Submitted to Advances in Intelligent Data Analysis Symposium (AIDA 2001)
at the Computational Intelligence Modeling and Applications Conference (CIMA 2001)
of the International Computer Science Conventions in Bangor, Wales, UK from June 19-22, 2001

A Learning Architecture for Intelligent Behavior in a Robot

Joseph A. Lewis, George F. Luger
{jalewis, luger }@cs.unm.edu
Department of Computer Science; FEC 323 University of New Mexico
Albuquerque, NM 87131

Abstract

We present a computational architecture for modeling intelligent behavior in a specific problem domain. Ours is a learning architecture that embraces a dynamical, embodied notion of intelligence relying on self-organizing emergent processes. We discuss *autopoeitic systems*, those systems whose components are engaged in an ongoing process of mutual self-maintenance. We believe autopoeisis to be an essential characteristic of intelligent systems. *Structural coupling*, or the capacity of the internal dynamics of an autopoeitic system to adapt to its changing environment, enables the architecture to utilize emergent representations of its environment. This representation makes possible further adaptive behavior. We describe the architecture in detail along with methods of development and training for a specific domain. We use our robot control architecture, Madcat (Lewis & Luger 2000), as an instantiated example of this architecture.

Introduction

We present a learning architecture for intelligent behavior in a robot that synthesizes several complementary approaches from the last decade. The architecture originates from a family of projects at Hofstadter's *Fluid Analogies Research Group* at Indiana University (1995). Specifically, it extends Melanie Mitchell's work on Copycat (1993). We have further refined that architecture using suggestions from McGonigle (1998), in the Madcat project (Lewis & Luger 2000). Madcat is an embodied instantiation of the architecture in a robot, which supports interactive learning about its environment through exploration.

In this paper we first describe refinements to the Copycat architecture motivated by recent ideas about embodied cognition. Next we discuss the role of emergent computation and why it is an important tool for adaptive representation. Then a brief look at how dynamical systems exhibit emergent representation will set the stage for a discussion of the components of the architecture. Concluding the theoretical background, we discuss Maturana and Varela's notion of *autopoeisis*—the supporting metaphor for our architecture. We then describe the components that implement this architecture.

From our experience with the Madcat project, we describe techniques for developing the architecture to address a specific problem domain. We also describe a method of training the architecture once it has been constructed. We conclude by identifying some additional problems for which this architecture might be appropriate.

It is useful to ground our discussion in some description of intelligent behavior. We link intelligence to the notion of *evolving complex adaptive systems (ECAS)* proposed by Steels (1996). Intelligence involves adaptation. Adaptive systems that we consider intelligent typically are complex, in the sense that we cannot easily predict how they will change internally as they respond to stimuli. Finally, intelligent systems evolve their capacities for complex adaptation over time.

Steels (1996) proposes four salient characteristics of ECAS. The first of these characteristics is *self-maintenance* (we prefer the term *autopoeisis* from Maturana and Varela (1980) who also describe a mutual maintenance relationship among system components). Steels' remaining characteristics for describing intelligence are *adaptivity*, *information preservation*, and, in response to the demands of a complex environment, a *spontaneous increase in complexity*. We also follow Steels (1996) suggestion that systems can meet these four criteria through a general purpose dynamical architecture and through capture of emergent properties, enabling the formation of concepts about and representations of the environment. We feel that the emergence of structures evolved through ongoing coupling with an environment is a defining feature of intelligence. We call this *behaviorally coupled representation*.

Embodied Cognition

Early artificial intelligence (AI) programs relied heavily on the physical symbol system hypothesis of Newell and Simon (1976). Those programs affected their world and created their understanding of it through the manipulation of symbol structures. Winograd's SHRDLU project is a prime example (1972). When limitations were met in these programs' behavior, one response was that these systems merely lacked a sufficiently large database of knowledge about their world. Even after the completion of very large "common-sense" knowledge databases, such as Lenat's CYC project (1990), there still has not been a consistent increase in the intelligence of programs relying on knowledge alone. Brooks (1991) responded to this problem with the subsumption architecture in which a few layers of well-defined feedback-controlled subsystems interact with the environment mostly independently of each other and without any kind of state or information representation. Some ideas from the subsumption architecture hailed the coming insights from embodied cognition. The move away from rigid symbolic representations has been supported by research such as (Crutchfield & Mitchell 1998) on emergent computation in cellular automata and (Hofmeyr & Forrest 2000) on distributed computation in an immunological model. We agree that wholesale rejection of representation

as a process within cognition misses something essential. Our architecture demonstrates the utility of an embodied emergent representation that is more fluid and adaptive than traditional symbolic representation.

Embodied cognition proponents argue that the evolutionary development of intelligent behavior in natural systems arises from their embodiment as physical organisms in constant interaction with their environment. Intelligent behavior develops to produce more effective coupling with the environment for greater adaptivity and survival. The nature of intelligence closely relates to the body and how it couples with an environment. What bodies "know", for instance about how to catch a ball or walk up a hill, plays a greater role in producing intelligent behavior than was previously appreciated.

There is a particular advantage to this tight coupling between the environment and intelligence, which Clark (1997) calls *scaffolding*. A system will adapt to utilize regular characteristics of its environment which have been frequently associated with some behavior to aid in the very production of that behavior. For example, "common sense" might suggest that in order to catch a ball one "computes" its trajectory to predict where it will fall within reach and then runs to that location. Clark points to research that indicates that instead one simply maintains a certain angle of gaze at the ball, which happens to entail running at a speed that puts one where the ball will land. A known relationship between the environment and the organism maintained through feedback can be a natural alternative to explicit use of cognitive resources. Deliberate scaffolding occurs when an organism relies on some state of the environment to trigger a particular response, as when we place a book on a table knowing that upon seeing it the next morning we will recall the need for it at the office.

The Madcat project extends Copycat both architecturally and by providing a means of environmental coupling. Madcat (a Nomad SuperScout II robot) explores its space, discovering the objects therein utilizing emergent structures that represent the surfaces in its environment. As it continues to explore, Madcat refines its representation into a map that augments its navigational capabilities. The environmental coupling comes from sixteen sonar sensors around the exterior, from which Madcat can take distance readings at variable intervals, typically several per second. Madcat also has information from a bumper sensor that encircles its exterior indicating the location of contact with objects and a color vision camera that uses very simple edge-detection for corroborating data and for location recognition. Madcat can also engage in interactive refinement of its emerging internal map, such as repeating the passage of an object from a different angle to corroborate its internal model. We feel that the ability to glean information from interactions with an environment is an important step towards realizing ECAS, and a distinction from knowledge-based AI programs. While the Madcat project is an example of physical embodiment in the world, the notion of embodiment can be applied to any domain, however abstract. Agents simply must be able to affect their embodying world and to rely on the results of those effects for adapting their behavior. Other feedback-based learning models can be found in Holland's (1986) classifier systems and reinforcement learning, e.g. (Sutton & Barto 1998).

Emergent Computation and Representation

One of the most difficult problems with traditional approaches to artificial intelligence is determining when and how to produce novel behavior. Such programs are notorious for getting stuck along the way to a solution. A related problem is *brittleness*, the inability to modify behavior just slightly from the prescribed behavior when the problem presents itself in a slightly different manner. Attempts to address these issues have led to the study of *emergent properties*: those which cannot be predicted by examining the state of the system and its rules prior to their appearance. When a system with the capacity for emergence is coupled to environmental demands, properties often emerge to meet those demands, leading to new adaptive features of the system.

We believe that a computational accounting of intelligence will likely utilize emergent properties. There are two parts of the Madcat architecture which make possible its emergent behavior. First, as in the original Copycat program, behavior of the architecture is achieved piecemeal by myriad tiny units of executable code, called *codelets* whose execution is managed by a component called the *coderack*. Each accomplishes only a step or two toward the realization of possibly several different goals in the program, but many are created as time passes and their collective execution produces the behavior of the system. There are many points of interaction among these codelets which provides opportunity for emergent behavior. The second feature that helps to avoid brittleness and provides an opening for emergence is randomness. Although the coderack tends to execute the most urgent codelets, it has a random component that allows it sometimes to choose from among the less urgent ones. This prevents it from being locked into a certain sequence of executions and provides the chance to discover other possibilities. Hofstadter (1995) devotes an entire chapter to the merits of randomness in the service of intelligence.

The code executed in this emergent fashion has many roles. One is to produce relevant behavior for the robot. Another is to discover relationships among the sonar readings at a given instant. As these relationships are discovered they (probabilistically) generate corresponding information structures in the program. The kinds of relationships that are being sought at any given instant is an emergent feature of the program: a combination of those it has already found, how the robot has been moving, whether the program has a clear model yet or not, etc. The component where these structures are built is called the *workspace*. Ultimately these structures give rise to the map which it is Madcat's goal to build.

As the navigational behavior of the robot emerges so do the internal data structures which serve as an aid in the robot's navigation. By design these structures are intricately layered, and woven throughout them is a context-dependent history of their construction. Moreover their evolution participates in controlling the evolution of the rest of the system as well. These are important differences between these emergent structures and their counterparts in programs that use traditional symbol manipulation. We say that the structures and the behavior *self-organize*.

Indeed something significant is taking place here that is common in dynamical systems. Structures often are the echo of some pattern in a process. Consider the so-called Bénard instability. A viscous oil is placed between two metal plates, one of which is heated. At first the heat passes through the oil by conduction, the molecules increasing their individual kinetic energies and conducting the heat through their collisions. As the temperature difference increases above a certain threshold, convection begins to occur. The fluid begins to self-organize into a regular grid of same-sized hexagonal convection cells. The walls and internal regions of these cells carry opposite convection currents. Structure emerges out of chaos echoing the patterns of the underlying interactions. This is the same deep connection between structure and process described above that is the heart of Madcat's functioning.

Another feature of dynamical systems that the Bénard instability demonstrates is that complex processes and the structures produced by their patterns can *spontaneously reorganize themselves in response to external stimuli*. This is reminiscent of the behavior of attractors in dynamical systems. In high phase-space a great diversity of data-points could be just over a threshold that triggers some different behavior. In our Madcat example, even the shift of focus of the codelets from searching for one kind of relationship to another could be triggered in similar fashion. This is the essence of a self-organizing system: its own behavior meshed with information from its environment gets fed back into the system, providing the opportunity to adapt its functioning toward particular effects.

In Madcat the *slipnet* provides the web of connections linking the behavior and the structures through time. The slipnet is a semantic network organized with spreading activation and multiple kinds of links among its nodes, some of which can change in length. The evolution of structures in the system impacts the topology of the slipnet, making the program's own behavior part of the context-dependent control. The slipnet actually models attractor-like behavior: as activity is added to its nodes from throughout the program they reach thresholds where they may rise to full activation. Groups of active nodes tend to correspond to certain behaviors and structures in the program. The success of codelets impacts the slipnet as does the building of structures, and the slipnet in turn controls which particular codelets and structures are engaged. In the next section we discuss autopoeisis, a formalization of this mutual self-maintenance, and the emergent features that it can produce. We believe Madcat is an instance of an autopoeitic system. Furthermore, identifying this as a cogent property of computational models of intelligence will provide a framework in which to build dynamical systems whose emergent features are adaptively embodied.

Autopoeisis

The term autopoeisis originates from research on two questions (Maturana and Varela 1980). The first was "what is the nature of perception," and the other "what is the organization of living systems". Autopoeisis, literally *self-making*, captures what Maturana and Varela believed was common to both. Each of the components of an autopoeitic system participates in the creation and maintenance of the others.

Living systems exhibit this property of mutual self-maintenance. Consider a cell membrane. Its layers of lipids and proteins regulate what molecules can enter and leave the cell, thus maintaining the chemistry of the cytoplasm; while it is that very chemistry that maintains the lipid-protein layers of the membrane. Cognitive systems (by Maturana's definition: systems engaged in perception) seem to be involved in a similar mutual feedback relationship among their components and the external features that affect their behavior.

Maturana and Varela further note that it is a hallmark of living systems that their external environments trigger changes in the internal structure of the system--changes that afford effective behavior--but that the nature of those internal changes are dictated by the dynamics internal to the system rather than being specified by the environment. They gave the name *structural coupling* to the changes triggered in the internal dynamics of a system by its need for effective behavior in an environment to which it is exposed. The information-preserving quality of the concept of structural coupling has been extended (Hendriks-Jansen 1998) through the study of evolution and ethology to provide a grounding of species' behavior, even meaning forms, in the history of the organism.

The idea of structural coupling is especially important in our consideration of the essentials of intelligent behavior. Structural coupling prescribes that a complex system will modify the nature of its internal dynamics in order to adapt its autopoeitic functioning to the changing external features of its environment. Furthermore, the nature of those changes will be established by the existing dynamics within the system, subject to its current state *and history*. This context-dependent coupling between internal states of the system and cogent external events comprises what we consider to be *representation*. If autopoeisis describes part of what generates intelligent behavior, then it offers the position that behaviorally coupled representation should indeed be a part of our models of intelligent behavior. We will have to develop models whose representations are emergent from an environmentally coupled dynamical system and which do not have to take

on a prescribed form (as is the case in strictly symbolic systems). These features—emergent embodied dynamical behavior with an emergent structural footprint for representation and feedback—are essential to the systems that exhibit autopoeisis, and we will hereafter take that term to include them. While there may be further requirements, we believe that a computational architecture for learning and intelligent behavior must at least be autopoeitic.

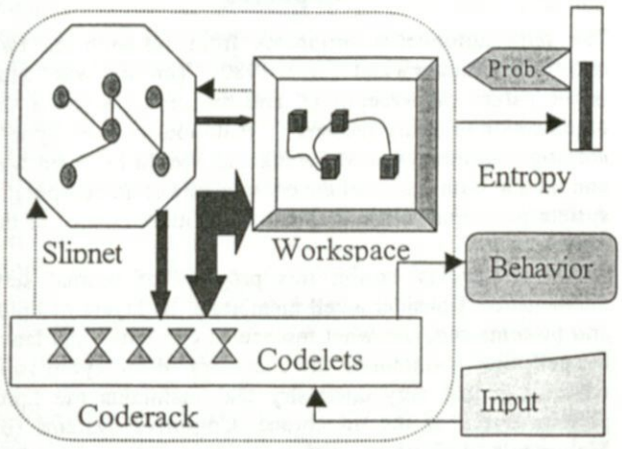

Figure 1: An Autopoeitic Architecture

A General Autopoeitic Architecture

As seen in Figure 1 (and discussed in the preceding sections) the three primary components of this architecture are the coderack, the workspace, and the slipnet. The arrows indicate the direction and influence of information among the components. As in Copycat, the entropy is a measure of coherence and consistency among the structures in the workspace. It provides feedback which influences the probabilistic decisions constantly made throughout architecture. The coderack is a stochastic priority queue that selects the next micro-action (codelet) for execution. The details of the other components have been extended and refined from their ancestors in Copycat. The workspace is the shared arena for the structures that are built as a side effect of the processes engaged by the interacting codelets. Finally, there is the context-capturing, dynamically deformable semantic network that provides control feedback called the slipnet.

The coderack is the simplest component, being essentially a stochastic priority queue. Of the three main components, its implementation is closest to its ancestor in Copycat . Each codelet is a subclass instance of an abstract *Codelet class* which carries an *urgency* value as well as private data for managing the codelet within the coderack. The urgency values of a codelet are assigned by the component that creates it. They are determined as a function of the relevance of the codelet to the current set of most important processes as captured in the activity of sets of nodes in the slipnet.

There are seven urgency values ranging from extremely low to extremely high and *bins* corresponding to each within the coderack. When a component wishes to create a new codelet it asks the coderack to *post* codelets of that type. The likelihood of this posting and the number posted is a variable value regulated by the entropy and context information from the slipnet. It is also one of the many parameters that can be *tuned* in the system. When the coderack chooses the next codelet for execution, it selects a bin randomly with a bias toward the higher urgencies that is mediated again by the entropy.

Each subclass of the Codelet class has one public method that is the executable code for that codelet. Which codelet subclasses to develop is a question specific to the structure-building or behavior-producing actions appropriate for the problem domain. One difference from its predecessor in Copycat is that codelets not only build structures in response to patterns in the data but also may take action which affect that data or the system's ability to perceive it. While there is no direct correlate of the notion of a process in the architecture, each codelet represents a small step toward the realization of one or more *pressures* within the system. Codelets are very similar to classifiers in Holland's classifier systems (1986).

The workspace is the locus of shared activity in the system, very similar to the shared message space of blackboard systems (as well as classifier systems). The workspace is essentially a "smart" substrate for the various types of structures the codelets choose to build. A three-dimensional array indexed by instance, type, and time, the workspace is able to provide linear-time access to randomly selected structures in general, or specified by type or time index or both. Furthermore, the random selection can be biased by the *salience* value of the structures, a bias which in turn is influenced by the context information in the slipnet. The amount of this bias is another one of the tunable parameters of the system.

Each structure is an instance of an abstract *WorkspaceObject class* which carry the indices and other information required to manage the structure within the search and storage mechanisms of the workspace. For example, each structure in the workspace carries two values: *importance* and *happiness*. The importance of a structure derives from the current importance of the concepts related to that structure as measured by the context information in the slipnet. The happiness of a structure reflects how well that structure has been connected together with other structures of appropriate type. These two values determine the salience at any given instant of a WorkspaceObject. How these values are used to compute salience is yet another tunable parameter of the architecture. Each subclass carries information specific to the structure it represents. The question of which WorkspaceObject subclasses to develop is specific to the structures that accompany and reflect the activity of the system as it engages a particular problem domain.

One significant difference between Copycat and Madcat is that there is one Copycat workspace for all the structures built from the initial exposure of the system to a problem. Our system expects to be continually exposed to an

evolving state of the world over time, so the workspace is actually composed of an array of temporally indexed *snapshots*, each of which is analogous to the Copycat workspace. One kind of structure Madcat can build in any snapshot of the workspace is a *temporal bond*, identifying relationships between structures in different snapshots. Thus, our architecture is capable of interacting with evolving patterns in the environment, which deepens the degree of embodiment. This architectural change supported the new problem domain: building spatial maps from a moving robot. The patterns among the temporal relations between individual sonar readings are used to establish models of the objects existing in the environment. The utility of this evolution of the architecture is clear for any problem that involves continued exposure to a changing environment.

The slipnet is a network of nodes that alone or in groups represent relevant concepts for the problem domain. It is fairly straightforward to determine the context-dependent micro-actions or codelets the system should have. It is also not very difficult to determine the relevant structures or workspace objects to build. However, it is a significant challenge to decide what are the relevant concepts--nodes or attractor-like conglomerations of nodes--in the slipnet for a particular domain. This is especially true since it is important to avoid introducing too much inductive bias which might undermine the adaptability of the program.

The slipnet consists of nodes and links. The nodes, which represent programmer-chosen concepts of significance for the problem, are connected together by links that are sometimes *labeled* by other nodes in the network. For example, the nodes *front* and *back* could be connected by a link labeled by the node *opposite*. Nodes have activation levels that are incremented whenever that concept is currently successful and which are decremented over time by decay. The activation of a node will spread across links to neighboring nodes whenever it exceeds a certain threshold (a tunable parameter). The higher the activation level of a node, the shorter the links for which that node is a label. So when opposite has a lot of activity, front and back are more closely linked.

The entropy is an overall measure of coherence and consistency among the developing structures in the workspace. Structure-specific measures like *importance* and *happiness* are used in conjunction with context information from the slipnet to produce this measure. As its analogue from physics would suggest, the entropy is high when disorder dominates the workspace and low when the structures therein couple tightly with one another and fit into consistent groups.

Almost all of the tunable parameters mentioned so far, as well as many others, have their effects mediated by the value of the entropy. For instance, when the entropy is very high the salience of an object has less impact on its likelihood of being selected randomly than when the entropy is low. This means that, before many coherent structures form, most items in the workspace are equally attractive to explore. Once more structures form, then those which have not yet been incorporated into the emerging structures are more likely to be selected.

Similarly, when the entropy is high the selection of the next codelet to execute is fairly random; whereas when the entropy falls the higher urgency codelets are selected with greater probability. Essentially this allows the system to shift from parallel, random search toward serial, deterministic search as it hones in on a useful solution. In the temporally sequenced problems that this architecture handles, the falling of the entropy can trigger the movement to the next snapshot. Entropy provides a coarse form of self-organizing feedback.

Development and Training

The first step in using this architecture for adaptive problem solving is to determine what will be its domain of interaction—what measurements provide its data. In the original Copycat program this was simply a sequence of letter-strings that formed an analogy problem. In the TableTop program (Hofstadter 1995) this was a two-dimensional arrangement of certain classes of objects. In Madcat it is the time-indexed set of sonar measurements taken from the perceptual systems of a moving robot.

The second step is to determine how the program will interact with the environment from which that data comes. Copycat had a single exposure to its source problem with no means of interacting with that environment. Madcat is embodied in its environment by its ability to exhibit movement commands for the robot from its emergent behavior.

The next step in developing this architecture for a specific domain is to determine the kinds of structures the program should build in the workspace. These are motivated by what the user wants the program to produce as in Copycat's analogy solution or Madcat's interactively refined world-map. These structures are akin to the structures placed in global space in a blackboard system, little information packets scrupulously advertised for the next layer of structure-building to select. They are often discovered recursively during the development of the architecture for a particular domain. They are typically proxy objects for measured data or bond structures for relationships between data or other structures. The interpretation of structures will typically depend on the program's run-time contexts. The structures actually in use during execution will evolve as the program explores the problem domain.

Once some basic set of structures is identified, the codelets that produce them are developed. In Copycat it typically takes two to three sequenced codelet executions to get a structure built. First a codelet is posted to determine roughly the merit of building some structure. Then if that is high enough a codelet is posted to determine how useful or important such a structure would actually be. Then if that is high enough a codelet is posted actually to build the structure. All this is mediated through context from the entropy and slipnet. In Madcat, because of real-

time constraints, most structure-building is decided and accomplished by a single codelet.

Codelets also obtain information from other parts of the architecture, and the choice of whether to stage this process must also be made. For instance, most codelets look for particular relations among particular types of objects. They select a structure or structures of the appropriate type using the type-specific biased random selection interface of the workspace. Then they use context information from the slipnet to decide whether some new structure built on these is appropriate.

Since the architecture is stochastic, specific sequencing cannot be guaranteed; thus, it is important to provide competition among structures. If a new one potentially conflicts with existing ones a competition allows a context-dependent decision about which structure to keep. Choosing the types of codelets establishes the possibilities for interaction and emergence of behavior. Some codelets will be responsible for taking measurements. Others identify relationships and patterns. Still others will produce behavior.

Finally, the slipnet is developed. This component represents the inductive bias of the system and it is wise to develop concepts that are relevant and useful for describing the structure of the domain without being too rigid or precise. Usually there will be a set of related nodes specifying possible relationships among the measurements the program takes of its environment. There may also be *meta*-concepts represented about how those measurements can be related at a higher level and so on. Some domains have a good deal of inherent structure, as with the transitive *successor* and *predecessor* relations among the letters of Copycat's letter-string domain, including even meta-relations like *opposite* between successor and predecessor. Developing a set of concepts and relations among them for describing such a structured domain starts with much information and is gradually refined. On the other hand, some domains may be less structured at first, as with Madcat's ring of sonar sensory data. In Madcat a simpler slipnet is used with only a few nodes and links to build simple similarity bonds among readings and groups of readings. This happens every snapshot. A more complex slipnet is used that carries information across snapshots and guides the construction of temporal pattern structures that are used to map objects.

The nodes in the slipnet are responsible for posting codelets according to the concepts with highest activation. The amount and likelihood of those codelets is another tunable parameter. It must also be determined when and how much to spread activation among nodes, how much to decay activation over time, how the links should change as activation changes, and so on. All these tunable parameters that impact the behavior of the system can be honed for particular behavior with some patience. For example, in Madcat, about two months went into the initial tuning of these parameters leading to the behavior seen in Figure 2.

Once constructed, training the architecture consists of an iterative process of varying the tunable parameters and determining the effect on the behavior in some suitably determined test suite. The space over which the behavior varies has many local peaks, so the problem domain should be widely represented in the test suite. A technique we are currently using is to externalize the values for the tunable parameters into a *Genotype class* and to use a genetic algorithm to evolve an optimized tuning of the values in that class. The usefulness of this approach depends largely on the ease of constructing a fitness function for the application to which the architecture is being applied.

The Madcat Architecture

We use Madcat throughout our discussion of this general autopoeitic architecture as a concrete example of how to apply the architecture to a specific problem domain. The Madcat project specifically develops this autopoeitic architecture as the control system for a Nomad Super Scout II robot whose task is to move about its space and construct an emergent, interactively refined map of its environment. The sequences of data from its 16 peripheral sonar sensors and color vision camera, taken at self-determined intervals, are synthesized to produce a map that enhances the behavior of the robot. For example, the development of internal structures reflecting objects in the environment allows the robot to avoid obstacles that it otherwise might not see (Lewis & Luger 2000). Figure 2 shows an example of the internal structures built by the architecture reflecting objects in its environment.

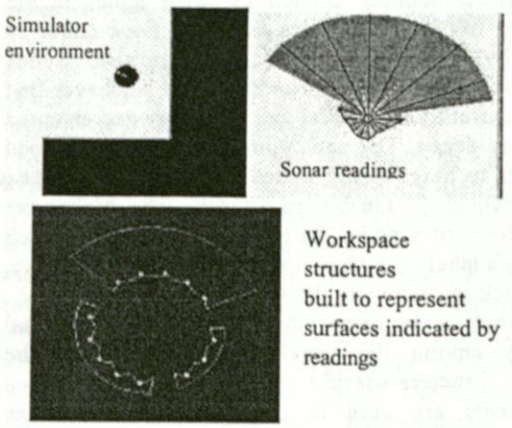

Figure 2 : Structures built for an object

The arcs around the sonar sensors in the bottom of the figure correspond to candidate surfaces that span those sensors. These form the emergent representation of the approaching corner, which not only provide a piece of its world-map but also augment the adaptive behavior of the robot. For instance, non-reflex navigational choices are made using this representation (e.g. exploration of an opening or repeat measurements to refine the map). Also certain surfaces may become undetectable during the approach but the representation serves as a model of what is there to guide the blinded low-level navigation skills.

Conclusion

We have developed the Madcat robot control architecture, which supports a robot learning about its environment through exploration. It was motivated theoretically by the recognition that its predecessor, Copycat, is an instance of an autopoeitic system as well as by ideas from the study of embodied cognition and emergent behavior in dynamical systems. Based on these insights, we have refined the architecture to incorporate interaction with the environment and perceptions of temporal patterns during that interaction. Using context-driven feedback provided by the slipnet, the program produces an interactively refined map of the robot's space. We believe this represents a novel contribution to the existing architectures for modeling intelligent behavior. We will continue to investigate this interactive verification and refinement between model and world as a basis for machine learning.

There are other problems to which we would like to apply this architecture in the hopes of gaining further insights into the connection between autopoeisis and intelligent behavior. Better synthesis between different sensory modalities, such as between sonar and color vision information would be of particular interest in the robotics domain. Also, we intend to explore how Madcat refines its emergent representation when it changes its environment, for example by moving objects in its environment. The robotics arena is an excellent domain for further research on issues in emergent and embodied cognition.

In a non-robotics domain, we believe that our architecture might also be adapted to form, from continued exposure to behavior-sentence pairings, a kind of "grounded" meaning representation for certain embodied linguistic forms. Some early work toward this end was done in the beginning stages of the Madcat project (Rogati, Lewis & Luger 2000).

Acknowledgments

This research has been supported at the University of New Mexico by the NSF CISE Research Infrastructural award CDA-9503064 and NSF 9800929, and an international extension to work at the University of Edinburgh (NSF 9900485). The contributions of Brendan McGonigle, Matthew Fricke and Monica Rogati have been invaluable.

References

Brooks, R. (1991) Intelligence Without Representation. Reprinted in Luger, G. (ed.). 1995. *Computation & Intelligence*. 343-364. Cambridge: MIT Press.

Clark, A. (1997). *Being There*. Cambridge: Bradford Books/MIT Press.

Hendriks-Jansen, H. (1996). *Catching Ourselves in the Act*. Cambridge: MIT Press.

Hofmeyr, S. and Forrest, S. (2000) Architecture for an Artificial Immune System. In *Evolutionary Computation Journal*. (in press)

Hofstadter, D. et. al. (1995). *Fluid Concepts and Creative Analogies*. NY:Basic Books.

Holland, J. (1986). Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. Reprinted in Luger, G. (ed.). 1995. *Computation & Intelligence*. 275-304. Cambridge: MIT Press.

Hordijk, W., Crutchfield, J. and Mitchell, M. (1998) Mechanisms of Emergent Computation in Cellular Automata. In *Parallel Problem Solving from Nature, Fifth International Conference*. Berlin: Springer.

Lenat, D. and Guha, R. (1990). *Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project*. Addison-Wesley.

Lewis, J and Luger, G. (2000). A Constructivist Model of Robot Perception and Performance. In *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society*. Philadelphia: Mahwah, NJ: Lawrence Erlbaum Associates, Publishers.

Maturana, H. and Varela, F. (1980). *Autopoeisis and Cognition*. Dordrecht, Holland:D. Reidel.

McGonigle, B. (1998). Autonomy in the making: Getting robots to control themselves. In *International Symposium on Autonomous Agents*. Lanzarote: Oxford University Press.

Mitchell, M. (1993). *Analogy-Making as Perception*. Cambridge: Bradford Books/MIT Press.

Newell, A. and Simon, J. (1976). Computer Science as Empirical Inquiry. Reprinted in *Computation & Intelligence*, ed. Luger. G. (1995) Cambridge, MIT Press.

Rogati, M., Lewis, J. and Luger, G. (2000). A Deformable Semantic Network as a Tool for Context-Based Ambiguity Resolution. Internal tech report, UNM CS department. (NASA PURSUE Student Conference Paper).

Steels, L. (1996). The origins of intelligence. In *Proceedings of the Carlo Erba Foundation Meeting on Artificial Life*. Berlin: Springer-Verlag.

Sutton, R. and Barto, A. (1998). Reinforcement Learning. Cambridge: MIT Press.

Winograd, T. (1972) *Understanding Natural Language*. NY: Academic Press.