Annals of Operations Research 75 (1997) 69-101 (Special Issue on Artificial Intelligence and Management Science)

"Go to the Ant": Engineering Principles from Natural Multi-Agent Systems

H. Van Dyke Parunak (van.parunak@altarum.org)
+1 734 302-4684 (voice), +1 734 302-4991 (fax)
Altarum Institute
3520 Green Court, Suite 300
Ann Arbor, MI 48105-1579, USA

Abstract

Agent architectures need to organize themselves and adapt dynamically to changing circumstances without top-down control from a system operator. Some researchers provide this capability with complex agents that emulate human intelligence and reason explicitly about their coordination, reintroducing many of the problems of complex system design and implementation that motivated increasing software localization in the first place. Naturally occurring systems of simple agents (such as populations of insects or other animals) suggest that this retreat is not necessary. This paper summarizes several studies of such systems, and derives from them a set of general principles that artificial multi-agent systems can use to support overall system behavior significantly more complex than the behavior of the individuals agents.

1. Introduction

Table 1 shows that the history of software is one of increasing localization and encapsulation.

Originally, the basic unit of software was the complete program. Arbitrary jumps of control made it difficult to manipulate any unit other than the individual line of code and the entire program. Data often occupied the same deck of cards as the program, and the entire deck, code and program, was the responsibility of the programmer, who thus determined the behavior of the complete program before it began execution.

The "structured programming" movement designed programs from smaller packages of code, such as structured loops and subroutines, with a high degree of local integrity. Though a subroutine's code was encapsulated, its state had to be supplied externally through arguments, and it gained control only when externally invoked by a call.

The next generation was object-oriented programming, which localized not only a segment of code but also the variables manipulated by that code. Originally, objects were passive, and gained control only when some external entity sent them a message.

Table 1: Increasing Software Localization

	Monolithic Program	Structured Programming	Object-Oriented Programming	Agent-Oriented Programming
How does a unit behave? (Code)	External	Local	Local	Local
What does a unit do when it runs? (State)	External	External	Local	Local
When does a unit run?	External	External (called)	External (message)	Local (rules; goals)

The next step, software agents, gives each object its own thread of control and its own internal goals, thus localizing not only code and data, but also invocation. Such an “active object with initiative” is the most basic manifestation of a software agent, sometimes called an “autonomous agent” to emphasize that it does not need external invocation, or a “responsible agent” to emphasize that it watches out for its own set of internal responsibilities. With code, state, and control all localized within the agent, little or no integration is required to launch an application. In the ultimate agent vision, the application developer simply identifies the agents desired in the final application, and they organize themselves to perform the required functionality.

Such an approach to software has many advantages over traditional top-down techniques. The behavior space of the entire system is much larger than could be developed with traditional techniques: one hundred agents, each with ten behaviors, require the programming of only 1000 individual behaviors, yet provide a behavior space on the order of 10^{100}, a number far larger than the total number of elementary particles in the universe. Integration and maintenance costs, traditionally two of the largest expenses in developing a software system, are greatly reduced. There is evidence that interacting systems of agents can provide adaptability and robustness only dreamed of in conventional systems.

The appeal of agent architectures depends on the ability of populations of agents to organize themselves and adapt dynamically to changing circumstances without top-down control from a programmer or system operator. This claim at first blush appears incredible, and some researchers hedge their bets by designing complex agents that emulate human-level intelligence and that can reason explicitly about their coordination with one another. Such an approach reintroduces many of the problems of complex system design and implementation that drove the evolution of increasing software localization in the first place. Naturally occurring systems of simple agents (such as populations of insects or other animals) offer evidence that this retreat is not necessary. This paper summarizes a number of studies of such systems, and abstracts from them a set of general design principles that can be applied to artificial agent-based systems in which the behavior of the whole is significantly more complex than that of any of the individuals.

Any effort to compare software agents with entities in the natural world must take account of some fundamental differences between these two classes of objects. Section 2 develops a formal definition of “agent” that covers both cases, points out the similarities and differences between them, and draws some preliminary conclusions about applying software agents in physical contexts. Section 3 develops several examples of naturally occurring multi-agent systems. On the basis of these examples, Section 4 proposes seven principles that can help design artificial systems of fine-grained agents with emergent systematic behavior. Section 5 discusses how such principles can be evaluated for deployment in real-world applications.

2. Theoretical Context

A disciplined comparison between natural agent systems such as ant colonies and systems of computerized agents depends on a model of agenthood that is sufficiently general to cover both cases. Constructing such a common model shows us important points of difference between the two domains, differences that we must take into account in engineering effective multi-agent systems.

A fundamental insight is that agents cannot be considered independently of the environment in which they exist and through which they interact [Müller 1996]; [Ferber & Müller 1996]. This insight is often ignored in work on computational multi-agent systems, where researchers consider the environment as a passive communications framework and everything of interest is relegated to one or another of the agents. Even such purely electronic environments often prove embarrassingly active when an application is scaled up for real-world use, and those engineering agents for non-electronic domains (such as factory automation) must

consider the environment explicitly. So we begin by defining a Multi-Agent System as a three-tuple: a set of Agents, an Environment, and a Coupling between them:

$$MAS = <Agents, Environment, Coupling>$$

2.1 Agents

Each agent is a four-tuple:

$$Agents = \{Agent_1, \ldots, Agent_n\}$$

$$Agent_i = <State_i, Input_i, Output_i, Process_i>$$

- $State_i$ is a set of values that completely define the agent. The structure, domains, and variability of these values are not constrained by this definition, and differences in these features are responsible for much of the interesting variation among different kinds of agents.
- $Input_i$ and $Output_i$ are subsets of State, whose variables are coupled (in a way to be defined more specifically below) to the environment. We will occasionally speak of an agent's sensors and effectors. These are mechanisms that implement the coupling between the environment and the agent's Input and Output variables, respectively.
- $Process_i$ is an autonomously executing mapping that changes the agent's State. "Autonomously executing" means that the Process runs without being invoked from any outside entity. In computational terms, an agent has its own virtual CPU.

The existence of Input and Output imply that an agent is a bounded process, and Input and Output cross this boundary. Input and Output relate the agent immediately to the environment, and only mediately (through the environment) to other agents.

2.2 Environment

Environment is a two-tuple that is syntactically a subset of an agent:

$$Environment = <State_e, Process_e>$$

The important feature of this definition of the environment is that the environment itself is active. It has its own Process that can change its State, independent of the actions of its embedded agents.

$Input_i$ and $Output_i$ of the various agents are coupled to elements of $State_e$, but the environment does not distinguish which elements of $State_e$ are so coupled. That distinction depends on the agents that exist at any moment and the capabilities of their sensors and effectors. The lack of a distinguished Input and Output means that the environment, unlike an agent, is unbounded. The fact that environment, like agent, has State and Process means that putting a boundary and associated Input and Output values around an environment (and its associated agents) yields a higher-level agent. Thus this model lends itself naturally to aggregating low-level agents into higher-level ones.

2.3 Modeling State and Process

The exact nature of the coupling depends on how we model State and Process. Table 2 summarizes two approaches that have been taken.[1]

Table 2: Two Models of State and Process

Model	Discrete-Event Dynamical Systems	Time-Based Dynamical Systems
State	Discrete (symbols)	Continuous (real numbers)
Processing	Symbol manipulation, e.g., -- Rewrite rules -- Transition tables	Partial differential equations or difference equations
Progress	Discrete; event-based	Discrete (integer time) or Continuous (real time)
Time & State	Independent (clock speed)	Coupled (physical laws)
Communities	Computer Science, AI	Physics, Mathematical Ecology

Computer scientists, particularly those with a background in symbolic computation (as in artificial intelligence), naturally model systems as though their dynamics were governed by discrete events. Canonical models of such systems include rewrite grammars and automata driven by transition tables (such as a finite-state automaton or a Turing machine). Such a system leaps instantaneously from one discrete state to another on the basis of events that are not necessarily anchored to a numerical time parameter. As a result, there is no intrinsic coupling between numerically valued time and the state of the model. The system yields the same result when run on a fast computer as on a slow one, and only an outside observer with access to a real-time clock can tell the difference.

Students of the natural sciences naturally model systems as time-based dynamical systems, in which state variables can be real numbers and thus vary continuously, and in which the progress of the system is based not on the sequence of a series of events, but on a numerical time parameter. In such a system, processing speed is intimately linked to the physics of the domain, and any attempt to change the "clock speed" of the process can be expected to change the qualitative behavior of the system.

It can be argued that the most appropriate model to use for a system depends on the level of analysis. A digital computer, for example, is built on time-based continuous quantum wave functions at the subatomic level. When one moves to the level of holes and free electrons in a semiconductor, a discrete-event model is more appropriate, and continues to dominate through the level of gates, logic circuits, and program execution. However, it becomes inefficient to consider the state space of a very large network of computers as discrete, and time-based models with continuous state sometimes emerge for analyzing such systems.

Most work on artificial agents identifies itself with the artificial intelligence community, which views the problem of building an agent as one of artificial cognition. AI "orthodoxy" advocates the Physical Symbol System hypothesis, classically expounded by [Simon 1981], which asserts that a cognitive system is a discrete-event dynamical system.[2] As a result of this orthodoxy, most research on agents takes a discrete-event approach to

---

[1] I am indebted to my colleague A.D.Baker for helpful discussions on the relation between agent theory and systems engineering.

[2] Recently, [Port & vanGelder 1995] have articulated the view of a small but growing group of researchers who assert that cognition is a time-based dynamical system. For purposes of contrast with the Physical

modeling agents (and the environment, if the researcher models it explicitly at all). By contrast, most researchers who focus their attention on the environments (other than computational) in which agents must perform use time-based models, and these models also dominate discussions of the internal dynamics of lower life-forms such as ants and termites.

2.4 Coupling

An understanding of the coupling between agent and environment must consider four cases, corresponding to the interacting distinctions {Discrete-Event, Time-Based} x {Agent, Environment}. The simplest cases are when the dynamics of both agent and environment are modeled the same way (a situation we describe as homodynamic). Additional complications arise when the system is heterodynamic, with discrete-event dynamics for one element and time-based dynamics for another.

2.4.1 Homodynamic Systems

When both agents and environment are discrete-event, the coupling of $Input_i$ and $Output_i$ to $State_e$ is simply a mapping of registers from environment to agent or vice-versa. This model, which dominates artificial agent research today, leads to an (unrealistic) identity between an agent’s actions and the resulting change in the environment, which in turn contributes to classical AI conundrums such as the Frame Problem [Ferber & Müller 1996].

When both agents and environment are time-based, as in most research on mathematical ecology, the coupling takes the form of an energy flow. Because the state values can vary continuously, the change in a variable resulting from such a flow may be infinitesimal, depending on the Process in the receiving entity and the other energy flows in the system. Because we model all agent interactions through the environment, an agent can individually determine its next state, based on the flows it senses from the environment. However, no single agent can predict the effect of its actions on the environment, because these will depend on the actions of other agents as well.

2.4.2 Heterodynamic Systems

In engineering artificial agents for real-world applications, the agents are discrete-event and the environment is time-based. In this case the coupling is even more complicated. In addition to a mapping between the state spaces of agent and environment, the agent’s sensors must provide an energy sink and an analog-to-digital conversion to respond to energy flows from the environment, and its actuators must provide an energy source and a digital-to-analog conversion in order to provide energy flows to the environment.

When the agents are time-based and the environment is discrete-event, the role of the agent’s sensors and actuators is reversed: a sensor is now a D-A energy source, and an actuator is now an A-D energy sink. This particular combination of agent and environment has not been extensively explored. Ergonomic studies of how humans interact with computers may point the way to such investigations, but people who work with computers often do so in the role of symbol processors themselves, thus masking the more difficult questions of how the two models of State and Process interface.

Symbol System hypothesis, they call their approach the Dynamical System Hypothesis. This moniker, while memorable, is a little misleading, since both approaches use dynamical (that is, state-based) systems.

3. Natural Agent Systems

Each example in this section considers the desirable *system behavior* that emerges from the overall community, reviews the *individual responsibilities* of the agents in the system that yield this overall behavior, and discusses the *integration* of individual behaviors into the system behavior.

Claims in this section are based on emulation[3] of hypothesized individual behaviors in a community of software agents (active objects). If the behavior of the overall emulated system corresponds to the behavior of the natural system, the experimenter concludes that the individual behavior of the agents models that of the natural animals.

This approach is suggestive rather than conclusive. The individual behaviors emulated may not be the only ones that will lead to a given overall system behavior. The experimenter's judgment that the overall behavior of the emulation is "the same as" that of the natural system is highly subjective. In spite of these shortcomings, this approach can at least generate some interesting hypotheses. When our objective is to engineer artificial systems rather than derive explanations for natural ones, superficial correspondence is often sufficient.

3.1 Ants: Path planning

System Behavior.—Ants construct networks of paths that connect their nests with available food sources. Mathematically, these networks form minimum spanning trees [Goss et al. 1989], minimizing the energy ants expend in bringing food into the nest. Graph theory defines a number of algorithms for computing minimum spanning trees, but ants do not use conventional algorithms. Instead, this globally optimal structure emerges from the simple actions of the individual ants.

Responsibilities.—Each ant that forages for food has the same basic program, consisting of five rules that fire whenever their conditions are satisfied [Steels 1991].

1. Avoid obstacles. Whatever an ant does, it will not aimlessly push against a wall.

2. Wander randomly, in the general direction of any nearby pheromones (scent markers that many insects generate). If it does not sense any pheromones, an ant executes Brownian motion, choosing each step from a uniform distribution over possible directions. If it senses pheromones, the ant continues to wander randomly, but the distribution from which it selects its direction is weighted to favor the direction of the scent.

3. If the ant is holding food, drop pheromone at a constant rate as it walks. In the simplest simulations, the ant continues to move randomly. In others, it follows a beacon (e.g., a distinctive pheromone at the nest) that leads in the general direction of home. Both approaches yield the same global behavior. The homing beacon generates paths sooner, but continued random wandering works in the emulation as well.

4. If the ant finds itself at food and is not holding any, pick the food up.

5. If the ant finds itself at the nest and is carrying food, drop the food.

Integration.—Brownian motion eventually brings the ant arbitrarily close to every point in the plane. As long as the separation between nest and food is small enough compared with the range of the ant, a wandering ant will eventually find food if there is any, and (even without a beacon) a food-carrying ant will eventually find the

[3] In emulation, software entities mimic the behavior (observed or postulated) of entities in the problem domain. Simulation is a more general class of techniques, including emulation and also models that capture the aggregate behavior of the system but not the activities of the individual members.

nest again. In most cases, food is available only in some directions from the nest, and ants who wander off in the wrong direction will starve or fall to predators, but as long as there is food close enough to the nest and as long as there are enough ants to survey the terrain, the food will be found.

Because only food-carrying ants drop pheromone, and because ants can carry food only after picking it up at a food source, all pheromone paths lead to a food source. Once a full ant finds its way home, there will be paths that lead home as well. Because pheromones evaporate, paths to depleted food sources disappear, as do paths laid down by food-carrying ants who never reach home. Paths that touch the nest are easily found by outbound ants, and as long as they lead to food, they will be reinforced by those ants once they pick up food.

The initial path will not be straight, but the tendency of ants to wander even in the presence of pheromones will generate short-cuts across initial meanders. Because pheromone paths have some breadth, they tend to merge together into a trace that becomes straighter the more it is used. The character of the resulting network as a minimal spanning tree is not intuitively obvious from the individual behaviors, but does emerge from the emulation.

3.2 Ants: Brood Sorting

System Behavior.—An ant hill houses different kinds of things, including larvae, eggs, cocoons, and food. The ant colony keeps these entities sorted by kind. For example, when an egg hatches, the larva does not stay with other eggs, but is moved to the area for larvae. Computer scientists have developed a number of algorithms for sorting things, but no ant in the ant hill is executing a sorting algorithm.

Responsibilities.—The individual ant algorithm that yields system-level sorting behavior contains some behaviors similar to those in the path-planning problem [Deneubourg et al. 1991].

1. Wander randomly around the nest.

2. Sense nearby objects, and maintain a short memory (about ten steps) of what has been seen.

3. If an ant is not carrying anything when it encounters an object, decide stochastically whether or not to pick up the object. The probability of picking up an object decreases if the ant has recently encountered similar objects. In the emulation, the probability of picking up an object is

$$p(pickup) = (k^+/(k^+ + f))^2$$

where f is the fraction of positions in short-term memory occupied by objects of the same type as the object sensed and k^+ is a constant. As f becomes small compared with k^+, the probability that the ant will pick up the object approaches certainty.

4. If an ant is carrying something, at each time step decide stochastically whether or not to drop it, where the probability of dropping a carried object increases if the ant has recently encountered similar items in the environment. In the emulation,

$$p(putdown) = (f/(k^- + f))^2$$

where f is the fraction of positions in short-term memory occupied by objects of the same type as the object carried, and k^- is another constant. As f becomes large compared with k^-, the probability that the carried object will be put down approaches certainty.

Integration.—As in path planning, the Brownian walk eventually brings the wandering ants to examine all objects in the nest. Even a random scattering of different items in the nest will yield local concentrations of similar items that stimulate ants to drop other similar items. As concentrations grow, they tend to retain current

members and attract new ones. The stochastic nature of the pick-up and drop behaviors enables multiple concentrations to merge, since ants occasionally pick up items from one existing concentration and transport them to another.

The put-down constant k^- must be stronger than the pick-up constant k^+, or else clusters will dissolve faster than they form. Typically, k^+ is about 1 and k^- is about 3. The length of short-term memory and the length of the ant's step in each time period determine the radius within which the ant compares objects. If the memory is too long, the ant sees the entire nest as a single location, and sorting will not take place.

3.3 Termites: Nest Building

System Behavior.—Tropical termites construct mounds that can exceed five meters in height and ten tons in mass. These multi-story structures store food, house the brood, and protect the population from fire and predators. The existence of some of these structures has been documented for over 350 years, which is as long as they have been accessible to the European compulsion for chronological records. In spite of the complexity, durability, and effectiveness of these structures, no termite serves the role of a chief engineer, planning the structure and managing its construction.

Responsibilities.—Termites draw on the pheromone mechanism illustrated in the ant path-planning example, and on the wandering in both of the ant examples [Kugler et al. 1990].

1. Metabolize bodily waste, which contains pheromones. This waste is the material from which the termite mound is constructed.

2. Wander randomly, but prefer the direction of the strongest local pheromone concentration.

3. At each time step, decide stochastically whether to deposit the current load of waste. The probability of making a deposit increases with the local pheromone density, and the amount of waste that the termite is currently carrying. A full termite will drop its waste even if there is no other nearby deposit, and a termite that senses a very high local concentration of pheromones will deposit whatever waste it is carrying, even a relatively small amount.

Integration.—The probabilistic algorithm leads to the generation of scattered initial deposits. These deposits attract termites who wander close enough to smell them, and increase the probability that these visitors will make reinforcing deposits. Because pheromones decay over time, the most recent deposits at the center of the pile are the strongest, and the piles tend to grow upward rather than outward, forming columns. When two columns grow near one another, the scent of each attracts termites visiting the other, thus pulling subsequent deposits into the shape of an arch. A similar dynamic leads to the formation of floors joining multiple arches. When one floor is complete, the cycle repeats to construct the next.

3.4 Wasps: Task Differentiation

System Behavior.—Mature *Polistes* wasps in a nest divide into three groups: a single Chief, a group of Foragers who hunt for food, and a group of Nurses who care for the brood. These varied roles are filled by genetically identical wasps. The relative proportion of Foragers and Nurses varies with the abundance of food and the size of the brood, but the nest has no Human Resources department, and no wasp (not even the Chief!) computes what this proportion should be.

Responsibilities.—Each wasp maintains two parameters: a *Force* parameter that determines how mobile it is, and a *Foraging Threshold* that determines how likely the wasp is to go seek food. The brood maintains a

third parameter, *Demand*, that stimulates the foragers. The various wasp behaviors involve interactions of these parameters [Theraulaz et al. 1991].

1. When two wasps meet, engage in a face-off. The winner is chosen stochastically. The probability of success of individual *j* in confronting individual *i* given by the Fermi function of their forces,

$$p\left(F_i, F_j\right) = 1 \Big/ \left(1 + e^{h\left(F_i - F_j\right)}\right)$$

where *h* modulates the extent to which the outcome is predetermined. Thus the wasp with the higher force has a higher probability of winning the face-off, but the wasp with lower force will occasionally win. A quantum of force is transferred from the losing wasp to the winning wasp.

2. When the brood receives food, reduce its demand. The brood's demand at time *t* is given by

$$D(t) = D(t\text{-}1) + d(t\text{-}1) - W(t\text{-}1)$$

where *W* is work done by all individual foragers.

3. When a wasp is near the brood, determine probabilistically whether or not to forage. The probability of foraging is given by the Fermi function $p(s_j, D(t))$, where s_j is the foraging threshold for individual j. Successful stimulation reduces s_j by ξ (a learning coefficient), while failure to forage increases s_j by ϕ (a forgetting coefficient).

Integration.—Confrontation among wasps shifts force from one to another, a primitive form of communication. Foraging reduces the brood's demand and thus the brood's stimulation on nearby wasps, while stimulation reduces wasps' thresholds and thus triggers foraging. When a community of artificial wasps executes these behaviors over a period of time, the population stabilizes into three groups, corresponding to the division observed in the natural insects. A group with high force and low threshold corresponds to the foragers, who have both the strength to move about and the sensitivity to the brood to respond to their needs. A second group with low force and low threshold corresponds to the nurses, who also are attentive to the brood but, lacking force, cannot move about and must remain near the brood. Finally, a single wasp with high force and high threshold corresponds to the chief. The chief does not command or control the others, but grounds the force and threshold scales and (by wandering around the nest and facing off against the other wasps) balances these variables across the population.

3.5 Birds and Fish: Flocking

System Behavior.—Flocks of birds stay together, coordinate turns, and avoid collisions with obstacles and each other. Schools of fish exhibit similar coordinated behavior. Human societies address similar problems, in air-traffic control and convoys of ships, but conventional human solutions depend on sophisticated communication and central coordination structures not available to birds and fish, and cannot handle the density of coordinated entities that flocks and schools can.

Responsibilities.—Each bird or fish follows three simple rules [Reynolds 1987, Heppner 1990]:

1. Maintain a specified minimum separation from the nearest object or other birds.
2. Match velocity (magnitude and direction) to nearby birds.
3. Stay close to the center of the flock.

Integration.—The flock or school is a self-constraining structure in which each entity's individual actions simultaneously respond to and change the overall structure of the flock. Although each bird or fish senses only

the movements of its nearest peers, its responses to these movements propagate to others, so that the system as a whole exhibits global coordination.

3.6 Wolves: Surrounding prey

System Behavior.—A single wolf cannot kill a moose. The hooves of the larger and more powerful moose are more than a match for any single aggressor. The pack of wolves must first surround the moose, so that one can jump on its back while it is occupied with the others. Human SWAT teams use radio to coordinate such surrounding maneuvers, but wolves don't have walkie-talkies or similar long-range communication mechanisms.

Responsibilities.—This task, known in the literature as the "predator-prey" problem, was for some years a major challenge problem in the field of distributed artificial intelligence (DAI) [Korf 1992]. Many of the proposed solutions assumed reasoning and communication capabilities whose existence in wolves is doubtful at best. For example, the emulated wolves would communicate and negotiate their strategies with one another ("I'll head for the north side; why don't you go to the south?").

A simpler solution requires only rudimentary sensing and action on the part of both moose and wolves. When these actions are emulated on a hexagonal grid, six wolves will always capture the moose (with one wolf on each cell adjacent to the moose's cell) as long as it cannot run faster than they [Korf 1992]. [Manela & Campbell 1995] study the extension of these methods to predator-prey problems of different degrees of difficulty.

1. Moose: move to the neighboring cell that is farthest away from the nearest wolf. As long as its rate of movement is faster than that of the wolves, this strategy permits it to escape.

2. Wolves: move to the neighboring cell with the lowest score

$$S = d(moose) - k * d(wolf)$$

where *d(moose)* is the distance to the moose, *d(wolf)* is the distance to the nearest other wolf, and *k* is a tuning constant modeling a repulsive force between wolves.

Integration.—As in the example of birds and fish, each individual in the wolf-moose system both influences and is influenced by the entire system. Behavior of the overall system depends critically on the relative speeds of moose and wolves (since a fast moose can always escape a pack of slow wolves), and on the value of the parameter *k* that establishes the repulsion among wolves. When repulsion and attraction are suitably balanced, the wolves inevitably surround the moose, without any explicit communication or negotiation of strategies.

4. Engineering Principles

Many of the natural systems described above share some common principles of self-organization. As we learn to recognize and understand these principles, we can construct artificial systems that emulate the desirable behavior we observe in these natural systems. We begin by surveying principles suggested by other researchers, then discuss in more detail a set particularly useful for engineering artificial systems. For the sake of concreteness, each principle is illustrated with reference to one of two actual prototype implementations. The AARIA shop-floor scheduling and control system, which uses a supplier-consumer model to coordinate discrete manufacturing both within a single facility and between trading partners in a supply chain [Parunak et al. 1997]. CASCADE [Parunak et al. 1987] is a self-routing material handling system.

4.1 Other Lists of Principles

The paradox that simple individual behaviors can give rise to complicated overall behavior has been known for centuries. One ancient observer, King Solomon, knew from his father David of the elaborate court organizations of oriental kings and the preparations needed for military campaigns. He marveled that insects could accomplish both these tasks without central control. Thinking of the complex systems needed to maintain the palace commissary (The Bible, 1 Chronicles 27:25-31), he wrote, "Go to the ant ...; consider her ways, and be wise. Having no guide, overseer, or ruler, she prepares her bread in the summer, and gathers her food at harvest time" (The Bible, Proverbs 6:6). He knew the complexity of a military organization (The Bible, 1 Chronicles 11:10-12:37), and was impressed that "the locusts have no king, yet all of them go forth by companies" (The Bible, Proverbs 30:27). Nearly three thousand years later, a participant in the NCMS Virtual Enterprise workshop in 1994 commented, "We used to think that bugs were the problem in manufacturing software. Now we suspect they may be the solution!"

More recently, three works offer lists similar to the list in this paper. Each of these works has its own perspective, distinct from this paper's emphasis on software engineering, and the lists they offer reflect these distinctive objectives. There is not space here to discuss these lists in detail, but a brief summary will provide perspective for the principles highlighted in this paper. The bracketed terms after each item in these other lists indicate the principles in this paper that correspond to them.

4.1.1 Holland, *Hidden Order*

[Holland 1995] outlines an extensive research program devoted to understanding what he calls "complex adaptive systems" (CAS's), roughly analogous to what we call a Multi-Agent System. He offers four properties that such systems share, and three mechanisms that they employ. These features are analytic rather than synthetic. That is, they represent properties and mechanisms common to existing CAS's, not normative recommendations for how to construct new ones. The four properties of CAS's are:

1. Aggregation: CAS's are not constructed monolithically, but consist of smaller components (roughly, our "agents"), which are themselves aggregates of still smaller units. The behavior of the aggregate is often distinct from the individual behaviors of the parts. [Things; Small (Mass)]
2. Nonlinearity: The behavior of CAS's is not linear, and their interactions are thus not simply additive. [Things]
3. Flows: CAS's are characterized by flows of various substances through networks of agents. These flows exhibit two important effects: multiplication (in which one change produces a chain of others) and recycling (feedback loops). [Entropy]
4. Diversity: The agents in a CAS differ from one another. [Diversity]

The three mechanisms are:

1. Tagging: Agents in a CAS are able to recognize and differentiate among one another. [Share Information]
2. Internal Models: The internal structure of an agent in a CAS enables it to anticipate changes in its environment. [Share Information]
3. Building Blocks: An agent's internal model is made up of small, reusable modules, thus enabling it to capture a rich set of alternatives with a limited representational vocabulary. [Share Information]

4.1.2 **Resnick, *Turtles, Termites, and Traffic Jams***

[Resnick 1994] has a pedagogical objective. He wants to understand how people think about decentralized systems, and his five principles are “guiding heuristics” that people can use to learn to understand these systems.

1. Positive feedback isn’t always negative. Sometimes it leads to destructive oscillations, but in other cases it is critical to self-organization. [Entropy]
2. Randomness can help create order, by providing diversity among agents. [Diversity]
3. A flock isn’t a big bird. The behavior of an aggregate system is not the same as the individual behaviors of the lower-level units out of which it is constructed. [Small (Mass)]
4. A traffic jam isn’t just a collection of cars. Decentralized systems generate emergent objects that are distinct from any of the individual parts. [Things]
5. The hills are alive. The environment is an active process that impacts the behavior of the system, not just a passive communication channel between agents. [Entropy]

4.1.3 **Kevin Kelly, *Out of Control***

[Kelly 1994] seeks to identify the laws “governing the incubation of somethings from nothing.” Like ours and unlike those of Holland and Resnick, his principles are prescriptive rather than descriptive. However, his focus is considerably more philosophical and less oriented to engineering concerns than is ours.

1. Distribute being. Emergent behavior is distributed across the components, rather than localized in any single one. [Decentralize]
2. Control from the bottom up. Wide-ranging, concurrent changes frustrate central top-down control. [Decentralize; Small (Scope)]
3. Cultivate increasing returns. Positive feedback is essential to large, self-sustaining systems. [Entropy]
4. Grow by chunking. Large monoliths are unreliable. Complexity should be assembled incrementally from small independently-functioning modules. [Small (Mass)]
5. Maximize the fringes. A diverse entity is more adaptable and robust than a homogeneous one. [Diversity]
6. Honor your errors. Competitive superiority depends on breaking out of conventional patterns, which is indistinguishable from error. [Diversity]
7. Pursue no optima; have multiple goals. Survival and adaptation require an exploratory approach that is necessarily sub-optimal in a formal sense. [Plan and Execute Concurrently]
8. Seek persistent disequilibrium. A useful system must balance stability with constant change, otherwise it will degenerate either to convulsions or to death. [Entropy]
9. Change changes itself. In dealing with changing circumstances, complex systems change, and over time the rules of change themselves undergo change. [Share Information]

4.2 Agent Things, not Functions

Classical software engineering techniques lead many systems designers toward “functional decomposition.” For example, manufacturing information systems typically contain modules dedicated to functions such as

"scheduling," "material management," and "maintenance." Carried over to agent-based systems, these instincts lead to individual agents assigned to such functions as Logistics, Transportation Management, Order Acquisition, Resource Management, Scheduling, and Dispatching [Fox et al. 1993, Barbuceanu & Fox 1995].

The functional approach is well suited to centralized systems, but unprecedented in naturally occurring systems, which divide agents on the basis of distinct entities in the physical world rather than functional abstractions in the mind of the designer. A functional decomposition appears most natural when the distinction between agents and their environment is overlooked. A clear distinction between agent and environment (such as that forced by discrete-event dynamics in the agent and time-based dynamics in the environment), and a recognition that the environment's own process mediates agent interactions, force the designer to pay attention to topological boundaries between agents and environment and make it more difficult to assign agents to arbitrarily conceived functions.

Even in natural systems, functions are important, but they emerge from the interactions of the individual components rather than being explicitly represented as units in their own right. Holland's Nonlinearity principle explains how such unexpected behaviors come about. The temptation to functional decomposition reflects Resnick's fallacy of identifying a traffic jam with a collection of cars, and recognizing that fallacy is an important first step to avoiding the error. Holland's second sense of Aggregation (of agent behaviors rather than agents themselves) also focuses on this issue.

ERIM's experience with agent-based prototypes suggests that each functional agent needs detailed knowledge of many of the physical entities being managed. When the physical system changes, the functional agent needs to change as well. However, it is often possible to define generic behaviors for physically defined agents from which the required functionality will emerge, for widely varying overall populations of agents.

There are two important exceptions to this principle: legacy systems and watchdogs.

Most industrial agent applications are additions to existing systems. They need to interface with *legacy systems*, many of which are functionally oriented. For example, a shop-floor control system needs to interface with a factory-wide MRP system that is doing classical scheduling. A reasonable way to connect the legacy system to the new system is to encapsulate it as an agent. Even though the MRP system may be functionally defined, as a legacy program it is a well-defined "thing" in the application domain and so deserves agenthood. Its presence in the system will not preclude the development of techniques for emergent scheduling as long as its agent behaviors restrict it to serving as a link with the rest of the system rather than drawing on its centralized scheduling behavior.

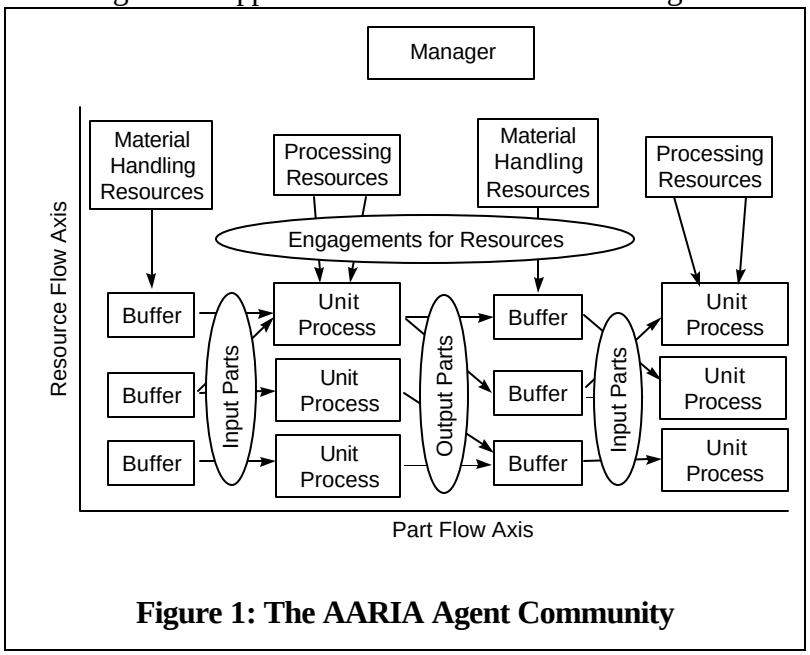

Figure 1: The AARIA Agent Community

Some system states that local agents cannot perceive may need to be monitored to ensure overall system safety or performance. A *watchdog agent* that simply monitors the behavior of a population of physical agents is not nearly as restrictive on future reconfiguration as one that does centralized planning and

action. It is best not to rely on watchdogs at all, but if they are used, they should sense conditions and raise signals but not plan or take action.

The ovals and rectangles in Figure 1 show the main agent classes in the AARIA agent community. Each material handling resource (such as a truck or a fork-lift), each processing resource (such as a machine tool or an operator), and each part is a separate agent. There are no separate agents for such standard functions as scheduling or logistics, but these functions emerge from the interaction of the local agents. AARIA's Manager agent is an example of a watchdog, who maintains a global view of the operation and raises signals to alert the other agents when something undetectable on the local level needs attention.

4.3 Keep Agents Small

Naturally occurring adaptive systems have parts that are small compared with the entire system, in mass, time, and space.

4.3.1 Small in Mass

Each termite is an almost negligible part of the entire termite hill. As a result, the behavior of the whole is stable under variations in the performance of any single member, and the collective dynamics dominate. This and similar examples suggest implementing artificial systems with large numbers of agents, each small in comparison with the whole system.

The motivation for this principle derives not from our theory of multi-agent systems, but from the experience of software engineers that the difficulty of designing, implementing, and launching computer-based systems increases exponentially with the size of the system. Small individual agents are easier to construct and understand than large monolithic systems, and the impact of the failure of any single agent will be minimal. In addition, a large population of agents gives the system a richer overall space of possible behaviors, thus providing for a wider scope of emergent behavior. Very roughly, the number of agents is a multiplicative factor in determining the implementation effort, but an exponent in determining the size of the overall system state space. The effort to code 100 agents with 10 behaviors each is on the order of $100*10 = 10^3$, but the resulting state space is on the order of 10^{100}.

Keeping agents small often means favoring specialized agents over more general ones, using appropriate aggregation techniques. For example, a complete manufacturing cell would be extremely complex as a single agent, but can be developed as a community of agents for individual mechanisms (e.g., one for the fixture, one for the tool, one for the load-unload mechanism, one for the gauging station). Our model of a multi-agent system directly supports such aggregation, since an environment (and its associated agents) can become a higher-level agent by defining its inputs and outputs to another environment. The BRIC architecture (Ferber 1995) is an important first step in developing such systems.

Holland's Aggregation property, Resnick's Flock/Bird distinction, and Kelly's rule, "Grow by Chunking," all contrast the mass of the system and that of the individual agents.

AARIA's assignment of a separate agent to each machine, each part, and each tool in a manufacturing enterprise provides agents that are much lighter weight than traditional shop-floor software systems. AARIA does not further decompose workstations into sensor and actuator agents, only because the scope of the project is to investigate issues at the level of the shop floor rather than individual device engineering. The industrial controls marketplace offers increasing support for agents as small as the sensor/actuator level,

including small controllers of traditional design (Rockwell Allen-Bradley, Mitsubishi, Zworld) as well as more novel architectures combining computation, control, and networking (Echelon, Basic Stamp).

4.3.2 Small in Time (Forgetful)

Naturally occurring agent systems can forget. Pheromones evaporate, and as a result obsolete paths leading to depleted food sources disappear rather than misleading members of the colony. The probability that a wasp will forage decreases as it successfully resists stimulation. Even the death of unsuccessful organisms in an ecosystem is an important mechanism for freeing up resources so that better adapted organisms can flourish.

The mechanism of forgetting is an important supplement to the emphasis in conventional AI systems on mechanisms for learning. In a discrete-event system, forgetting can be as complex as learning, since both represent discrete state transitions. In a time-based system, forgetting can take place "automatically" through the attenuation of a state variable that is not explicitly reinforced. The example of pheromones suggests that a time-based environment can support a "forgetting" function even for discrete-event agents.

CASCADE [Parunak et al. 1987] shows how an artificial agent system can forget. In CASCADE, each segment of a switched conveyor system is a separate agent that seeks to maintain its population of a given part type within certain limits. If the population rises above the upper limit, the segment seeks to spill excess parts to some neighboring segment, while if the population falls below the minimum, the segment sends requests out to neighboring segments. The probability that a segment will spill a part to a given neighbor is a state variable for the spilling segment, and is increased each time the neighbor requests a part and decreased each time the segment spills a part to the neighbor. Because the information on a neighbor's interest in a part is maintained as a real number rather than symbolically, obsolete behaviors are forgotten automatically as the value is modified to reflect more recent behavior. The rate at which the probability changes in response to interactions with neighboring segments is a tuning parameter.

4.3.3 Small in Scope (Local Sensing and Action)

The participants in natural systems usually can sense only their immediate vicinity. In spite of this restriction, they can generate effects that extend far beyond their own limits, such as networks of ant paths or termite mounds. Wolves are built close to the ground and so can't see very far, but manage to coordinate a hunt with other wolves over a wide area. Kelly's exposition of the principle, "Control from the bottom up," recognizes the superiority of many local interactions over a few global ones.

Though high-bandwidth wide-area telecommunications makes it increasingly easy to connect each artificial agent directly with every other agent, it may be beneficial to follow the natural examples and engineer agents that limit the recipients of their messages. Telecommunications technology means that these limitations need not be geographical, but natural examples suggest that effective systems will restrict communications in some way. Wherever possible, agents should define the audience that needs to receive the message, at least by subject-based addressing, rather than broadcasting information. Several lines of research suggest the need for such a restriction.

A suggestive analogy comes from the history of physics. Newton's classic equation for gravitational force

$$F = gM_1M_2/r^2$$

usefully *describes* the effects of gravity, but does not *explain* the underlying mechanism. Massive bodies do not figure out how hard to pull on one another by communicating their masses and measuring the distance that

separates them. Einstein's theory of general relativity replaces this impossible notion of action-at-a-distance with a model of local interaction in which masses warp space in their immediate vicinity and respond to the local geometry of the space in which they find themselves. Localization of agent interactions thus follows in the intellectual tradition of modern physics.

[Kauffman 1993] describes experiments with randomly constructed networks of Boolean operators that show that self-organization works better in sparsely-connected systems than in densely-connected ones (compare [Hogg et al. 1989]). The networks under study jump from one point in their state space to another as they evolve. A given network is finite and deterministic, so eventually it will enter a cycle of states, but given the size of a network's state space ($\approx 2^{1000}$), one might expect both the time needed to enter a cycle and the length of a cycle to be very long. Surprisingly, such networks can repeat in about 100 steps, with a period of repetition on the order of 10 to 20 steps! This self-organization appears only when the network is sparsely connected (for example, when each element is connected to no more than three or four others). Networks with very dense connectivity do not settle down easily into loops, but exhibit unstable behavior.

Software engineering offers another argument for local agent communications. [Dijkstra 1968] warned of the dangers of the Fortran GOTO statement, which gave the programmer the ability to jump from anywhere to anywhere in a program. This powerful tool led to tangled mazes of spaghetti code that were easy to break and almost impossible to correct and maintain. More disciplined structures proved to have the same expressive power, while supporting modularity and restricted interfaces that limited the propagation of faults. Global data reference has the same kind of engineering implications that global transfer of control does. In both cases, direct remote interactions are difficult for humans to understand, maintain, and control. In both cases, global effects can be obtained by propagation of local influences, much more robustly than by providing global influences.

Evidence most closely related to our model of multi-agent systems comes from experiments of [Takashina & Watanabe 1996] on the impact of sensor scope on the behavior of agents in a predator-prey simulation. They find a direct correlation between the entropy of the output from an agent's sensors and the quality of the agent's performance. For a given sensor bandwidth, as the range of an agent's sensors increases, entropy (and performance) first increase, then decrease. The work of [Deneubourg et al. 1991] on how ants sort their nests yields a similar result. An ant whose short-term memory is too long "sees" the entire nest at once and is unable to sort. In simple terms, giving an agent access to too much of the world may lead to sensory overload, reduced ability to discriminate, and lower performance. In terms of our <Agents, Environment, Coupling> model, if agents are small compared to the environment, their State will have fewer elements than the environment's State, and their Input and Output subsets of State will be even smaller. The accuracy of the model of the environment presented by an agent's Input will begin to decrease once the agent's scope of sensing is so large that the portion of the environment accessible to it contains more information than its Input can model. Thus "small in scope" is a direct consequence of "small in mass."

Both AARIA and CASCADE explicitly limit the scope of agent communication. Figure 1 shows how each class of AARIA agent (represented by a rectangle or an ellipse) interacts only with certain other classes of agents. In practice, this limitation is imposed through subject-based addressing and agent migration. In subject-based addressing, agents subscribe to certain classes of messages and thus are not distracted by messages irrelevant to their interests. Agent migration supports high-bandwidth conversations between two specific agents by moving them onto the same physical processor. In CASCADE, each segment of the material handling system communicates only with physically adjacent segments, and routing through the network takes place without using any global system map.

4.4 Decentralize System Control

Originally computers were large, expensive systems. Their physical size and sensitivity to their environment led to the "computer room," a specially constructed and climate-controlled sanctuary tended by a high priesthood of operators. Because computers were so expensive, there were few of them, and because computer rooms were expensive, even if a firm had more than one computer, they were often housed in the same facility. Computational needs throughout the facility were all handled at the same central location. Until the spread of time-sharing systems in the mid-1970's, most computers ran one application at a time, encouraging software engineers to take a large-grained, centralized approach to system computerization.

Such centralization is foreign to natural systems. To echo King Solomon, ants have no overseer, and locusts have no king. Even the distinguished individual in some insect colonies (such as the chief wasp in the *Polistes* hive or the queen honeybee) does not give orders, but instead serves as a communication bus for the rest of the community. In our model, only the environment has a state space large and rich enough to represent the entire system, and unless the environment is itself a computer, programming its process directly is a matter of physical and chemical engineering that is usually more difficult and fragile than programming an agent's (computer-based) process.

This observation does not mean that we should neglect engineering the environment; on the contrary, the <Agents, Environment, Coupling> model emphasizes that we must pay attention to both components of the system. However, the rich information interface that computers support is available only through the agents, and there are important reasons not to make one agent large and complicated enough to control the entire system.

- A central agent is a single point of failure that makes the system vulnerable to accident.
- Under normal operating conditions, it can easily become a performance bottleneck.
- Even if it is adequately scaled for current operation, it provides a boundary beyond which the system cannot be expanded.
- It tends to attract functionality and code as the system develops, pulling the design away from the benefits of agents and in time becoming a large software artifact that is difficult to understand and maintain.

Centralization can sometimes creep in when designers confuse a class of agents with individual agents. For example, one might be tempted to represent a bank of paint booths as "the paint agent," because "they all do the same thing." Certainly, one would develop a single class (in the object-oriented sense of the word) for paint-booth agents, but each paint booth should be a separate instantiation of that class.

The importance of distributed decentralized systems is axiomatic in Holland, Resnick, and Kelly. Kelly makes this point explicitly in his principles "Distribute being" and "Control from the bottom up."

4.5 Support Agent Diversity

We develop the concept of agent diversity in three steps.

1. Why should agents be diverse? Ecological examples and our model show the value of diverse populations.
2. How can diversity be established and maintained? Random processes and repulsive fields are important tools.
3. What changes are necessary in management philosophy to support diversity? Managers need to recalibrate their acceptable thresholds for risk and redundancy.

4.5.1 The Value of Diversity

The difference in size between an individual agent's State and that of the environment not only favors small agents and decentralized control, but also encourages diversity among agents. The environment's State contains information concerning both opportunities that agents should exploit and threats that they should avoid. The more of the environment's State the agents can sense and modify, the better they can exploit those opportunities and avoid those threats. Any single agent can model and manipulate only a small portion of the environment, and a population of completely identical agents will do no better, since they will still cover only the same subset of the environment's state. A population of diverse agents will cover more of the environment's state and thus provide better performance. We can illustrate this principle mathematically for the homodynamic discrete-event case by observing that if the state of Agent_i is State_i and the environment has State_e, we want to maximize

$$\left\| \left(\bigcup_i Input_i \cup Output_i \right) \cap State_e \right\| \Big/ \left\| State_e \right\|$$

where $\|x\|$ measures the number of different state variables in x.[4]

While diversity is not the same as population size, the two are correlated. In a physical environment, it is impossible for two agents to occupy the same place at the same time. Thus two otherwise identical agents will be at different locations in the environment. They will differ in that element of their respective States that records their location. This simple but crucial element of diversity enables them to monitor different elements of the environment's state, and thus collectively be more robust than a single agent could be. The important observation is that the advantage of the larger population lies not merely in numbers, but in the diversity that results from physical exclusion laws.

Natural populations often have a "critical size" that is much larger than the simple breeding pair that a standard accounting paradigm might justify. If the population falls below this level, the colony dies out. Once an ant finds food, it generates a pheromone trail to guide other ants to it, but the critical initial discovery depends on having enough ants wandering around that some will stumble across food, wherever it may be. Unused alternatives (unsuccessful scouts) are insurance for unforeseen change, not waste.

The example of similar agents at different locations illustrates that diversity is not the same as incompatibility. The diversity in location among ants is able to benefit the society as a whole because in many other respects the ants are interchangeable. They all like the same kind of food, and they all lay down and sense the same pheromones. Their diversity of location would be of no value if they were not similar enough to one another to share in the benefits that diversity conveys.

Diversity can be quantified by formalizing these insights. Treating this problem adequately is beyond the scope of this paper, but one might begin by defining the square symmetric matrix M over Agents such that

$$M_{ij} = \left\| State_i \cap State_j \right\| \Big/ \left\| State_i \cup State_j \right\|$$

[4] This example is only an illustration. For example, sometimes two agents with the same state variables (e.g., "Location") can sample different aspects of the environment. Full formalization of the concept of diversity is beyond the scope of this paper.

A homogeneous population is one for which every element of M is 1. An incompatible population is one for which some element of M^n is 0 for every value of n (that is, some subpopulations of agents cannot interact with one another). A diverse population is one that is neither homogeneous nor incompatible. Measures of diversity can be derived in terms of various matrix theoretic characteristics, such as the average value of the elements of M or the least value of n for which M^n has no zero elements.[5]

Artificial systems engineered for leanness are liable to fail if their environment changes unexpectedly. One price of being able to thrive in a constantly changing environment is a willingness to support a diverse set of resources that go beyond the needs of the immediate environment [Preiss 1995]. Holland captures this insight as his Diversity principle; Kelly, as "Maximize the fringes."

AARIA supports diversity by instantiating agents for every individual machine, operator, tool, and other resource available in the factory. It cannot enforce diversity. A management decision to equip a ship with only milling machines will result in a shop that cannot produce rotational parts. However, if management adds lathes, AARIA will enable processes that need them to identify and exploit them.

4.5.2 Achieving Diversity: Randomness and Repulsion

Resnick's Randomness principle recognizes that formally random behavior can be a good way to achieve the diversity a population of agents needs in order to adapt. Randomized agents attack a problem in a Monte Carlo fashion that does not require a detailed advance model of the domain, and lend themselves to much simpler coordination and inferencing algorithms than approaches that require an explicit reason for every decision.

The use of Fermi functions and other weighted probabilistic mechanisms in insect communities is an example of using randomness to achieve diversity. Both AARIA and CASCADE rely on similar random mechanisms to identify and exploit opportunities. As described above, a conveyor segment in CASCADE selects a neighbor to which to spill an excess part probabilistically, in effect by rolling a die that is weighted by past experience with each neighbor's requirements. In AARIA, individual resources search their projected utilization level stochastically to decide when to offer their services to a unit process.

The wolf and bird examples show how a simple repulsion among agents can maintain diversity of location. The same technique could be applied to other aspects of the relation among agents. For example, in a machine job shop, a new job can be processed more rapidly if the necessary tooling is already on the machine. Machine agents representing otherwise identical machine tools in such a shop might use a repulsive force with respect to their tooling characteristics to see to it that the currently unused machines have diverse tooling, thus increasing the chances that a new job will find a machine ready to run it.

---

[5] Again, this formalization is intended only as an illustration. Among other shortcomings, it is restricted to homodynamic discrete-event systems, it ignores the probability that the environment's Process will couple states sensed or altered by otherwise disjoint populations of agents, it does not take into account possible differences in input and output capabilities among agents with the same underlying state variables, and it does not consider the degree of similarity among the Processes of different agents.

4.5.3 Management Concerns: Risk

An ant who wanders off in search of food where there is none may never return to the nest. Traditional industrial analysis would consider the insect an example of wasted resource or under-utilization, and would insist that ants file detailed exploration plans (instead of wandering randomly) and head home when their energy level reaches one-half (instead of going beyond the point of no return in hopes of refueling at a yet-to-be-discovered food source). An ant hill that took such measures to guarantee the security of its ants would be less effective in identifying outlying food sources, and thus less robust overall.

Kelly canonizes this observation as "Honor your errors," observing that error and innovation are indistinguishable before the results are in. Managers need to recognize risky behavior as an important mechanism for providing the diversity a population needs to survive change.

The adaptability of an economy as a whole depends on the existence of a wide range of attitudes toward risk, ranging from highly conservative (optimized to the current environment, but unable to change) to highly daring. Most of the latter group will fail, but those who succeed enable the society to adapt to new conditions, and enjoy dramatic rewards. Compare the rule of thumb among some venture capitalists that investment returns are maximized by a portfolio in which no more than 20% of the startups succeed [Morley 1995].

4.5.4 Management Concerns: Redundancy

Compared with conventional systems, natural agent-based systems seem wasteful. They allocate far more resources to a task than a global analysis would require, and sometimes seem to throw resources away. For example, an ant hill's ignorance of where food may be found requires a large army of scouts, some of whom wander far from both nest and food and either starve or are taken by predators.

The example of the ant hill shows that redundancy supports diversity in two ways. The diversity in location among the ants enhances the colony's chances for finding food and thus surviving, while the small size of one ant in comparison with the colony means that several individuals can perish without endangering the entire community. These two insights reflect our earlier distinction among homogeneity, diversity, and incompatibility. Because diverse agents are not homogeneous, they can monitor an environment that is much more complex than any single agent. Because they are not incompatible, the community as a whole can tolerate the loss of any one individual, drawing on the overlapping capabilities of others.

Sometimes lack of redundancy in the application domain leads to centralization. If every product in a plant must pass through a single paint booth, it is all too easy to give the agent representing that booth strong central characteristics. For example, with multiple paint booths, booths might bid for products, but with only one, it calls the shots unilaterally. In such a case, it is sometimes possible to reduce this effect by imagining that there are two or more such agents, and letting the "real" one regularly win run-time competitions. Only the "real" agent would ever win a bid for work, because the others would always report that their machines are off-line. If there is ever a need for new machines, they can be added without modifying the overall design.

Redundant agents can step in for one another because agents do not communicate directly with one another, but only mediately, by making changes in the environment that are subsequently sensed by other agents. Any agent that senses these changes can respond to them, and failure of an individual agent does not bring the system down as long as there are other agents sufficiently similar to sense the same environmental changes and respond appropriately to them.

AARIA supports redundancy in production capability by using a negotiation protocol to identify potential suppliers for the inputs to a given unit process. For example, a unit process that needs a milling machine issues

a request indicating the required machine class in the subject, and through subject-based addressing, all machines of that class that are currently on the network will receive the request. Through a negotiation, the unit process discriminates among available machines on the basis of criteria such as operating cost and availability, and selects a platform on which to execute. If one machine breaks down or is fully loaded, this approach permits another to take its place, as long as the physical shop has redundant capabilities.

4.6 Provide an Entropy Leak

The Second Law of Thermodynamics observes that closed systems progressively become more disordered over time. It is not obvious that a large collection of agents will organize itself to do useful things. The Second Law warns that the result of such an architecture may be disorder.

Natural agent-based systems do organize themselves with striking efficiency. A common explanation is that a system can become more organized if energy is added to it from the outside (for example, by the metabolism of the food gathered by an insect hive). The addition of energy is necessary for self-organization, but hardly sufficient. Gasoline in construction equipment can erect a building, but the same gasoline in a terrorist's bomb can destroy it.

In natural systems, agents can organize themselves at the macro level because their actions are coupled to a dissipative or disorganizing process at a micro level. The system can reduce entropy at the macro level by generating more than enough entropy at the micro level to pay its second-law debt. To adopt another metaphor, it provides an entropy leak to drain disorder away from the macro level (where useful work is done) to the micro level (where it won't interfere with the system's function).

In one model of this leakage, micro-level dissipation in the environment generates a flow field that the agents can perceive and reinforce and to which they can orient themselves [Kugler & Turvey 1987]. Insect colonies leak entropy by depositing pheromones whose molecules, evaporating and spreading through the environment under Brownian motion, generate entropy. The resulting flow produces a field that the insects can perceive and to which they orient themselves in making further pheromone deposits. Figure 2 generalizes this example in terms of the three fundamental processes: micro dissipation, macro perception of the micro flow field, and macro reinforcement of the micro dissipative mechanism.

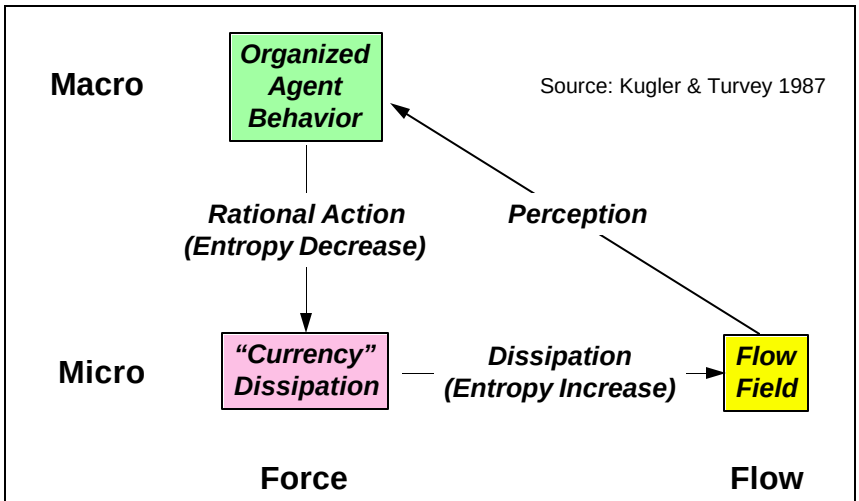

Figure 2: Macro Organization through Micro Dissipation

In other cases, the marker dissipates among the agents themselves rather than in the environment, so that the individual agents belong to the "Micro" scale level, leaving only the emergent structures at the macro level. One example is the flow of force among wasps that drives the emergence of the three roles in the hive. Another example is the movement of currency in a market economy. Money benefits its holders only when they spend it. As it spreads from purchasers to buyers, entrepreneurs perceive the resulting flow field and

orient themselves to it, resulting in the self-organization of structures such as supply chains and geographic economic centers.

Artificial agent communities will be more robust and better able to organize themselves if they are designed to include a dissipative mechanism such as a currency. This mechanism should have three characteristics:

1. It must flow (either among agents or through the environment), thus setting up a gradient field.
2. The agents must be able to perceive this field and orient themselves to it.
3. The agents' actions must reinforce the field (positive feedback).

Principles and mechanisms related to the need for an entropy leak are in the forefront of the lists presented by Holland, Resnick, and Kelly. All three discuss the need for positive feedback (Holland under the rubric of Flow and Kelly as Increasing Returns). Kelly's paradox of "Persistent disequilibrium" reflects the tension between high-level order and low-level disorder. Resnick's insistence that "The Hills are Alive" points to the importance of the environment, which is usually where we need to dump excess entropy, and points up the importance of understanding the environment explicitly in designing a multi-agent system.

In AARIA, each agent participates in an economy in which it buys the products and services it requires from others and sells its own products and services. The flow of currency among agents guides their decisions, thus providing an entropy leak that supports self-organization of the entire system. The use of a currency to drive self-organization in many implementations is documented in [Clearwater 1996]. [Drogoul 1995] offers a fascinating example of how a simple dissipative mechanism of another sort can enable a rudimentary algorithm to play a respectable amateur game of chess.

4.7 Enable Agents to Share Information

Natural systems exchange information among members of the population, at three levels: the species, the individual, and the society. Sexual reproduction exchanges information from one generation of a species to another, by passing on successful characteristics in the form of chromosomes to offspring. Individual organisms can also pass on skills post-embryonically. For example, young black bears learn to rob food caches hung from trees by watching older bears. The society as a whole can learn even if individual members do not, as in the development of pheromone paths in an insect colony. In each case, a community reduces the need for expensive search by finding ways to cache and share accumulated knowledge. In the <Agents, Environment, Coupling> model, species and individuals learn by modifications to agents' State and Process components, while societies learn by modifying the environment's State.

Artificial agent systems can often use similar mechanisms. While learning in a single agent can require sophisticated techniques, methods for learning across generations, such as genetic or evolutionary programming [Fogel 1995, Mitchell 1996, Koza 1992], have proven their maturity in numerous applications, and changing a community's structure to enable its members to respond better to a changed environment is straightforward for agent architectures.

Holland's Tagging, Internal Model, and Building Block mechanisms support the sharing of information among agents, while Kelly's concept of change changing itself reflects the accumulation and transmission of knowledge from one generation to another.

AARIA agents share information explicitly, in the content of messages that they pass to one another. In addition, the AARIA architecture supports genetic modification of various aspects of agent behavior, though this feature is not yet implemented.

4.8 Plan and Execute Concurrently

Traditional systems alternate planning and execution. A firm develops a schedule each night that optimizes its manufacturing operations the next day. Unfortunately, changes in the real world tend to invalidate advance plans. Manufacturing engineers in industries as diverse as semiconductors, aerospace, agricultural equipment, and motor vehicles agree that a daily schedule is usually obsolete less than an hour after the day begins, and from that point on serves to guide expediters, not to tell when specific manufacturing resources will be working on particular jobs. As Kelly recognizes in his principle, "Pursue no optima," theoretical optima are useless if they depend on a steady state that is never achieved.

The problem is a natural consequence of a system in which the environment as well as the agents has a Process that can autonomously modify the environment State with which agents interact. Although traditional artificial intelligence has devoted considerable attention to the "plan, then execute" model, [Agre & Chapman 1987] show the remarkable functionality of concurrently reasoning about a situation and acting in it, in a highly dynamic environment (Pengo, an arcade video game) that would frustrate traditional planning models.

Natural systems do not plan in advance, but adjust their operations on a time scale comparable to that in which their environment changes. The coherence of their behavior over time is maintained by the dynamics of their interactions, not imposed by an external plan or schedule. To achieve the robustness exemplified by these systems, artificial agents should seek to avoid the "plan then execute" mode of operation and instead respond dynamically to changes in the environment.

AARIA's mechanism for scheduling tasks on resources is an example of concurrent planning and execution. The actual time at which a job will execute may not be known until the job starts. The resource does not schedule a newly-arrived job at a fixed point in time, but estimates probabilistically the job's impact on its utilization over time, based on information from the customer about the earliest and latest acceptable delivery times. The width of the window within which the job can be executed is incrementally reduced over time as needed to add other jobs to the resource's list of tasks. If the resource is heavily loaded, the jobs organize themselves into a linear sequence, but if it is lightly loaded, the actual order in which jobs are executed is decided at the moment the resource becomes available, depending on the circumstances that exist at that time.

5. Evaluation

The approach to system design and management outlined in this paper is at odds with the centralized top-down tradition in systems engineering, and potential users will reasonably want assurance that the approach will work better than conventional methods. The question usually arises in terms of the contrast between local and global optimization. Decision-makers fear that by turning control of a system over to locally autonomous agents without a central decision-making body, they will lose value that could have been captured by a more global approach.

The benefits of agent-based approaches over centralized ones are conditional, not absolute. In a stable environment, a centralized approach can be optimized to out-perform the initial efforts of an opportunistic distributed system. If the distributed system has appropriate learning capabilities, it will eventually become as efficient, although a strong enough optimized competitor may kill it before it learns enough. However, modern market conditions are marked by rapid and unpredictable change, not stability. [Agre 1988] argues more generally that change and contingency are inescapable features of the real world. In this case the appropriate comparison for systems designers is not between local and global optima, but between static and adaptable systems.

Managers should evaluate the competing options in a particular case theoretically, strategically, tactically, and practically.

Theoretically, there are decentralized mechanisms that can achieve global coordination. For example, economists have long studied how local decisions can yield globally reasonable effects, and recently these insights have been applied to a number of domains that have not traditionally been considered as economic, such as network management, manufacturing scheduling and control, and pollution control [Clearwater 1996]. Recent studies in mathematical ecology and artificial life are another locus of emerging theoretical insights.

Strategically, managers must weigh the value of a system that is robust under continual change against one that can achieve a theoretical optimum in a steady-state that may never be realized. A company that anticipates a stable environment may well choose centralized optimization. One that installs agent-based software does so because it cannot afford to be taken by surprise.

Tactically, the life-cycle software costs are lower for agent-based systems than for centralized ones, because agents can be modified and maintained individually at a fraction of the cost of opening up a large, complex software system. In systems that must be modified frequently, losses due to suboptimal performance can be more than recovered in reduced system maintenance expenses.

Practically, agent-based systems that follow these principles are not untried. [Parunak 1996] and [Parunak 1998] report on a number of industrial agent-based systems that have been piloted or deployed in regular operation. The agents in these systems regularly reflect the principles outlined here rather than those of centralized systems, and their growing acceptance in competitive business environments is evidence of the benefit they bring their adopters.

6. Summary

Recent trends in software architecture are leading to the widespread use of software agents, or active objects with initiative. Some researchers design these agents with the same principles that were developed for monolithic systems. Examination of naturally occurring agent-based systems suggests some radically different design principles for the next generation of computer systems. While particular circumstances may warrant deliberate exceptions, in general:

1. Agents should correspond to things in the problem domain rather than to abstract functions.
2. Agents should be small in mass (a small fraction of the total system), time (able to forget), and scope (avoiding global knowledge and action).
3. The agent community should be decentralized, without a single point of control or failure.
4. Agents should be neither homogeneous nor incompatible, but diverse. Randomness and repulsion are important tools for establishing and maintaining this diversity.
5. Agent communities should include a dissipative mechanism to whose flow they can orient themselves, thus leaking entropy away from the macro level at which they do useful work.
6. Agents should have ways of caching and sharing what they learn about their environment, whether at the level of the individual, the generational chain, or the overall community organization.
7. Agents should plan and execute concurrently rather than sequentially.

References

[Agre & Chapman 1987] P.E.Agre and D.Chapman, “Pengi: An Implementation of a Theory of Activity.” *Proceedings of AAAI-87*, 268-272.

[Agre 1988] P.E.Agre, “The Dynamic Structure of Everyday Life.” Ph.D. Dissertation, Dept. of Computer Science, Massachusetts Institute of Technology.

[Barbuceanu & Fox 1995] M.Barbuceanu and M.Fox, “The Architecture of an Agent Based Infrastructure for Agile Manufacturing: Extended Abstract.” IJCAI-95 Workshop on Intelligent Manufacturing.

[Clearwater 1996] S.H.Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation.* Singapore: World Scientific.

[Deneubourg et al. 1991] J.L.Deneubourg, S.Goss, N.Franks, A.Sendova-Franks, C.Detrain, and L.Chretien, "The Dynamics of Collective Sorting: Robot-Like Ants and Ant-Like Robots." In [Meyer & Wilson 1991], 356-365.

[Dijkstra 1968] E.W.Dijkstra, “Go to statement considered harmful.” *Comm. ACM* 11 (March) 147-148.

[Drogoul 1995] A.Drogoul, “When Ants Play Chess (Or Can Strategies Emerge from Tactical Behaviors?).” In C.Castelfranchi and J.-P.Müller , *From Reaction to Cognition: Selected Papers, Fifth European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW ’93)*, 13-27.

[Ferber 1995] J.Ferber, *Les systèmes multi-agents: vers une intelligence collective.* Paris: InterEditions.

[Ferber & Müller 1996] J.Ferber and J.-P.Müller, “Influences and Reactions: a Model of Situated Multiagent Systems.” *Proceedings, Second International Conference on Multi-Agent Systems (ICMAS-96)*, 72-79.

[Fogel 1995] D.B.Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.

[Fox et al. 1993] M.S.Fox, J.F.Chionglo, and M.Barbuceanu, “The Integrated Supply Chain Management System.” Working Paper, Enterprise Integration Laboratory, University of Toronto, available at http://www.ie.utoronto.ca/EIL/public/iscm-intro.ps .

[Goss et al. 1989] S.Goss, S.Aron, J.L.Deneubourg, and J.M.Pasteels, "Self-organized Shortcuts in the Argentine Ant." *Naturwissenschaften* 76, 579-581.

[Heppner 1990] F.Heppner, "Of Flocks and Chaos." *Bioscience* 40:6, 429-30.

[Holland 1995] J.H.Holland, *Hidden Order: How Adaptation Builds Complexity.* Reading: Addison-Wesley.

[Kauffman 93] S.A.Kauffman, *The Origins of Order: Self Organization and Selection in Evolution.* Oxford University Press.

[Kelly 1994] K.Kelly, *Out of Control: The Rise of Neo-Biological Civilization.* Addison-Wesley.

[Korf 1992] R.E.Korf, "A Simple Solution to Pursuit Games." *Working Papers of the Eleventh International Workshop on Distributed Artificial Intelligence.*

[Koza 1992] J.R.Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press.

[Kugler et al. 1990] P.N.Kugler, R.E.Shaw, K.J.Vincente, and J.Kinsella-Shaw, "Inquiry into intentional systems I: Issues in ecological physics." *Psychological Research*.

[Kugler & Turvey 1987] P.N.Kugler and M.T.Turvey, *Information, Natural Law, and the Self-Assembly of Rhythmic Movement*. Lawrence Erlbaum.

[Manela & Campbell 1995] M.Manela and J.A.Campbell, "Designing Good Pursuit Problems as Testbeds for Distributed AI: A Novel Application of Genetic Algorithms." In Castelfranchi and Pierre Müller , editors, *From Reaction to Cognition: Selected Paper from the 5th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, MAAMAW ’93*. Lecture Notes in Artificial Intelligence 957. Berlin: Springer, 231-252.

[Meyer & Wilson 1991] J.A.Meyer and S.W.Wilson, *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press.

[Mitchell 1996] M.Mitchell, *An Introduction to Genetic Algorithms*. MIT Press.

[Morley 1995] R.Morley, personal communication.

[Müller 1996] J.P.Müller, *The Design of Intelligent Agents*. Lecture Notes in Artificial Intelligence 1177. Berlin: Springer.

[Parunak 1996] H.V.D.Parunak, “Applications of Distributed Artificial Intelligence in Industry.” In G.M.P.O’Hare and N.R.Jennings, editors, *Foundations of Distributed Artificial Intelligence*. New York: John Wiley.

[Parunak 1998] H.V.D.Parunak, “Industrial and Practical Applications of DAI.” In G.Weiss, editor, *Introduction to Distributed Artificial Intelligence*. MIT Press (forthcoming).

[Parunak et al. 1987] H.V.D.Parunak, J.Kindrick, and B.Irish, “Material Handling: A conservative Domain for Neural connectivity and Propagation.” *Proceedings of the Sixth National Conference on Artificial Intelligence, AAAI-87*, 307-311.

[Parunak et al. 1997] H.V.D.Parunak, A.D.Baker, and S.J.Clark, “The AARIA Agent Architecture: An Example of Requirements-Driven Agent-Based System Design.” *Proceedings of the First International Conference on Autonomous Agents (ICAA-97)*.

[Port & vanGelder 1995] R.F.Port and T.vanGelder, *Mind as Motion: Explorations in the Dynamics of Cognition*. Cambridge, MA: MIT Press.

[Preiss 1995] K.Preiss, “Mass, Lean, and Agile as Static and Dynamic Systems.” Agility Forum Perspectives on Agility Series, Volume PA95-04, ISBN 1-885166-07-9.

[Resnick 1994] M.Resnick, *Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds*. MIT Press.

[Reynolds 1987] C.W.Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model." *Computer Graphics* 21:4 (July), 25-34.

[Simon 1981] H.A.Simon, *The Sciences of the Artificial*. 2nd Edition. Cambridge, MA: MIT Press.

[Steels 1991] L.Steels, "Toward a Theory of Emergent Functionality." In J.A.Meyer and S.W.Wilson, eds., *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*. MIT Press, 451-461.

[Takashina & Watanabe 1996] T.Takashina and S.Watanabe, “The Locality of Information Gathering in Multiagent Systems.” *Proceedings, Second International Conference on Multi-Agent Systems (ICMAS-96)*, 461.

[Theraulaz et al. 1991] G.Theraulaz, S.Goss, J.Gervet and J.L. Deneubourg, "Task Differentiation in Polistes Wasp Colonies: A Model for Self-Organizing Groups of Robots." In [Meyer & Wilson 1991], 346-355.

[vanGelder & Port 1995] “It’s About Time: An Overview of the Dynamical Approach to Cognition.” [Port & vanGelder 1995] 1-44.

