Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

SIMULATING HYBRID CONNECTIONIST ARCHITECTURES

Trent E. Lange, Jack B. Hodges, Maria E. Fuenmayor, Leonid V. Belyaev

Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90024

ABSTRACT

The symbolic and subsymbolic paradigms each offer advantages and disadvantages in constructing models for understanding the processes of cognition. A number of research programs at UCLA utilize connectionist modeling strategies, ranging from distributed and localist spreading-activation networks to semantic networks with symbolic marker passing. As a way of combining and optimizing the advantages offered by different paradigms, we have started to explore hybrid networks, i.e. multiple processing mechanisms operating on a single network, or multiple networks operating in parallel under different paradigms. Unfortunately, existing tools do not allow the simulation of these types of hybrid connectionist architectures. To address this problem, we have developed a tool which enables us to create and operate these types of networks in a flexible and general way. We present and describe the architecture and use of DESCARTES, a simulation environment developed to accomplish this type of integration.

1. INTRODUCTION AND MOTIVATION

Connectionist networks have recently been the subject of a tremendous rebirth of interest, as researchers have begun to explore their advantages for cognitive models ranging from low-level sensory abilities to high-level reasoning. Connectionist models employ *massively parallel* networks of relatively simple processing elements. These models draw their inspiration from neurons and neurobiology, as opposed to existing symbolic artificial intelligence (AI) models, which are generally based on serial Von Neumann architectures.

A few designers have implemented connectionist networks directly in hardware to achieve the networks' true parallelism (e.g. [Akers & Walker, 1988] and [Lazzaro *et al*, 1988]). Hardware implementations of specific networks, however, are far too inflexible and expensive for the exploration and testing of different types of models. Because of this, there is a great need for software tools to allow researchers to design and simulate the many kinds of connectionist network models.

1.1. Symbolic and Subsymbolic Connectionist Models

While there are several existing connectionist simulators, none allows the simulation of *hybrid* networks that integrate elements from more than one paradigm of connectionist modelling. This paper describes the simulation of hybrid connectionist networks composed of heterogeneous elements in DESCARTES (Development Environment for Simulating Connectionist ARchiTEctureS) [Lange *et al*, 1989a].

Connectionist networks are made up of a large number of relatively simple computing elements, called *nodes*, which are connected together by *links* , [Feldman & Ballard, 1982]. Within the connectionist approach there are three paradigms, each having its own advantages and disadvantages: *Distributed Connectionist Networks* (DCNs), *Localist Connectionist Networks* (LCNs), and *Marker-Passing Networks* (MPNs).

DCNs (such as the models described by [Rumelhart & McClelland, 1986]) use simple, neuron-like processing elements which represent knowledge as distributed patterns of activation across those elements. Each node in a DCN has a numeric activation level, computed by an *activation function* applied against the node's weighted input links. This activation is communicated to the rest of the network by spreading through its output links after being passed through the node's *output function*.

DCNs, sometimes known as *Parallel Distributed Processing* or *Subsymbolic* models, are interesting because they have learning rules that allow statistical category generalization, they perform noise-resistant associative retrieval, and they exhibit robustness to damage. Distributed models, however, have (so far) been sequential at the knowledge level, lacking both the representation of structure needed to handle complex conceptual relationships and the ability to handle dynamic variable bindings and to compute rules.

LCNs (as exemplified by the models of [Waltz & Pollack, 1985] and [Shastri, 1988]) also use simple, neuron-like processing elements with numeric activation and output functions, but represent knowledge using semantic networks of conceptual nodes and their interconnections. Unlike DCNs, localist networks are parallel at the knowledge level and have structural relationships between concepts built into the connectivity of the network. Unfortunately, they lack the powerful learning and generalization capabilities of DCNs. They also have had difficulty with dynamic variable bindings and most other capabilities of symbolic models.

MPNs (as exemplified by the models of [Charniak, 1986] and [Hendler, 1988]) also represent knowledge in semantic networks and retain parallelism at the knowledge level. Instead of spreading numeric activation values, MPNs

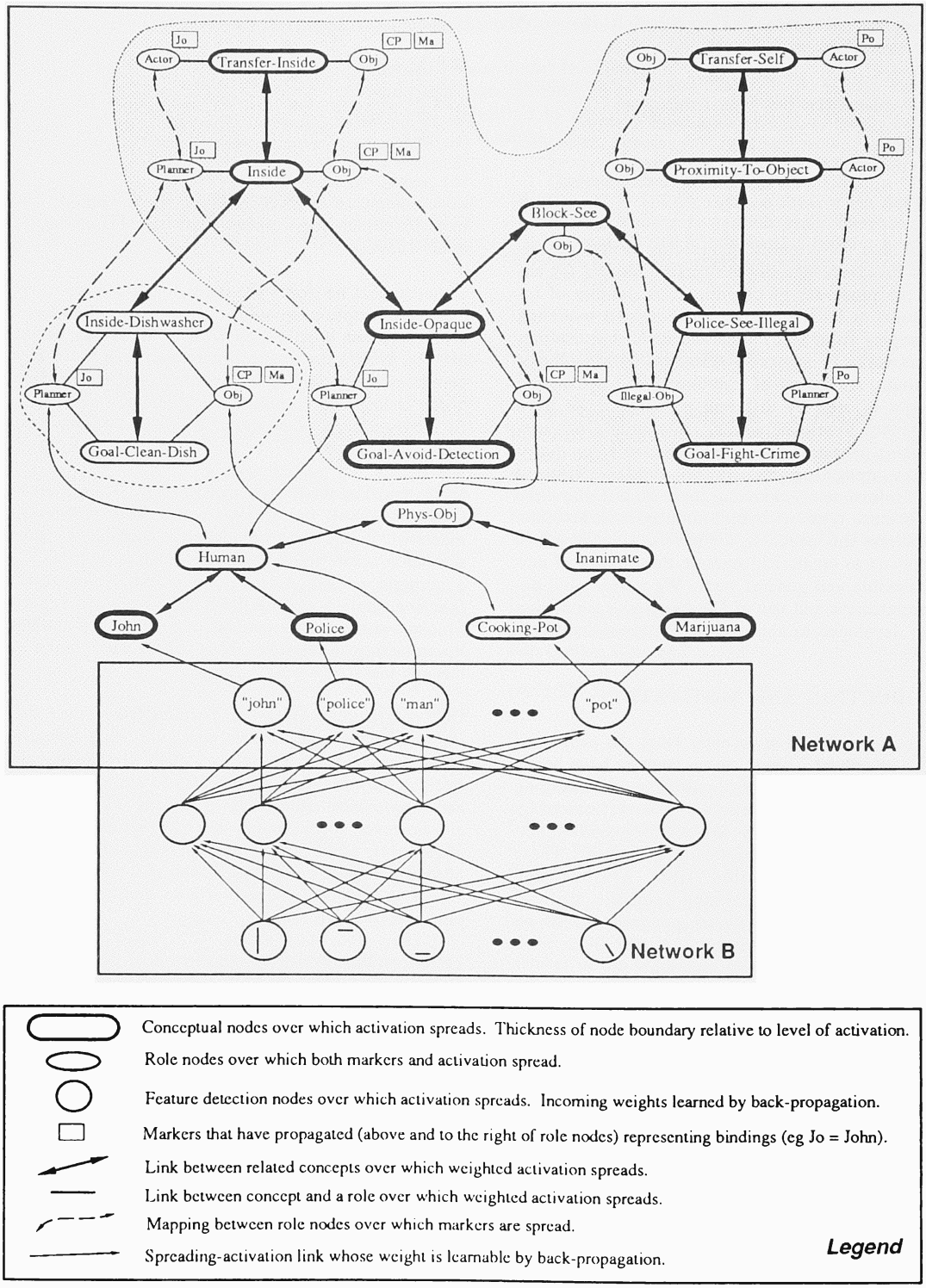

Figure 1: The sentence *"John put the pot in the dishwasher because the police were coming."* illustrates the utility of integrating semantic networks (Network-A) and distributed networks (Network-B). The darkest area represents the most highly-activated set of nodes representing the network's plan/goal analysis of the sentence. Not all markers are shown. Location role nodes and other parts of the network are also not displayed.

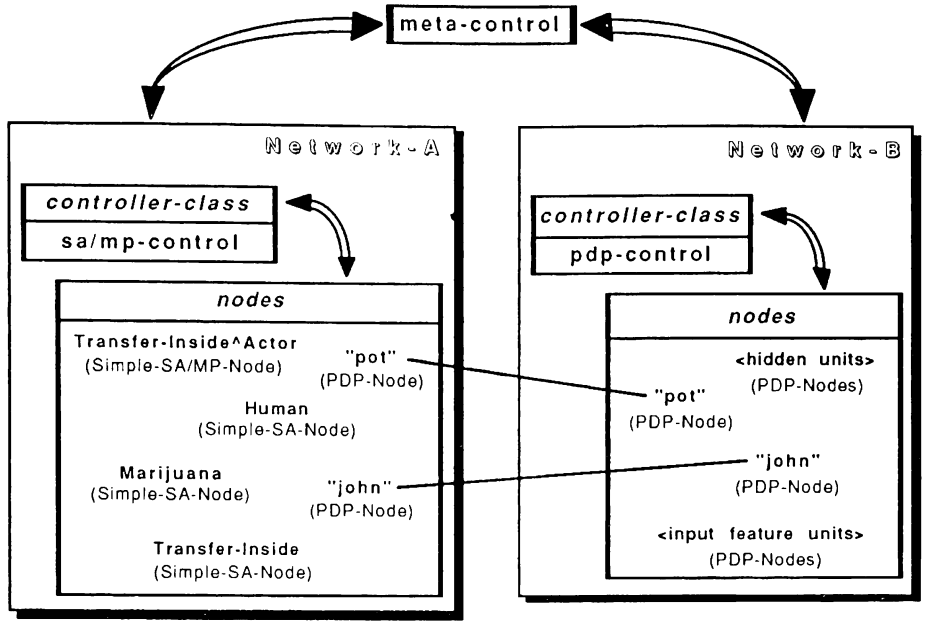

Figure 2: DESCARTES Processing Architecture applied to **Hiding Pot**. Shown in each network are a few of their nodes, with the class of each node being declared in parentheses below their names. PDP-Nodes "pot" and "john" are shared by both networks.

propagate symbolic markers, and so support the variable binding necessary for rule application, while preserving the full power of symbolic systems. On the other hand, they do not possess the learning capabilities of DCNs or exhibit the constraint-satisfaction capabilities of LCNs.

1.2. Hybrid Connectionist Models

Research at UCLA has spanned the range from subsymbolic to symbolic connectionist models [Dyer, 1989]. A number of us have begun to construct hybrid architectures which use what we term Multiple Interacting Networks, or MINs; heterogeneous connectionist networks that communicate via shared elements. A neurophysiological approach [Nenov & Dyer, 1988] effectively uses MINs for visual/verbal association by modeling heterogeneous neuronal characteristics in separate networks. We have also been exploring the use of MINs for higher cognitive tasks, such as planning, creativity, story invention, and political negotiations. In political negotiations research, for instance, MINs are used to simulate the multiple perspectives of negotiating parties.

Another approach is to build models that combine the bottom-up processing features of DCNs with the top-down processing features of LCNs and MPNs. Figure 1 shows **Hiding Pot**, an example wherein elements from each paradigm are combined using MINs. This allows us to approach a problem that would be difficult, if not impossible, using a single paradigm. **Hiding Pot** shows a simplified network built to understand the sentence, *"John put the pot inside the dishwasher because the police were coming."* Network-A in Figure 1 utilizes an MPN to do role-binding and an LCN to activate and combine evidence for individual schemas. These then combine their functionality to support predictions and perform inferencing and disambiguation.

One might also want to combine different connectionist approaches by having separate networks that communicate with each other, where each one performs a different cognitive task. Network-B in Figure 1 is a DCN, trained to recognize words from line segments [McClelland & Rumelhart, 1986, chap. 1]. By integrating these two approaches, we can simulate cognitive processes at the different levels of abstraction necessary for modeling reading and understanding.

Network-A interacts with Network-B through shared lexical nodes. Once a word has been recognized, it passes activation to the concepts related to the word. For example, the node for concept John gets activation from the word node "John" which is shared by both networks. Activation then propagates along the chain of related concepts in the network as contextual evidence for disambiguation. Markers are passed over the role nodes across marker passing links between corresponding roles to represent role-bindings and perform the needed inferencing.

While there are several existing connectionist simulators, none allows the simulation of multiple interacting hybrid networks, as in **Hiding Pot**, that integrate elements from more than one paradigm of connectionist modelling. We have developed the DESCARTES simulation environment specifically to address this kind of integration. DESCARTES enables researchers to design, simulate, and debug hybrid

```
(Simple-SA-Node Transfer-Inside :in-links (SA-Link ("put"                  0.75)
                                                 (Inside                 1.00)
                                                 (Transfer-Inside^Actor  0.50)
                                                 (Transfer-Inside^Obj    0.50)
                                                 (Transfer-Inside^Loc    0.50)))

(Simple-SA/MP-Node Transfer-Inside^Actor :in-links (SA-Link (Transfer-Inside 1.0))
                                                 (MP-Link Inside^Planner))
```

Figure 3: Creation of Transfer-Inside and Transfer-Inside^Actor nodes, with forward-referencing.

connectionist architectures that combine elements of distributed, localist, and marker-passing networks.

2. DESCARTES ARCHITECTURE

DESCARTES is a package designed for simulating network processing, network interaction, and integration of networks into an overall processing environment. The system consists of two interactive components: network *elements*, such as nodes and links, their associations, and their functionality, and *processing controllers*, which organize network elements and coordinate their processing. The components of this architecture, as applied to **Hiding Pot**, are shown in Figure 2.

2.1. Processing Controllers

When DESCARTES is loaded and running, the required processing controllers are a *meta-controller* (a supervisor for all elements and sub-controllers present in the run-time system) and at least one *network controller* (a supervisor for an individual network and its elements). The architecture described in Figure 2, and implemented in **Hiding Pot**, is controlled by a meta-controller (Meta-Control) which coordinates the two networks (Network-A and Network-B). Each of these networks has a local network controller which coordinates the processing of its elements. In this case the controller for Network-A is of class SA/MP-Control, which combines both spreading-activation and marker-passing functionality.

2.2. Network Elements

DESCARTES has a number of predefined node classes with different functionalities. Three of these node classes are used in the **Hiding Pot** example: (1) Simple-SA-Node, used in **Hiding Pot** for conceptual elements, such as Human and Transfer-Inside, (2) Simple-SA/MP-Node, used for roles, such as Transfer-Inside^Actor, and (3) PDP-Node, used for feature detection in Network-B, such as the node representing the lexical entry "pot". Figure 3 provides an example of node creation for the Simple-SA-Node and Simple-Sa/MP-Node classes.

Simple-SA-Node is a basic class of spreading-activation nodes with default activation and output functions. Simple-SA/MP-Node is another standard node class, which combines the functionality of Simple-SA-Node with that required for marker passing. Finally, PDP-Node is the simplest class of DCN-type nodes — spreading-activation nodes that modify the weights on their input links by backpropagation [Rumelhart *et al.*, 1986, chap. 8].

Many other common node and link types are predefined, with a variety of activation, threshold, and output functions. More complicated classes are also available, including gated nodes and links, along with more neurally-realistic nodes that communicate via output spikes, such as the artificial neural oscillators of [Vidal & Haggerty, 1987]. The functionality of DESCARTES objects can easily be extended by combining the default class definitions of the object hierarchy with user-defined modifications, a process described in [Lange *et al.*, 1989b].

2.3. Structured Networks

Some connectionist models have a consistent structure between groups of nodes in the network. In a semantic network, for example, a node representing the head of a frame might always be connected via a certain type of link to each of its roles, which in turn might always have a node for their fillers. Groups of nodes forming winner-take-all networks are always completely interconnected with constant inhibitory weights. Rather than force the user to repetitively define all nodes and connections for each such structured group, DESCARTES has a facility that allows the programmer to optionally define a structured growing method for each node class. A node's growth method automatically creates the node's expected structured incoming and outgoing nodes and connections. This feature allows knowledge base definitions to act as keys for network creation rather than as exhaustive listings of the networks' nodes and their connectivity.

2.4. Simulation in DESCARTES

Once the networks have been designed and built, the user starts the simulation by (1) optionally defining the cycling, termination, and display sequence for each network, (2) initializing the meta-controller to clear out all activation and markers, (3) activating or marking the desired nodes, and (4) starting the cycling sequence and specifying the number of global cycles to run. An example of this process is shown in Figure 4, but for a complete description see [Lange *et al.*, 1989b].

Figure 4 shows the initial activation and markers needed to process the phrase *"John put the pot inside the dishwasher because the police were coming."* The first define-cycling command in the figure specifies that the meta-controller spread activation in Network-A once per global

cycle, while only passing markers once per every three global cycles. Both activation and markers will cycle until stability, their default termination condition. For analysis of the network's activity, the user has defined that a trace of the markers' propagation be shown and that the status of the nodes be displayed every ten cycles. The second define-cycling command defines that Network-B is not to be cycled in this example.

```
(define-cycling %Network-A :sa-cycle-every     1    ;1
                           :marker-cycle-every
                           :marker-trace       T
                           :display-every     10)
(define-cycling %Network-B :sa-cycle-every    NIL)
(init meta-control)                                 ;2

(clamp-output %w-put          1.0)                  ;3
(clamp-output %w-john         1.0)
(clamp-output %w-pot          1.0)
(clamp-output %w-dishwasher   1.0)
(clamp-output %w-police       1.0)
(clamp-output %w-were-coming  1.0)
(mark %transfer-inside^actor %marker_John)
(mark %transfer-inside^obj   %marker_Marijuana)
(mark %transfer-inside^obj   %marker_Cooking-Pot)
(mark %transfer-inside^loc   %marker_Dishwasher)
(mark %transfer-self^actor   %marker_Police)

(cycle 50)                                          ;4
```

Figure 4: Example of DESCARTES' control language.

In general, the networks' cycling sequences need only be set once per session (if at all), although all sequencing and displaying parameters may be re-specified in mid-simulation. Activations and markers of nodes may be changed at any time.

3. IMPLEMENTATION

DESCARTES is implemented in COMMONLISP, the ANSI Lisp standard, and the COMMONLISP Object System, CLOS, which provides hierarchical inheritance for DESCARTES classes and ensures flexibility by allowing the user to utilize pre-defined functional classes to customize network semantics.

DESCARTES's control language is simple and effective, enabling the designer to easily set up and test different network configurations using either pre-defined or user-defined elements. At the same time, the system has been designed with ease of network debugging in mind, with history and output facilities that offer researchers valuable methods for interpreting network behavior.

As previously stated, all nodes, links, controllers and markers in DESCARTES are objects; specifically, instances of DESCARTES classes. Presented below is a detailed description of a single node and a single link in Hiding Pot; a detailed description of the simulation cycle; and an example of a network with four nodes and six links, which demonstrates basic concepts of DESCARTES

3.1. Nodes and Links

DESCARTES functionality can be illustrated with a single node (Inside) and a single link (L18) from the **Hiding Pot** network. Figure 5 shows the internal organization of the node Inside, as created from the definition in Figure 3. Inside is an instance of the class Simple-SA-Node, and contains the slots defined in (or inherited by) Simple-SA-Node. Figure 5 shows these slots and their values for the node Inside.

```
Node Inside, class Simple-SA-Node, sa-asleep-since NIL
INTERNAL-ID....: "Inside"
IN-LINKS.......: ("L15" "L16" "L17" "L18" "L19" "L20")
OUT-LINKS......: ("L2" "L21" "L28" "L35" "L41" "L68")
ACTIVATION.....:  2.291 (0.725 0.788 0.335 0.500)
THRESHOLD......: 0.05
DECAY-RATE.....: 0.8
```

Figure 5: Display on node Inside.

The internal-id slot is common to all user-accessible objects in DESCARTES, serving as a handle for the object. In-links and out-links contain the lists of input and output links, respectively. They connect the node to other nodes in the network, and allow activation and markers to be passed. The activation slot contains the node's current activation, as well as its activation history. In Figure 5, Inside is shown after cycle 7 of the simulation, so the first value shown is the current (unnormalized) activation, the next one (the first value in parentheses) is the activation on cycle 6, and so on. The threshold slot contains the minimum activation required for Inside to have an output, an activation functionality provided in DESCARTES's class hierarchy. Finally, the decay-rate slot holds the value of the activation's linear decay coefficient. Thus, Simple-SA-Node's activation is calculated by its activation function as follows:

$$A_{new} = (1-\partial) * A_{old} + \sum w_i * a_i$$

where A_{new} is the new activation, A_{old} is the activation from last cycle, ∂ is the node's decay-rate, and $\sum w_i * a_i$ is the sum of weighted activations over the node's in-links.

Networks, such as the ones described here, are inherently parallel computational models. Since most computers are still quite sequential, DESCARTES provides a functionality to speed up network processing on sequential architectures. This functionality allows the controller to only process those nodes, whose activation or markers change during a cycle. Thus a node whose activation remains constant at cycle W, need not be processed on that cycle. Such a node is said to be *asleep* at cycle W. It is important to note that in a typical semantic network, most nodes' activation and output remain constant on any given cycle. By declaring those nodes as asleep, the number of nodes processed during a given cycle is reduced, which improves DESCARTES performance on large networks.

Figure 6 shows the internal organization of link L18, one of Inside's input links, connecting Inside^Planner to Inside. Link L18 is an SA-LINK, which indicates it can

```
Link L18, class SA-LINK
INTERNAL-ID.: "L18"
SOURCE......: "Inside^Planner"
SINK........: "Inside"
ACTIVATION..: 0.11
WEIGHT......: 0.5
```

Figure 6: Display on link L18.

pass weighted spreading activation. Its source slot points to Inside^Planner (the actual value stored in that slot is a pointer to a CLOS object), the node from which L18 originates. The link's sink slot points to Inside, the node to which L18 delivers activation or markers. The weight slot specifies the link's weight--the strength of the connection between source and sink. Finally, the activation slot indicates the value that will be passed to the sink node during the Update procedure (see below). Activation on a link is calculated as follows:

$$Activation_{Link} = Weight_{Link} * Output_{LinkSource}$$

Activation of a link becomes one of the inputs of its sink node on the subsequent simulation cycle. The cycling mechanism is used to simulate parallel processing on the network(s) and is described below.

Most of the slots on Simple-SA-Node and SA-LINK are common to the other classes of nodes and links in DESCARTES; as previously mentioned, all user-accessible objects have an Internal-ID slot, all links have source and sink slots, etc.. Other classes, however, often add their own specific slots for processing (such as markers) or redefine the functionality of a class.

3.2. The Simulation Cycle

DESCARTES is designed in such a way that networks can be cycled in parallel or serially. The meta-controller provides for timing coordination between the networks. Networks cycled in parallel behave as if they were a single net, even though they need not operate at the same frequency or functionality. A particular model may have a network of inhibitory nodes cycling at a faster rate than a network of excitatory nodes with which it interacts, at the same time as symbolic markers are being passed over each, and backpropagation is being performed within sub-networks of the model. With serial cycling, one network may wait until another network completes a specified number of cycles or reaches stability before starting to cycle itself.

Each global network cycle is comprised of four steps: (1) determination of which networks need to be cycled, (2) update of active nodes in the cycling networks, (3) spread from active nodes in the cycling networks to their out-links, and (4) report any requested output.

Determining Active Networks: The meta-controller determines which of the networks in the system need to be cycled in parallel on the given cycle, according to defaults and any define-cycling commands. In Figure 4, spreading-activation nodes in Network-A will be cycled on every global cycle, while marker-passing nodes will be cycled only on global cycles 1, 4, 7, and so on, until termination (stability).

Update: Each active node in the cycling networks queries its incoming links for new activation and/or markers. Spreading-activation nodes calculate their new activation by applying their activation function, while marker-passing nodes store any new markers they have received.

Spread-To-Out-Links: Each active node in the cycling networks calculates its output (either activation or markers) and sends it to its outgoing links. The output of spreading-activation nodes is calculated by applying their output function, while the output of marker-passing nodes is generally their new markers. The links receive input from their source nodes and *wake up* (declare active) their sink nodes, if they are *asleep* (inactive).

Report Output: The final step of a cycle entails querying the cycling networks for results. Each network controller can optionally display the status of important nodes at specified cycles (Network-A's status will be displayed every 10 cycles in Figure 4) or trace new activation and/or markers. DESCARTES currently has a number of output options useful for system design and debugging.

3.3. A Simple Example

To illustrate how activation actually spreads through the network, consider Figure 7. It shows the first four cycles of a simulation of a simple network. This network contains two classes of nodes: max-nodes and sum-nodes. Max-nodes and sum-nodes are very similar to Simple-SA-Nodes, except that the activation function for max-nodes is the *maximum* activation of its in-links, while the activation function of sum-nodes is the *sum* of the activations of its in-links.

At cycle 1, node A is clamped with activation of 1.0, which means that A will keep its output at 1.0, regardless of its inputs. The other nodes in the network are asleep, and therefore are not updated. A then sends its output to its out-links, which will wake up node B. On cycle 2 A goes to sleep because its activation remains unchanged. B checks its input links and calculates its activation. B is a max-node, so its activation is the maximum of all of its inputs, in this case, 0.5. B then sends activation to its out-links, waking C and D.

At cycle 3, B goes to sleep, and both C and D are awake. They each calculate their activation by summing the inputs and pass it through their out-links. Node D then goes to sleep on cycle 4. Node C, however, remains active, since its input link had notified it that there is new activation from node D. Note that node A remains asleep, since it is clamped at a specific value, and inputs will not change its activation or output. Node B, however, becomes active, and will recalculate its activation at cycle 4 by taking the maximum of its inputs.

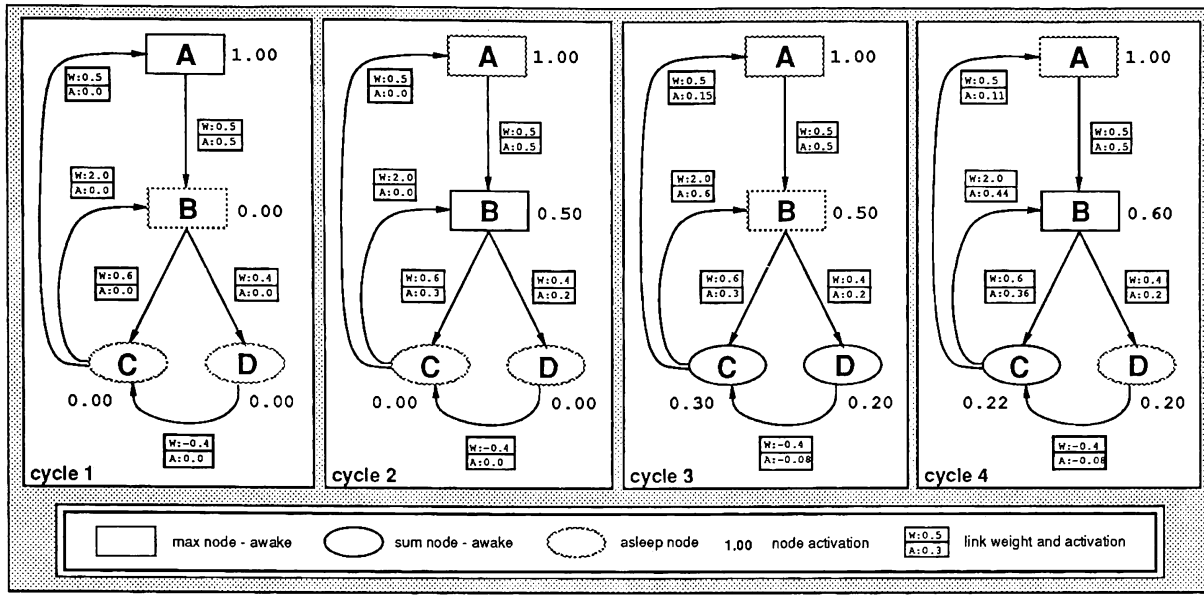

Figure 7: An example of a simple spreading-activation network.

This network will cycle 12 times before it reaches stability, at which time all nodes will be quiescent.

4. SIMULATION OF HIDING POT

Returning to the **Hiding Pot** example, figure 8 shows an interactive session in DESCARTES after the initialization of Figure 4 for *"John put the pot inside the dishwasher because the police were coming"*. After the cycle command is given, the markers originally marked on the role nodes of Transfer-Inside and Transfer-Self propagate along the connections shown in Figure 1, marking new nodes every third cycle. Concurrently, activation spreads along the Simple-SA-Nodes of the network until the ten cycles are completed. At this point the activations (first values) and activation histories of the last few cycles (in parentheses) of the desired nodes are displayed. Notice that the activation of node Cooking-Pot (0.235) slightly exceeds that of Marijuana (0.208) early in the simulation run. The display-connections command shown is one way to allow the user to "debug" the network, by displaying the incoming links that affect a node's activation (Transfer-Inside in this case).

The next (cycle 100 :display-on-exit-p T) command tells the *meta-controller* to cycle 100 times (or by default until termination) without displaying either the trace or network status until complete. The marker-passing portion of the network becomes stable (and thus stops cycling) when the markers have propagated throughout the network by cycle 18.

Activation continues to spread until the network becomes spreading-activation stable at cycle 42. The path of most highly-activated nodes runs, in this case, through nodes Transfer-Inside, Inside, and Inside-Opaque, forming part of the network's interpretation that John was trying to hide his Marijuana rather than clean his Cooking-Pot.

5. SIMULATOR INFORMATION

A complete description of DESCARTES's functionality and test-bed cases can be found in [Lange *et al.*, 1989b]. The largest test case simulated to date is an implementation of a ROBIN [Lange & Dyer, 1989b] network in the domain of **Hiding Pot**. It consists of two interacting LCNs built from four node classes and five link classes, with a total of 12,400 nodes and 40,000 links.

DESCARTES will be made available to all interested users. Enquiries about access to the simulator should be sent to <DESCARTES@CS.UCLA.EDU>.

6. RELATED WORK

Some of the recent tools constructed for building and simulating connectionist architectures are (1) the Rochester Connectionist Simulator (RCS) [Goddard *et al.*, 1987], (2) the PDP Software Package [McClelland & Rumelhart, 1988], (3) MIRRORS/II [D'Autrechy *et al.*, 1988], and (4) GENESIS [Wilson *et al.*, 1988]. RCS is a spreading-activation simulator which allows units to have any amount of associated data. There is no specification language for construction of the net, but the system provides a library of commonly used network structures and units. The PDP Software package includes a number of programs for simulating the DCN models in [Rumelhart & McClelland, 1986]. MIRRORS/II and GENESIS, the most recent of the four systems, have both features: a high level non-procedural language for network construction and an indexed library of commonly used networks.

```
> (cycle 10)
Cycling...

Newly marked nodes in Network-A on cycle 4:
Inside^Planner.................: (<Marker_John>)
Inside^Obj.....................: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Inside^Loc.....................: (<Marker_Dishwasher>)
Proximity-To-Object^Actor......: (<Marker_Police>)

Newly marked nodes in Network-A on cycle 7:
Goal-Avoid-Detection^Planner...: (<Marker_John>)
Goal-Avoid-Detection^Obj.......: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Goal-Clean-Dish^Planner........: (<Marker_John>)
Goal-Clean-Dish^Obj............: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Goal-Fight-Crime^Planner.......: (<Marker_Police>)
Inside-Dishwasher^Planner......: (<Marker_John>)
Inside-Dishwasher^Obj..........: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Inside-Dishwasher^Loc..........: (<Marker_Dishwasher>)
Inside-Opaque^Planner..........: (<Marker_John>)
Inside-Opaque^Obj..............: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Inside-Opaque^Loc..............: (<Marker_Dishwasher>)
Police-See-Illegal^Planner.....: (<Marker_Police>)

Newly marked nodes in Network-A on cycle 10:
Block-See^Planner..............: (<Marker_John> <Marker_Police>)
Block-See^Obj..................: (<Marker_Marijuana> <Marker_Cooking-Pot>)
Block-See^Loc..................: (<Marker_Dishwasher>)

Status of nodes in Controller1 from cycle 11 thru 7:

                                  11     10    9     8     7
Cooking-Pot....................:  0.536 (0.235 0.200 0.251 0.242)
Marijuana......................:  0.491 (0.208 0.174 0.210 0.199)
Transfer-Inside................:  2.143 (0.946 0.784 1.000 0.843)
Inside.........................:  2.699 (1.000 1.000 0.979 1.000)
Inside-Dishwasher..............:  1.630 (0.723 0.604 0.772 0.582)
Inside-Opaque..................:  1.417 (0.610 0.465 0.571 0.350)

> (display-connections %transfer-inside :length 5)

Node: "Transfer-Inside", class FRAME, sa-asleep-since NIL
Activation.............: 2.115 (2.115 0.784 1.000 0.843)
In-Links:
Inside.................: 2.237 (2.237 1.000 0.979 1.000) (<L2 0.9>)
Transfer-Inside^Actor.: 0.438 (0.438 0.199 0.218 0.228) (<Marker_John>) (<L3 0.5>)
Transfer-Inside^Loc...: 0.478 (0.478 0.217 0.243 0.253) (<Marker_Dishwasher>) (<L5 0.5>)
Transfer-Inside^Obj...: 0.443 (0.443 0.201 0.221 0.229) (<Marker_Marijuana> <Marker_Cooking-Pot>) (<L4 0.5>)
W-Put..................: 0.103 (0.103 0.121 0.107 0.114) (<L1 0.75>)

(cycle 100 :display-on-exit-p T)

Cycling......                  Network-A Marker-Passing stable at cycle 18
Cycling....................... Network-A Spreading-Activation stable at cycle 42

                      43     42    41    40
Cooking-Pot.:        0.506 (0.203 0.209 0.209....)
Marijuana.:          0.580 (0.233 0.233 0.233....)
Transfer-Inside.:    2.106 (0.846 0.846 0.846....)
Inside.:             2.491 (1.000 1.000 1.000....)
Inside-Dishwasher.:  1.543 (0.619 0.619 0.619....)
Inside-Opaque.:      1.573 (0.631 0.631 0.631....)
```

Figure 8: Example interactive session in Hiding Pot after setup in Figure 4.

Both have more sophisticated and flexible control mechanisms than RCS and the PDP Software Package, with MIRRORS/II emphasizing simulations using LCNs and GENESIS emphasizing realistic, biologically-based models.

The flexibility and symbolic capabilities afforded by DESCARTES' object-oriented implementation in COMMONLISP and CLOS comes at a small expense in simulation speed in comparison to the C-based implementations of RCS, the PDP package, and GENESIS. The only case where the difference in speed should be significant, however, is in simple backpropagation networks requiring thousands of learning epochs, for which the PDP package might be more appropriate. Except for GENESIS, all of the above-mentioned simulators are geared toward monotonic distributed or localist spreading-activation networks. None of them have the concept of hybrid multiple interactive networks as part of their design, especially those which can pass symbolic markers.

7. CONCLUSIONS

We have presented a development tool, DESCARTES, which provides researchers with the capability to combine Distributed Connectionist Networks, Localist Connectionist Networks and Marker-Passing Networks within a single simulation environment. The most important theoretical contribution of DESCARTES is the concept of Multiple Interactive Networks with intra- and inter-network heterogeneity. As a tool, it provides a simple, portable, and versatile environment for designing and testing different cognitive models. These capabilities make DESCARTES a powerful tool for researchers in Artificial Intelligence, Cognitive Modelling, and Connectionism.

ACKNOWLEDGEMENTS

This research has been supported in part by a contract with the JTF program of the DOD and a grant from the Office of Naval Research (no. N00014-86-0615). DESCARTES has been implemented on equipment donated to UCLA by Hewlett-Packard, Inc., and Apollo Computer, Inc. We would like to John Reeves, Colin Allen, Michael Dyer, and Eduard Hoenkamp for their helpful comments on previous drafts of this paper.

REFERENCES

D'Autrechy, C. L., Reggia, J. A., Sutton, G. G., & Goodall, S. M. (1988): A General-Purpose Simulation Environment For Developing Connectionist Models. *Simulation, 51(1)*, p. 5-19.

Charniak, E. (1986): A Neat Theory of Marker Passing. *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, PA, 1986.

Dyer, M. G. (1989): Symbolic Neuroengineering for Natural Language Processing: A Multi-Level Research Approach. In J. Barnden and J. Pollack, editors, *Advances in Connectionist and Neural Computation Theory*, Ablex Publishing, 1989. In press.

Goddard, N., Lynne, K. J., & Mintz, T. (1986): *Rochester Connectionist Simulator*. Technical Report TR-233, Department of Computer Science, University of Rochestor.

Hendler, J. (1988): *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Lange, T. & Dyer, M. G. (1989a): Dynamic, Non-Local Role-Bindings and Inferencing in a Localist Network for Natural Language Understanding. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, p. 545-552, Morgan Kaufmann, San Mateo, CA (Collected papers of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic, Denver, CO, November 1988).

Lange, T. & Dyer, M. G. (1989b): Frame Selection in a Connectionist Model of High-Level Inferencing. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society (CogSci-89)*, Ann Arbor, MI, August 1989.

Lange, T., Hodges J., Fuenmayor, M., & Belyaev, L. (1989a): DESCARTES: Development Environment For Simulating Hybrid Connectionist Architectures. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society (CogSci-89)*, Ann Arbor, MI, August 1989.

Lange, T., Hodges J. B., Fuenmayor, M., & Belyaev, L. (1989b): *The DESCARTES Users Manual.* Research Report, Computer Science Department, University of California, Los Angeles.

McClelland, J. L. & Rumelhart, D. E. (1988): *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA.

Nenov, V. I., & Dyer, M. G. (1988): DETE: A Connectionist/Symbolic Model of Visual and Verbal Association. *Proceedings of the IEEE Second Annual International Conference on Neural Networks (ICNN-88)*, San Diego, CA, July 1988.

Rumelhart, D. E., & McClelland, J. L. (1986): *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Volumes 1-2, MIT Press, Cambridge, MA.

Shastri, L. (1988): A Connectionist Approach to Knowledge Representation and Limited Inference. *Cognitive Science, 12*, p. 331-392.

Vidal, J. & Haggerty, J. (1987): Synchronization in Neural Nets. *Proceedings of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic (NIPS-87)*, Denver, CO, November 1987.

Waltz, D. & Pollack, J. (1985): Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. *Cognitive Science, 9(1)*, p. 51-74.

Wilson, M. A., Upinder, S. B., Uhley, J.D., & Bower, J. M. (1988): GENESIS: A System for Simulating Neural Networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, p. 485-492, Morgan Kaufmann, San Mateo, CA (Collected papers of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic, Denver, CO, November 1988).

AUTHORS' BIOGRAPHIES

TRENT LANGE is a Ph.D. student in Artificial Intelligence in the Computer Science Department at UCLA. He received a B.S. in Computer Science in Engineering from UCLA in 1985. His main area of research is in connectionist models of high-level inferencing, but he is also interested in short-term sequential memory, artificial neural oscillators, and connectionist simulation environments.

Trent Lange
Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90025
lange@CS.UCLA.EDU, (213) 825-5199

JACK HODGES is a Ph.D. student in Artificial Intelligence in the Computer Science Department at UCLA. He received B.S. and M.S. degrees in Aerospace Engineering from the University of Michigan in 1976 and 1978. His research addresses the representation and processing of simple mechanical devices necessary to support naive improvisation and invention.

Jack Hodges
Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90025
hodges@CS.UCLA.EDU, (213) 825-5199

MARIA FUENMAYOR is a Ph.D. student in Artificial Intelligence in the Computer Science Department at UCLA. She received B.S. and M.S. degrees in Computer Science from the Universidad Simon Bolivar (Caracas, Venezuela) in 1980 and 1983. Her main area of research is in connectionist models of planning and creativity. Other areas of interest are logic and object-oriented programing.

Maria Fuenmayor
Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90025
maria@CS.UCLA.EDU, (213) 825-5199

LEONID V. BELYAEV is a Ph.D. student in Artificial Intelligence in the Computer Science Department at UCLA. He received an A. B. degree in Computer Science from University of California, Berkeley, in 1986. His main research interest is in the area of connectionist models of semantic and episodic memory, and participation in political discussions. He is also involved in research of Soviet perspectives on arms control agreements.

Leonid V. Belyaev
Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90025
leo@CS.UCLA.EDU, (213) 825-5199