APPROVAL SHEET

Title of Thesis: A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

Name of Candidate: Donald P. Miner
PhD in Computer Science,
2010

Thesis and Abstract Approved: _____
Dr. Marie desJardins
Associate Professor
Department of Computer Science and
Electrical Engineering

Date Approved: _____

Curriculum Vitae

Name: MY-FULL-NAME.

Permanent Address: MY-FULL-ADDRESS.

Degree and date to be conferred: DEGREE-NAME, GRADUATION-MONTH GRADUATION-YEAR.

Date of Birth: MY-BIRTHDATE.

Place of Birth: MY-PLACE-OF-BIRTH.

Secondary Education: MY-HIGH-SCHOOL, MY-HIGH-SCHOOLS-CITY, MY-HIGH-SCHOOLS-STATE.

Collegiate institutions attended:

University of Maryland Baltimore County, DEGREE-NAME MY-MAJOR, GRADUATION-YEAR.
MY-OTHER-DEGREES.

Major: MY-MAJOR.

Minor: MY-MINOR.

Professional publications:

FULL-CITATION-INFORMATION.
FULL-CITATION-INFORMATION.

Professional positions held:

EMPLOYMENT-INFO. (START-DATE – END-DATE).
EMPLOYMENT-INFO. (START-DATE – END-DATE).

ABSTRACT

Title of Thesis: A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

Donald P. Miner, PhD in Computer Science, 2010

Thesis directed by: Dr. Marie desJardins, Associate Professor
Department of Computer Science and
Electrical Engineering

This is the abstract. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum

A Framework for Predicting and Controlling System-Level Properties of Agent-Based Models

by

Donald P. Miner

Thesis submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science
2010

This is my dedication.

ACKNOWLEDGMENTS

These will be written out later:

- John Way: My excellent high school computer science teacher; he was the first teacher to truly inspire and make me interested in something
- Richard Chang: Unofficial undergrad advisor; Inspired me to work on hard problems with his classes/my undergrad thesis; Indirectly helped me want to pursue graduate school
- Marie: advisor; introducing me to MAS
- Bill Rand: introducing me to NetLogo
- Forrest Stonedahl: working on a similar problem; a conversation which was a turning point in my research focus to ABMs; introduced me to the term 'meta-model'
- Tim Oates: Suggestions dealing with the ML portion of my research
- Undergraduate Researchers (Peter, Kevin, Doug, Nathan?): Helping with researching new domains
- Marc Pickett: Good friend that is always willing to listen to research ideas; Helped me throughout grad school
- Senior grad students who helped me as a young grad student: Adam Anthony, Eric Eaton, Blaz Bulka
- Other supporting CS graduate students: Wes Griffin, Yasaman Haghpanah, Niels Kasch, James MacGlashan, JC Montminy, Sourav M, Patti Ordonez, Soumi Ray, and Brandon Wilson.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION AND MOTIVATION

The behavior of individual agents in an agent-based model (ABM) is typically well understood because the agent's program directly controls its local behaviors. What is typically not understood is how changing these programs' agent-level control parameters affect the observed system-level behaviors of the ABM. The aim of this dissertation is to provide researchers and users of ABMs insight into how these agent-level parameters affect system-level properties. In this dissertation, I discuss a learning framework named the ABM Meta-Modeling Framework (AMF) that I have developed that can be used to predict and control system-level behaviors of agent-based models. With this framework, users can interact with ABMs in terms of intuitive system-level concepts, instead of with agent-level controls that only indirectly affect system-level behaviors.

1.1 Agent-Based Models

Agent-based models are used by scientists to analyze system-level behaviors of complex systems by simulating the system bottom-up. At the bottom of these simulations are individual agents that locally interact with other agents and the environment. All the behavior in an ABM, from agent-level local interactions to system-level behaviors, emerge from these local interactions, which are governed by the individual *agent programs*. ABMs can

be used to understand how changes in individuals' *agent-level parameters* affect *system-level properties*.

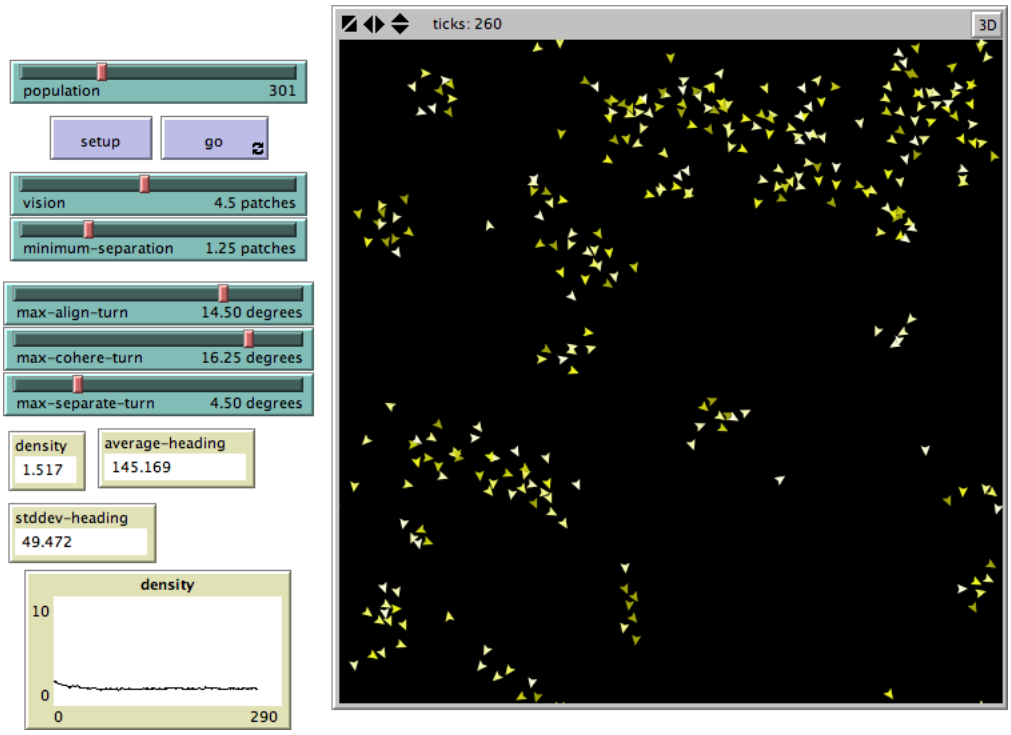

FIG. 1.1. A screenshot of NetLogo's graphical user interface while executing a flocking simulation.

Agent-level control parameters adjust the behaviors of agent-based programs. However, scientists are not typically interested in the local interactions between agents— they are interested in the resulting system-level behaviors that result. For example, researchers that studied agent-based models of lane formations in army ants were interested in the traffic patterns of the lanes, not the individual behaviors of the ants (Couzin & Franks 2003). In other work, researchers that studied locusts were interested in determining at what critical density locusts begin to swarm and destroy crops (Buhl *et al.* 2006). Typically, scientists analyze ABMs by viewing visualizations of the environment or gathering statistical data on the simulation. For instance, NetLogo, an agent-based modeling programming

environment (Tisue & Wilensky 2004), has monitors, plots and visualizations to convey system-level properties to the user. In Figure 1.1, monitors are displaying *density*, *average-heading* and *stddev-heading* statistics for a flocking domain. In addition, a plot of density shows how it has changed over time. These tools are used by a researcher to generate a mental model of how the agent-level control parameters of the flocking domain (the sliders seen in the user interface) affect these system-level properties.

Although using ABMs for researching agent-based systems has been proven useful in a number of domains, there is a glaring conceptual disconnect from the user's perspective, between the agent-level controls and the system-level properties. The classical ABM control method of adjusting agent-level properties is unintuitive because they only indirectly affect the system-level properties through emergence. With the current methodology, a simulation has to be executed in order to observe what the values of the system-level properties will be. A time consuming iterative process of guess-and-check is the only way to configure the system to have it exhibit a desired system-level behavior. A determination of what an ABM will do at a system-level, given only the agent-level parameters, is not possible with current software.

The main goal of the ABM Meta-Modeling Framework is to bridge the gap between agent-level parameters and system-level properties. AMF reduces the learning curve of an agent-based model since users are interacting with the system at the system-level, instead of at the agent-level. Qualitative analysis of an ABM's system-level properties will be a more efficient process since researchers deal with an abstraction of the system's controls. In addition, the models learned by AMF can be inspected to gather quantitative data about the correlations between system-level properties and agent-level parameters.

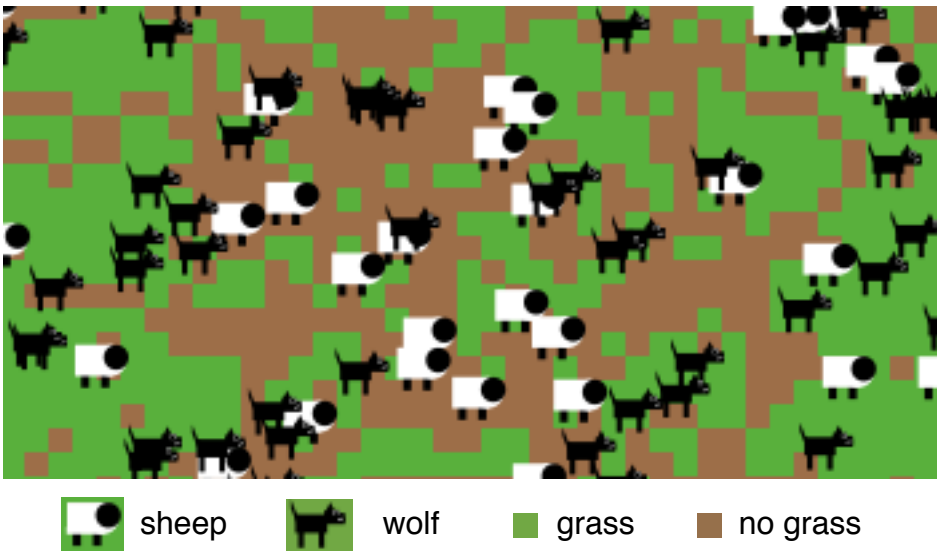

FIG. 1.2. A screen shot from NetLogo's Wolf Sheep Predation model.

1.2 Wolves, Sheep and Grass

Throughout this dissertation, I will use NetLogo's Wolf Sheep Predation model (Wilensky 1997b), which is bundled with NetLogo's standard Model Library,[1] as an example to explain concepts. A snapshot of its NetLogo visualization is shown in Figure 1.2. This multi-agent model simulates a food chain consisting of wolf agents, sheep agents and grass in a two-dimensional space. The model is controlled by seven agent-level control parameters, which directly affect the following agent behaviors:

- The system is initialized with *initial-number-sheep* sheep and *initial-number-wolves* wolves.
- Wolves and sheep move randomly though the space.
- Wolves and sheep die if they run out of energy.

[1] http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation

- Wolves eat sheep if they occupy the same space in the environment. Wolves gain *wolf-gain-from-food* units of energy from eating sheep. The sheep dies.
- Sheep eat grass if they are on a location of the environment that has grass. Sheep gain *sheep-gain-from-food* units of energy from eating grass. The grass dies in that grid location.
- Every time step, each sheep and each wolf has a chance (*sheep-reproduce* and *wolf-reproduce*) to reproduce asexually. Both the parent and the child split the parent's original energy evenly (i.e., parent's energy divided by two).
- Grass regrows after *grass-regrowth-time* number of time steps.

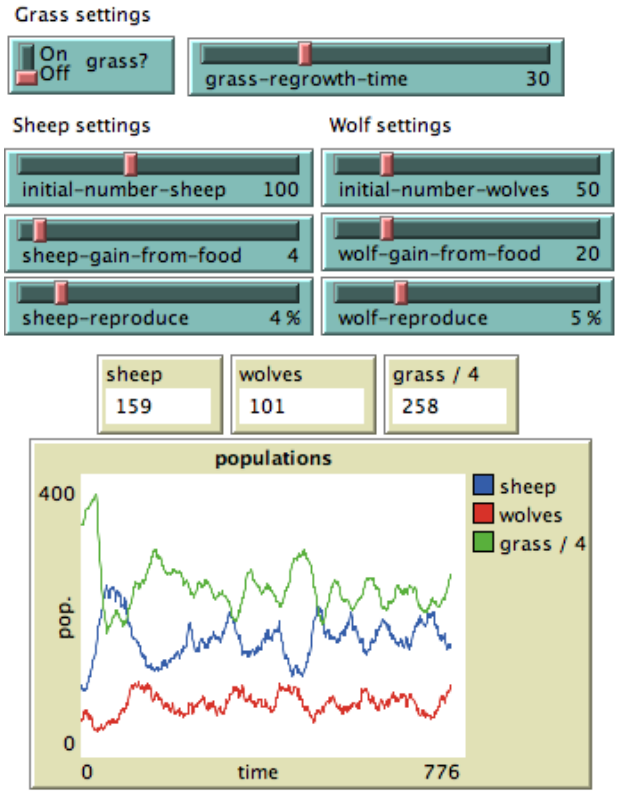

FIG. 1.3. The control and monitor interface for the Wolf Sheep Predation model.

The system-level concepts we are interested in are the number of sheep, the number of wolves and the number of grid locations containing grass. In NetLogo, these properties are displayed with monitors and a plot, as seen in Figure 1.3. The number of each population of agents may change continuously, but the average number of sheep converges. Another interesting feature is some ecosystems fail: either sheep or both sheep and wolves go extinct.

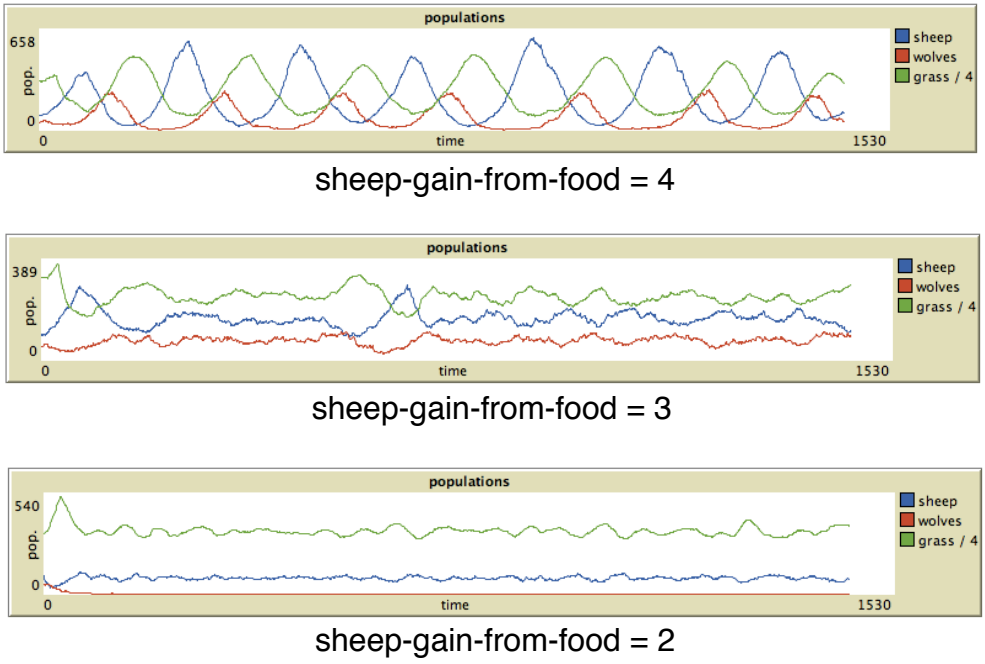

FIG. 1.4. Differences in populations based on changes of the *sheep-gain-from-food* parameter.

After working with this ABM for some time, a user will begin to realize that changes in the control parameters will yield different types of behavior. For example, by setting *sheep-gain-from-food* to 2, 3, and then 4, major differences in system-level behavior are apparent by viewing the graphs in Figure 1.4. When the value of *sheep-gain-from-food* is 4, the system rhythmically exhibits major changes in all three agent populations. When the

value is 2 or 3, the population remains relatively stable, but the average population values are different. When the value is low enough (e.g., 2) the wolves go extinct.

The Wolf Sheep Predation model is a good example of the intuitive disconnect between agent-level parameters and system-level properties. There is no clear *explicit* relationship between the controls presented in the user interface and the resulting system-level properties. An experienced user may have a qualitative understanding of the correlations, but would not be able to predict quantitative concepts, such as the average number of sheep after 2000 time steps. In Chapter 9: Results, I will show that the intuitive disconnect in this domain can easily be solved by AMF.

1.3 Overview of the ABM Meta-Modeling Framework

The foundation of this work is framing the problem of building a meta-model of an ABM as two sub-problems: the *forward-mapping problem* and the *reverse-mapping problem*. In Chapter 6: The Forward-Mapping Problem, I will discuss how AMF maps given values of the agent-level parameters to expected system-level property values with standard regression approaches. In Chapter 7: The Reverse-Mapping Problem, I will discuss how AMF maps a set of desired system-level property values to a set of agent-level parameters that would generate this behavior. My general approach to solving the reverse-mapping problem is to interpolate configurations using the forward mapping to approximate a smooth and continuous surface. This interpolated surface represents the space of configurations that would satisfy the system-level requirements set out by the user. Also, in Chapter 7, I will discuss alternative methods for solving the revers-mapping problem.

AMF is simple and has only a few configuration points. This allows researchers to focus on the analysis of the system, instead of on the details of AMF. The framework consists of three major steps: sampling, solving the forward-mapping problem and solving

the reverse-mapping problem. In these three steps, the only configurations the user must perform are: define how to measure system-level properties of interest, provide the ranges of parameters to be sampled, and plug in a regression algorithm.

I will show in Chapter 9: Results that my framework is able to generate models of system-level behavior. For example, AMF is able to predict the number of sheep and wolves in the Wolf Sheep Predation model, given the configuration parameter values (the forward-mapping problem). Also, AMF is able to make suggestions for the values for the control parameters, given the the desired system-level property outcome (the reverse-mapping problem).

A more comprehensive overview of AMF is provided in Chapter 2: The ABM Meta-Modeling Framework.

1.4 Summary of Contributions

My main contribution presented in this dissertation is an in-depth analysis of meta-models of agent-based models. This analysis includes a discussion of methods for using regression to build models of the correlations between agent-level parameters and system-level properties. In addition, this dissertation contains a survey of ways that that meta-models can be used to inspect system-level behaviors of agent-based models.

The ABM Meta-Modeling Framework encapsulates my methodology for building meta-models of ABMs. The implementation of AMF as software serves as a proof-of-concept to show that my approach is implementable and applicable to a variety of domains. The software itself is a contribution, since it is available to be used by researchers interested in building meta-models of NetLogo ABMs. The design of the general framework is a contribution as well, since it could be implemented to interact with other agent-based modeling systems similar to NetLogo, or totally independent agent-based models.

1.5 Dissertation Organization

This dissertation is divided into nine chapters, including this one. Chapter 2: The ABM Meta-Modeling Framework explains each framework component in detail, explains how a new user would tailor AMF to a new ABM, discusses implementation details and gives an introduction to the forward- and reverse-mapping problems. Chapter 3: Related Work compares and contrasts approaches similar to AMF in motivation, with AMF. Chapter 4: Background provides information about NetLogo and algorithms used by AMF. Chapter 5: Defining System-Level Properties discusses the numerous different ways that system-level properties of ABM can be defined. Chapters 6 and 7 discuss my solutions to the forward- and reverse-mapping problems. Chapter 8: Using Meta-Models is a survey of different ways the meta-models generated by AMF can be used to analyze system-level properties of ABMs. Chapter 9: Results evaluates AMF on a domain-by-domain basis and provides explicit examples of how AMF has been used. Chapter 10: Conclusions and Future Work summarizes this dissertation, provides additional thoughts I have regarding this work and possible directions for future work.

Chapter 2

THE ABM META-MODELING FRAMEWORK

The ABM Meta-Modeling Framework builds meta-models that map the values of agent-level control parameter values to system-level property values, and vice versa. Learning these mappings are separate problems, which I call the *forward-mapping problem* and the *reverse-mapping problem*. To solve these problems, a user of AMF "plugs in" a regression algorithm of their choice. The framework uses this regression algorithm to learn the mappings and then provide the user with interfaces to query them. Once the mappings are learned, the user can query either for *prediction* or *control*. A prediction query uses the forward mapping to determine values for the expected values for system-level properties, given the system's configuration. A control query uses the reverse mapping to suggest values for agent-level parameters, given desired values for system-level properties. Most of the implementation details and inner workings of AMF are abstracted away from the user, who only has to attend to a limited number of configuration points.

In this chapter, I will discuss the design goals of AMF, the framework structure, software implementation details, and how well AMF conforms to the design goals.

2.1 Design Goals of The ABM Meta-Modeling Framework

My specific goal in designing AMF was to make controlling and interacting with agent-based models more intuitive. In addition to this central goal, AMF strives to be:

- Domain independent – the design of AMF should minimize the amount of configuration for each new domain,
- Algorithm independent – any regression algorithm should be able to be plugged into AMF,
- Accurate – AMF should generate accurate predictions and control suggestions,
- Fast for the user – interactions with the models generated by AMF should require minimal computational time.

Domain independence is paramount because of the variety in which ABMs come. I have designed AMF such that the same general approach would work for any ABM. Also, I strove to minimize the amount of configuration needed to apply AMF to a new domain. These constraints I have set on the design make AMF broadly applicable to a number of domains, without the need of in-depth domain knowledge. To reinforce this claim, I have tested AMF on a number of diverse domains and used the same general approach for each.

Algorithm independence in a learning framework is important because different algorithms may be more effective for modeling different agent-based models. In general, the learning algorithms that will be discussed in this dissertation will satisfy the requirements for modeling most agent-based models. However, an in depth analysis of which types of algorithms should be used for different classes of ABMs is outside the scope of this dissertation research. In addition, algorithm independence allows AMF to scale with new advances in machine learning research, since future state-of-the-art regression algorithms can be plugged in just as easily as current approaches.

Accuracy and fast user response time appear to be obvious design goals. However, achieving these goals require sacrifices in performance in other portions of the framework. AMF requires a significant amount of computational time to sample different configurations of the target ABM. These large training sets can be used to build static meta-models of ABMs that are both accurate and fast to query. In contrast, an active learning approach would be able to learn models faster, but would require interaction with the user, increasing the amount interaction. Likewise, optimization approaches (e.g., hill climbing) could be used to generate arbitrarily accurate results, but typically require numerous iterations and would significantly increase the response time for a user's query. This is because each step would have to run the ABM to calculate the fitness score, which could take several seconds. In summary, I am making the assumption that users studying ABMs with AMF are more interested in achieving more accurate results for their research and interacting with the models quickly, than spending less time sampling.

2.2 Framework Structure

The framework is split into several phases: sampling, solving the forward-mapping problem, solving the reverse-mapping problem, querying for prediction, and querying for control (i.e., suggesting a configuration). I explain with Figure 2.1 how the different phases interact with one another. Sampling makes observations from the actual ABM and then feeds the newly generated data set to the forward-mapping solver. The result of solving the forward mapping is a function f, which is used to predict behavior and to develop the reverse mapping f^{-1}. From an exterior perspective, the framework only has a limited about of input and output: AMF takes in observations of an agent-based model and then provides interfaces to query for prediction and control. These queries interact with the user, as well as the agent-based model.

Many configuration points exist, which allow users of AMF to modify the behavior of the framework. However, the individual phases take the same output and provide the same output, no matter the configuration. This is what makes AMF a framework and not a collection of algorithms. For example, even though different regression algorithms can be used to learn the forward mapping, the forward mapping still provides predictions of how an ABM will behave, from the user's perspective.

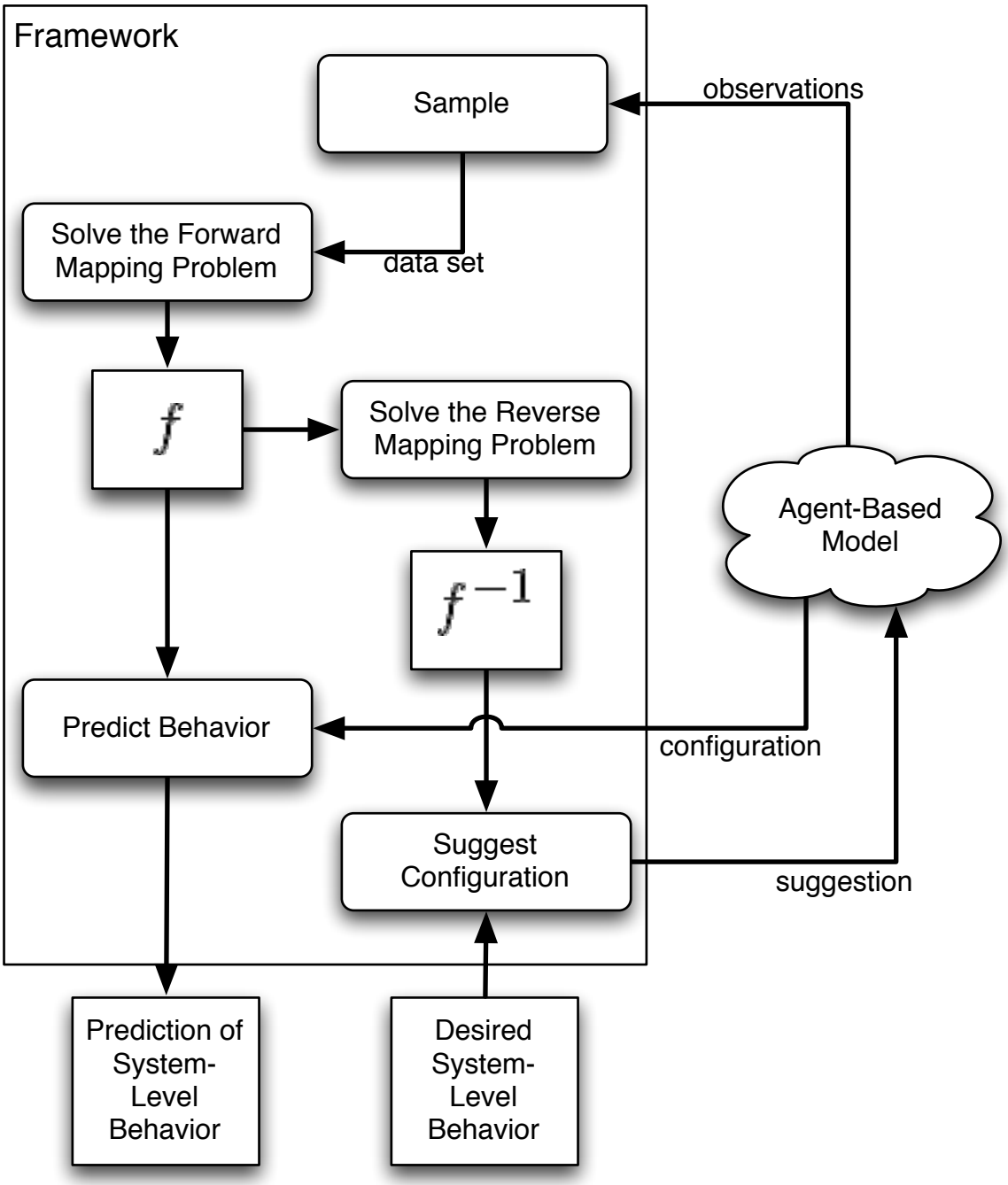

FIG. 2.1. An overview of the phases of AMF and how data flows between them.

2.2.1 Defining System-Level Behavior Properties

System-level properties of an ABM must be defined by the user of AMF. The measurement of a system-level property is a statistical or mathematical calculation based on the state of the ABM over some period of time (or "ticks"). The one assumption made by AMF is that the measurement is *stable*, meaning that as the same configuration is sampled repeatedly, the measurement's value should vary minimally. One way to measure stability is to calculate the standard deviation of measured values of several runs of the sa e system. The need for this assumption, ways to conform to it, and a more detailed explanation of how to define system-level properties is provided in Chapter 5: Defining System-Level Behaviors.

For example, there are several system-level behavior properties we measure in the Wolf Sheep Predation model. Three of the most obvious are the average number of sheep, average number of wolves, and average amount of grass over a significant number of ticks. These are measured by individually summing the number of sheep, wolves, and grass living at each time step and dividing by the number of ticks. Although the number of sheep and wolves change rhythmically, the average values typically converge to a single value after about 10,000 ticks.

2.2.2 Sampling

The first phase is *sampling*. In this phase, numerous observations are made on different configurations of an agent-based model. A data set is generated that contains the independent variables (the agent-level control parameter values) and the resultant system-level behaviors that are measured, for each observation. This process can be significantly time consuming, dependent on several factors:

- Execution time of the model – the models will be executed numerous times, so the

longer a model takes to execute, the longer the whole set of experiments will take to execute,

- Granularity – more detailed samplings will take more time, since more points must be sampled,
- Dimensionality – more agent-level control parameters result in a larger search space and naturally more points to sample,

A 120 observation sample of the NetLogo *Fires* model[1] (Wilensky 1997a), requires about thirty seconds[2] to generate. Meanwhile, a large sample of 50,000 observation of a Reynolds boid flock (Reynolds 1987) took approximately four days.

The user is required to have a minimal amount of domain knowledge to specify to AMF which ranges of values should be sampled. AMF is not able to automatically infer which value ranges are interesting, and thus must be explicitly defined.

This sampling process is easily parallelizable. Since each experiment is independent of one another, the set of all experiments can be segmented among a number of systems and processors to significantly reduce the computation time. Once all of the experiments are done executing, the results can be merged into one data set.

For the scope of this dissertation, I limit AMF to use a simple random sampling method (i.e., randomly select points within a specified range) or a systematic sampling method (i.e., given ranges, sample evenly spaced points). I acknowledge that an intelligent sampling strategy can be improve the performance of many machine learning techniques, however, none are used for the sake of focusing on other portions of the framework. In future work, different sampling techniques' effect on accuracy and speed could be measured.

[1] See Chapter 9 for detailed results

[2] Most experiments are executed on a 2.2GHz Intel Core 2 Duo running Mac OS X

2.2.3 The Forward-Mapping Problem

The forward-mapping problem is:

> Develop a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that maps a provided configuration vector $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ to several system-level behavior properties $\hat{\mathbf{y}}$:

$$\hat{\mathbf{y}} \leftarrow f(\mathbf{x})$$

The set of values $\mathbf{x}$ consists of all the agent-level parameters (independent variables) in the data set provided by the sample step. An individual mapping is learned for each system-level behavior measurement provided by the user.

This problem is solved with straightforward regression, such as k-nearest neighbor. In this phase of AMF, the regression algorithm is "initialized," if necessary. For example, linear regression would need to solve the least-squares problem. Meanwhile, an algorithm like k-nearest neighbor would not need to initialize anything.

Once this phase is completed, the user is presented with f, an interface to the mapping built by the regression algorithm. The forward mapping is primarily used to *predict* what the system-level property values will be, given the system's configuration. For example, a learned mapping could be used to determine the average number of sheep given a configuration vector, without having to run the system. Some sample predictions, given particular system configurations, are shown in Table 2.1. The values for a system configuration represent *grass-regrowth-time*, *sheep-gain-from-food*, *wolf-gain-from-food*, *sheep-reproduce*, and *wolf-reproduce*, respectively.

A more in depth definition of the forward-mapping problem, specific examples of regression methods used in AMF, and the role of regression is given in Chapter 6: The Forward-Mapping Problem.

Table 2.1. Sample Predictions of Behavior in the Wolf Sheep Predation Model

Configuration	Average # Sheep	Average # Wolves	Average # Grass
$(30, 4, 20, 4, 5)$	162.8	76.1	964.8
$(30, 3, 26, 7, 5)$	122.8	90.4	1135.2
$(14, 3, 26, 7, 5)$	144.3	163.8	1621.4
$(5, 3, 17, 7, 5)$	946.9	3.4	1065.4

2.2.4 The Reverse-Mapping Problem

The reverse-mapping problem is:

> Produce a mapping $f^{-1} : \mathbb{R} \rightarrow \hat{S}$ from a given system-level behavior properties $\mathbf{y}$ to a set of configurations $\hat{S} = \{\hat{\mathbf{x}} | f(\hat{\mathbf{x}}) = \mathbf{y}\}$ (i.e., S is the set of behaviors that will produce behavior $\mathbf{y}$):

$$\hat{S} \leftarrow f^{-1}(\mathbf{y})$$

The problem of developing the mapping f^{-1} is what I call an "inverted regression problem" and has many unique challenges. The main challenge is f^{-1} does not describe a functional mapping. This is because f^{-1} describes a one-to-many relationship, since f is many-to-one. Therefore, AMF cannot use standard regression techniques to solve this problem. Instead, the default behavior of AMF is to approximate the inverse of the forward mapping, with a method that I developed. This approach has the benefit of using the forward mapping to develop the reverse mapping, so no additional input from the user or data set are needed.

The reverse mapping can be used to suggest a system configuration that will exhibit a specific system-level properties. I call this process *control* of the ABM, since it is controlling the ABM at a system-level by suggesting configurations. For example, the reverse mapping could be used to suggest a configuration of $(30, 4, 20, 4, 5)$ for a desired system-

level property of 162.8 average sheep (see Table 2.1). However, using the reverse mapping is more involved than using the forward mapping, because f^{-1} returns a set of possible solutions, not a singular suggestion. If the mapping is to be used for control, any configuration in this set will satisfy the system-level requirements. Therefore, an additional step must be taken to extract a point from the set if it is to be used to control an ABM.

The implementation details of developing reverse mappings and how to use the reverse mappings are discussed in Chapter 7: The Reverse-Mapping Problem.

2.2.5 Summary of Configuration Points

The following is a summary of configuration points discussed in this section. For each new domain, the user must specify the following:

- A list of agent-level control parameters, and within what ranges they should be sampled,
- A list of system-level behavior properties and the way to measure them.

In addition, the user may configure the following if they wish to stray from the defaults:

- The sampling strategy (default: random sampling),
- The regression algorithm to be used for the forward mapping (default: k-nearest neighbor),
- The method for the reverse mapping (default: approximate the inverse of the forward mapping).

2.3 Software Implementation Details

So far, I have discussed AMF abstractly and have avoided specific implementation details. This is because the framework is a methodology and could be reimplemented in a number of different ways or to work with a number of different ABM simulation systems. In this section, I discuss the details of my proof-of-concept implementation that was used to run many of the experiments documented Chapter 9: Results.

My implementation works directly with NetLogo[3] to sample, predict and control NetLogo ABMs. All interactions with NetLogo, such as running experiments, retrieving a current configuration or pushing a suggested configuration are handled with Java through NetLogo's Java API. The rest of the framework is implemented in Python. Each step of AMF returns its result as a file, so that it can be passed to the next step or saved for later use. For example, the forward mapping writes the model to a file so that it can be used by both the prediction script and the reverse mapping script.

Since each domain has different variable names and nuaces, the user must implement a Java program that interacts with NetLogo's API. An abstract base class for interaction is provided to guide the development of this program. Next, the user lists the configuration parameters and the ranges of these values so that the sampling can begin. Measuring the system-level parameters can either be calculated within NetLogo or in Java. For example, I modified the standard Wolf Sheep Predation model to keep track of the number of sheep at each tick. Then, to extract the value, NetLogo's API is used to retrieve the sum of the sheep divided by the number of ticks. A similar calculation could be performed within Java program by retrieving each piece individually, then performing statistics on them. Performing the statistics in Java have the benefit of not having to change the NetLogo ABM source code. The data set is written to a file, row by row. Therefore, the results of several

[3]More about NetLogo is covered in Chapter 4: Background.

instances of the sampling, executing on different machines, can be easily concatenated. An example of an interaction program for Wolf Sheep predation is given in Appendix A.1.

The regression algorithms used with AMF must conform to a standard API and are passed as arguments into the forward, reverse mapping and prediction scripts. Therefore, no configuration of these core scripts is needed, since they are "pluggable."

An overarching "master script," written in Python, runs each step automatically, limiting the amount of direct interaction with the framework software.

In addition to the core framework software, I have developed a toolkit that queries the models generated by the forward- and reverse-mapping models. The core tools include:

- Query the forward mapping for a prediction by passing in a configuration,
- Query the reverse mapping for a set of possible solutions by passing in a desired system-level behavior configuration,
- Visualize the mappings.

Each of these will be discussed in detail in Chapter 8: Using Meta Models.

2.4 Analysis of AMF vs. the Design Goals

In summary, I align the actual implementation of AMF with the design goals.

Domain independence— My framework's implementation is domain independent, to an extent. AMF reduces the forward-mapping problem to a classical regression problem of learning the correlation between the agent-level parameters and the system-level properties. My reverse-mapping problem solution is also domain independent because it interacts exclusively with the forward mapping, which is domain independent.. However, my implementation of AMF still requires the user to specify the agent-level parameters and how to measure the system-level properties. This is a reasonable, since automatically detecting

configuration points and having a computer determine behaviors of interest in an ABM would be a challenging unsupervised learning problem.

Algorithm independence— Any regression algorithm can be plugged into my implementation, as long as they conform to the standard API. A simple Python wrapper can be written to adapt an already existing third party regression algorithm to work with my implementation of AMF. My default process of inverted regression requires nothing special of the regression algorithm, because the inversion learning process uses the algorithm's standard forward-mapping behavior.

Accuracy— No extra error is incurred by AMF, itself; the accuracy of AMF depends on the regression algorithm used and the amount of time spent sampling. If AMF is not providing accurate results with state of the art regression algorithms, either the correlations cannot be learned with current technology or not enough time has been spent sampling.

Fast for the user— Most computation time is spent sampling and learning the models. These operations are offline and do not affect the response time of real-time interaction with the user. The response time of the forward mapping and the reverse mapping are proportional to the running time of the regression algorithms, bur are often require less than a few seconds.

Chapter 3

RELATED WORK

Idea of system-level control.... robot swarms (McLurkin 2004) – tightly coupled with domain, describes the behaviors as actions (not properties), splits behaviors into hierarchies.

ABMs have been used to study behaviors in biological systems... Domain specific work interested in system-level behavior: ant lane formation (Couzin & Franks 2003), marching locusts (Buhl *et al.* 2006), fish schools (Parrish, Viscido, & Grunbaum 2002). Most of these studies are qualitative in nature.

particle swarm optimization: emperical study (Shi & Eberhart 1998); more general approach: (Van den Bergh & Engelbrecht 2006)

physics-based control policy (Spears *et al.* 2004) – similar to our approach, but the models are tightly coupled to the domain. the system was designed with system-level models in mind. Algebraic inversion for inverse mapping is nice. Inspires the nonlinear regression approach in AMF.

Macroscopic models of swarm robot systems (Lerman & Galstyan 2002)(Lerman, Martinoli, & Galstyan 2005) – similar in motivation, but models the system in a more specific way (FSAs). Specific to systems in which agents can be modeled as FSAs. Our approach is more general, since it just looks at parameters.

Inversion of neural networks:

- (A Linden and J Kindermann “Inversion of multilayer nets” 1989 – optimization problem solved by gradient descent)
- (Bao-Liang Lu “Inverting Feedforward Neural Networks using Linear and Nonlinear programming” 1990 – formulate the inverse problem as a nonlinear programming problem, a separable programming problem or a linear programming problem)
- (S. Lee and R.M. Kill “Inverse Mapping of continuous functions using local and global information” 1989 – iterative update towards a good solution)
- (Michael I. Jordan work with robot arm “Forward Models: supervised learning with a distal teacher” – asks the question, what configuration of the robot arm will yield this behavior?)

All of these are optimization techniques. They do not return an actual mapping. Also, some of the techniques are restricted to neural networks (and are thus not algorithm-independent).

Experimental platform for messing around with ABMs: (Bourjot – “A platform for the analysis of artificial self-organized systems” 2004) (relevant?)

Chapter 4

BACKGROUND

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 5

DEFINING SYSTEM-LEVEL PROPERTIES

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

For example, in the Wolf Sheep Predation domain, one simple system-level measurement could be the average number of wolves over a thousand time steps. This number can be misleading, because certain configurations will sometimes result in the wolves going extinct (i.e., zero wolves). Other times, the same configurations will have the wolves converge to a stable non-zero population. Therefore, the expected value for the average number of wolves $\hat{w}$ to be:

$$\mathrm{E}(\bar{w}) = \mathrm{P}(extinct) * (\bar{w}|extinct) + (1 - \mathrm{P}(extinct)) * (\bar{w}|\neg extinct)$$

where $extinct$ is whether the wolves went extinct or not, $\mathrm{P}(extinct)$ is the probability wolves go extinct, and $(\bar{w}|\neg extinct)$ is the average number of wolves, given they did not go extinct. $\hat{w}$ is not very stable, because sometimes it is zero and sometimes it is $(\bar{w}|\neg extinct)$.

Making a prediction based on the expected value of $\hat{w}$ will always have a specific amount of error associated with it. To remedy this problem

Chapter 6

THE FORWARD MAPPING PROBLEM

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 7

THE REVERSE MAPPING PROBLEM

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 8

USING META-MODELS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 9

RESULTS

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Chapter 10

CONCLUSIONS AND FUTURE WORK

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Appendix A

CODE SAMPLES

A bunch of code samples will be put here. These will serve as examples to show how to do things or to show how easy they are

A.1 Sample Java/NetLogo Sampling Program

This code example will show how to write a sampler for AMF.

REFERENCES

[1] Buhl, J.; Sumpter, D.; Couzin, I.; Hale, J.; Despland, E.; Miller, E.; and Simpson, S. 2006. From Disorder to Order in Marching Locusts. *Science* 312(5778):1402–1406.

[2] Couzin, I., and Franks, N. 2003. Self-organized lane formation and optimized traffic flow in army ants. In *Proceedings of the Royal Society of London, Series B*, volume 270, 139–146.

[3] Lerman, K., and Galstyan, A. 2002. Mathematical model of foraging in a group of robots: effect of interference. *Autonomous Robots* 13(2):127–141.

[4] Lerman, K.; Martinoli, A.; and Galstyan, A. 2005. A review of probabilistic macroscopic models for swarm robotic systems. In *Swarm Robotics Workshop: State-of-the-art Survey*, 143–152. Springer.

[5] McLurkin, J. 2004. *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots*. Ph.D. Dissertation, Massachusetts Institute of Technology.

[6] Parrish, J.; Viscido, S.; and Grunbaum, D. 2002. Self-organized fish schools: an examination of emergent properties. *Biological Bulletin, Marine Biological Laboratory, Woods Hole* 202(3):296–305.

[7] Reynolds, C. W. 1987. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings)*, 25–34.

[8] Shi, Y., and Eberhart, R. 1998. Parameter selection in particle swarm optimization. *Lecture notes in computer science* 591–600.

[9] Spears, W.; Spears, D.; Hamann, J.; and Heil, R. 2004. Distributed, physics-based control of swarms of vehicles. *Autonomous Robots* 17(2):137–162.

[10] Tisue, S., and Wilensky, U. 2004. NetLogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, 16–21.

[11] Van den Bergh, F., and Engelbrecht, A. 2006. A study of particle swarm optimization particle trajectories. *Information Sciences* 176(8):937–971.

[12] Wilensky, U. 1997a. NetLogo Traffic Basic model. Technical report, Northwestern University.

[13] Wilensky, U. 1997b. NetLogo Wolf Sheep Predation model. Technical report, Northwestern University.