Learning Concepts in Software Agents

Uma Ramamurthy[*], Stan Franklin[**], Aregahegn Negatu[***]
[*]St. Jude Children's Research Hospital[1]/University of Memphis
332 North Lauderdale, Memphis, TN 38105, U.S.A.
urmmrthy@memphis.edu
[**]University of Memphis[2]
Department of Mathematical Sciences, Memphis, TN 38152, U.S.A.
stan.franklin@memphis.edu
[***]University of Memphis
Department of Mathematical Sciences, Memphis, TN 38152, U.S.A.
negatua@msci.memphis.edu

Abstract

This concept-paper explores issues related to learning new concepts in software agents which inhabit dynamic domains. We argue that agents learn based on what they already know and agents solve new problems which they encounter by making analogies to previously solved problems of similar type. We explore these issues within the scope of the Cognitive Agent Architecture and Theory (CAAT) [Franklin, 1997]. The architecture of a multi-agent system, CMattie which is based on the CAAT strategy and inhabits an email-based dynamic domain, is described. CMattie gathers information from humans, composes announcements of next week's seminars, maintains a mailing list, mails the weekly seminar announcement to members of that mailing list, and learns new variations to seminars and other such events which have to be announced to the members of the mailing list as her domain changes with new types of seminar-like events. Learning mechanisms being implemented in this system which enable CMattie to adapt to her changing domain are described. Through her learning, CMattie acquires new domain-specific concepts, thus adapting to her dynamic domain.

1. Introduction

Computer networks provide an infrastructure for an efficient information exchange and processing; as a result they improve individual and group performances. E-mail is one of the most used tools that help people transfer and manipulate information. In our academic setting, e-mail service is so important that many standard activities are based on it. Introducing intelligent agents into computer systems can automate some simple and not so simple routine organizational tasks so that human workers will be freed from such routine but tedious work to concentrate on some other higher-level and non-routine tasks.

1.1 Cognitive Agent Architecture and Theory

We have been participating in an intelligent system research using the Cognitive Agent Architecture and Theory (CAAT) strategy [Franklin, 1997]. Basic cognitive processes include learning, memory (long term & short term), concept formation, perception, attention, problem solving, decision-making, thinking, etc. An autonomous agent that exhibits many or most of these cognitive processes (functions) can be referred to as a cognitive agent [Franklin, 1996] [Franklin, 1997]. Agent architectures may be characterized according to their capabilities (learning, planning, etc.), their properties (learning: deliberative, reflexive, monatomic, non-monatomic, etc.) and the nature of environments (static, dynamic, etc.) and domains for which they are designed. In general, agent architecture can be stated as a subset of a system that contains and manages the underlying and primitive resources of an agent. A methodical study of these resources and their management will assist in determining the necessary, sufficient and optimal distribution of resources for the development of agents that exhibit some type of intelligence.

In the design space [Sloman, 1995], we develop specifications (what capabilities and what properties/characteristics in each capability are needed for a specific domain) for a control structure which in turn has some type of underlying theory to support it. So, in CAAT strategy, architecture design entails the creation of theories of cognition to reinforce or modify existing theories or to serve as a hypothesis for a new theory. Conversely, any

[1] First author supported in part by Cancer Center Support CORE Grant, P30 CA 21765 and by American Lebanese Syrian Associated Charities (ALSAC).

[2] Second author supported in part by NSF grant SBR-9720314 and by ONR grant N00014-98-1-0332

improvements in theory should lead to possible enhancements of the architecture. This synergy between theory and architecture should lead to productive experimental work in artificial intelligence, cognitive science and other related fields.

1.2 VMattie

VMattie is a software agent which gathers information from humans, composes announcements of next week's seminars, maintains a mailing list and mails the weekly seminar announcement to members of that mailing list, all without human supervision. VMattie "lives" in a UNIX-based system. It is an intelligent software agent living in a complex dynamic environment, with multiple high-level perceptions and actions satisfying many drives.

VMattie is a multiagent system with all internal actions accomplished by microagents called **codelets**. VMattie's architecture is based on multi-layer, feed-forward neural network based categorization mechanism and Behavior Networks [Maes, 1989] [Maes, 1990]. The Behavior Networks component of VMattie is actually an extension of the classic Behavior Networks. Using the Behavior Networks component, VMattie deals with message variables, with variable drives and with activation from internal states, thus providing informal planning and action selection in this system. The message understanding module of VMattie is composed of a feed-forward neural network mechanism working with codelets to achieve the natural language understanding within a narrow-domain when VMattie perceives its input, namely incoming mail messages.

In VMattie's architecture shown in Figure 1, the three units -- Drives, Behavior Network and Attention Registers -- form the action-selection module with important extensions. Each behavior is implemented by a collection of codelets. The two units -- Input Processing Knowledge and Input Processing Workspace along with the associated codelets -- form the message understanding module of VMattie.

Several distinct drives operate in parallel in this system. The salient ones amongst these drives are:

1. To understand every incoming mail message,
2. To acknowledge every incoming mail message,
3. To maintain complete information on every ongoing seminar (in its Tracking Knowledge Base),
4. To keep the mailing list current, and
5. To mail out the weekly seminar announcements on time.

The urgency of these drives varies with incoming mail messages and the nearing of the time for mailing the weekly seminar announcements. This variation in drive-urgency is an enhancement over the classic Behavior Network architecture. Drives provide activation to behaviors.

The behavior net consists of various behaviors with their associated links: successor links, predecessor links and conflictor links [Maes, 1989]. In VMattie, behaviors get instantiated, thus supporting variables. For example, the add-address-to-list behavior asks which address to add to the list. The behaviors consist of preconditions, activation, add-list and delete-list accommodating variables in the contents of any of these. Activation spreads through the instantiated behaviors.

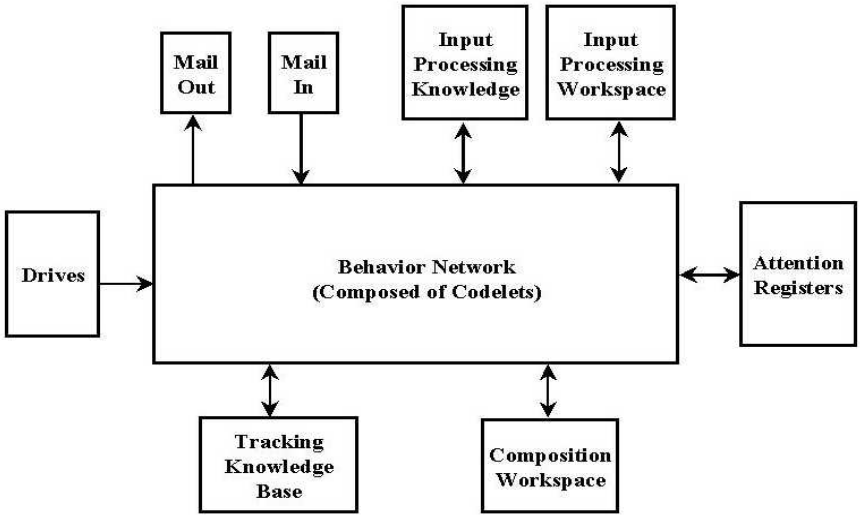

Figure 1: VMattie's Architecture

Attention Registers hold information extracted from an incoming mail message. Each register holds the content of a field, so that codelets which need it can get it from the Attention Registers. Such fields include name-of-the-organizer, email-address, seminar-name, etc. When filled, Attention Registers provide activation to behaviors that can use their contents.

The Input Processing Knowledge holds the declarative knowledge needed to understand the incoming mail messages. For example, it knows the various forms of "Wednesday" -- Wed., wed., Wednesday, etc., and the

names of the buildings where seminars are held. Input Processing Workspace holds the contents of an incoming mail message while codelets associated with the Workspace attempt to understand the mail message and intermittent results. In regular consultation with the Input Processing Knowledge, inferences are made. The most significant inference of all is the type of the incoming mail message. Once an incoming mail message is understood -- every significant phrase or word is given a field name and the type of the incoming mail message is inferred, then the primitive codelets transfer this information to the Attention Registers. Primitive codelets are not directly connected to any behavior or neural network node, and perform housekeeping functions in the VMattie system.

The Tracking Knowledge Base unit holds information used in composing outgoing mail messages. Examples of such information are default data for ongoing seminars as regards the name of the seminar, the organizer of the seminar, the usual seminar time and place. Behaviors update the default data. This unit maintains the mailing list for announcements with help from behaviors. The Tracking Knowledge Base also holds the templates for the different types of outgoing mail messages from the VMattie system.

The outgoing mail messages are composed in the Composition Workspace. In this unit, choosing of the template and filling the appropriate fields takes place. The information for the composition comes from the Tracking Knowledge Base and the Attention Registers.

2. CMattie

CMattie is the next experiment under the CAAT strategy. CMattie "lives" in a UNIX-based system. She is a software agent that gathers information from humans, composes announcements of the next week's seminars, maintains a mailing list and mails the weekly seminar announcements to members of that mailing list, all without human supervision.

In CMattie, new mechanisms have been added to the VMattie's architecture. This has resulted in a modified, more complex architecture for CMattie (shown in Figure 2). CMattie has a limited capacity Global Workspace [Baars, 1988] which is capable of broadcasting to all the processes. An attention mechanism, the Spotlight, controls access to the Global Workspace of coalition of processes.

CMattie has memories based on Sparse Distributed Memory architecture [Kanerva, 1988] and case-based memory [Kolodner, 1993]. CMattie has an emotional mechanism that enables her to be emotional when appropriate. CMattie has a sense of self-preservation which enables her to be concerned with her environment, namely her resource needs and the status of the UNIX-based system that she "lives" in. Most important of all, CMattie learns. She learns new concepts and new behaviors related to those newly learnt concepts in her dynamic domain.

CMattie has an episodic memory which acts as an intermediate memory. This is a case-based memory system. In her episodic memory, she stores sequences of mail messages that form episodes in her domain. Using this episodic memory, she can relate to new events which are similar to past events in her domain and which she is capable of understanding from her built-in domain knowledge. This memory acts as intermediate memory as information stored here is used in learning domain knowledge and if found relevant to the overall drives of the system, the information is passed on to the long-term memory of the system, the Sparse Distributed Memory.

Due to the very nature of her domain, namely, interaction through mail messages, reinforcement learning is best suited for many of CMattie's learning mechanisms. Reinforcement learning [Kaelbling, 1996] is a trial and error approach to learning. The agent learns by adjusting the

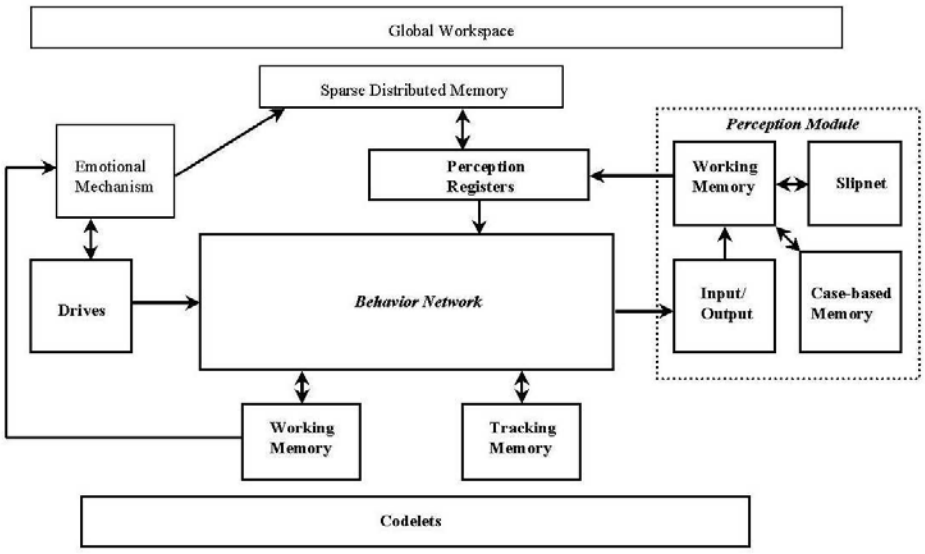

Figure 2: CMattie's Architecture

mapping from her domain states to her actions based on positive and or negative feedbacks. CMattie has built-in domain knowledge and built-in behaviors. When her domain changes, she attempts to interpret the new events in her domain in terms of her built-in knowledge and behaviors. This elicits some feedback from the members on her mailing list who send her mail messages. Based on that feedback, CMattie learns that she needs to modify some of the concepts in her domain knowledge, thus creating new concepts and further, new behaviors.

The various aspects of learning in CMattie are outlined below:

1. Associative learning in the codelets or processes based on Pandemonium Theory [Jackson, 1987].
2. Learning occurs automatically in the Sparse Distributed Memory architecture which is being used as long-term memory in CMattie.
3. The Slipnet in CMattie's Perception Module forms part of her built-in domain knowledge. CMattie learns new Slipnet nodes and fits them into her existing Slipnet.
4. For the new Slipnet nodes, CMattie learns associated new codelets.
5. When a change occurs in CMattie's domain knowledge, most often it calls for new behaviors. Based on the changes in her domain knowledge and feedback from her environment, she modifies her existing behaviors to learn new behaviors for her action-selection.

The last three mechanisms of learning in the CMattie system listed above are some form of reinforcement learning. In this paper, we focus on learning mechanisms for items (3) and (4) above.

3. Perceptual Learning

The Perception Module of CMattie is based on the Copycat architecture [Mitchell, 1993]. The Copycat system is based on the thesis that analogy-making lies at the core of understanding and analogy-making is itself a process of high-level perception. The Copycat system interprets and makes analogies between situations in an idealized microworld involving letter-string analogy problems. The basic objects in this microworld are the 26 letters of the alphabet. A typical analogy problem in this world is as below:

abc → abd
ijk → ?

i.e., one is given the change from string **abc** to the string **abd** and asked to come up with a similar change for the target string **ijk**.

Copycat's architecture consists of 4 parts: (1) The **Slipnet**, (2) **Workspace**, (3) **Pool of Codelets**, and (4) **Temperature**. The **Slipnet** is a network of nodes and links which represent the permanent concepts of the system. Copycat builds perceptual structures in the **Workspace** to represent the system's current understanding of the given analogy problem. The **Pool of Codelets** consists of various perceptual and higher-level structuring agents called **Codelets**, waiting to run. A **codelet** is a small piece of code that carries out some local task. These agents build and sometimes destroy the perceptual structures in the **Workspace**. The **Temperature** measures the amount of disorganization or entropy in the system's understanding of the given analogy problem and controls the degree of randomness used in making decisions.

The domain of the Copycat system is predefined and fixed. Hence there is no learning in this system. On the contrary, CMattie "lives" in a dynamic domain. As her domain changes, to enable her to perceive this dynamism, she learns new concepts in her Perception Module.

3.1 Concepts and Features

Concepts are objects or logical units of ideas that are used to represent entities. In the domain of CMattie, Seminar, Seminar Organizer, E-mail Address, Periodicity, Location, Title of Talk, etc. are considered concepts. Such concepts need to be represented in such a way that cognitive processes relevant in the domain will manipulate these concepts effectively.

In CMattie, concepts are manipulated continuously. After sensing raw data from the environment, a perceptual cognitive activity takes place. The perception involves building of instances of known concepts, detection and creation of new concepts, making appropriate relations among concepts. In our design, we like to have a simple conceptual representation, yet powerful enough to be used in a relatively complex domain. The basic characteristics of a concept are represented by its features or attributes. In a given situation, a feature of a concept has a specific measure called “value”. For instance, a “seminar” concept has a feature called “name” whose values could be “Complex System” or “Cognitive Sciences”. A concept may have more than one feature. The same “seminar” concept can have another feature called “organizer” whose values could be “S. Franklin” or “M. Garzon”. A concept can be a feature for another concept. For instance, “organizer” which is a feature of the “seminar” concept is by itself a concept which has a feature called “e-mail address” with a value, say “franklin@msci.memphis.edu.”

In CMattie, concepts are defined not necessarily by a single node in the Slipnet of her Perception Module. A concept has a core and a set of features. Each of these may be individual nodes or group of nodes. The various nodes of the Slipnet are connected to one another through weighted links between them. One of the concepts in CMattie's built-in domain knowledge is the **Seminar**. The **Seminar** concept has the following features:

- **Name** of the seminar
- **Organizer** of the seminar
- **Location** where the seminar will be held
- **Date** of the seminar
- **Day** of the seminar
- **Time** at which the seminar will be held
- **Speaker** of the seminar
- **Title of talk** for the seminar

- **Periodicity** of the seminar

In Figure 3, a small segment of the Slipnet from the Perception Module of CMattie is shown. The direction of the links indicates the possible spread of activation between the nodes.

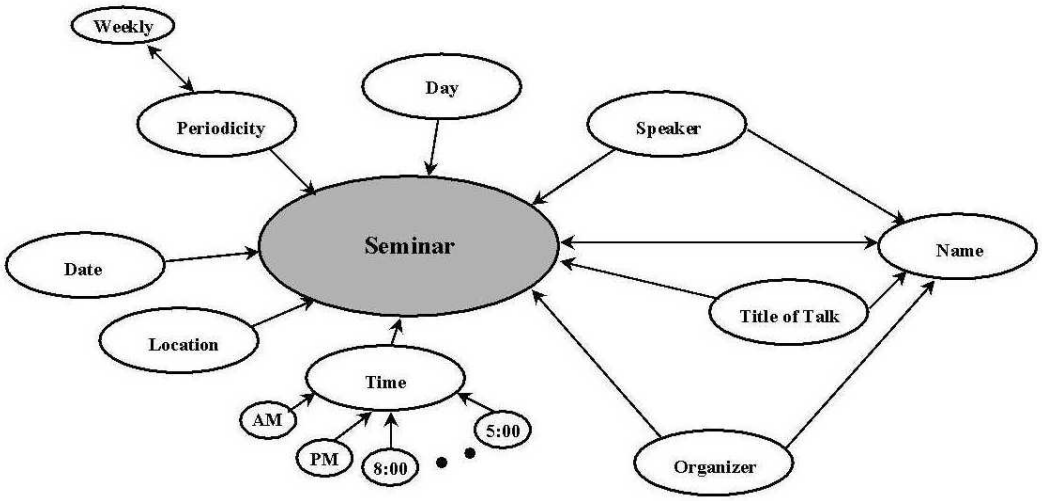

Figure 3: Some nodes and weighted links in CMattie's Slipnet

As can be seen from the above figure, **Time** is a feature of the **Seminar** concept. But **Time** by itself is also a concept with a set of its own features. The concept **Seminar** is much deeper and has a higher **depth value** than the concept **Time**. The links are weighted and these weights aid in reinforcement learning in the Slipnet.

Next question we need to ask is "how does CMattie learn new concepts ?"

3.2 Learning new Concepts

CMattie's senses are the mail messages she receives. In her Perception Module, the Slipnet contains most of her domain knowledge. Using this domain knowledge, she understands the input, namely mail messages, sent to the CMattie system. This understanding process occurs in the Perceptual Working Memory through limited amount of natural language understanding. Through her built-in domain knowledge, CMattie is capable of understanding different types of messages related to the mailing list she maintains and messages related to seminars.

CMattie has a limited number of seminars already defined in her Slipnet. CMattie "knows" about these seminars through the built-in **seminar** concept and its features: She "knows" --

- a seminar is held once a week;
- it has an organizer and a name;
- each week, it might have a different speaker;
- it has a different title-of-talk;
- it is usually held at the same location, on the same day of the week and at the same time, unless stated otherwise.

When a Seminar-Organizer sends her a message announcing a seminar with a seminar name that she has never seen before, she attempts to treat such a message as similar to seminars that she already knows. The learning mechanism here is based on the premise that **the agent learns based on what it already knows**. When the message understanding mechanism in the Perception Module attempts understanding this message, the system recognizes that it is a **initiate-seminar-message** for a seminar, but the **name** of this seminar is not part of the built-in domain knowledge. In such a situation, CMattie has built-in mechanisms to converse with the sender of the message to determine if the sender wishes to initialize a new seminar. She sends an acknowledgement to the sender of the message stating that a new seminar with that seminar name will be initialized in the system, with the sender as the organizer for that seminar. Reinforcement is provided to CMattie by the response she might or might not get for such an acknowledgement. Based on such a feedback, she creates a new node for this seminar name and links it to the **name** node (which is also a feature of the seminar concept) in the Slipnet. When a new node is generated, the underlying codelets for that node are generated as well. This process is quite straight forward, as the new codelets are based on existing similar codelets for other seminar name nodes. Once this process is complete, CMattie has understood the incoming message and the Perception Module sends the relevant fields from the understood message to the Perception Registers.

The second type of learning which takes place in the Perception Module is when CMattie learns concepts which are not completely identical to the built-in seminar concept, but slightly different from it. In CMattie's domain, the Colloquia, Dissertation Defenses, Dissertation Committee Meetings, Faculty Meetings fall in the category of concepts which are not totally identical to the seminar concept. This learning mechanism is based on the premise that **every new**

situation/problem is viewed by the agent in terms of a previously solved problem (analogy-making). When CMattie receives a message of such a non-seminar event, say a Colloquium, she treats it as a **speaker-topic message** for a seminar. She sends an acknowledgement to the sender stating that she is initializing a new seminar by the name "Colloquium Seminar" in the system and the sender is the organizer for this seminar. This misunderstanding effects one or more of the following events dependent on the sender:

- The wrong acknowledgement might elicit a negative response from the sender, starting an episode. The "conversation" between CMattie and the sender is stored in the case-based memory system of the Perception Module as an episode. This episode provides the reinforcement and along with the limited natural language understanding based on this episode, CMattie learns that Colloquium is similar to the seminar concept with slightly different features. In the case of colloquium, the **periodicity** feature has a different value. CMattie learns that through reinforcement from her domain.
- The sender might ignore this wrong acknowledgement and CMattie includes the Colloquium Seminar in her weekly seminar announcement.
- CMattie sends out the weekly seminar announcement with the Colloquium Seminar listed in it. That announcement might elicit a negative message from the sender of the original Colloquium message, starting a "conversation" between CMattie and the sender. This episode is stored in CMattie's case-based memory to aid in her learning what a colloquium is.
- The sender of the original Colloquium message might ignore the weekly announcement with the Colloquium Seminar listed in it. But he/she might respond to the reminder sent by CMattie the following week, when CMattie doesn't receive a **speaker-topic message** for the Colloquium Seminar. This, again, generates an episode aiding CMattie to learn about a colloquium.
- The sender of the original Colloquium message ignores all the reminders. This acts as a feedback to the CMattie system to modify the **periodicity** feature of the seminar concept, giving rise to a new concept which is similar to the seminar concept with a modified set of feature-values.

Irrespective of which of the above events occur, CMattie eventually learns the new concept called Colloquium which is closely related to the seminar concept. This concept is of the same depth as the seminar concept with a modified set of features. CMattie also learns related new codelets for the colloquium concept, by copying and modifying the codelets for the built-in seminar concept, to be able to correctly perceive and understand a **Colloquium message** when she receives one the next time.

4. Conclusions

In this paper, we briefly described the architecture of a software agent, CMattie, whose design considerations are based on the CAAT strategy. We also showed how CMattie learns new concepts to adapt to her dynamic domain. Our focus here was to discuss perceptual learning to detect changes in her domain, generate new concepts in CMattie's domain knowledge so as to enable her to respond appropriately to novel situations in her domain. The learning mechanisms are based on the premises: (1) Agents learn from what they already know, and (2) Every new situation/problem is viewed in terms of a previously solved problem. At the time of writing this paper, the VMattie system has completed preliminary testing and is starting on a beta test and the CMattie system is being implemented. We plan to focus on issues related to unlearning in the future stages of this project.

References

Bernard J. Baars. (1988). "A Cognitive Theory of Consciousness", Cambridge University Press.

Stan Franklin. (1995). "Artificial Minds", MIT Press.

Stan Franklin and Art Graesser. (1996). "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag.

Stan Franklin. (1997). "Autonomous Agents as Embodied AI," Cybernetics and Systems, special issue on Epistemological Issues in Embodied AI, 28 p. 499-520.

John V. Jackson. (1987). "Idea for a Mind", SIGART Newsletter, No. 181, July 1987, p. 23-26.

Leslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore. (1996). "Reinforcement Learning: A Survey", Journal of Artificial Intelligence Research 4 p. 237-285.

Pentti Kanerva. (1988). "Sparse Distributed Memory", MIT Press.

Janet Kolodner. (1993). "Case-based Reasoning", Morgan Kaufmann Publishers.

Pattie Maes. (1989). "How to do the right thing", Connection Science Journal, Vol. 1, No. 3, March 1989.

Pattie Maes and Rodney A. Brooks. (1990). "Learning to Coordinate Behaviors", Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI Press/MIT Press, p. 796-802.

Melanie Mitchell. (1993). "Analogy-Making as Perception", MIT Press.

Aaron Sloman. (1995). "Exploring Design Space and Niche Space", Proceedings of the Fifth Scandinavian Conference on AI, Trondheim, May 1995, Amsterdam: IOS Press.

Hongjun Song and Stan Franklin (forthcoming), Action Selection using Behavior Instantiation.