

# CritSIM

## A Program for Simulating Society

David B. Kronenfeld, Ph.D.  
Department of Anthropology  
University of California  
Riverside, CA 92521

Phone

Office 951.827.4340

Message 951.827.5524

Fax 951.827.5409

Email:

david.kronenfeld@ucr.edu

Jerrold E. Kronenfeld, Ph.D.  
Kronenfeld Designs  
3100 Purple Martin Lane  
Indialantic, FL 32903

Phone

321.960.2912

Email

jerry@kronenfelddesigns.com

**TABLE OF CONTENTS**

Table of Contents .....	ii
List of Figures .....	iii
1 Background .....	1
2 Emergent Properties – Our Simulation .....	2
3 The Substantive Problem .....	5
4 SIMULATION Development .....	8
4.1 CritSIM I .....	9
4.2 CritSIM II .....	10
4.2.1 Object Oriented Design Efforts .....	12
4.2.2 Virtual Flock Issues .....	15
4.2.3 Development Issues .....	16
4.3 CritSIM III .....	18
5 Summary and Conclusion .....	23

**LIST OF FIGURES**

Figure 1 Critter Update Logic.....	10
Figure 2 Example Object Code.....	12
Figure 3 Objects Representing Inheritance.....	13
Figure 4 Polymorphism Example .....	14
Figure 5 Entity Object Tree .....	14
Figure 6 Aggregate Object Types .....	16
Figure 7 Sample Output from Sequencing Study .....	17
Figure 8 Entity Processor Model .....	20
Figure 9 CritSIM III Target Architecture .....	22

## 1 BACKGROUND

The present project is aimed at an experimental explication of one aspect of a general view of culture. With that in mind, we will first briefly explain what we mean by “culture”, and then turn to the aspects that our simulation addresses. Following this background, we will first discuss our particular substantive problem and then some significant considerations that affect the actual simulation.

Anthropology, beyond being taken to be whatever it is that anthropologists study, has commonly been described as the study of “culture”. “Culture” seems, in general, to be taken to be whatever it is that people in one cultural community do that makes them different from people in some other cultural community—which seems kind of vague. But a couple of attributes do pop out, and a couple of more can be inferred.

Culture is “collective”—that is, it pertains to communities rather than to individual people. Culture involves “doing” something. Since it varies from community to community, we infer that the specifics of any given culture are learned as opposed to innate. We infer from our ability to recognize cultural similarities across members of a community, that culture has to be patterned. We infer from the variability we see across members of a cultural community, and from the adaptability of culture to changing external (and internal, for that matter) circumstances that whatever it is within individuals that produces their cultural behavior has to be a productive generative system or a set of such systems (something like a set, or sets, of “rules” or consistent regularities). Finally, I would add that, for the most part, people seem unaware of the rules that underlay their behavior; they seem to learn the regularities from their experience with members of the cultural community and then to infer the rule-like regularities from the samples of behavior they see or participate in. Note that “culture” thus conceived—as the systems of rules and regularities that pertain to a community—does not actually exist; all that is “real” are the representations of these rules and regularities that each individual separately infers. But what allows us to study culture as if it exists are two features. 1) Communication and interaction—talking and doing—among members of a community forces and reinforces systematic parallels

among the various individual representations, which are constantly reshaped in response to feedback that their talking and doing elicits from other community members. 2) Members of cultural communities presume the existence of community-defined rule-like regularities and adapt their behavior to these. Thus the various community members all converge on a shared representation even if it seems very unlikely that they ever actually get there.

In addition to the above observations which seem to me to apply to the commonly held anthropological view of culture even if many holders seem maybe unaware of some of them, I would like to offer some further observations which, while “obvious” to me, pertain to aspects of culture that are less widely recognized.

1. We ALL each belong to a variety of cultural communities, rather than just to one. Sometimes, for some of us, the contrasting cultures appear to be deeply different from one another (and hence different cultures) while in other cases the contrast is among similar sub-cultures of a single culture. Some of these differences involve contrasting “alternative” cultures, while others involve “inclusion” relations of successively greater levels of specification. From this I infer that our representations of culture are composed of “fragments” such that we form our representation of each of a set of contrasting cultures from a combination of a common coding of the fragments that the contrasting ones share and separate codings of the fragments on which they differ. There then has to be some way of easily combining the various sets of fragments into something that feels like a single whole to the various actors.
2. The cultural regularities that I speak of involve not simply our cognitive knowledge, but also our awareness of the the goals that might drive relevant action, the emotions that might influence it, the social context in which our action (or someone else’s action) is situated, our presentation of self, our sense of the personalities of other actors, and so forth.
3. Culture thus conceived is NOT any automatic generator of our individual behavior, nor, for the most part, any firm set of “Thou Shalt”s. It is, rather, something more like a set of canonical scenes and scenarios that we, as individual “users” apply in our construction of our own behavior—according to our individual aims, knowledge, perceptions, etc. —as an exercise of our individual “agency”, and that we call on in our attempts to make sense of (or interpret) the behavior of those around us.

## **2 EMERGENT PROPERTIES – OUR SIMULATION**

At least since Durkheim raised the problem in the latter part of the 19th Century, issues involving society’s “emergent properties” have been important to all of the social sciences. I want to note—and emphasize—here that by “emergent properties” we do NOT refer simply to

---

some novel outcome configuration, but instead mean an emergent functioning system with its own behavioral rules. It was this that Durkheim asserted could NOT be reduced to some combination or cascading of individual behavior, and this that characterized the epistemological turf of his emerging discipline of “sociology” (under which umbrella we anthropologists quickly snuck in !). One of our immediate goals in the simulation work has been to demonstrate that the reduction of emergent systems of social behavior to individual “psychology” **IS** possible—via the kind of interaction between the individual and the collective to which the individual belongs and interacts with that lies at the heart of Durkheim’s work. We take these interactions as feedback mechanisms between the individual and the group. Such feedback was not known in Durkheim’s time—and not known to him—but it does seem to be something that he was groping toward—and it represent a very different form of psychology from that that whose relevance he denied. That is, we are suggesting (or “allowing”) that our “reduction” may **NOT** be the one Durkheim rejected, and thus perhaps might not have been so objectionable to him. At the same time we do want to be clear that our simulation relies totally on individual perceptions and decisions, and thus presents a genuine reduction of an emergent social phenomenon to individual psychology.

Our other immediate goals are more problem specific, and will be introduced in our discussion of our actual simulation.

A longer range goal of our simulation project is to attack the issue of emergent collective representations—distributed cognitive structures, if you will. We aim to do this by introducing cognitive content that relates to aspects of our critters’ environment and with “exchanges” that relate to inter-critter interaction, and then to use feedback mechanisms like those in our social simulation joined with a push for patterning stored content to lead the cognitive systems of critters in a “flock” to converge.

The cognitive modeling idea derives in part from my exploration elsewhere of the the kinds of cognitive units that might make up culture. In particular, I have been working on the kind of unit that seems most closely related to culturally patterned action—what I, following Quinn, D’Andrade, Strauss and others, have referred to as “cultural models”. These are the scenes and scenarios that I spoke of earlier, and contrast with other kinds of cultural cognitive structures, including cultural conceptual systems (such as ethnobotanical and kinterm systems)

and cultural modes of thought (general cognitive orientations from which more specific cultural models can be constructed on the fly when needed and which serve, where possible, to provide common themes across a culture's range of specific cultural models).

The Critter Simulation (CritSIM) project represents a set of increasingly complex exercises designed to study aspects of emergent systems in an agent-based simulation environment. Three issues in particular are of interest:

1. Do such properties actually exist? That is, are there some properties of social systems which are not predictable from the behavior of individual members of those systems—or from the rules which generate individual behavior?
2. If such emergent social properties exist, what attributes characterize them? How do they come to get implemented in the behavior of individuals? On the assumption that there exists no external “collective consciousness”, how are pertinent individuals (their behavior and/or the rules which generate their behavior) linked together in the same supra-individual system?
3. How and why did such systems arise? What costs do they carry for their participants? And what benefits do they confer that offset the costs? How did individuals come to be linked into such systems? What learning was required of them, and how did that learning take place? How did such learning come to produce an internalized productive generative representation of relevant parts of the emergent system—vs. the kind of rigidly memorized rote behaviors which old learning theory might have suggested, but which would be clearly inadequate to the task?

Our long-term goal in this project is to explore all of these questions via a computer simulation, but in a way that starts with simple issues and gradually builds in additional complexity.

The computer simulation enables us to experiment with relations, motivations, abilities, and linkages in ways that would be both impossible and immoral if tried on real humans. It allows us to run these experiments quickly enough to see real results as we go and allows us to repeat them enough to generate useful statistical distributions.

We have adopted an incremental approach (vs. trying to do everything all at once) in order to make sure that we always have good analytic and operational control over our simulation—to make sure that we always understand what the program is doing and how it

produces its observed effects<sup>1</sup>. We are adopting a minimalist approach in which we are trying to find the simplest combination of concepts, procedures, and parameters that are sufficient to produce the given output features. Our goal thus is, first, a capability claim—that the mechanisms we model are demonstrably capable of producing the effect/capability/system in question—and, then, second, a plausibility claim—given Ockham’s razor considerations of economy, to make sure our mechanisms seem plausible representations of what happens in nature. Our minimalist approach means additionally that we are **NOT** trying to reproduce behavior that is “realistic” in all regards (e.g., actual appearance, specific eating habits, actual forms of locomotion, etc.) but only behavior that is relevant to the particular mechanism being explored.

### 3 THE SUBSTANTIVE PROBLEM

In a previous paper (Kronenfeld and Kaus 1993) David Kronenfeld and Andrea Kaus implemented—in a simple Turbo Pascal program—a first stage: a single social unit (“flock” or “herd”, if you will) searching for and consuming food in a simple bounded space containing food caches. Food caches were varied in number, size, and location (within the given space), and locations of food could be in either a regular or an irregular pattern. The individuals—“critters” or “birds” (or “starlings”)—moved randomly within the space until they were aware of (“saw”) food, at which point they moved toward the food; when at food they “ate” until the food cache was exhausted. Explicit parameter values included how far the critters could “see” and how fast they could move. A “society” (let us say “SOC”) capability was introduced (as a parameter which could be turned on or off) which consisted of a simple feedback loop between the individual critter and its perception of where the approximate centroid of the “flock” was relative to it; if the individual “bird” found itself further than a certain distance (a parameter value) from

-----

<sup>1</sup> In the early days of artificial intelligence programs such an approach would be characterized as a “white box” one in which the inner connections--the “wiring diagram”--were known--as opposed to a “black box” one in which one saw the input and the output, but had no understanding of how the one came to produce the other.

the “flock” then it moved toward the “flock” and that move took precedence over other possible moves. As an alternative mechanism to SOC that might produce some comparable effects another capability was introduced—the ability of an individual to recognize whether or not another individual “saw” food (let us say “SEESPOTTER”); this parameter could be turned on or off and required that a recognition distance be inputted. In runs, we (DK and AK) explored the interaction of our initial food cache and critter parameters, examined the effects of SOC, compared SOC’s effects with SEESPOTTER’s effects, and looked to see what happened when both SOC and SEESPOTTER were on.

This initial version was purely perceptual/behavioral (in effect, was simply “innate”), and thus involved no “learning”, no “understanding”, no “representations” (individual or collective), and no calculation—and, in this sense, involved no cognition.

Our (again, DK and AK) initial evaluation of the effects of SOC was based on the intuitive observational criterion of whether the “flock” with SOC on moved like a single (collective) entity as opposed to the scatter we got with SOC off; this contrast was based in part on the idea that the simplest society is a mob, and that some kinds of birds (especially Starlings) seem to move as a mob. The initial stage of the simulation did pass this test. But, easy though the test was, it did clearly demonstrate that realistic collective behavior (exhibiting Durkheimian “emergent properties” could be generated by critters that had only individual perceptions, decision processes, and movement—as long as their attention to their environment included attention to other critters in their group. And we noted that this cybernetic view of society involved nothing in any sense mystical !<sup>2</sup>

Then, in order to quantify and measure our results—and as a way of exploring the selective payoffs of society— we turned to the effects of our parameters on grazing efficiency (the only obviously plausible ecological constraint built into the structure of the design at that point). Two kinds of efficiency emerged as important. One was the speed (and effectiveness)

-----

<sup>2</sup> An eventual aim in the larger project is to apply a similar approach to cognitive content.

with which the food caches were found and cleaned out, while the other was the degree to which food consumption was spread out across members of the flock. SOC always slowed down the finding and consumption of food, no matter what settings were assigned to any other parameters. But when the birds were slow-moving with only a limited visual field, and when food was clumped (especially, irregularly clumped) SOC had a large evening-out effect on the inter-bird variance in food consumption (important if a breeding population were to be maintained); to the degree that the birds' visual fields were large (relative to the distances apart of food caches) and their movements fast, this effect was lessened. With large visual fields and fast movement, SEESPOTTER not only sped up the finding and cleaning out of food caches but also did so in a way that kept the inter-bird variance low. But with small visual fields and slow movement rates SEESPOTTER could have deleterious effects—by constantly pulling birds toward food caches that were exhausted (or nearly exhausted) by the time they got there.<sup>3</sup>

From the beginning, the CritSim project has been intended as a sequence of increasingly complex simulations. The 1993 version, with Kaus, aimed at modeling the behavior of a single herbivorous flock relative to a resource environment. Successive later versions are planned to involve, first, introducing con-specific competition—that is, more than one flock of the same species. Next is to be interspecies competition, involving multiple herbivorous species, and that is to be followed by the introduction of carnivores. Finally, the ultimate goal is to introduce some learned cognitive content. Each of these increments involves interesting analytic problems.

Con-specific competition, if it is not to trivially reproduce the single group condition, has to involve some sort of “repulsion” among the membership of competing flocks—which can

---

<sup>3</sup> For a fuller and more detailed presentation of the Kronenfeld and Kaus results see Kronenfeld and Kaus 1993. Copies can be obtained from DK.

mean that members of different flocks don't mix (share the same immediate space) or that members of different flocks don't share food caches. "Repulsion" can involve the "weaker" flock backing off from a confrontation by hovering just outside of the interactive "range" or can involve a "rebound" where the "weaker" flock actively heads off in a different direction. "Weaker" can be measured by relative flock numbers—either absolute flock membership numbers, or numbers of members actually on the scene of interaction, or numbers within some range of the interaction—or by some sort of aggressiveness (maybe related to some kind of "home" territory), or by hunger state (hungry and hence weak or hungry and hence more aggressive), and so forth.

Interspecies competition requires allowing for a complex environment in which the different species can focus on different resources.

Carnivores imply death! Death requires birth, and the combination of the two requires the introduction of an energy economy.

#### **4 SIMULATION DEVELOPMENT**

The present effort represents the third iteration in an evolutionary effort to study emergent properties through the use of simulation. The first step (done a dozen years ago) was an effort to test the underlying concepts using a very simple model. The second phase represented a more recent effort to modernize the code that was used, to look more deeply into the implementation of that simple model, and to extend it slightly. The current effort aims at implementing an increasingly more complex societal model and developing a simulation architecture (our "objective system") which would be more general than that of the first two phases and that would support future studies.

Our overall architectural goal is to develop a design for the objective system that could serve as a general platform for agent-based simulation studies. This architecture will be based

upon the first two iterations of these societal studies as well as other distributed simulation efforts that the authors have supported in the past. Once this architecture has been defined and its bare structure developed, our plan is to implement a series of evolving studies that would permit the concurrent development of ever more complex societal models with ever more capable simulation elements.

In this section we first will briefly describe the architecture of the first and second iterations, and then offer some discussion of the results from those efforts, especially as they relate to the problem of designing an agent-based simulation. We will next provide an architectural overview of the objective architecture.

#### 4.1 CRITSIM I

The initial effort, CritSIM I, represented a “quick and dirty” study to see if an extremely simple model could produce interesting effects. This effort used only the cultural goals embodied by the parameters SOC and SEESPOTTER, previously described. The architecture was equally simplistic and the analysis was at a very coarse level. Despite these limitations, **it worked**: The exercise produced a virtual flock<sup>4</sup> whose behavior resembled that of a “mob” (such as a flock of starlings).

CritSIM I was developed in the 1993 time frame on a PC platform operating under DOS. The Turbo Pascal system was used as the development environment. Its architecture, consistent with approaches used at the time, embodied a batch, sequential-procedural model:

- Read in the run parameters
- Execute successive update cycles
- Print out a summary of the results

The key elements of this approach were the modeling of the critters themselves and how they were updated at each cycle.

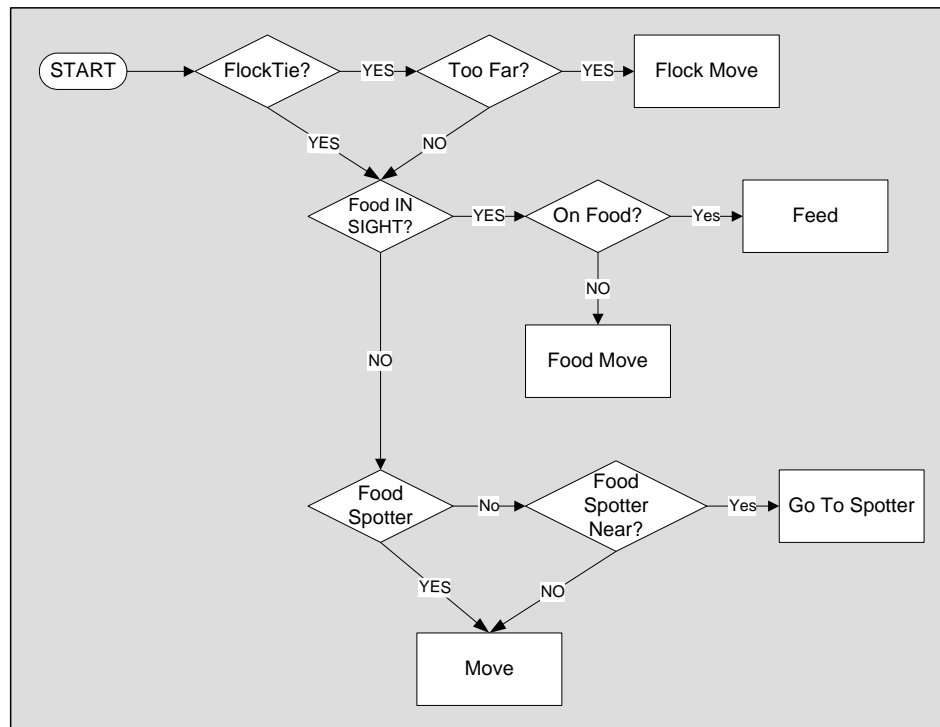
-----

<sup>4</sup> In CritSIM there is no flock per se. There is a mob of critters which, as a group, is referred to as a flock. Thus the term “flock” is for descriptive purposes only and does not affect the behavior of the simulation. For that reason we will refer to it as a virtual flock when it is important to underscore the lack of any group mechanism.

The critters were modeled as stateless entities without memory. The logic for determining a critter's activity during an update cycle (Figure 1) was not dependent upon what the critter was doing in the previous cycle but only upon

- Location of the “virtual” flock, i.e., the centroid of the nearest 80% of the other critters as seen by each critter
- Whether food is in sight (SOC)
- Whether another critter who sees food is in sight (SEESPOTTER)

From a programming standpoint, a “critter” referred to an entry in a critter table. At each update cycle, the program processed a row at a time, in fixed order, and starting at the top of the table.



**Figure 1 Critter Update Logic**

## 4.2 CRITSIM II

CritSIM II, executed some twelve years after CritSIM I, was initiated with four goals in mind:

1. Rehost the program onto a more modern platform and operating environment
2. Perform a more thorough test on the effects of the various simulation parameters (both explicit and implicit)
3. Extend the societal model to include con-specific competition
4. Design our “objective” architecture

CritSIM II was again hosted on a low end PC under Microsoft’s operating system *de jour*, Windows XP. The development environment was Borland’s Delphi 7. The major changes in the program’s architecture arising from this change of environment were

- **Interactive vs. Batch Processing:** Program operations are controlled by the user through interaction with various screen choices.
- **Event-Driven Interface:** Windows, and those development environments like Delphi which generate Windows’ programs, support an event-driven processing architecture. Rather than executing procedures in a fixed, defined order, processes are initiated by interrupts that correspond to user actions (e.g., clicking on a menu or moving the mouse cursor) or program actions (completion of one action triggering the start of another activity).
- **Object Oriented Design:** Windows supports and Delphi requires the use of “objects” as part of the programming paradigm.

Of these, the Object Oriented (OO) design<sup>5</sup> had the most impact. In Section 4.2.1 the object design employed in CritSIM II is discussed.

As part of the CritSIM II effort we both tested the performance of the model and looked at the difficulties of code development. The most important issues that we identified fell into two broad areas: Virtual Flock issues and Design Issues. These issues and our approach to resolve them in the next version of the program are discussed below (Sections 4.2.2 and 4.2.3).

-----

<sup>5</sup> No attempt will be made here to define or discuss Object Oriented (OO) design in detail. A number of good discussions of this subject may be found on the World Wide Web, for example at [http://en.wikipedia.org/wiki/Object\\_oriented\\_design](http://en.wikipedia.org/wiki/Object_oriented_design).

### 4.2.1 Object Oriented Design Efforts

In non-object environments (e.g., CritSIM I under DOS) the characteristics of and operations against and by the elements of the simulation (e.g., the critters and food caches) resided in different parts of the program and were implemented independently. Under an Object Oriented design an entity's characteristics (or properties) and activities (or methods) are defined as part of a single structure, the "object". The use of this architecture element made the simulation development much easier and provided a number of insights into design elements of the objective system.

The Object Oriented approach is based upon three concepts: Encapsulation, inheritance, and polymorphism. Encapsulation refers to the integration of properties (descriptions of some aspect of the entity) and methods (actions by or against the entity) into a single structure that effectively models the entity in question. Figure 2 shows sample code defining two simple objects (using the Delphi version of Pascal). The first, TCritter<sup>6</sup> defines a simple critter. The critter is characterized by a location, age and species. A single process is defined for the object that displays a icon of the critter at its location on the simulation display field. The second object is similarly defined. Note that through encapsulation everything about the object is defined within the object.

<pre> TCritter=OBJECT PRIVATE   LocX, LocY      : DOUBLE;   Specie         : TSpecies;   Age            : DOUBLE; PUBLIC   PROCEDURE Display(sf: TSimField); END;           </pre> <p style="text-align: center;">(a) Critter Object</p>	<pre> TCache=OBJECT PRIVATE   LocX, LocY      : DOUBLE;   Specie         : TSpecies;   Size           : DOUBLE; PUBLIC   PROCEDURE Display(sf: TSimField); END;           </pre> <p style="text-align: center;">(b) Food Cache Object</p>
--	---

**Figure 2 Example Object Code**

---

<sup>6</sup> By convention, most type definitions in Delphi begin with the letter T for "type". Hence, the statement:  
TCritter=OBJECT  
Indicates that the code to follow defines the definition of a critter type.

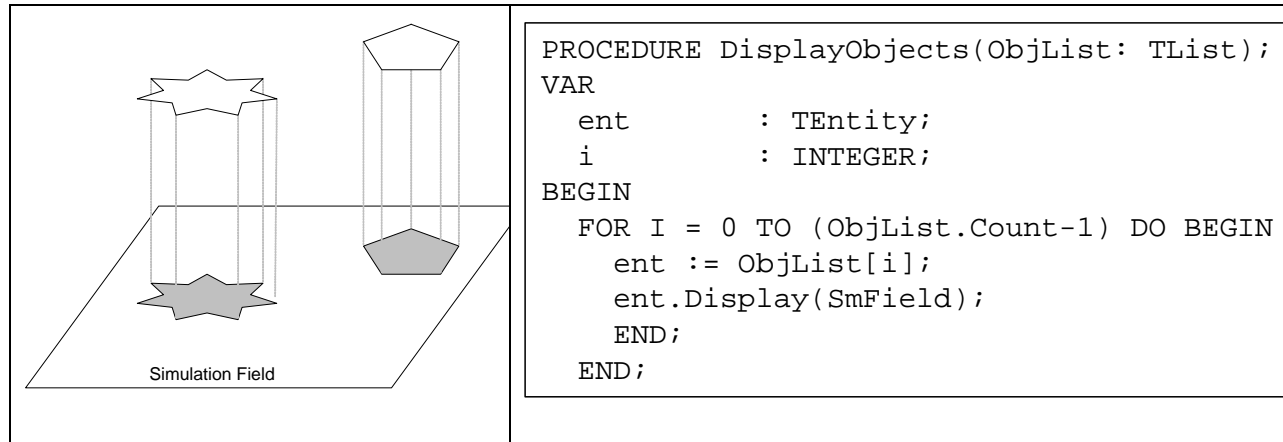
The second pillar of OO programming is inheritance. Inheritance allows one object to be defined in terms of the definition of another object. Figure 3 illustrates this concept. A (generic) entity object (TEntity) is defined which contains common properties and methods. The critter and cache objects of Figure 3 are defined in terms of this generic object and inherit properties (LocX, LocY, and Specie). Note that inheritance represents a tremendous simplification in the evolving of an increasingly complex set of entities. At each stage the old definitions are extended without having to redevelop the old, existing code. Furthermore, if a change in the base definition is required (e.g., introduction of a third coordinate position, LocX) it need only be added to the base class.

<pre> TEntity=OBJECT PRIVATE   LocX, LocY      : DOUBLE;   Specie          : TSpecies; PUBLIC   PROCEDURE Display(sf: TSimField); ABSTRACT; END; (b) Generic Entity Object </pre>	
<pre> TCritter=OBJECT(TEntity) PRIVATE   Age             : DOUBLE; PUBLIC   PROCEDURE Display(sf: TSimField); END; (b) Critter Object </pre>	<pre> TCache=OBJECT(TEntity) PRIVATE   Size           : DOUBLE; PUBLIC   PROCEDURE Display(sf: TSimField); END; (c) Food Cache Object </pre>

**Figure 3 Objects Representing Inheritance**

The final pillar of OO programming is polymorphism, which means “allowing a single definition to be used with different types of data (specifically, different classes of objects)”. This characteristic permits a great simplification of supporting code. For example, in the sample objects in Figure 3, an abstract method (Display) was defined in the base object type, TEntity. In each of the derived objects, this method is implemented in a way appropriate for the specific derived object. This has two advantages: The code used to display an entity on the simulation

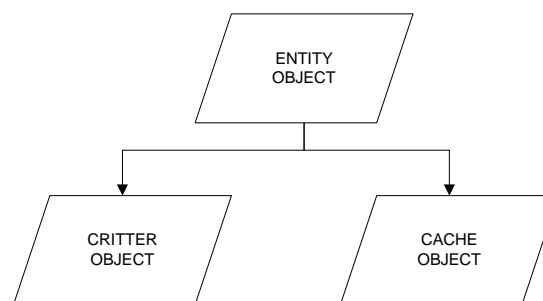
field need not invoke special code to address different objects and the code need not be modified to add new objects to the simulation. Figure 4 illustrates the use of polymorphism to display different object types onto a simulation field.



**Figure 4 Polymorphism Example**

The major benefits to the CritSIM II efforts arising from the use of the OO design were:

- It served to greatly facilitate the code development
- It served to simplify the exposure and study of the parameters underlying the simulation model
- It served as a test bed for studying the implementation of elements of a key design in the objective architecture



**Figure 5 Entity Object Tree**

#### 4.2.2 Virtual Flock Issues

The initial object model (Figure 5) consisted of one abstract object and two concrete objects:

- **Entity Object:** an abstract object type that defines properties and methods common to all simulation elements. No actual entities of this object type are actually created (i.e., “instantiated”). Rather, this class serves as a basis from which the two object types below are derived.
- **Critter Object:** This class defines the simulation critters. Critters move and they consume the caches
- **Cache Object:** This class represents vegetative food. Caches are immobile and exist to be eaten.
- 

Initial tests of the simulation with just these objects exposed a number of performance and/or design problems with this simple scheme. These problems were associated with building up a statistical picture of the flock as part of the analysis of a run and with the sensing by each critter of its virtual flock. The computation of statistics about the virtual flock (e.g., centroid and dispersion) required an external procedure with access to all flock entities. Whether implemented as a standalone (i.e., non-object) method or was encapsulated in an object, this constituted an operation on the virtual flock. Note that positing this type of “flock” measurement in no way interferes with the simulation but merely collects information about it.

The second problem related to the need for each critter to know the location of the centroid of its virtual flock. This can be done by each critter in a number of ways:

- The critter can “scan” all entities it can see, select those that belong to its virtual flock, and compute the centroid of those.
- The critter can maintain a list of relatives (i.e., critters of the same virtual flock) and simply loop through these.
- The system can define an object to keep a list of all related critters in the virtual flock and each critter can then simply query this virtual flock object to get the information it needs for its individual decision making.

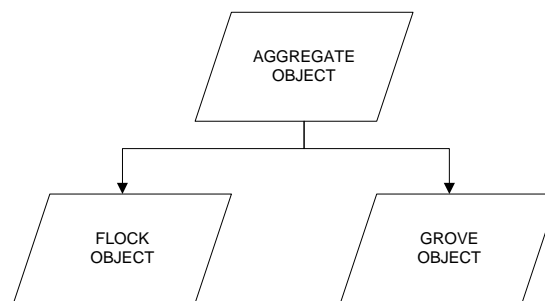
At one level all three of these approaches can be developed so that they do exactly the same thing. They merely differ in the amount and type of overhead needed to implement them. None need obviate the independence of the entity methods from the aggregate objects. They latter represents coding constructs.

The third of these has two advantages over the first two:

- Since the system already needs a virtual flock object (for the analysis purposes previously discussed) , this approach requires the least redundant coding
- If, in the future, we wish to study alternative societal rules that might lead to a group behavior (e.g. following the most dominant other critter visible), this approach will be the simplest to modify.

For these reasons, we added a second object tree to the CritSIM II structures (Figure 6). This tree contained a single abstract object type (Aggregate) and two concrete ones (Flocks and Groves).

Some design features of the CritSIM I model were left largely unchanged in the transition to CritSIM II: The critters remained stateless and without memory. The update logic remained the same. And the sequence of updating critters remained fixed.



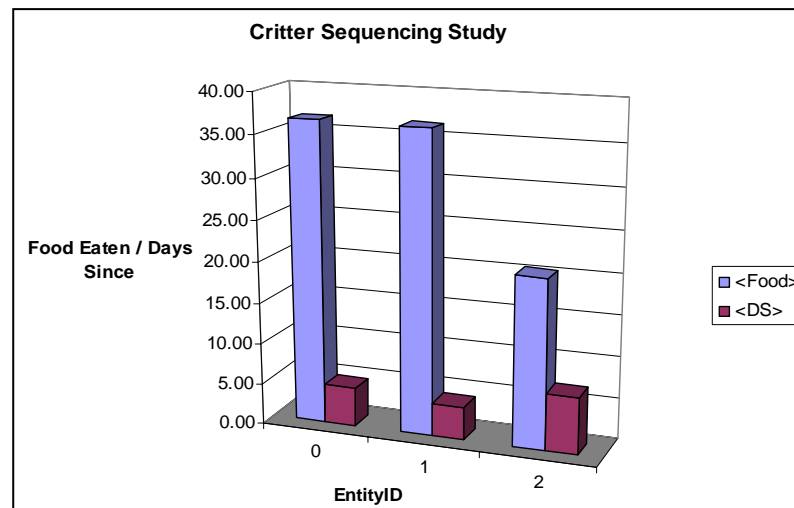
**Figure 6 Aggregate Object Types**

### 4.2.3 Development Issues

From our development and analysis efforts several problems, both substantive and developmental in nature were identified. Two of the substantive problems could be directly traced to the underlying architecture: inter-parameter relationships and fixed order of processing.

There are a number of interrelated time and distance parameters defined in the code: distance to the visual horizon, maximum distance per cycle move, maximum distance from the flock, distance between caches, etc. Although these distances are conceptually independent, we found that certain values led to anomalous behavior based upon simulation constructs and not on physical or societal rules. In the project developed, these raised and emphasized the need to study the parameter domains thoroughly as part of the development plan.

More serious was the effects of the fixed order of processing that were noticed. Since the critters were always updated in a fixed order each cycle, some critters always had first shot at the food in the nearby caches and, hence, tended to be better fed than others. Figure 7 shows this effect. In this run, the three critters, bound by FlockTie were competing for food from a number of small caches. Critters 1 and 2 were able to feed successfully much more effectively than critter 3 as illustrated by their average food consumption over a number of runs. Critter 3 also tended to go longer without food (the plot of <DS>, average days since feeding).



**Figure 7** Sample Output from Sequencing Study

Although in nature some critters may be more successful feeders, such success will not be dependent upon an artificial order in some table.

To correct this problem, the objective system either would need to contain some mechanism for randomizing the order of updates or would need to have the updates occur asynchronously. We chose the latter option.

A second type of problem noted was of a developmental nature: The code complexity increased greatly as the number and types of entities increased. This problem was further exacerbated as the behavioral models of the critters were made more complex. While such problems are clearly soluble (many existing simulation systems show this), they require a

significant development effort, usually with a team of developers. Our goal was to develop an objective architecture that could be supported by a limited staff (initially 1).

### 4.3 CRITSIM III

The design and implementation of the next generation of CritSIM has been driven by a couple of considerations, including those discussed in the previous section. In addition, we are interested in developing a more general event driven simulation environment that could support not only the expanded study of societal issues that are at the heart of the CritSIM effort but also other research areas being pursued by the authors.

One such area of research addresses equilibrium states among forest tree species and the dynamics of reestablishing an equilibrium condition following a change in the environment (such as a decrease in rainfall or increase in mean temperature). Although CritSIM and this problem address totally different application disciplines, from a simulation standpoint, they share many of the same traits: Each can be modeled using independent agents (critters and trees) operating independently and asynchronously against an environmental background. We would like to develop a simulation support environment that could support both of these areas.

We considered two approaches: (1) the expansion and extension of the CritSIM II approach (embedding entities, aggregates of entities, and tables/objects of aggregates) and (2) a distributed processor approach. The former represents an extension of the simulation model used in CritSIM II but appears to offer significant challenges in being extended to support the next level of research. The latter approach utilizes independent processes to represent each entity operating in the simulation environment and has been used extensively to support real-time, man-in-the-loop simulation exercises. Our experience using the latter environment has led us to try and use this environment as the basis for our next generation simulation (CritSIM III).

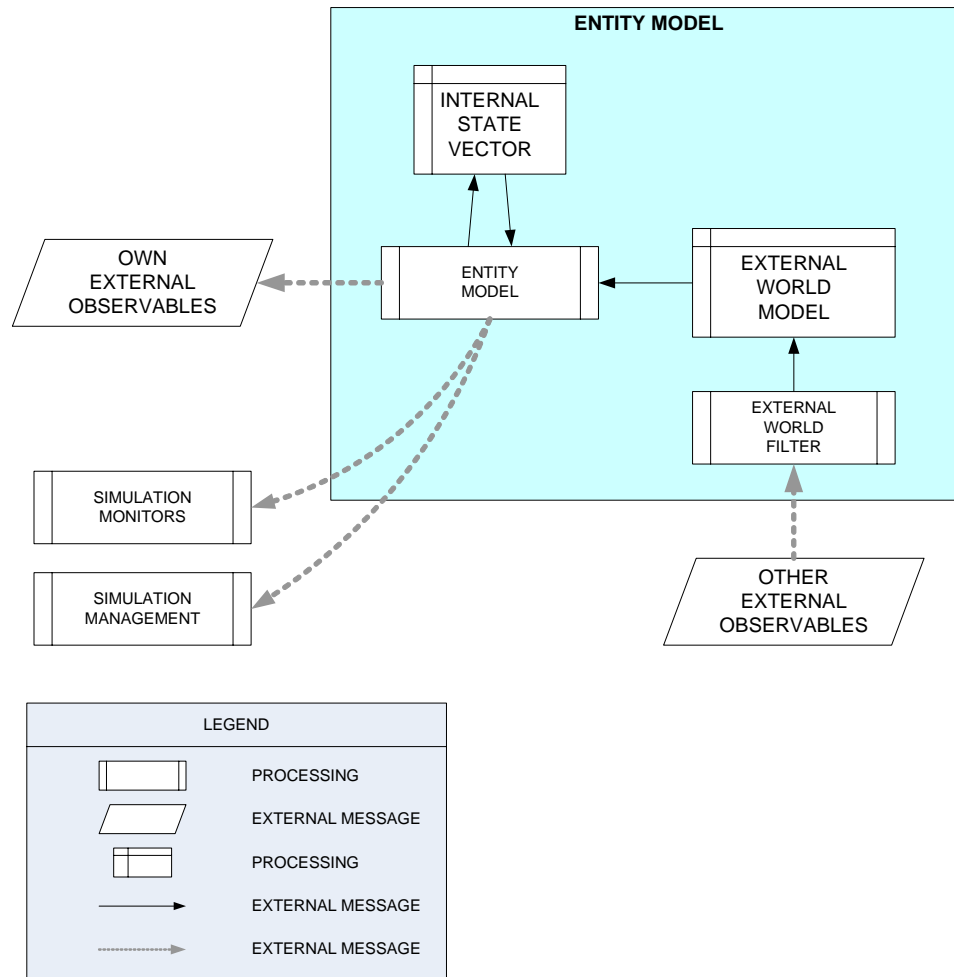
The basic approach was developed by the Defense Advanced Research Projects Agency (DARPA) and the Army training community under two architectures: Distributed Interactive Simulation (DIS) and High-Level Architecture (HLA). Distributed Interactive Simulation (DIS) was the first of these distributed simulations and was developed to support real-time, man-in-the-

loop exercises. The High Level Architecture (HLA) represented an effort to correct some perceived problems with DIS (time management and exercise control, for example) and to integrate the real-time simulation community with the constructive simulation community as represented by the Aggregate Level Simulation Protocol (ALSP).

The basic rationale behind the DIS approach is that each player in the simulation is represented by an independent process (or program). Each entity (program) keeps an accurate model of its state, with the details of the model dependent upon the type of entity and its required degree of accuracy needed for the simulation. During the simulation each entity broadcasts to the world at large those elements of its state vector that are observable to the outside world (position, velocity, changes in its external appearance, etc.). Simultaneously, each entity monitors the broadcasts from other entities in the simulation so that it can maintain a model of the world it “sees” around it, including other entities. Figure 8 depicts the generic entity model.

This approach has a number of specific advantages for us:

- It permits a true, agent-based simulation with each critter operating asynchronously at its own pace and in reaction to its own stimulae
- Separation of the code into a number of smaller and simpler independent programs greatly simplifies the development, maintenance, and expansion of the system.
- Each entity need be modeled only to the level of detail necessary to support a specific exercise. For example, food trees (as background entities) can initially be modeled as fixed capacity caches with no resupply. At a later date, growth and resupply characteristics can be added without modifying the rest of the simulation
- Extending or adding entity types is particularly easy. A hierarchical set of objects (such as has been initialized in CritSIM II) will serve as the bases for the entity and aggregate processes. Adding a new entity species would involve creating a new object, descendent from an existing one, modifying a few parameters, and then applying a process wrapper around it.
- If processing performance should become an issue, the architecture supports the distribution of the simulation elements across networked platforms



**Figure 8 Entity Processor Model**

In implementing this approach we will use neither a pure DIS nor a pure HLA approach, though we distinctly lean toward the simpler DIS architecture<sup>7</sup>. Each of these has certain objectives that carry too large an overhead for the current effort. Rather we will utilize some concepts from each of these systems but implement them locally:

-----

<sup>7</sup> The primary elements we plan to extract from HLA are its use of defined object models to formally design and document the elements (appearance and operation) operating in a given exercise.

- Primarily use a simple broadcast model for communication between entities, for example when an entity publishes its external state updates
- Use targeted (or point-to-point) communication between selected elements, for example between a entity element and its parent aggregate element.
- Use an exercise manager to manage the simulation
- Use the HLA concept of Object Models to define the entities to be simulated and their internal and displayed characteristics.

Figure 9 depicts the architecture we are developing for CritSIM III. The elements of this architecture are:

- **Simulation Manager:** The purpose of the simulation manager is to serve as the primary interface supporting the user's setting up of the exercise. It will serve to spawn a set of integrated element processes<sup>8</sup> at the commencement of the simulation; start, pause and stop the simulation, and to monitor available resources. The Simulation Manager will also support non-integrated element processes to join the simulation. This latter ability will permit the testing of new processes without modifying the basic setup of an existing exercise scenario.
- **Aggregate Processes:** Independent processes that will maintain information about aggregations of entities (such as flock centroid, dispersion, and average health). These processes will spawn the entity processes that make up the aggregation and then monitor the external state vectors broadcast by those entities. These processors will also perform as servers for its constituent entities providing a common calculation of some of the aggregate properties, such as centroid. In this manner we plan to relieve the entity processors of some of the computational requirements associated with maintaining an internal picture of its environment.
- **Entity Processes:** The processors will represent individual entities, both agents (active participants) and environmental elements (e.g., some food sources). Behavior, energetics, sex, and other such critter processes will be incorporated in these units.
- **Monitor Processes:** A variety of processes will be developed to permit the collection, storage, and analysis of data generated in the simulation. Currently planned monitor processors are (1) Logger to record all data committed to the network, (2) Plane View Displays to show a bird's eye view of the simulation field, (3) statistics monitors to

-----

<sup>8</sup> Here, an integrated element process is defined to be one which is defined in the setup of the simulation and whose processes execution code is know to the Simulation Manager.

generate and display tables and plots of selected simulation metrics (available food, number and average health of selected critters, etc.).

- Environmental Processes:** This class of processes will define how the simulated environment is set up. The versions to date have only one type of such processes: the laying down of the cache patterns. Two lay down approaches are supported: random and gridded. Given that these caches are supposed to represent vegetation and that there are better models available for creating more realistic lay downs<sup>9</sup>, we plan to integrate those models into this element. Other areas of the environment (e.g., rainfall and temperature) will be modeled as need for this and other studies that will utilize this simulation platform.

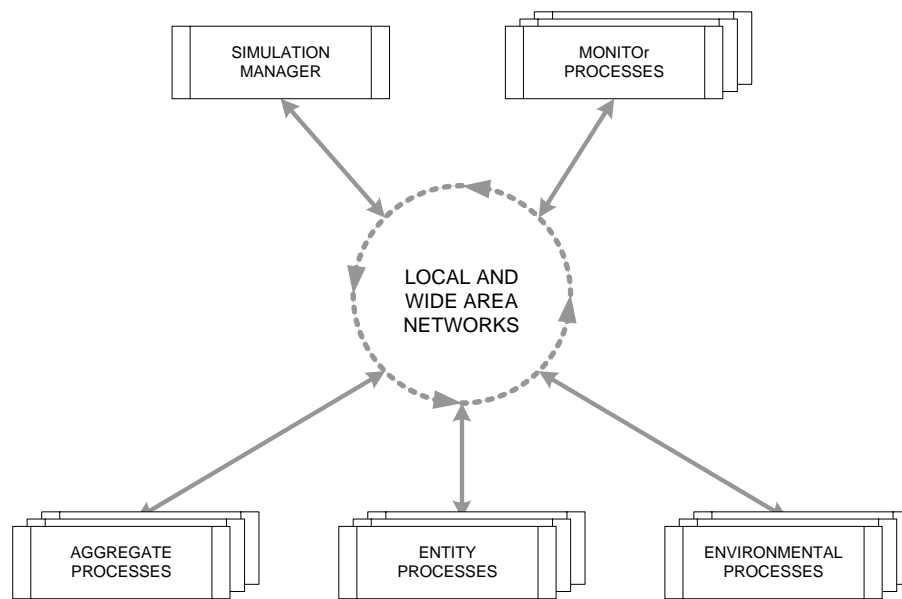


Figure 9 CritSIM III Target Architecture

-----

<sup>9</sup> Kronenfeld, BJ and Wang, YC, 2007. Accounting for Surveyor Inconsistency and Bias in Estimation of Tree Density from Presettlement Land Survey Records. Canadian Journal of Forest Research. (in press)

## 5 SUMMARY AND CONCLUSION

The Critter Simulation effort seems to be progressing smoothly, if sporadically, along the path of evolving capabilities and complexity. The first step demonstrated the potential utility of the approach. The second stage expanded the domain of study (slightly) and revealed difficulties with the initial architecture. The third stage (currently under way) will represent the transfer of the system to an existing architectural model (the DIS paradigm) and the further extension of the complexity addressed. The CritSIM III development will proceed through several stages: Initially the communication infrastructure necessary to support the distributed agent model will be developed and tested. Secondly, the objects of CritSIM II will be rehosted as standalone processes and tested within the distributed environment. Particular attention will be paid to testing any timing issues associated with this implementation. Finally, additional entity objects will be derived to extend the study to multiple species and to develop an energy regime necessary to support the birth and death of entities.

The current authors have brought widely different sets of experience together in looking at this problem. On the one hand is the awareness of the problem and the issues involved (the domain knowledge, if you will). On the other hand is the experience and knowledge of the methods and tools that might be used to explore the problem from different directions. Thus far the authors feel that this multi-discipline approach has been fruitful.