

# An Adaptive Substrate for Agents in an ALife Simulation

Joseph Lewis

Zach Barrow

Department of Computer Science  
San Diego State University  
5500 Campanile Dr. MC 7720  
San Diego, CA 92182  
(v): 619-594-2014 (f): 619-594-6746  
{ [lewis@cs.sdsu.edu](mailto:lewis@cs.sdsu.edu) , [barrow@rohan.sdsu.edu](mailto:barrow@rohan.sdsu.edu) }

SYS Technologies  
5095 Murphy Canyon Rd Suite 240  
San Diego, CA 92123  
(v): 858-244-6649  
(f): 619-594-6746

**Keywords:** Predator-prey, agents, Copycat, Starcat, adaptation

*Abstract—We describe a novel configuration of the adaptive computational framework Starcat, a descendant of the Copycat architecture of Mitchell and Hofstadter. The arrangement of components of this instance of the framework realizes an artificial life simulation, a dynamic collection of predator-prey interactions. Each of the agents in the simulation has its behavior regulated by a concept-network known as the slipnet. Various experiments are presented with different slipnets to determine what sorts of collective behavior can arise from their interactions.*

## 1.0 Introduction

Predator-prey simulations are legion in ALife research, some of which are discussed in [1,2]. The work we present in this article investigates yet another such simulation, but it is one wherein the behavior of the agents is produced by a novel cognitive architecture. This architecture is inherited from the Copycat program of Hofstadter and Mitchell [3,4].

We have been engaged in the development of a general framework for adaptive computation called Starcat. Starcat is a generalization of the original Copycat architecture, designed among other things to ease its instantiation for particular application domains. Previous work has discussed this framework [5,6]. The application we discuss here, called Multicat, is one of several current applications being used to test the Starcat adaptive computational framework. There is much in common between Starcat and its ancestry in Copycat, Metacat [7], and Madcat [8,9] (see [4] for a partial list of the other domains to which this architecture has been applied). We refer to this lineage collectively as the \*-cat architectures, also read as “starcat” but where the \* is meant to be interpreted similarly to the Kleene closure operator, encompassing all such related projects. The details of the \*-cat architectures are discussed below.

### 1.1 Introduction to the \*-cat Architectures

The original Copycat architecture addressed the problem domain of letter-string analogies. It included three components. Other instantiations of the Starcat architecture may modify somewhat these components or introduce new ones. For the purposes of understanding basic functioning, it will suffice to consider the three Copycat components. These are the workspace, the slipnet and the coderack. The Copycat architecture, like all emergent computational systems, embraces the notion from the study of complex adaptive systems that global-scale behavior is achieved by the myriad interactions of short, locally-focused agents—here, pieces of executable code known as codelets. Codelets are emitted from the slipnet to be enqueued in the coderack and later dequeued and sent to the workspace where they are executed. When they are executed in the workspace they attempt to build micro-scale perceptual structures, the collective effect of which is to direct the program toward its next action (in Copycat, to produce an answer). The success of such structure-building activity in the workspace adds activation to various nodes in the slipnet. The slipnet is a collection of nodes and links among those nodes that roughly represent concepts from the problem domain and their interrelationships. The activation levels of nodes and lengths of links between them can change over time, making the slipnet an adaptive structure that captures what are currently the most relevant concepts and relationships for the program’s interaction with its “environment” (some problem instance). Highly active nodes in the slipnet emit codelets into the coderack, as well as spread activation to linked neighbor nodes. In order to ensure that the system does not get stuck only following one path of possible perceptual activity, the coderack, essentially a priority queue, is probabilistic. This means that, while it tends to select the higher-urgency codelets stored within it, it has some chance of selecting the lower-urgency ones as well. The system is guaranteed to engage in what Hofstadter calls the “parallel terraced scan”—several competing possibilities for action are explored, each in proportion to their promise in the current problem. Finally, there is a feedback mechanism that gives rise to the self-organization in the

system. This is the temperature, a measure of the coherence of the perceptual structures being created in the workspace. When the temperature is high (very little coherent structure), as at the beginning of a run, the system tends to be rather random, parallel and data-driven in its perceptual behavior (the building of structures in the workspace). When the temperature is low (much coherent structure), as toward the end of a run, the system tends to be more deterministic, serial and goal-driven.

The Madcat architecture was an attempt to transplant the essence of the Copycat architecture into a real-world domain with constantly changing input. The target domain was mapping and navigation in a mobile robot. The Madcat architecture used codelets to build perceptual structures pertaining to spatial relationships among sonar readings from the robot in workspace-like components called snapshots. The workspace proper contained multiple snapshots, and it was the task of still other codelets in the system to build structures between these snapshots that captured persistent patterns among that data. This engendered the introduction of a new component, the mapnet, in which these long-term patterns could be captured. Lawson and Lewis [10] have discussed the significance of this work in the context of understanding the role of representation in cognition and spatial awareness. Madcat, wherein some generalization of the ideas of Copycat to arbitrary domains was engaged, led ultimately to the development of the Starcat framework.

## 2.0 Design of the Starcat Architecture

The designers of both Copycat and Madcat recognized that the basic architecture could be generalized. The Starcat architecture is a framework that generalizes the ideas in these earlier works, and which can be instantiated for particular problem domains. The motivation for the experiments here, and others currently underway from different problem domains and with different configurations, is to test this framework. The results of such tests will guide the ongoing effort of refining the framework. In particular we are watchful for those features of various problem domains that introduce new demands on the generic framework. For more information about the Starcat project, see <http://starcat.org>.

Probably the most important change from previous \*-cat projects to Starcat is that the components are completely asynchronous and event-driven, each running in its own thread. Furthermore, the drivers for behavior of these components are decoupled from the data structures that are the essence of the components. Both of these changes carry more deeply the biological metaphor of the cellular cytoplasm that dates back to the first predecessors of Copycat [4]. Moreover, this facilitates domain-driven variations regarding the connectivity, number and nature of the components as well as the activity which engages them. Codelets are still the mediator of interactions among components, though now to an even greater degree. Each codelet has behavior that is specific to any of the components that might be listening for codelet events. The codelets have become like RNA, with specific interpretations dependent on the particular environment (component) in which they are executed.

There are several other changes in Starcat worthy of note. Component update behavior (e.g., recalculating the temperature of the structures in the workspace or performing the spreading of activation between and emission of codelets from slipnet nodes) was once regulated by a central control algorithm triggering updates in lockstep fashion after some number of codelet executions in the workspace. Now the update behavior is more general and can range from the same lockstep behavior (possible with semaphores) to arbitrary updates for each component regulated by its own update thread—none using a central control. Also, the semantics of codelets have been expanded. A codelet that executes in the workspace can add activation to different nodes depending on whether it succeeds or fails to build a structure. Furthermore, a slipnet node can be configured to emit codelets in a continuous fashion, in proportion to its activation, or as with the predecessor architectures, it may be configured to emit codelets only when it is fully active. Finally, there are many parameters that the generalized framework has exposed for ease of variation. How these parameters affect system behavior is domain-specific, requiring post-instantiation experimentation; but the design facilitates this. In particular, the slipnet, which has always been the most domain-dependent component, is expressible in XML format, more easily facilitating experimentation with different configurations by domain experts as much as by developers. Other aspects of the program's functioning are specified in a similar way. It will be from the building of more and more applications that we will know which of these aspects are most useful.

## 3.0 Multicat System Design

Multicat is an application using the Starcat framework. It is an ALife simulation wherein different types of agents interact in a resource-restricted environment. Because Multicat is a program implemented using Starcat, the agents in the simulation are significantly different in their design and capabilities than in standard ALife simulations where agents are rather simplistic and static in the types of behavior they manifest. Multicat agents are intended to take advantage of the adaptive qualities provided by the Starcat framework. This is largely achieved with the first important design component: a slipnet per agent. This will be elaborated below. The environment in which the agents interact is also a component in the Starcat

framework, playing the familiar role of the workspace (except here it is shared among agents) and providing a “domain physics” for the agents (constraints on their movement and an energy source/sink). The final component of the design is the algorithm which dictates how the agents run. It is considerably different in nature from the standard Starcat process of instantiation and engagement. There are also certain parameters outside of the Starcat framework that are peculiar to the implementation of the agents in Multicat. These parameters will be briefly discussed as they relate to the overall behavior of agents.

### **3.1 Slipnet and Coderack**

The implementation of the agents in the ALife simulation is probably the most important aspect of the design of Multicat. In previous instantiations of \*-cat applications such as the original Copycat, the program runs with one slipnet, coderack, and workspace. In Multicat, each agent has its own dedicated slipnet and coderack. These components are encapsulated within each agent. Such a design enables each agent in the simulation to have potentially very different behaviors from any other agent. In previous \*-cat applications, the slipnet is used as the long-term memory and intelligence of the system. In Multicat it is used as the main source of behavior for agents. All actions are precipitated by activations of nodes within the agent’s slipnet. There are two basic types of nodes in an agent slipnet: emotion nodes and action nodes. Emotion nodes do not directly cause a specific action to occur when active, but their placement in the slipnet, using links to action nodes or other emotion nodes, does provide a path of spreading activation for activities related to a particular emotion.

One of the issues that Starcat attempts to overcome is the difficulty in configuring a slipnet for a given problem domain. Unfortunately, it cannot at the moment figure out what a “good” slipnet is, which is highly problem-dependent, but it does have the aforementioned facility for creating slipnets using configuration files written in XML. Indeed, finding proper configurations of slipnets with which to derive successful agent behavior is a prime experimental goal of Multicat. There are many agents interacting in a particular run, and consequently, many opportunities for testing different slipnet configurations. Agents could have potentially many different slipnets, but all agents in Multicat share a relatively small set of actions that they can undertake.

### **3.2 Codelets**

Agents interact with the environment and each other through the execution of codelets in the workspace (called the SimulationWorld in Multicat). Codelets for an agent are probabilistically chosen based on their urgency in the coderack. Since each agent maintains its own coderack, it maintains its own state of current and potential activity without regard to other agents. Internal to each agent the interaction between the slipnet and coderack is exactly the same as in previous \*-cat architectures, but that interaction drives a single element in the application rather than the application itself.

### **3.3 SimulationWorld**

The SimulationWorld is a modified version of the workspace in Copycat. Since Starcat attempts to abstract the process that Copycat uses to solve letter-string analogy problems in a domain-neutral way, the current Starcat workspace is primarily a storage space, with methods for storing and then retrieving structures randomly, biased by object salience and selected by object type.

The workspace provides an object hierarchy of the common objects found in previous \*-cat applications, including descriptors, description types, bonds, groups, correspondences, etc. In Multicat, each agent is a workspace object that exists in the SimulationWorld. The workspace in Starcat handles all objects in a polymorphic way, so it is configurable in terms of what objects one would like to use. Of course, one can also create their own types to use in the workspace. Such is the case of agents in the SimulationWorld. Unlike the coderack and slipnet, the SimulationWorld is a shared component and is used as the environment for the agents. Every environmental construct in the SimulationWorld is either a HexObject (Agent or Grass living in a world of hexagonal grid elements), a Bond (currently there are only TargetBond objects), or a hexagonal tile (an empty space). One or more agent or grass objects can occupy a single hex object, and bonds can form between agents. These structures are all extensions to the basic structures provided by the core Starcat workspace.

### **3.4 Control Algorithm**

The final design feature to consider is the process flow for the Multicat application as a whole. The algorithm is different from Starcat’s asynchronous model because it is attempting to run as an ALife application. Achieving this requires a more controllable algorithm to ensure that each agent gets treated fairly and to make the assessment of agent performance easier (because we do not have to take into account thread issues). In Multicat, the control algorithm is similar to the original Copycat. Each agent in each “round” (Multicat runs go to 1000 rounds), gets to pop (execute) 15 codelets, then does a slipnet update, and then the SimulationWorld updates. The choice of 15 codelet pops is inspired by Copycat’s use of that number in its control algorithm.

### 3.5 Miscellaneous Agent Parameters

Agents are also imbued with state that is not appropriately stored in any of the traditional Starcat components. Each agent, for instance, has both energy and health. These values are decreased with every action the agent takes, except for eating and resting, which increase said values, respectively. An agent dies when its health (not energy) goes to zero or below. These parameters have been chosen arbitrarily and are meant to provide a kind of ‘currency’ of activity to limit the amount of action agents can undertake. Some parameters also extend beyond the agents themselves into the codelets that drive agent behavior. Codelets are interesting in that they control the agent, but the agent itself is also using codelets to execute behavior on its behalf. This results in agent action properties (internal state) that have been externalized into these behavior codelets to affect their decisions. This should be kept to a minimum, but from a design perspective, it is necessary in some cases, e.g. to filter the types of objects in the SimulationWorld that an agent can eat.

## 4.0 Experimental Setup

The design of Multicat makes possible variation of two main aspects: 1) each agent type can be driven by different slipnets (and the instantaneous state of each instance of these slipnets varies dynamically); and 2) within each simulation run, different combinations and numbers of agent types exist. There are four main agent types (slipnets): HA1, HA2, PA1, and PA2. The agent-type names are arbitrary, but the essential differences between the agent types will be discussed as a function of their slipnet construction. Following this, we explain the population permutations that are explored in each simulation run.

### 4.1 Agent Types and Corresponding Slipnets

As previously mentioned, there are four main agent types corresponding to four slipnet configurations. We will discuss the common aspects across all slipnets and then the differences among the agent types. Multicat is an entirely different problem domain than Copycat, so many of the semantics that describe the slipnet in Copycat do not make sense. In Multicat, most slipnet nodes correspond to actions rather than workspace structures. There are also three “emotion” nodes in each slipnet, Hunger, Contentedness, and Fear. These nodes are laterally-linked (i.e. there are no semantics to the link, it carries only spreading activation) to action nodes and define what an agent will do when certain emotions are activated. Fear, hunger, and contentedness are all activated proportional to certain agent and environmental conditions. An agent is only content if there are few agents in proximity, its health and energy are decent, and it is not being targeted. Fear works in the opposite manner. Hunger occurs if agent energy and/or health are low. There are a number of action nodes available to the agents. Actions nodes break down into a couple of broad categories: moving, attacking, targeting, and eating. Targeting is different than attacking in the fact that targeting just creates a bond between agents. Attacking is an actual attempt at damaging a targeted agent. The agent has a probability of successfully attacking and damaging, of doing no damage, or of being damaged in the attack. The damage is based on the agent’s energy or the attacked agent’s energy if damage happens to be done to the attacker. All of the agent types have a facility for eating (either grass which is randomly generated, or dead agents). Also, all agents move, although some move for different reasons and there are a couple of types of movement. Other actions are dependent on the slipnet configurations. We turn now to the specific slipnet configurations for the four types of agents used in the experiments.

*HA1:* HA1 agents are “active” hunting agents. When hungry, they probabilistically move towards areas of higher agent density and attempt to make target bonds to prey agents. They also tend to move when content. Both HA1 agents and HA2 agents target and attack agents that have targeted them when they are in a state of fear (high activation of that slipnet node).

*HA2:* HA2 agents do not move when content. Other than this, their slipnets are identical to HA1 agents. They represent a more “lazy” population of hunters who only rest rather than move when content.

*PA1:* PA1 agents are much like HA1 agents in that they will move or rest when content and they also move around when hungry. PA1 agents are aggressive in their defense, targeting and attacking agents that have targeted them.

*PA2:* PA2 agents are lazy agents like HA2 agents. Additionally, when in a state of fear, they do not attack but move towards areas of lower agent density, hopefully away from agents targeting them.

### 4.2 Population Configurations

We conducted nine experiments using the four agent types. These nine experiments correspond to the nine possible combinations of agent types in which there is at least one hunter and one prey type in the simulation. Table 1 below summarizes the populations of the agents in each run. Each experiment consists of 100 runs of the simulation, with results averaged by agent type since we are looking for overall behavior by agent type. The gathered metrics for each agent type in each experiment include the number of times each node was activated, the number of times each type of codelet was popped (executed by the agent), and various statistics like the number of times each agent attacked, was attacked, ate

something, or moved. Finally, the average age of the agent type when it died was recorded. The averages for each agent type in each experiment are then processed to obtain the minimum, maximum, average, median, and standard deviation of each result over the 100 runs in each experiment.

Table 1: Agent Populations by Type and Experiment

Agent Type	Exp1	Exp2	Exp3	Exp4	Exp5	Exp6	Exp7	Exp8	Exp9
HA1	20	20	20				20	20	20
HA2				20	20	20	20	20	20
PA1	40		40	40		40	40		40
PA2		40	40		40	40		40	40

### 4.3 Experimental Results

The statistics show some unexpected results in the behavior of the agents. In this discussion of results, only the averages will be used to focus on the main qualitative results for hunter and prey agents. Table 2 shows the average behavioral statistics for each agent type in each experiment in which it participated.

Table 2: Average Agent Statistics by Agent Type and Experiment

	Age@death	Attack	Attacked	Eat	Move	Rest
<u>HA1</u>						
Exp1	39.18	4.66	2.64	24.54	46.51	18.19
Exp2	24.26	7.30	0.00	9.84	20.44	9.66
Exp3	25.99	6.89	1.39	12.72	23.49	7.70
Exp7	26.25	4.18	2.77	12.28	23.97	8.47
Exp8	21.83	5.64	1.30	7.48	15.96	6.69
Exp9	24.14	5.85	1.86	10.93	20.17	6.62
Column Avg.	26.94	5.75	1.66	12.96	25.09	9.55
<u>HA2</u>						
Exp4	43.67	3.85	2.97	26.90	2.97	21.70
Exp5	25.86	7.10	0.00	9.28	5.98	10.63
Exp6	25.86	6.96	1.30	11.61	5.68	7.97
Exp7	29.57	3.95	2.90	13.29	3.12	10.69
Exp8	22.99	5.26	1.42	7.07	4.35	6.73
Exp9	23.28	5.67	1.78	9.49	5.16	5.62
Column Avg.	28.54	5.47	1.73	12.94	4.54	10.56
<u>PA1</u>						
Exp1	39.18	4.66	2.64	24.54	46.51	18.19
Exp3	20.76	2.20	1.32	1.97	14.62	5.80
Exp4	20.16	1.49	1.93	1.89	13.42	5.40
Exp6	20.89	2.29	1.32	2.20	14.69	6.08
Exp7	18.37	1.43	2.66	1.37	10.16	3.31
Exp9	19.48	1.86	1.82	1.75	12.37	4.52
Column Avg.	23.14	2.32	1.95	5.62	18.63	7.22
<u>PA2</u>						
Exp2	29.75	0.00	3.65	5.64	31.68	11.25
Exp3	29.44	0.00	3.63	5.01	31.15	9.05
Exp5	31.17	0.00	3.55	5.67	33.99	12.88
Exp6	28.22	0.00	3.80	4.92	28.87	8.34
Exp8	26.50	0.00	4.09	4.40	25.70	8.60
Exp9	28.21	0.00	3.98	4.82	28.88	8.91
Column Avg.	28.88	0.00	3.78	5.08	30.05	9.84

We now present the most significant agent behavioral results:

- Hunter agents had equivalent or higher average lifespans compared to prey agents across experiments (the column averages, 26.94 and 28.54 for HA1&2 vs. 23.14 and 28.88 for PA1&2).
- PA2 had the highest average lifespan within each of the experiments in which it was involved.

- HA1, HA2, and PA1 had the highest single-experiment average lifespans (39.2, 43.7, and 39.2, respectively). These were significantly higher values than that of PA2, whose highest average lifespan was 31.17 in experiment 5.
- HA2 moved very few times across all experiments, with the highest average being 6.0. This is in contrast to HA1 and PA1 agents with a highest average movement count of 46.5.
- PA1 agents were attacked around twice as many times as PA2 agents across all experiments.
- Despite the fact that PA2 agents were designed to be “lazy”, they moved the most of any agent type on average, across all experiments. This likely stems from a combination of fear, hunger and the longer lifespans they had in which to rack up lots of moves.
- Prey agents in runs with both hunter types did not have significantly lower lifespans or suffer more attacks.
- Hunter lifespans were not generally higher with both prey types present.
- Hunter agents ate much more than they attacked suggesting a “scavenging” characteristic to their behavior.

## 5.0 Discussion

The lifespan statistic is an important one in determining the success of agents in the experiments as it is the highest-level measure of success that can be considered. It is unclear why HA1 and PA1 agents had such a high relative lifespan in Experiment 1. One reason may be that the agents are always moving which tends to have a built-in evasion effect in the prey agents, while at the same time providing the opportunity for HA1 agents to find food and eat it as they travel. The latter hypothesis is somewhat offset by the fact that HA2 agents had the highest average lifespan of all at 43.7 in Experiment 4, being notably “lazy” agents and only moving when hungry. Possibly the most interesting lifespan result is that the PA2 agent type did not have the highest average lifespan in any one experimental run, but did have a higher average lifespan across all runs suggesting it was well adapted to the other agents with which it interacted.

Movement statistics are also very important, as it is the initial activity that often results in the occurrence of other activities. There was a large range between average movement counts. HA2 agents moved far less than their prey counter-parts, the PA2 agents, even though both are not “active” agent types that move when content. This could be due to the fact that PA2 agents move when threatened resulting in a lot of fear-based movement on their part.

Most agent types had relatively consistent statistics across all experiments in which they were involved, except for PA1 agents in Experiment 1. In this case, every statistic increased over the same statistics in other runs. The most obvious major cause of this is the PA1 agent type's ability to co-exist with HA1 agents, but it is unclear as to why. The constant movement of both types of agents may play a significant role in this result.

The significant differences in behavior between agent types support the notion that the slipnet can be used as an instrument to control agents in such an ALife simulation. The low average ages of all agents across all experiments suggests that the agents present in the experiment are not as good at the business of survival as we expected them to be. The simplicity of slipnets has a lot to do with the lower-than-expected lifespans of the agents, but there are other aspects than the slipnet that contribute to the overall behavior of the slipnets. Many of the codelets that produce action have random components involved in decision processes rather than ones that probabilistically follow a biased distribution. These random decisions dampen the emergent effects seen in Copycat. In Copycat, nearly every decision made was based on some value directly or indirectly derived from activation levels in the slipnet. Thus, the activations in the slipnet of the agents are more disconnected from their manifestation in the SimulationWorld through codelets than they could be. Future development of Starcat will attempt to provide a framework to support domain-neutral implementations of the Copycat concepts of strength, importance, temperature, salience, and happiness as influences of system behavior, which are absent in Multicat. Nevertheless, the simple behavioral tendencies of Multicat show interesting consequences. Agent behavior can be seen to adapt to changing perceptions of the environment.

It is encouraging that there is a diversity of agent behavior given the relatively small changes made in the slipnets of the agents. It is also important to note that complex behavior has been implemented through very simple slipnets. To make this clear, the slipnets are extremely basic (a few nodes and connections), but the resultant behavior of moving, attacking, eating, and resting comes about through feedback between the environment and slipnets via codelet execution. None of the agents has a hardcoded response to any stimuli, only a spreading of activation down paths that result in an action. The slipnet in the Copycat application is far larger than that of the slipnets in Multicat agent types. Copycat's answers to letter string analogy problems were often surprising and novel. Though no such novel results occurred in the behavior of the agents, we hope to produce more rich behavior in the agents entirely through the configuration of their slipnets.

The experimental results presented here are the first from a series of proposed experiments that will attempt to iteratively modify the slipnets of the agent types and to tie codelet decisions back to activation levels in the slipnets, in the hopes of producing interesting and/or better behavior. Interesting behavior is analogous to Copycat's “deep” analogy solutions. Behaviors that are unintended and unexpected would be welcomed, but did not occur in the current set of experiments. An

example of unexpected emergent behavior would be agent-clustering (for example, herding of prey for safety) or agent-coordination (for example, predators banding together and sweeping the space for prey). Better behavior can be quantified by higher statistics of proactive actions to better the agent's state of well-being (eating and resting) and the lifespan of the agent, and by lower statistics of actions that negatively impact the agent's health.

## 6.0 Conclusions

We have presented a predator-prey style ALife simulation whose agents are, in effect, miniature instances of an adaptive architecture. Using the Starcat framework, each agent possesses a slipnet and a coderack to produce behavior via codelets and all agents live in and interact through those codelets in a shared instance of the workspace. This novel configuration of components and component relationships achieves several things. First, it is proof that the Starcat framework supports the use of this adaptive architecture in configurations that differ substantially from its predecessors. Second, this particular application helps to reify the notion that, at its core, this architecture pertains to perception and action—that is, it is a useful cognitive model. Third, it brings the complex adaptive systems hypothesis—that globally coherent behavior can arise from the interactions of very many locally acting agents—to bear not only at the level of the simulation (the interactions among agents) but also at the next level down (the internal production of behavior by the agents). In future experiments, this last point may bear much fruit; for there has been speculation [10] that cognitive systems indeed exhibit this multi-layered repeated design.

We have reviewed the significant advances realized in the Starcat framework and showed that this more general, asynchronous and event-driven reframing of the predecessor projects, whose key controlling values are now more easily varied, can indeed be used to produce interesting behavior. In these experiments the slipnet is used in a different fashion than it has been before, focusing more on the production of behavior than the building of structure, though that feature is present also (the target-bonds initiated by an attacking agent). The XML configuration files for the slipnet and other parameters facilitated the exploration of slipnet impact on behavior. We also showed that other factors, such as local state of agents not regulated by the slipnet, can be an effective part of the instantiation of this framework for an application domain. Here the four slipnet types are shown to impact behavior; in future experiments we intend to show more clearly how the changing state of each agent's slipnet realizes its own individual adaptation of behavior to its environment. Also a part of future work will be the incorporation of slipnet-driven bias in the random decisions made by codelets, for example in selecting another entity for interaction. Some particular agent behaviors discussed here are interesting from the perspective of an ALife simulation; but the primary achievement of this project so far has been the novel instantiation of the Starcat framework and its testing in this problem domain.

## 7.0 References

- [1] Stone, P. & Veloso, M. "Multiagent Systems: A Survey from a Machine Learning Perspective", *Autonomous Robotics*. Vol. 8, No. 3, 2000.
- [2] Taylor, C., Jefferson, D., Turner, S. & Goldman, S. "RAM: Artificial life for the exploration of complex biological systems", *Artificial Life* (Langton, C., ed.). Addison-Wesley, Reading, MA, pp. 275-295, 1989.
- [3] Mitchell, M. *Analogy-Making as Perception*. MIT Press, 1993.
- [4] Hofstadter, D., et. al. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, 1995.
- [5] Lewis, J. & Lawson, J. "Computational Adaptive Autonomy: A Generalization of the Copycat Architecture", *Proceedings of the 2004 Las Vegas International Conference on Computer Science*. CSREA Press, Las Vegas, NV, pp. 657-663, 2004.
- [6] Lewis, J. & Lawson, J. "Starcat: An Architecture for Autonomous Adaptive Behavior", *Proceedings of the First Annual Hawaii International Conference on Computer Sciences*. Honolulu, HI, pp. 537-541, 2004.
- [7] Marshall, J. "Metacat: a self-watching cognitive architecture for analogy-making", *Proceedings of the 24th Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, Mahwah, NJ, pp. 631-636, 2002.
- [8] Lewis, J. *Adaptive Representation in a Behavior-Based Robot: An Extension of the Copycat Architecture*. Ph.D. Dissertation, University of New Mexico, Albuquerque, NM, 2001.
- [9] Lewis, J. & Luger, G. "A constructivist model of robot perception and performance", *Proceedings of the 22<sup>nd</sup> Annual Conference of the Cognitive Science Society*. Philadelphia, PA, pp. 788-793, 2000.
- [10] Lawson, J. & Lewis, J. "Representation Emerges from Coupled Behavior", *Workshop Proceedings of the 2004 Genetic and Evolutionary Computation Conference*. Seattle, WA, 2004.