesis are the following:

- 1) Aid by Dr. Peacor's guide and advice, I worked with other researchers and helped implement a computational system that addressed the shortcomings of traditional approaches. Because traditional modeling techniques are insufficient to address the complexities associated with phenotypic plasticity and trait-mediated interactions in natural systems, our researches in DOVE propose a bottom-up approach to understand the dynamics of ecological communities and uncover complex and functional relationships associated with phenotypic plasticity.
- 2) In DOVE, we exemplify the capabilities of computational methodologies and approaches in interdisciplinary research. We adopted several inter-related computational methodologies and approaches, including agent-based modeling, genetic algorithm, to emulate individuals' behaviors and dynamic environments. Thus, ecologists can manipulate various levels of dynamic complexities of

ecosystems with great flexibility, and study phenotypic plasticity and its consequences efficiently.

8.2 Future Work

In current version of DOVE, we assume that the genetic evolution happens in the sexual reproduction and the behaviors of an organism do not change during its lifetime. This is true for many low-level consumer species. However, if we consider the complex ecosystems having some highly intelligent species, this assumption is probably not applicable. Therefore, I suggest incorporating other learning and behavioral approaches to address this issue. For example, we can use Bayesian Network to emulate the learning process of natural organism, so that digital organisms can learn from their experiences and become smarter by altering their behaviors accordingly. To incorporate those learning and behavioral systems, we can build different interfaces in Intelligence class to various kinds of decision-making “brains”. Such “brains” can either be hardwired in the codes, or be linked as external components built in coding-free modeling tools, such as Bayesian Network tools from Netica and Microsoft.

In dove, we apply the methodologies from computer science to propose a new research approach in ecology. On the other hand, the research in biology or ecology also presents the challenges and opportunities for computer scientists in many subjects, e.g. the study of natural adaptation leading to genetic algorithm, and neural network hinted by biology research. In DOVE, we present an interesting fact to the genetic algorithm community: how to explain that the emulated implicit fitness mechanisms allow organisms to evolve towards an optimal solution without specifying an explicit objective function on system level. Traditionally, researchers employ the explicit objective function

for the optimization purposes with definite goals (Goldberg, 1989, etc.). Some researchers tried to model each predator-prey relationship into a static objective function so that multi-objective functions/optimizations may be applied to the ecosystem scenarios (Laumanns M., Rudolph G., Schwefel H. P., 1999). However, for ecological systems that we modeled in DOVE, we do not have prior knowledge about global preferences to individuals' behaviors before a simulation starts. Actually, all global patterns or behaviors of ecological system emerge from relatively simple interactions among resources and organisms. Therefore, we adopt a more general and implicit approach by emulating the natural ecosystem and setting up some implicit fitness mechanisms, e.g., the interactions rules among species and resource requirement for sexual reproduction. Our preliminary experiments confirmed our hypothesis that under these implicit fitness mechanisms, organisms can evolve towards optimal solutions without the users having to specify an explicit objective function. From this perspective, I believe that emulating implicit and characteristic rules, not formularizing global objective/fitness function, is a more natural and appropriate approach to the research of complex systems in many disciplines.

Appendix A: How to Run DOVE

DOVE is built in SWARM development environment, and can run in Linux, or in Window OS through Cygwin package. Following the SWARM program standard, DOVE has all commands and parameters of SWARM programs. Among these standard command line parameters, the most useful is:

`-b` or `-batch`

Run in batch model

In batch model, Instead of putting up fancy graphics, batch DOVE read data from files to control the model and write the data out to other files for analysis.

An important feature of DOVE is that we use UM-ExpTools4 objects to make it easier to add and process run time parameters, to get random variants in replicable ways, all of which will make it easy to use Drone package. To use the parameter options from UM-ExpTools4, users need add `-D` with the sub-parameters of global and resource/species parameter profiles (we explained parameter setting later in this appendix).

`-D, --drone-param=PARAM`

Parameter from drone.

There are two classes of parameters:

1. Global parameters: is related to the global setting, such as world dimension, habitats, import/export.
2. Resource and species parameters: is related to the specific parameters for different species and resources.

These are read in from two different files, and (most of them) can be set on the run command, to override defaults or values read from files.

1. Global parameters

You can see them by running:

```
./dove -Dh
```

These include parameters inherited from the RParameters object in UM-ExpTools.

The **bold abbreviation** of parameters can be used with **-D** to override the default value in command line. We will explain the details of global and resource/species parameters in Appendix B.

reportFileName (**RFN**) = report

reportFileNameSuffix (**RN**) =

reportFrequency (**RF**) = 1

outputDirName (**OD**) = .

stopT (**T**) = 4

seed (**S**) = 0

inputFileName (**IF**) = rparameters.in

rDebug (**D**) = 0

And these include parameters specific to Dove, but which are "global", i.e., not related to particular Resources or Species.

worldXSize (**X**) = 40

worldYSize (**Y**) = 40

yDivider (**yD**) = 10

numResourceTypes (**R**) = 1

homogeneousResource (**hR**) = 0

antiExtinctionSteps (**aES**) = 20000

topToBottomMultiplier (**tTBM**) = 3

maxGlobalPredators (**mGP**) = 1

parmFileName (**pFN**) =

importFileName (**iFN**) = import.pop

exportFileName (**eFN**) = export.pop

importOrganismCount (**iOC**) = 50

exportPopAtEndFileName (**ePAEFN**) =

All the global parameters can be set in the input file (via the -DIF= parameter), and/or on the run-time command. For example

```
./dove -b -DIF=parameters0.in -DT=50 -DyDivider=20
```

tells dove to read parameters from the file parameters0.in but then overrides the T (stopT) and yDivider values as indicated. (The -b tells it to run in batch mode.)

Note all the global parameters are written to the start of the report file, in a format that can be used as input, for example, suppose you did a run and had a report file named "report". You can re-run a particular run including the seed via

```
./dove -b -DIF=report -DRN=rerun0
```

This will produce a report file named report.rerun0, (If the run is producing species specific output in the species files, it will also append the RN (run number) to those file names, too.)

2. Resource and Species (Organism) parameters.

These are parameters specific to particular Resources or Species, as specified in the RESOURCE and ANIMAL blocks.

There is a global parameter that tells dove the name of the file that contains these:

parmFileName (pFN) =

So for example,

```
./dove -b -DT=19 -DpFN=parfile0.txt
```

tells dove to use defaults for global parameters, except stopT (T) = 19, and to read Resource/Species parameters from the parfile0.txt.

There is also a parameter which can be used to override parameters in specific Resource or Species objects. This is for use on command lines, e.g., for Drone to use,

overRide (OR) =

For example:

```
./dove -b -DT=19 -DpFN=parfile0.txt -DoverRide = Species.Bluegill.MoveRadius = 2
```

tells dove to run in batch mode, with a stopT of 19 steps, using default global parameters, but reading in the Resource and Species parameters from the file parfile0.txt, but overriding the value of MoveRadius to set it to 2, for the Bluegill species.

Appendix B: Parameters Setting

1. Global Parameters

- worldXSize: This reads in an integer that defines the size of the X dimension for the world
- worldYSize: This reads in an integer that defines the size of the Y dimension for the world
- yDivider: This reads in an integer that defines where the top habitat ends and the bottom begins. If the goal is to run the system with one habitat set the yDivide to one higher than the worldYSize.

- **numResourceTypes**: This reads in the number of resource types defined in Resource/Species parameter files. DOVE allows multiple types of resources to grow in the same cell.

- **antiExtinctionSteps** This reads in an integer which defines the number of steps before the anti extinction is turned off. This is used to allow populations to evolve but then seeing if they will eventually go extinct. Currently all species will have anti extinction turn off at the same time.

- **topBottomMultiplier**: This represents how much more likely an organism is to be eaten on the top habitat versus the bottom habitat.

- **maxGlobalPredators**: This represents the maximum number of global predators in the world.

- **globalPredationTimeOnMean** This represents the average time before the number of global predators change.

- **globalPredationTimeOnVar** This represents the variability in the time before the global predators change.

- **globalPredationTimeOffMean** This represents the average time that the global predator is away after a global predator time.

- **globalPredationTimeOffVar** This represents the volatility in the time that the global predator is away after a global predator time.

- **globalPredatorDietNames** This is a list of all the resources and organism species that the global Predator eats.

- **globalPredatorDietProbs** This represent the probability of the global predator eating the organism and resource species listed in the globalPredatorDietNames attribute. The

probabilities represent the chance of the global predator eating prey based on the different actions the prey is doing. If the diet entry is a resource there will only be one number, if the diet entry is an organism there will be between two and three percents.

- **globalPredatorTraceOn:** This indicates whether DOVE will trace the global predator activities and put the results in file for analysis purpose.

- **parmFileName:** This is a string variable that holds the default name of the Resource/Species parameters file.

- **importFileName:** This is a string variable that holds the default name of the import file.

- **exportFileName:** This is a string variable that holds the default name of the exportfile.

- **importOrganismCount:** This reads in the number of the organisms that will be imported in the initial phase of program execution.

- **exportPopAtEndFileName:** This is an optional parameter for the suffix name of population file(.pop). Using this parameter will cause exports at the end of each run.

2. Resource/Species Parameters

- **Resource**

For all of resource definition variables, there should be one entry per resource. This means that if there are two resource types the Names should be entered like: "Clover, Grass".

- **Names:** This is a list of resource names.

- **SizesX:** This is a list of the size of the resource patch. As a default (and in most cases) it should be set the same as wolrdSizeX. It would be set differently if for instance

there was a reason to want part of the world to have more or less resources than the other. This is likely to not occur but may make further enhancements easier later.

- **SizeY:** This is a list of the sizes of the resource patches. As a default (and in most case it should be set the same as `worldSizeY`. See `worldsizeY` for more in depth description.

- **InitialTopValueMean:** This is a float that describes the mean value for the initial resource on the top. It should be listed for each resource.

- **InitialTopValueVar:** This a float which represents the variance of the initial resource level on the top habitat. It should be listed for each resource.

- **InitialBottomValueMean:** This is the same as `resInitValMeanTop` but for the bottom habitat.

- **InitialBottomValueVar:** This is the same as `resInitValdDevTop` but for the bottom habitat.

- **GrowthRateTop:** This is the growth rate at which the resources on the top grow each step. It should be listed for each resource.

- **CarryCapacityTop:** This is the maximum resource level on the top. It should be listed for each resource.

- **GrowthRateBottom:** This is the same as `resGrowthTop` but for the bottom habitat

- **CarryCapacityBottom:** This is the same as `resCarryTop` but for the top habitat

- **Animal**

For all of resource definition variables, there should be one entry per resource. This means that if there are two resource types the Names should be entered like: "rabbit, hamster".

- **Name:** This is where the names of each organism is entered. It should be listed for each species.

- **Number:** This variable represents the number of organisms at initialization per species. It should be listed for each species.

- **orgTraceOn** This variable represents whether the trace is on or off (1=on). The organism trace prints a file for each organism that includes its actions at each step, its gene and its reproduction history. It should be listed for each species.

- **MoveRadius:** This variable represents the radius (how many cells) the organisms can move in one time step. A directly diagonal cell is still considered one space away. moveRadius should be listed for each species.

- **EatRadius:** This variable represents the radius around which an organism can eat. If the radius is set to 0 it means an organism can only eat on its own space. A directly diagonal cell is still considered one space away. eatRadius should be listed for each species.

- **SwitchOn** This is currently not used and should be removed.

- **InitialAgeMean:** This represents the average age that an organism has when the population is created. It should be listed for each species.

- **InitialAgeVar:** This represents the variance of the age that organisms receive when the population is created. It should be listed for each species.

- **DeathAgeMean:** This represents the average maximum age that an organism can attain before dieing of old age. It should be listed for each species.

- **DeathAgeVar:** This represents the variance in the maximum age that an organism can attain before dieing of old age. It should be listed for each species.

- **DeathProb:** This represents the death probability that is caused by situations other than aging or predation.
- **InitialResource:** This represents the resource that the organism has when they are born and after they reproduce. It should be listed for each species.
- **ReproductionResource:** This represents the resource level needed for an organism to reproduce. It should be listed for each species.
- **StepLoss:** This represents the amount of internal resources that an organism loses each step. It should be listed for each species.
- **NumActions:** This represents the number of actions that an organism can perform. If it is set to one it can only sit. If two it can sit and feed. If three it can sit, feed and switch. It should be listed for each species.
- **NumPredRiskLevels:** This represents the number of different predator risk levels the organism can observe. It should be listed for each species. Associated parameter **HighPredRiskLevel** indicated the largest value of Predator Risk Level.
- **NumFoodLevels:** This represents the number of different relative resource levels that the organism can observe. 1 means do not use, 2 means look at whether the resource is higher or lower than what we have seen recently. It should be listed for each species.
- **NumHungryLevels:** This represents the number of different internal resource levels the organism can observe. It should be listed for each species.
- **Relative Resource Level:** This measures how high the resources/food are compared to **HighAvailResLevel**.
- **HighAvailResLevel :** This defines the highest resource on an absolute scale.

- **FoodMemory:** This defines how many steps organism can memorize about the foods that it observed.

- **AntiExtinctionOn:** This represents whether anti-extinction is on or off. 1 is on. It should be listed for each species.

- **AntiExtinctionNumber:** This number represents the threshold of Anti-Extinction. If it is less than the anti extinction number then each creature is cloned a certain number of times.

- **AntiExtinctionMultiplier:** This is the number of organisms that get cloned for each organism if anti-extinction is used. It should be listed for each species.

- **FindBestEatSpotProb:** This represents the chance that an organism will find the best spot to feed if feed is the chosen action. If it is set to 1 then the organism would always move randomly. FindBestEatSpotProb should be listed for each species.

- **MutationProb:** This represents how often a gene mutates. It should be listed for each species.

- **MutationMean:** This represents the average change in a gene when it is being mutated. It should be listed for each species.

- **MutationVar:** This represents the variance of the change in a gene when it is being mutated. It should be listed for each species.

- **DietNames:** This is a list of all the resources and organism species that the organism eats. It should be listed for each species.

- **DietProbs:** This represent the probability of the organism eating the organism and resource species listed in the DietNames attribute. The probabilities represent the chance of the organism eating prey based on the different actions the prey is doing. If the diet

entry is a resource there will only be one number, if the diet entry is an organism there will be between two and three percents.

- **DietRefuge:** This represents the percentage of the available resource that organism can not eat at any cell.

EatPercent: This represents the percentage of the available resource that organism can eat after DietRefuge mechanism. Thus, the resource consumed by organism is $(1 - \text{DietRefuge}) * \text{EatPercent}$.

GeneticEvolution: This switch can turn on/off genetic evolution mechanism. That means there is no gene mutation and recombination during reproduction.

DefaultImportFile: This is a string parameter which holds the default import file name for this animal if GeneticEvolution is turn off. In this situation, instead generating new gene sequence for offspring, DOVE will import a existing gene sequence from this file.

Appendix C: Import/Export

Importing organisms

A common way to run the model is:

1. Run it with par files that set up some resources, predators, prey after running under these conditions for a while export the pops.
2. Restart, but load only the resources and predators, so that one can import the organisms from the (end of) the previous run(s).

To do this, the "Org" parfile (loaded with parmFileName) must still specify:

- resources

- species (ANIMAL) info for each species, including all to be imported at any time in the run.

If animals are to imported, just start with number=0. For examples, the file should have entries like

```
BEGIN ANIMAL
```

```
  Name = Consumer1
```

```
  Number = 0
```

```
  ...
```

```
END ANIMAL
```

This must be done because at the start of the run, when things like the global predators are created, they must be able to find the species info for all prey listed for them, including prey that will only be imported later in a run.

Export Results

Right now there are 1 + numSpecies output files. The default names are:

- report -- the base report file.

It has the global parameters at the top, including date/time run, seed, etc. The name of this is set by -DRFN=name and a suffix can be added via -DRN=nn, eg, -DRN=00 produces report.00. This file then contains one line per time step with various aggregate measures written on each line.

- report-sp<dd> -- a file for each species, e.g. report-sp00, report-sp01, etc.

Each file contains detailed info about a particular species, one line for each time step.

Note the "report" part of the name is the same name as the base report file (from the

RFN parameter), and the suffix is the same as on that base file, e.g. -DRN=00 will produce report.00 and report-sp00.00 &etc.

- `<name>.<rn>.<specName>.pop` population files

`<rn>` is the runNumber and `<specName>` is the species name. The runNumber is the same as the suffix added to report.* files, and is set by Drone for each run it does with different RNG in a "case", and can be set on the run command with -DRN=x . These files are produced at the end of batch runs when this parameter:

`exportPopAtEndFileName (ePAEFN) =`

is given a value, e.g. `exportPopAtEndFileName=popAtEnd` would produce (in batch runs) files named `popAtEnd.00.C1.pop`, `popAtEnd.00.C2.pop`, `popAtEnd.01.C1.pop`, `popAtEnd.01.C2.pop`, and so on, for two species, C1 and C2, for runs 00, 01 and so on.

Appendix D: Using Drone Tool

Drone is a tool to run experiments. It makes it easy to:

- sweep parameters (1, 2, or more)
- N runs for each case
- organize data into output directories and files
- automatically log experiments
- easy to replicate a whole experiment, case or run
- run on multiple machines at once

Basically, you have to tell drone, through a "control file"(.ctrl) which includes

- what program to run

- how many runs to do
- where to put the output
- the basic (non-changing parameters)
- any parameters to be varied across cases
- comment for the log

We list most important parameters for DOVE as following:

- Environment setting:
 - rootDir: all your simulation output goes in directories under this directory.
 - programDir: the directory that contains the program to run
 - programName: the simulation program to run
 - emailAddr: your email address to notify the completion of simulation.
 - numRuns: how many runs to do.
- Constant global and resource/species parameters

These are a set of "base values" that act as defaults. These are specified between the begin input file

@begin

...

@end

end input file

The format is the same as the global parameters input via `-DIF = parameter`, as noted earlier in Appendix B. (note that resource/species parameters are specified in a file named in the global section).

- "Sweep" parameters

These are parameters to have values varied for the different "cases" of this experiment.

We can override the parameter values set as above, e.g.:

```
sweep yDivider 5 10 15
```

to have a set of runs for each of those yDivider values.

We can also select world dimension by sweeping a pair of parameters like this:

```
sweep (X Y) (10 10) (10 20) (20 10) (20 20)
```

Now you are ready to run the experiment:

```
./drone-linux exp01.ctrl
```

drone will fire up and if all is well, it will (create if needed and) use these directories

expdata/ experiment-1 experiment-2 and etc for each run of drone with this rootDir

errorlogs/ look for files here if errors

logs/ info for doing re-runs if you want

Drone can allow DOVE to run on multiple machines at once. There are the instructions to follow:

1. Get mmdrone:

```
mkdir ~/mmdrone
```

```
cd ~/mmdrone
```

```
cp ~mcharter/Drone/mmdrone .
```

```
cp ~mcharter/Drone/drone-linux .
```

2. Create .bashrc.drone and .bashrc.regular:

```
cd
```

```
cp .bashrc .bashrc.regular
```

```
cp .bashrc .bashrc.drone
```

edit .bashrc.drone and change the line:

```
export PS1
```

to be:

```
export PS1="DRONEPROMPT"
```

3. Create control file as before except set this so it dishes out jobs:

```
useLocalHost = 0
```

4. Create hosts file, set this to point to the hosts file:

```
param hostDir = $rootDir/hostfiles
```

and set the name of the file the hosts are in:

```
param hostFile = hosts
```

so if the rootDir is /users/mcharter/RePast/ABModel1/drone there should be a file named "hosts" in /users/mcharter/RePast/ABModel1/drone/hostfiles with the list of hosts

5. Be sure this parameter is set thus:

```
param cacheIPAddresses = 0
```

6. open a terminal window and make the width large (150 chars or so), then do:

```
./mmdrone exp1.ctrl
```

Appendix E: Experiment Profiles

The simulation parameters profiles for the experiments in Chapter 7 are list below:

Global parameters:

```
@begin
```

```
reportFileName = report
reportFrequency = 1
stopT = 50
rDebug = 0
worldXSize = 30
worldYSize = 60
yDivider = 30
numResourceTypes = 1
homogeneousResource = 0
antiExtinctionSteps = 10000
topToBottomMultiplier = 8
# set maxGlobalPredators to 0 to disable the present of Global Predator, 1 to enable
maxGlobalPredators = 1
globalPredationTimeOnMean = 25
globalPredationTimeOnVar = 100
globalPredationTimeOffMean = 25
globalPredationTimeOffVar = 100
globalPredatorDietNames = (Consumer1,Consumer2)
globalPredatorDietProbs = ([0.001,0.006,0.004])
globalPredatorTraceOn = 0
# note we have to specify this full path for drone runs:
parmFileName = ./dat/parOrg_2Hx2P_SF.in
exportFileName = export.pop
```

@end

BEGIN RESOURCE

Name = Resource

SizeX = 30

SizeY = 60

InitialTopValueMean = 150.0

InitialTopValueVar = 15.0

InitialBottomValueMean = 150.0

InitialBottomValueVar = 15.0

GrowthRateTop = 1.03

CarryCapacityTop = 400.0

GrowthRateBottom = 1.03

CarryCapacityBottom = 400.0

END RESOURCE

BEGIN ANIMAL

Name = Consumer1

Number = 50

MoveRadius = 1

EatRadius = 0

SwitchOn = 0

InitialAgeMean = 75

InitialAgeVar = 5
DeathAgeMean = 300
DeathAgeVar = 0
DeathProb = .005
InitialResource = 60
ReproductionResource = 400
StepLoss = 0
NumActions = 3
NumPredRiskLevels = 2
HighPredRiskLevel= 1
NumFoodLevels = 1
NumHungryLevels = 1
NumAvailResLevels = 1
HighAvailResLevel = 400
FoodMemory = 0
AntiExtinctionOn = 0
AntiExtinctionNumber = 40
AntiExtinctionMultiplier = 1
FindBestEatSpotProb = 0.0
MutationProb = 0.1
MutationMean = 0.0
MutationVar = 0.1
DietNames = (Resource)

DietProbs = ([1.0])

DietRefuge = ([0.1])

EatPercent = ([0.35])

GeneticEvolution = 1

DefaultImportFile = ./dat/Even1Hx2P.Consumer1.pop

END ANIMAL

BIBLIOGRAPHY

Abrams, P. A. , B. A. Menge, G. G. Mittelbach, D. Spiller, and P. Yodzis. 1996. The role of indirect effects in food webs. Pp. 371-395. in: G. Polis and K. Winemiller, eds. Food webs: dynamics and structure. Chapman & Hall, New York, NY.

Abrams, P. A. 1995. Implications of dynamically variable traits for identifying, classifying and measuring direct and indirect effects in ecological communities. *Am Nat* 146:112-134.

Agrawal, A. A. 2001. Phenotypic plasticity in the interactions and evolution of species. *Science* 294:321-326.

Alexandridis K. T., Lei Z., Pijanowski B. C. 2003, Simulating Land-Use Entelechy using Multi Agent-Based Behavioral Economic Landscape (MABEL) Model, The Agent 2003 Conference on: Challenges in Social Simulation, Oct. 2003

Axelrod, R. 2002. The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration. Princeton Univ. Press.

Bar-Yam, Y. (1997). Dynamics of Complex Systems. Addison-Wesley.

Bender, E. A., T. J. Case, and M. E. Gilpin. 1984. Perturbation experiments in community ecology: Theory and practice. *Ecology* 65:1-13.

Booch, G. (1994). Object-Oriented Analysis and Design. Benjamin/Cummings, second edition.

Brown. J. H., 1994. Complex Ecological Systems, in Complexity: metaphors, models and reality. Santa Fe Institute Studies in the Sciences of Complexity, Vol. XIX, pp 419-449

Bunn and Larsen, 1997, Exploring Oil Market Dynamics, J.D.W. Morecroft and B. Marsh, in Systems Modelling for Energy Policy, chapter 10, pp. 167-203, Wiley, 1997.

DeAngelis, D. & L. Gross, L. 1992. Individual Based Models and Approaches in Ecology: Populations, Communities, and Ecosystems. Chapman and Hill.

Deaven, D., & Ho, K. (1995). Molecular geometry optimization with a genetic algorithm. *Phy. Rev. Let.*, 75 , 288-291.

Epstein, J.M. and R. L. Axtell 1996. Growing artificial societies: social science from the bottom up. MIT Press.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Mass.

Goodman E. D., K. Seo, R. C. Rosenberg, Z. Fan, J. Hu, and B. Zhang, "Automated Design Methodology for Mechatronic Systems Using Bond Graphs and Genetic Programming," 2002 NSF Design, Service and Manufacturing Grantees and Research Conference, January 7-10, 2002, San Juan, Puerto Rico

Holland J. H. 1975. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, Michigan. .

Holland, J. H. 1995. Hidden Order: How adaptation builds complexity. Perseus Publishing

Holland, J. H. & Miller J. H. Artificial Adaptive Agents in Economic Theory, *American Economic Review*, May 1991, pp. 365-370

Holland, J.H. 1986. Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In *Machine Learning: An Artificial Intelligence Approach*, Volume II, Michalski, R. S., (Eds). Morgan Kaufman Publishers.

Holt, R.D. 1987. Population dynamics and evolutionary processes: the manifold roles of habitat selection. *Evol. Ecology* 1:331-347.

K Alexandridis K. T., Lei Z., Pijanowski B. C. 2004, Sequential Decision-Making Process Simulation of Base-Agents in a Multi Agent-Based Economic Landscape (MABEL) Model, soon appear in *Journal of Environmental & Resource Economics*

Lanzi, P. L. and Rick L. Riolo. 2002. Recent Advances in Learning Classifier Systems. In "Advances in Evolutionary Computation.

Laumanns M., Rudolph G., Schwefel H. P., 1999, A Spatial Predator-Prey Approach to Multi-Objective Optimization: A Preliminary Study, *Proceedings of the Fifth Conference on Parallel Problem Solving from Nature*

Lei Z., Pijanowski B. C., Alexandridis K. T. 2003, Simulation and Distributed Architecture of Multi Agent-based Behavioral Economic Landscape (MABEL) Model within Swarm, The Agent 2003 Conference on: Challenges in Social Simulation.

Leibold, M.A. 1989. Resource edibility and the effects of predators and productivity on the outcome of trophic interactions. *Amer. Nat.* 134:922-949.

Levine, S. H. 1976. Competitive interactions in ecosystems. *American Naturalist* 110:903-910.

Lima, S. L. 1998. Stress and decision making under the risk of predation: Recent developments from behavioral, reproductive, and ecological perspectives. *Stress and Behav.* 27: 215-290.

Mitchell M. and Newman M., 2002, Complex systems theory and evolution, *Encyclopedia of Evolution*, M. Pagel (ed.), Oxford University Press, New York.

Murch and Johnson, 1999, *Intelligent Software Agents*, 66.

Parker, D. C. , S. M. Manson, M. A. Janssen, M. Hoffman, and P. Deadman. 2001. Multi-Agent Systems for the Simulation of Land-Use and Land-Cover Change: A Review, Workshop on Agent-Based Modeling for Land Use and Cover Change.

Peacor, S.D., and E.E. Werner. 2001. The contribution of trait-mediated indirect effects to the net effects of a predator. *PNAS* 98:3904-3908.

Rick L. Riolo, Scott Page, 2002, *Nonlinear System: Agent-based and Computer Intensive Modeling*

Sahiner B., H. P. Chan, D. Wei, N. Petrick, M. A. Helvie, D. D. Adler, and M. M. Goodsitt., 1996, Image feature selection by a genetic algorithm: Application to classification of mass and normal breast tissue. *Medical Physics*, 23(10):1671--84.

Serra R. and Zanarini G., 1990, *Complex Systems and Cognitive Processes*. Springer Verlag.

Shoham, Yoav., 1993, Agent oriented programming. *Artificial Intelligence*, 60:51-92.

Tollrian, R. and Harvell, C. D., eds. 1999. The ecology and evolution of inducible defenses. - Princeton University Press, New Jersey.

Weisbuch, W. , 1991, Complex Systems Dynamics, Addison-Wesley

Werner, E. E. and S. D. Peacor. 2003. A review of trait-mediated indirect interactions. Ecology 84: 1083-1100.

Wooldridge M., 1997, Agent-based Software Engineering. In IEE Proceedings on Software Engineering, 144(1), pages 26--37

Wootton, J. T. 1993, Indirect effects and habitat use in an intertidal community: interaction chains and interaction modifications. American Naturalist 141:71-89.

Yodzis, P., 1988, The indeterminacy of ecological interactions, as perceived by perturbation experiments. Ecology 72:1964-1972.

MICHIGAN STATE UNIVERSITY LIBRARIES



3 1293 02504 4078