The Swarm Application Framework

Don Miner and **Marie desJardins** and **Peter Hamilton**
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County
1000 Hilltop Circle, Baltimore, Maryland 21250
{don1, mariedj}@umbc.edu

Abstract

The Swarm Application Framework (SAF) is a tool that makes the development of swarm applications more intuitive. Traditionally, swarm applications are created by programming several low-level rules. This approach leads to several problems in designing and testing swarms, which serve as inspiration for the features of SAF. SAF encourages a new paradigm for designing swarm applications: engineers can interact with a swarm at the abstract (swarm) level instead of the individual (agent) level. In this paper, we discuss the design of the framework, how agents and rules in SAF operate, and a planned rule abstraction feature.

Introduction

Developing swarm applications, in which the individual actions of many agents result in emergent system behavior, presents unique challenges. To build such systems, an engineer programs the rules that the agents must follow, which *happen* to produce emergent behavior. Many examples of programming by rules exist in the literature (Flake 1998)(Kennedy & Eberhart 2001). In this paper, we propose the Swarm Application Framework (SAF), which improves the way in which these applications are created. Our goal is to allow the engineer to design swarm applications from the *abstract* (swarm) level instead of the *individual* (agent) level.

The classic example of a swarm application is Reynolds' computer animation technique, "Boids" (Reynolds 1987), in which particles flock together in groups, following three rules: attract to the center of mass; match neighbors' velocity vectors; and avoid closeness. None of these rules explicitly tell the boids to flock; they are simply telling the boids to perform simple behaviors. A more complex swarm application might consist of onboard computer systems in automated cars that drive on streets. Many rules would be necessary to control the individual cars and traffic lights. These rules may be different from agent to agent, depending on the type of agent. For example, ambulances may follow different rules than passenger vehicles. Debugging this swarm application would be extremely difficult: because of the swarm's complexity and heterogeneity, it may be impossible in practice to determine which rules are causing an undesired swarm-level behavior. Tuning an existing application in such a domain might involve adjusting many low-level variables and creating new rules that interact with existing ones.

SAF allows the engineer to design swarm applications from the top down, so the design problem becomes more manageable. Rules are organized into a hierarchy, so modifying behaviors can be done with top-level rules instead of modifying multiple low-level rules. For example, suppose that in the boid application we wanted to make the individuals "looser": that is, further apart and less cohesive. This modification requires weakening all three rules. With Reynolds' original implementation, the engineer would have to modify each rule. With a boid program written in SAF, the programmer could create an abstract rule called "looseness" and modify this instead of each low-level rule. When the looseness rule is changed, it propagates the changes down to the low-level rules. This top down, abstract design process can be extended to more complex applications such as the traffic example. We are also developing methods to specify abstract rules given a body of existing rules and to have the framework learn the variable correlations autonomously.

The goals of SAF are:

- Make the development of swarm applications intuitive.
- Make rule abstraction a simple process, thus promoting the use of rule hierarchies.
- Provide a collection of modules that developers can use to quickly build new applications.
- Enable engineers to focus on the behaviors and properties of the swarm, rather than on low-level details of individual agent behaviors.

Architecture Overview

SAF contains two layers: the *framework* and the *library*. Applications written in SAF reside below these in the *application* layer.

The framework layer is the most abstract and determines the fundamental organization of all SAF applications. This layer contains abstract classes that are the core components of SAF applications. Only classes that have to do with

swarm behavior are included in this layer. The framework layer contains top-level classes such as Rule, Agent, Environment, and Sensor, as well as more specific classes such as Invoked Rule, Watch Rule, and Passive Agent.

The library layer contains an array of tools and other useful classes for swarm applications. The classes in this layer are situational (yet domain-independent) and may not be used in every application. Modules in this layer include communications, basic rules, visualization packages, data analysis tools, and basic environments. This layer will grow as development of the framework continues and common themes are identified.

The application layer is where swarm applications are developed. The engineer extends the classes provided by the more abstract layers. Analytics, user interfaces, and other supplementary components created by the developer reside in this layer.

SAF Agents and Rules

Agents in our framework are the deliverable end result of an application written in SAF. Although applications are designed at the swarm level, the final compiled version is deployed into an agent program that is its own individual process. This separation ensures the flexibility of how these agents can be deployed. Given the necessary communication abilities, these agents could be placed over an array of desktop computers, a collection of individual robots, or hundreds of mobile wireless devices.

Each SAF agent is made up of two major components: a body of rules and a data repository. The rules dictate and control the behavior of the agent by performing all of the sensing and acting. They are organized as a hierarchy, where rules at the top are the most abstract. For example, "form geometric shape" could be an abstract rule for a group of agents that form shapes in a two-dimensional Euclidean space. There are many underlying rules such as "form corner," "avoid closeness," and "become corner" that support the abstract "form geometric shape" rule. The agent also contains a data repository, which allows the rules to maintain the state of the agent, while the rules themselves can remain stateless, simplifying their design.

Implementation Status

We have implemented a prototype of the framework and are using it to develop simple swarm applications in a two-dimensional Euclidean environment. These swarm applications use simple rule hierarchies to abstract low-level factors. In one application we have developed, agents form geometric shapes by sensing their neighbors and changing their own color, using no other means of communication between agents. Screenshots of this application in action can be seen in Figure 1. More screenshots and videos are available on the SAF project website[1].

Learning Abstract Rules

Rules are the cornerstone of the SAF framework, driving all agent behavior. Our rule abstraction mechanism in SAF is one of the major reasons why creating swarm applications with SAF is easier than creating them from scratch. In many cases, abstracted rules can be defined explicitly by the swarm application writer. We are currently exploring methods by which new abstracted rules can be *learned*, given a set of existing rules. The properties to be exhibited by the new abstract rule would be specified by the user; SAF would then use simulation and generalization to modify and combine existing rules to achieve the new desired behavior.

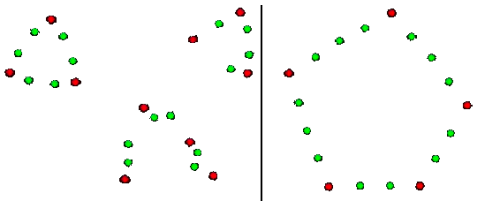

Figure 1: SAF Agents in the process of forming triangles (Left) and agents that have formed a pentagon (Right).

Consider the example of agents forming a circle in a two-dimensional environment, which can be accomplished with three rules:

- Avoid closeness to other agents.
- Tend towards the center of mass.
- Move towards the average distance from the center of mass. That is, calculate other agents' distance from the center of mass, average the values, and then tend towards this average.

Each of these rules has a scalar factor that specifies its relative strength. Over time, an equilibrium is reached and the agents form a stable circle. The radius and shape of this circle depends on the scalar factors in each of the three rules. Now suppose that the programmer wishes to abstract these rules into a single rule to "form a circle of radius r." Our research goal is to learn the relationship between the factors in the three rules so that the abstracted rule "knows" what to set the scalar factors to when a radius is specified. With the learned abstract rule, the user can command the swarm to form a circle of a specific radius instead of modifying the individual rules. Such learning can also be transferred to other situations through generalization to swarms with a different number of agents and different environmental constraints. The abstraction based, top-down swarm design framework provided by SAF, combined with the proposed learning extension, will provide a valuable and flexible tool for designing a variety of useful swarm applications.

References

Flake, G. 1998. *The Computational Beauty of Nature: Computer Explorations of Fractals, Chaos, Complex Systems, and Adaptation*. MIT Press.

Kennedy, J., and Eberhart, R. C. 2001. *Swarm Intelligence*. Morgan Kaufmann.

Reynolds, C. W. 1987. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.* 21(4):25–34.

[1] http:/maple.cs.umbc.edu/~don/projects/SAF/