The Swarm Simulation System: A Toolkit for Building Multi-agent Simulations

Nelson Minar [*]
Roger Burkhart [†]
Chris Langton [‡]
Manor Askenazi [§]
http://www.santafe.edu/projects/swarm/

June 21, 1996

Abstract

Swarm is a multi-agent software platform for the simulation of complex adaptive systems. In the Swarm system the basic unit of simulation is the *swarm,* a collection of agents executing a schedule of actions. Swarm supports hierarchical modeling approaches whereby agents can be composed of swarms of other agents in nested structures. Swarm provides object oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. Swarm is currently available as a beta version in full, free source code form. It requires the GNU C Compiler, Unix, and X Windows. More information about Swarm can be obtained from our web pages, http://www.santafe.edu/projects/swarm/.[1]

[*]Santa Fe Institute 1399 Hyde Park Rd. Santa Fe, NM 87501. `<nelson@santafe.edu>`

[†]Deere & Company John Deere Road Moline, IL 61265 `<roger@ci.deere.com>`

[‡]Santa Fe Institute 1399 Hyde Park Rd. Santa Fe, NM 87501. `<cgl@santafe.edu>`

[§]Santa Fe Institute 1399 Hyde Park Rd. Santa Fe, NM 87501. `<manor@santafe.edu>`

[1]Swarm has been developed at the Santa Fe Institute with the support of Grant No. N00014-95-1-1000 from the Office of Naval Research and Grant No. N00014-94-1-G014 from the Naval Research Laboratory, both acting in cooperation with the Defense Advanced Research Projects Agency. Swarm benefited from earlier support from The Carol O'Donnell Foundation, Mr. and Mrs. Michael Grantham, and the National Science Foundation. SFI also gratefully acknowledges invaluable support from Deere & Company for this project.

1 Computational Approaches to Complex Systems

In the sciences, especially in the study of complex systems, computer programs have come to play an important role as scientific equipment. Computer simulations — experimental devices built in software — have taken a place as a companion to physical experimental devices. Computer models provide many advantages over traditional experimental methods, but also have several problems. In particular, the actual process of writing software is a complicated technical task with much room for error.

Early in the development of a scientific field scientists typically construct their own experimental equipment: grinding their own lenses, wiring-up their own particle detectors, even building their own computers. Researchers in new fields have to be adept engineers, machinists, and electricians in addition to being scientists. Once a field begins to mature, collaborations between scientists and engineers lead to the development of standardized, reliable equipment (e.g., commercially produced microscopes or centrifuges), thereby allowing scientists to focus on research rather than on tool building. The use of standardized scientific apparatus is not only a convenience: it allows one to "divide through" by the common equipment, thereby aiding the production of repeatable, comparable research results.

In complexity research, at the Santa Fe Institute and elsewhere, we rely heavily on computers in the course of our investigations. We spend a lot of time constructing our own experimental apparatus in software, the computational equivalent to blowing our own glassware. Unfortunately, computer modeling frequently turns good scientists into bad programmers. Most scientists are not trained as software engineers. As a consequence, many home-grown computational experimental tools are (from a software engineering perspective) poorly designed. The results gained from the use of such tools can be difficult to compare with other research data and difficult for others to reproduce because of the quirks and unknown design decisions in the specific software apparatus. Furthermore, writing software is typically not a good use of a highly specialized scientist's time. In many cases, the same functional capacities are being rebuilt time and time again by different research groups, a tremendous duplication of effort.

A subtler problem with custom-built computer models is that the final software tends to be very specific, a dense tangle of code that is understandable only to the people who wrote it. Typical simulation software contains a large number of implicit assumptions, accidents of the way the particular code was written that have nothing to do with the actual model. And with only low-level source code it is very difficult to understand the high-level design and essential components of the model itself. Such software is useful to the people who built it, but makes it difficult for other scientists to evaluate and reproduce results.

In order for computer modeling to mature there is a need for a standardized set of well-engineered software tools usable on a wide variety of systems. The Swarm project aims to produce such tools through a collaboration between scientists and software engineers. Swarm is an efficient, reliable, reusable software apparatus for experimentation. If successful, Swarm will help scientists focus on research rather than on tool building by giving them a standardized suite of software tools that provide a well equipped software laboratory.

2 Multi-agent Discrete Event Simulation

The modeling formalism that Swarm adopts is a collection of independent agents interacting via discrete events. Within that framework, Swarm makes no assumptions about the particular sort of model being implemented. There are no domain specific requirements such as particular spatial environments, physical phenomena, agent representations, or interaction patterns. Swarm simulations have been written for such diverse areas as chemistry, economics, physics, anthropology, ecology, and political science.

The basic unit of a Swarm simulation is the agent. An agent is any actor in a system, any entity that can generate events that affect itself and other agents. Simulations consist of groups of many interacting agents. For example, an ecosystem simulation could consist of agents representing coyotes, rabbits, and carrots. In an economic simulation, agents could be companies, stockbrokers, shareholders, and a central bank. Simulation of discrete interactions between agents stands in contrast to continuous system simulations, where simulated phenomena are quantities in a system of coupled equations.

Agents define the basic objects in the Swarm system, the simulated components. A schedule of discrete events on these objects defines a process occurring over time. In Swarm, individual actions take place at some specific time; time advances only by events scheduled at successive times. A schedule is a data structure that combines actions in the specific order in which they should execute. For example, the coyote/rabbit simulation could have three actions: "rabbits eat carrots," "rabbits hide from coyotes," and "coyotes eat rabbits". Each action is one discrete event: the schedule combines the three in a specific order, e.g. "each day, have the rabbits eat carrots, then they hide from the coyotes, then the coyotes try to eat the rabbits". The passage of time is modeled by the execution of the events in some sequence.

3 Swarms

The fundamental component that organizes the agents of a Swarm model is an object called a "swarm." A swarm is a collection of agents with a schedule of events over those agents. For example, a swarm could be a collection of 15 coyotes, 50 rabbits, a garden with carrots, and a simple schedule: the rabbits eating the carrots and hiding and the coyotes eating the rabbits (Figure 1). The swarm represents an entire model: it contains the agents as well as the representation of time.

In addition to being containers for agents, swarms can themselves be agents. A typical agent is modeled as a set of rules, responses to stimuli. But an agent can also itself be a swarm: a collection of objects and a schedule of actions. In this case, the agent's behavior is defined by the emergent phenomena of the agents inside its swarm. Hierarchical models can be built by nesting multiple swarms. For example, one could build a model of a pond inhabited by single celled animals. At the highest level, a swarm is created that contains agents: the swarm represents the pond, and each agent represents one animal. The behavior of cells could be defined simply as some algorithm, but a cell is itself a collection of organelles:

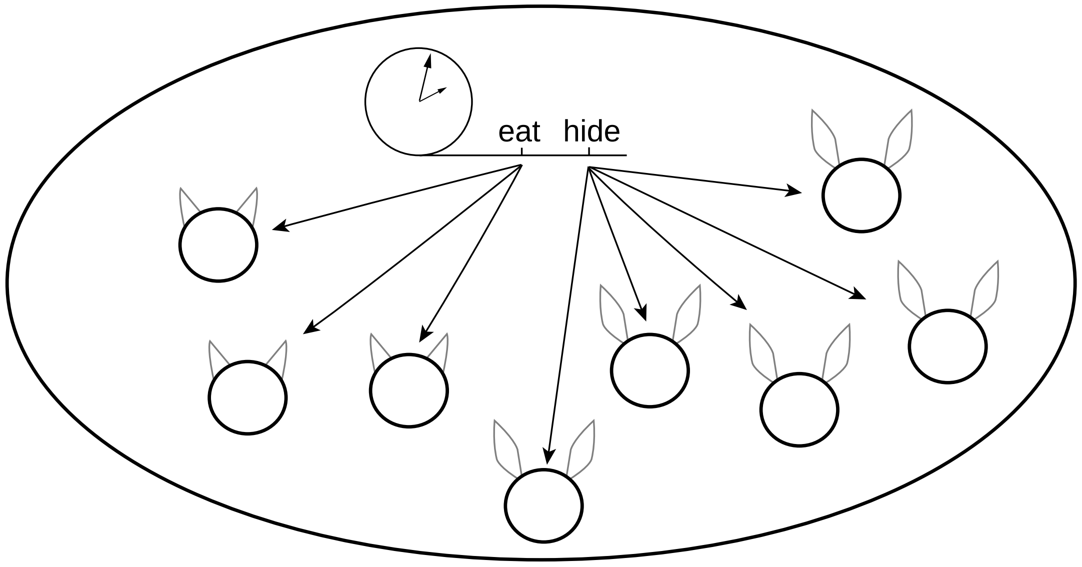

Figure 1: A Swarm of Rabbits and Coyotes

a nucleus, mitochondria, endoplasmic reticulum. Another way to represent a cell is as a swarm of agents, the organelles. Two models are being combined: the pond as a swarm of cells and the cell as a swarm of organelles.

The ability to build models at various levels can be very powerful. Swarm allows users to explicitly build and test multi-level models. A swarm can explicitly represent an emergent structure, a group of agents behaving cohesively as a single agent. Because swarms can be created and destroyed as the simulation executes, Swarm can be used to model systems where multiple levels of description dynamically emerge.

Another use of multiple swarms is to support the modeling of agents that themselves build models of their world. As noted in Hogeweg's MIRROR system [4], in some simulations, especially those where the agents have a cognitive component, an important factor in the system dynamics is an agent's own beliefs about its world. In Swarm, agents can themselves own swarms, models that an agent builds for itself to understand its own world. For example, in an economic simulation of a swarm of companies the researcher might be interested in the theory each company has of its competitors' actions. To model this in the Swarm system, the model builder would give each company-agent its own swarm: these private swarms implement each company's model of the world.

The formalism of the swarm is a natural way of encapsulating a simulation: a swarm simply represents a group of agents and their schedule of activity. The modularity and composability of swarms allows for a flexible modeling system. Swarms can be nested to directly represent multi-level simulations, and they can be used by the agents themselves as models of their own world.

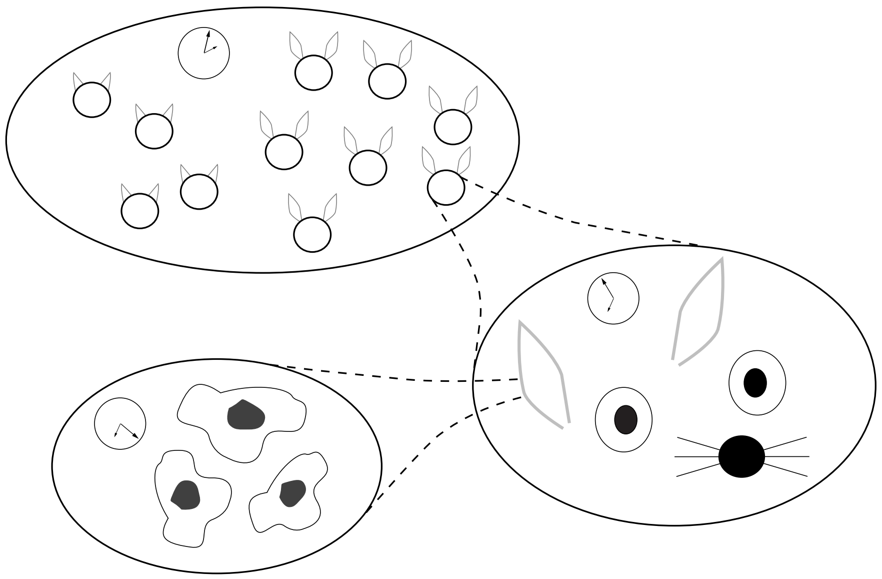

Figure 2: Hierarchical Swarms: Rabbits, Rabbit Parts, Individual Cells

4 Object Oriented Technology

The logical structure of swarms of agents interacting through discrete events is implemented in a straightforward way in Objective C, an object oriented (OO) language. In OO programming software consists of the definitions of various classes of objects. An object is a combination of instance variables for the object's state and methods that implement the object's behavior [7]. In Swarm an agent is modeled directly as an object. Types of agents (generic coyotes) are classes, and specific agents (a particular coyote) are objects, instances of the Coyote class. Each object carries with it its own state variables, but the generic definition of its behavior is provided by the class.

The instance variables for an object directly represent the state of the agent. For example, a particular rabbit's mass could be stored as an integer variable in the class Rabbit. The methods of an object implement the behavior of the agent. Rabbits hide: this is implemented by having a "hide" method defined on class Rabbit. The schedule of activity for a model is then simply a partially ordered series of such actions to be performed on objects. Each action specifies a method to be executed on a target object.

The Swarm system itself is an object framework: a set of class libraries that are designed to work together. There are seven core libraries in Swarm: `defobj`, `collections`, `random`, `tkobjc`, `activity`, `swarmobject`, and `simtools`. The first four libraries are support libraries with potential use outside of Swarm; the last three are Swarm-specific. There are also currently three domain-specific libraries available for Swarm model builders: `space`, `ga`, and `neuro`. The details of each library is discussed below.

In addition to being a natural way to implement multiagent simulations, OO programming is also a convenient technology for building libraries of reusable software. Users can

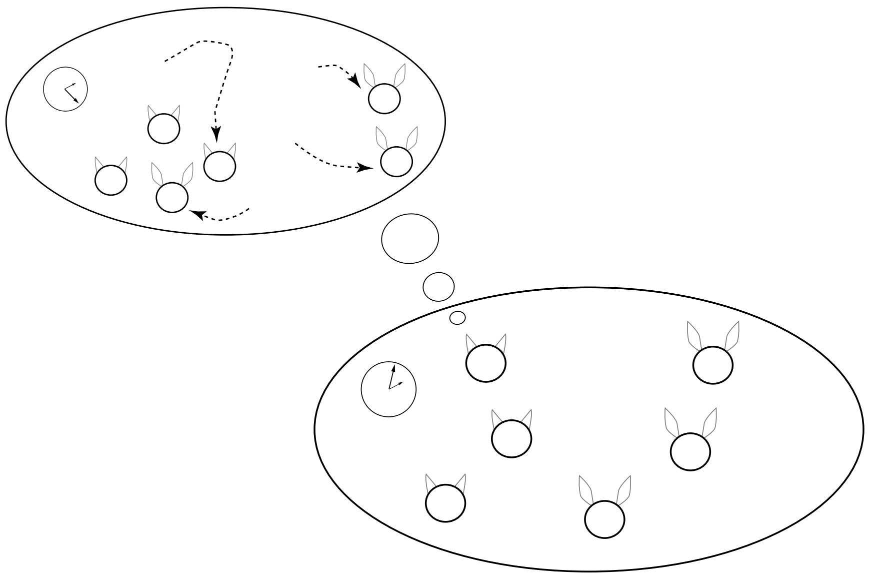

Figure 3: Coyote with a Model for Hunting

start to build models by directly instantiating useful classes from the Swarm libraries. If there is no particular class that has the precisely needed behavior, one can take a preexisting class and specialize it, adding new variables and methods via inheritance. Inheritance and OO encapsulation make writing reusable code easier than traditional procedural programming. This in turn makes it easier for people to share Swarm modeling software they have written, facilitating the exchange of ideas and techniques within simulation communities.

One addition to standard OO programming that Swarm implements is the "probe" facility. In most computer programs, it is enough that the program does what you want — that Windows doesn't lose your files, that your word processor can print out your papers. But in simulation it is important that all aspects of the computation be observable, that it be easy for the researcher to measure data from a running model. The Swarm system defines the ability for any object to be probed. Probes allow any object's state to be read or set and any method to be called in a generic fashion, without requiring extra user code. Probes are used to make data analysis tools work in a general way and are also the basis of graphical tools to inspect objects in a running system.

5 Structure of a Swarm Simulation

The core of a Swarm simulation is the modeled world itself. In the simplest case, a model consists of one swarm inhabited by a group of agents and a schedule of activity for those agents. The agents themselves are implemented as objects. Agents are created by taking a class from the Swarm libraries, specializing it for the particular modeling domain, and then instantiating it, one object per agent. For example, an agent that is a neural network could start by taking a general purpose neural network class from the `neuro` library, adding extra methods needed for the specific type of network, and then creating an instance of it to be the actual neural network.

In modeling it is common (but not universal) to speak of agents as living in an environment. Many simulation platforms fix the environment as a particular type; two dimensional grids are common. A distinguishing feature of Swarm is that there is no design requirement for a particular kind of environment. For instance, a coyote/rabbit model might have the rabbits living in the environment of a garden. In Swarm, this environment is itself just another agent. The garden is simply an instance of a user-defined garden agent, perhaps based on a cellular automata to simulate growth of carrots. The garden agent might have a special status in the model, but in the underlying software it is treated no differently than any other agent. In the general case, the environment for the agents is the agents themselves: some agents might have a larger influence than others, but in Swarm they are considered fundamentally equivalent.

Once a user has defined the agents and established their relationships, the last step in building the model itself is to put the agents together into a swarm. The user writes a schedule of activity for the agents, defining how time is simulated in the system by creating a set of actions in a specified ordering. Schedules are built by creating instances of data structures from the `activity` library, filling them in with ordered object/message pairs. Once the schedule is completed the model swarm is ready to be executed.

A model running by itself is not very interesting: data collection tools are needed to observe the model and record what is happening. In Swarm measurement happens by the actions of observer agents, special objects whose purpose it is to observe other objects via the probe interface (Figure 4). For example, one observer agent might be watching the number of rabbits and producing a time series graph of population dynamics. Another observer agent could track the spatial distribution of the coyotes, storing data to a file for later analysis.

The observer agents themselves are a swarm, a group of agents and a schedule of activity. By combining this swarm with a model swarm running as a subswarm of the observer, a full experimental apparatus is created. By using hierarchical swarms to separate data collection from the model, the model itself remains pure and self-contained, a simulated world under glass. Different observer swarms can be used to implement different data collection and experimental control protocols, but the model itself remains unchanged.

6 Swarm Libraries

Swarm libraries serve two major functions. The libraries are a set of classes that model builders can use by direct instantiation. For many objects, especially highly technical ones such as schedule data structures, it's likely that all a user will ever do is use the classes as provided. But in addition, one can use Swarm libraries by subclassing them, specializing particular classes for particular modeling needs. Both modes of using the Swarm libraries are important; Swarm is designed to facilitate both as appropriate.

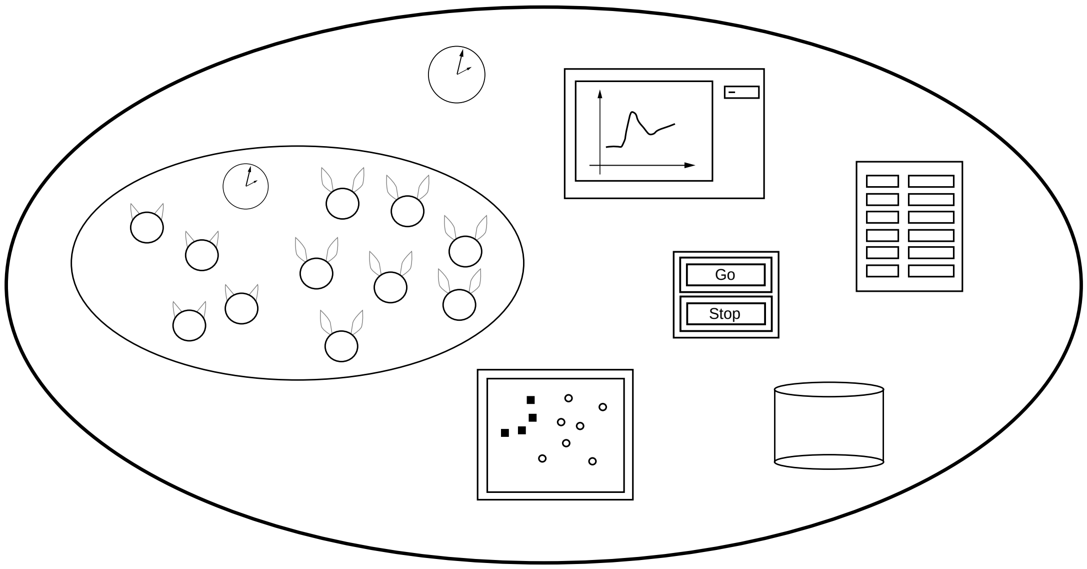

Figure 4: A Swarm of Observer Agents Measuring a Model

6.1 Simulation Libraries

The main novelty in the Swarm system is the design of the simulation-specific libraries themselves. The `swarmobject`, `activity`, and `simtools` libraries are the center of the Swarm modeling paradigm; their use is essential to all Swarm models.

The `swarmobject` library contains the core classes upon which agents in Swarm models are based. At the center of this library are two classes: `SwarmObject` and `Swarm`. `SwarmObject` is the root class for all simulated agents: all agent classes inherit behavior from it. `SwarmObject` defines the basic interface for memory management as well as the support for probes. In addition, the `Swarm` class is in the `swarmobject` library. Model swarms and observer swarms are written by using code inherited from this base class.

The `activity` library contains the heart of the simulation mechanism, the scheduling data structures and execution support. The underlying structure of the activity structures are partial orders, time dependencies between two events. A common special case of partial orders is a time-ordered schedule, a clock with specific events at particular times. These schedules are implemented in `activity` as a collection of actions sorted by timestamp.

Ambiguity can occur in partial orders and time-based schedules as a result of two or more actions scheduled at the same time or in the same relative order. Swarm resolves such ambiguity by defining a “concurrent group type,” an explicit indication of how to execute a group of actions that are defined at the same time. Options include running the group in an arbitrary, fixed order; running the group in a random order every time; or actually running each action concurrently, for future implementation on parallel machines. The explicit notation of a concurrent group type helps to expose and remove any hidden assumptions in the time structure of a model.

The final Swarm library specifically for simulation is the `simtools` library, a miscellaneous collection of classes needed to build simulations. `simtools` contains classes to control the execution of the entire simulation apparatus. Two modes of operation are supported: a fully graphical mode for interactive exploration and a batch mode for offline data collection. `simtools` also contains the data analysis and display support. The library contains classes that can generate summaries of statistical data, draw time series graphs, etc. Data analysis objects are completely generic in their application; users specify the particular data to collect by creating probes on the observed objects.

6.2 Software Support Libraries

In addition to simulation support libraries, Swarm also contains libraries that were written to support modeling but could also have applications beyond simulation. Whereas the simulation-specific libraries are used to define and execute the Swarm style of simulation, the `defobj`, `collections`, `random`, and `tkobjc` libraries do the more prosaic job of encapsulating the basic engineering tasks that are needed to write effective software.

`defobj` and `collections` are a set of general-purpose tools for building OO programs. `collections` implements the container classes used to track objects in a system: maps, lists, sets, etc. `defobj` defines the infrastructure for the Swarm object model. It augments the basic Objective C runtime to support tunable implementations, a split between object interface and implementation that provides highly optimized specific solutions to customizable forms of abstract object designs. This framework is used by the `collections` library to provide a group of classes that are simple to use but have complicated machinery underneath to efficiently handle a wide variety of object tracking needs.

The `random` library gives the user a suite of random number generators. In computer simulation the quality of random number generators is absolutely essential: it is very easy to have subtly wrong results because of a generator with biases or correlations. The Swarm random number library includes several classes of generators drawn from published research. It includes a bibliography as well as tested implementations. Because the library itself is object oriented it is simple to have multiple, independent random number streams, a feature that aids repeatability of experiments.

The final library of basic programming support in Swarm is `tkobjc`, a graphical user interface library based on Tcl/Tk [6], which in turn is based on X windows. The classes in `tkobjc` encapsulate the basic objects needed to make a user interface: buttons, raster displays, histograms, graphs, forms, etc. The library itself is intended to be very simple and easy to use, implementing enough for common graphics needs. If a `tkobjc` class is not powerful enough, a user can use the lower Tcl/Tk level to write more specific code.

6.3 Model-specific Libraries

In addition to the support and simulation libraries required for all Swarm applications there are several optional libraries that can be used for particular modeling domains. Libraries exist to support two dimensional spaces, genetic algorithms, and neural networks. Development effort for Swarm so far has concentrated on the required libraries; now that the Swarm infrastructure is in place, effort is shifting towards building more model-specific libraries. There are many opportunities for user-contributed code.

The Swarm distribution comes with a simple space library, a set of classes for two dimensional discrete lattices. These sorts of spaces are common in ecosystem simulations. The base class in `space` is an two dimensional array that stores objects or integer values at particular grid points. Several classes inherit from this base to provide dynamics, for instance a cellular automata approximation to diffusion. The current `space` library is merely a suggestion of the kinds of environments a model could use: in the future, we plan to have spaces with continuous values and dynamics defined by differential equations. Spaces with other topologies are also crucial: three dimensions, non-discrete coordinates, and arbitrary graph structures are all needed by applications.

The final libraries currently available for Swarm are the `ga` and `neuro` libraries. This code represents Swarm's first user-contributed libraries, software written by Juan J. Merelo while visiting the Santa Fe Institute. `ga` provides a set of classes for basic genetic algorithms; `neuro` implements a variety of neural networks. As Swarm matures, we hope that it will grow with the contributions of users.

7 User Community

Swarm is a service to the community of researchers building computers simulations. Computer programs have become an important aspect of experimental science and yet few general purpose tools exist to help people write models. The goal of Swarm is to provide consistent experimental tools in the form of libraries of carefully designed, implemented and tested code. Such libraries can also be developed and exchanged within user communities that organize and maintain the model-building components applicable to their shared needs.

Swarm defines a structure for simulations, a framework within which models are built. The core commitment is to a discrete-event simulation of multiple agents using an object-oriented representation. To these basic choices Swarm adds the concept of the "swarm," a collection of agents with a schedule of activity. The swarm is the basic structural element of a model: it is the basic collection that allows scaling within large models, and also supports the modeling of complex, multi-level dynamics including agents that model their own world. In all these choices, the goal of Swarm is to enable a higher level of representation for simulations, thereby making it easier to understand, implement, repeat, and communicate computer models.

Since Swarm's first limited beta release in October 1995, some thirty groups of users have installed Swarm and are actively writing models with it. There is already one finished paper that presents results accomplished with Swarm [5]; in the coming months we expect a series of further models developed by Swarm users, with published results following close behind.

In the next few months we will have a finished, version 1.0 release of Swarm. Our goal is to create a distribution that is simple, complete, well documented, and bug-free. Our longer term goal is to foster a community of modelers, researchers who use computer software for experimental science. Swarm is already helping to provide focal point for discussion of simulation techniques and methodology. Swarm can also facilitate the sharing of modeling components and libraries within particular research communities, fostering an important form of intellectual exchange. Finally, a formalized framework for model definition establishes a necessary standard of specification for computer programs used as tools in experimental science.

References

[1] Roger Burkhart, Manor Askenazi, and Nelson Minar. Swarm release documentation. Available as http://www.santafe.edu/projects/swarm/swarmdocs/set/set.html.

[2] NeXT corporation. *Object Oriented Programming and the Objective C Language.* Addison-Wesley, 1993. Also available as `http://www.next.com/Pubs/Documents/OPENSTEP/ObjectiveC/objctoc.htm`.

[3] D.L. DeAngelis and Louis J. Gross. *Individual-based models and approaches in ecology.* Chapman and Hall, 1992.

[4] P. Hogeweg. Mirror beyond mirror, puddles of life. In Chris Langton, editor, *Artificial Life.* Addison-Wesley, 1989.

[5] John Carnahan, Song-gang Li, Carlo Costantini, Yeya T. Touré, and Charles E. Taylor. Computer simulation of dispersal by *Anopheles Gambiae s.l.* in West Africa. In *Artificial Life V*, 1996.

[6] John K Ousterhout. *Tcl and the Tk toolkit.* Addison-Wesley, 1994.

[7] David Robson. *Smalltalk-80: the language and its implementation.* Addison-Wesley, 1985.