Journal of Artificial General Intelligence 5(1) 47-66, 2014
DOI: 10.2478/jagi-2014-0002

Submitted 2014-07-10
Accepted 2014-11-30

The Action Execution Process Implemented in Different Cognitive Architectures: A Review

Daqi Dong DDONG@MEMPHIS.EDU
Stan Franklin FRANKLIN@MEMPHIS.EDU
Department of Computer Science and the Institute for Intelligent Systems, University of Memphis, Memphis TN 38152, USA

Editor: Moshe Looks

Abstract

An agent achieves its goals by interacting with its environment, cyclically choosing and executing suitable actions. An action execution process is a reasonable and critical part of an entire cognitive architecture, because the process of generating executable motor commands is not only driven by low-level environmental information, but is also initiated and affected by the agent's high-level mental processes. This review focuses on cognitive models of action, or more specifically, of the action execution process, as implemented in a set of popular cognitive architectures. We examine the representations and procedures inside the action execution process, as well as the cooperation between action execution and other high-level cognitive modules. We finally conclude with some general observations regarding the nature of action execution.

Keywords: action execution, cognitive model, cognitive architecture

1. Introduction

Action plays an essential role in an autonomous agent (Franklin and Graesser, 1997): The agent interacts with its environment by performing actions to achieve its goals. From different fields of study such as psychology, neuroscience, and cognitive science, researchers have provided evidence and formulated hypotheses to explain how human action works. Marc Jeannerod, citing the work of Searle (Searle, 1983), built upon the concept that covert action representation is followed by overt, real execution of action. In detail, "…the conceptual content, when it exists (i.e., when an explicit desire to perform the action is formed), is present first. Then, at the time of execution, a different mechanism comes into play where the representation loses its explicit character and runs automatically to reach the desired goal" (Jeannerod, 2006, pp. 4-5). Jeannerod suggests that action representation (preparation) and action execution are two different processes. A similar idea of distinguishing action execution from action preparation is proposed by Milner and Goodale as well. In their work on the two visual systems (1992; 2008), they proposed two cortical systems, the ventral and dorsal streams, providing "vision for perception" and "vision for action" respectively. Regarding the roles of the two streams in the guidance of action, the perceptual mechanism in the ventral stream identifies a goal object, and helps to select an appropriate course of action, while the dorsal stream "is critical for the detailed specification and online control of the constituent movements that form the action" (Milner and Goodale, 2008, p. 775). In our recent work (Dong and Franklin, 2014), we have proposed as well that human action represents two aspects, the understandable and the executable. On the one hand, an action represents the behavioral result of the agent's mental process, and thus is able to be recognized by the agent. On the other hand, the action involves low-level environmental information that enables the agent to execute the action through appropriate use of its actuators in the environment. Action execution transforms a goal-directed action into low-level executable actions. Additional studies regarding human action, especially certain neuroscientific evidence, can be found in recent review papers (Castiello, 2005; Grafton, 2010).

In the field of cognitive modeling, the challenge of creating a real-life computational simulation of the human mind calls for efforts to develop biologically-inspired intelligent software agents and robots. The evidence and hypotheses mentioned above provide a basis for the development of a cognitive model for an agent's action and its execution.

Cognitive architectures are designed to be the basis for creating general, autonomous agents that can solve a wide variety of problems using a wide range of knowledge; they define and organize the primitive computational structures that store, retrieve, and process knowledge to pursue the agent's goals (J. Laird, 2012). A collection of cognitive architectures have been reviewed in recent studies (Duch, Oentaryo, and Pasquier, 2008; Goertzel, Lian, Arel, De Garis, and Chen, 2010; Langley, Laird, and Rogers, 2009). Regarding actions and their execution processes, brief summaries have been made, such as "… a cognitive architecture must also be able to execute skills and actions in the environment. In some frameworks, this happens in a completely reactive manner, with the agent selecting one or more primitive actions on each decision cycle, executing them, and repeating the process on the next cycle. This approach is associated with closed-loop strategies for execution, since the agent can also sense the environment on each time step. The utilization of more complex skills supports open-loop execution, in which the agent calls upon a stored procedure across many cycles without checking the environment. However, a flexible architecture should support the entire continuum from fully reactive, closed-loop behavior to automatized, open-loop behavior, as can humans." (Langley et al., 2009)

Here we examine such cognitive models of actions, and especially of the action execution process as it is implemented in different cognitive architectures. The emphasis is placed on three questions: 1) What are the comprehensive representations and functional procedures of action execution? 2) How do action preparation/selection and action execution cooperate? and 3) What kind of specific designs are useful for generating actions that both achieve the agent's motivations and execute appropriately using actuators in the environment?

The cognitive model of action in different cognitive architectures might be implemented variously because of the model's tasks. For example, an action implemented in a simulated chess match could be an abstract move, such as moving a piece one square to the left, or the low-level actions implemented in a simulated tennis match which are necessary for controlling the player's muscles (actuators) during the game. A chess action is driven only by the agent's internal goal; it is not affected by the environmental situation—except the abstract representation of the position of the pieces—nor does it require the maintenance of specifications for its actuators. On the other hand, actions in tennis are generated on the basis of both the player's internal goals as well as the present environmental situation. Tennis, unlike chess, requires an action execution process that enables the agent to act in an uncertain and dynamic environment[1]. The action execution process is the focus of this review.

1. A task environment is uncertain if 1) the agent sensors do not detect all aspects that are relevant to the choice of action, or 2) the next state is unable to be determined by the current state and the executed action; and the environment is dynamic if it can change while an agent is deliberating (Russell and Norvig, 2009).

Even for the cognitive architectures in which action execution has been considered, the architecture may or may not completely implement it as a standard component. Some architectures implement a complete action execution process, such as ACT-R (J. Anderson, 2007) and LIDA (Franklin, Madl, D'Mello, and Snaider, 2014); other architectures only transform prepared/selected high-level actions into general low-level actions, and leave the action execution process to a domain-dependent external program, such as Soar (J. Laird, 2012). We review both types of architectures' action execution models below.

The next section describes a set of popular cognitive architectures with models for action execution. For each of these architectures, we first give a brief introduction and overview of the architecture's major components and functions; then we examine its specific implementation of action execution. Section 3 concludes with a comparative summary of the action execution processes reviewed in Section 2.

2. Action Execution Processes of Cognitive Architectures

In this section, we review the action execution processes of different cognitive architectures in the ensuing subsections. In each of the subsections, after an introduction to the architecture, we examine the representations and procedures inside the action execution process, as well as the cooperation between action execution and other high-level cognitive modules.

2.1 4D/RCS

The review content of 4D/RCS mentioned here is mainly in response to a paper by Albus and Barbera (2005). We cite the paper for this whole subsection, unless other explicit citations or quotations are mentioned.

Real-time Control System (RCS) is a cognitive architecture designed to enable multiple levels of intelligent behaviors, achieved by a multi-layered hierarchy of sensory-interactive intelligent control process nodes. The most recent version, 4D/RCS, embeds the 4-D approach (Dickmanns, 1992, 2000), a machine vision technology, within the RCS control architecture.

Each node in the architecture contains sensory processing (SP), world modeling (WM), value judgment (VJ), behavior generation (BG), and a knowledge database (KD) (Albus and Barbera, 2005). A SP process receives input from sensors; SP and WM processes cooperate to filter, attribute, and classify the input data as a perception process; WM processes create and update the recognized states of the world in the KD; a BG process accepts tasks and plans, and executes behaviors to accomplish those tasks; a VJ process evaluates the results of tentative plans, and saves evaluation results in the KD.

Process nodes act hierarchically. The BG processes form a command tree: each input task is decomposed into a plan consisting of subtasks for subordinate BG processes. Information maintained in the KD is shared between WM processes in nodes above, below, and at the same level within the same sub tree. Sensory data flow up the SP hierarchy typically forms a graph; and these data are populated by the WM in the KD at each level.

A WM predicts what will change in the world as the result of an action, and what will stay the same, giving its solution to the frame problem. Specifically, the location and direction of motion of objects in the world are represented in an image or map, and a simple comparison between one frame and the next distinguishes what changes from what does not in a dynamic environment.

A BG process receives tasks from a supervising BG process as input. The receiving BG process has a planner that decomposes each task into a set of coordinated plans for subordinate BG processes. During this period, tentative plans are proposed by the BG planner; the VJ evaluates the probable results of those plans as predicted by the WM; and a plan selector in the BG planner will choose the plan with the greatest value as the current plan. "For each subordinate there is an executor that issues commands, monitors progress, and compensates for errors between desired plans and observed results. The Executors use feedback to react quickly to emergency conditions with reflexive actions. Predictive capabilities provided by the WM may enable the executors to generate pre-emptive behavior" (Albus and Barbera, 2005).

In each node, the content of the selected current plan is moved from the planner into an "executor plan buffer" that initiates and guides the upcoming execution. This buffer is an interface between the planner and executor processes, and also the interface between deliberative and reactive processes.

At the top level of the architecture, the task is defined by the agent's goal that is established typically by a human operator outside of the agent[2]. At each successive level in the hierarchy, tasks from the level above are decomposed into subtasks that are sent to the subordinate levels below. Finally at the bottom level, decomposed task commands are sent to actuators to generate movements.

In each node of 4D/RCS, the execution is driven by a selected plan so as to reflect the requirements of the agent's goal in a bottom-up fashion that is reactive to the sensory input. Thus, at the lower levels of the architecture, the process nodes generate goal-directed reactive behaviors, while at the higher levels, the process nodes enable decision-making and deliberative behaviors. 4D/RCS has implemented both the action preparation/selection and the action execution processes hierarchically; it allows a more gradual, and thus smoother, transformation from the agent's motivations, represented by a top-level task, to low-level actions that are directly applied to the agent's actuators.

2.2 ACT-R

Adaptive Control of Thought-Rational (ACT-R) is a cognitive architecture, a theory for simulating and understanding human cognition based on numerous facts derived from psychological experiments (Budiu, 2013). ACT-R consists of two types of modules: memory modules and perceptual-motor modules. Action preparation (selection) and action execution are implemented in ACT-R by using these two types of modules separately. An agent's motivation is achieved by choosing a proper action in memory modules, and the action is appropriately executed in the motor modules.

There are two types of memory modules in ACT-R: declarative memory and production memory. Declarative memory, represented in structures called chunks, maintains knowledge that people are aware of, and can share with others through a set of buffers. Procedural memory, encoded in production rules, represents knowledge outside of their awareness that is expressed in their behavior rules (ACT-R 6.0 Tutorial, 2012). A production rule is a condition-action pair. The condition specifies a pattern of chunks that must be in declarative memory's buffers for the production to apply; a production fires if its condition matches the chunks in the buffers. The action specifies some actions, all of which are to be taken when the production fires (ACT-R 6.0 Tutorial, 2012). "[A] critical cycle in ACT-R is one in which the buffers hold representations

2. A 4D/RCS agent is not autonomous because its original goal is not driven by its own motivation or agenda but comes from outside, created by a supervisor, such as its operator.

determined by the external world and internal modules, patterns in these buffers are recognized, a production fires, and the buffers are then updated for another cycle" (J. R. Anderson et al., 2004).

ACT-R's perceptual-motor modules provide an elementary cognitive layer by which to couple the environment with the high-level cognition layer, including declarative memory and production memory (Byrne and Anderson, 2001). Perceptual-motor modules embedded in ACT-R 6.0 was heavily influenced by Kieras and Meyer's EPIC system (1996). Their major difference is that, only one production rule fires each time in ACT-R, while EPIC allows multiple rules fire, a parallel cognitive processing.

The motor module in ACT-R 6.0 is developed based on EPIC's manual motor processor (module). It is designed for modeling a simulated hand to operate a virtual keyboard and mouse (Bothell, n.d.). The motor module "receives commands from the production system that specify a movement style (e.g., PUNCH, as in punch a key) and the parameters necessary to execute that movement (e.g., LEFT hand and INDEX finger)" (Byrne and Anderson, 2001). The movement is generated through three phases: preparation, initiation, and execution. Below we describe each phase in turn.

In the preparation phase, the motor module builds a list of "features" which guide the actual movement; the features include the movement's style and parameters. For an example, in the movement of "punch the key below the left index finger", three features of PUNCH, LEFT, and INDEX are involved in the preparation. (Byrne and Anderson, 2001). The amount of time that preparation takes depends on the number of features that need to be prepared—the more that need to be prepared, the longer it takes.

The motor module maintains a history of the last set of features that it prepared. The actual number of features that need to be prepared depends upon two things: the complexity of the movement to be made and the difference between that movement and the previous movement. On one end of the scale, the motor module is simply repeating the previous movement, then all the relevant features will already be prepared and do not require preparation. On the other end, a request could specify a movement that requires the preparation of full features because which have not been made in previous movements.

By default, the first 50ms after the preparation is movement initiation (Bothell, n.d.). After that, the movement may be executed if the motor module is not already executing a movement (Byrne and Anderson, 2001). "If a movement is currently being executed, then the newly prepared movement will be queued and will not be executed until the current movement and all other movements in the queue have been executed" (Byrne and Anderson, 2001).

In ACT-R, "[t]he world with which a model interacts is called the device. The device defines the operations which the perceptual modules can use for gathering information and the operators available to the motor modules for manipulating the device." (Bothell, n.d.). The executed movement sent out from the ACT-R motor module is passed to the related device module in order to carry out the execution in the real world.

There is typically no direct communication between the perceptual and motor modules in ACT-R[3]. The data passed to the motor module always comes from the high-level declarative or production memory. This is almost the only significant conceptual difference between LIDA's SMS (see Section 2.9) and ACT-R's motor module.

3. There is only very limited direct connectivity between perceptual and motor modules. Spatial information in particular is communicated directly.

2.3 BECCA

This subsection heavily relies on a paper by Rohrer (2012). We cite it for the whole subsection unless we explicitly cite or quote otherwise.

A Brain-Emulating Cognition and Control Architecture (BECCA) is developed to address the problem of natural world interaction (NWI). NWI is the set of all tasks by which an agent pursues its goal in an unstructured physical environment. BECCA consists of an automatic feature creator and a model-based reinforcement learner to capture structure in the environment and to maximize rewards respectively. BECCA issues action commands to a world module and receives back a reward signal and observations in the form of sensory input and basic features. The world module is not the part of the standard BECCA architecture. Rather, it maintains simulations of the world, agent embodiment, actuators, and so on (see details later in this section).

The feature creator identifies patterns in the input. Sensory inputs are formed into groups based on how often they are co-active, and patterns within each group are identified as features and added to a feature space. The creator also maps the input into that feature space at each time step; the strongest feature voted by the projected input is activated. Features can be built hierarchically into higher-level features. Low-level features progressively activate high-level features, and the final set of activated features is passed to the reinforcement learner as a feature simulation.

In the reinforcement learner, a feature activity vector maintains the recent incoming feature simulations. A salience filter selects a single feature from the vector for attention[4], and the working memory maintains a brief history of attended features.

The reinforcement learner forms a model of the world and uses that model to select actions that will maximize the amount of reward. “The model consists of a list of feature-space transitions in the form of cause-effect pairs, each with an associated count and reward value. At each time step, the previous working memory is compared to the list of causes and the attended feature is compared to the list of effects. If a similar pair exists within the model, its count is incremented, and its reward value is adjusted toward the current reward. If there isn’t a sufficiently similar pair, the previous working memory and attended feature are added as a new cause-effect pair” (Rohrer, 2012). In this way, the model is formed and updated. “The count associated with each transition establishes its frequency of observation, and the reward value represents the expected reward associated with making that transition” (Rohrer, 2012).

In the feature space, a transition represents a path segment that, when linked to other segments, may take a BECCA agent to its desired state from the current one. To predict the likely effect of a transition, the agent ranks the expected transition by 1) the matching strength between the current state and the cause of the transition, and 2) the count of the transition, and selects the transition with the highest rank.

Many effects are conditional on the actions selected by the agent. If an expected transition has both high similarity to the current state and high reward, and involves an agent’s action on it, that action will be selected and executed.

There are two types of action selection in BECCA, deliberative and reactive. Their major difference regards the content of the current state that is used in the prediction of transitions and in the action selection. In deliberative action selection, only the working memory (the recently attended features) is used to seed predictions and action selections from the model, while the entire feature activity vector is used in reactive action selection. The final selected action results

4. From the viewpoint of LIDA (see Section 2.9), this selection acts as the beginning of an agent’s consciousness, and the most salient feature is the consciousness content.

from a nonlinear sum of the actions selected by the two selection modes. The deliberative action is also fed back to the working memory so that the action's effect can be recorded as part of the previous working memory content when the model is trained on the following time step.

A world module maintains 1) the environment—the simulation of the real world with which humans interact; 2) the physical embodiment of the agent—the virtual actuators and sensors of the hardware and the mechanisms that couple them; 3) preprocessing between sensors/actuators and the BECCA agent; and 4) a reward calculator providing reward value to the model of reinforcement learner[5]. From the viewpoint of action, a preprocessing step occurs between the selected action and the actuators; this step may include the incorporation of coordinated multi-actuator motions, fixed motion primitives, and heuristic goal pursuit subroutines.

An action processing mechanism unique to BECCA is its two-step action selection process: 1) Certain transitions, cause-effect pairs, are predicted (selected), and 2) an action is selected from the predicted transitions if the effect of the transition with the highest expectation relies on that action. These three components, cause, action, and effect, may be represented and organized differently in other architecture. For examples, in many other architectures such as ACT-R and Soar (see Section 2.10), a production rule is used to represent a condition-action pair, corresponding to cause and action in BECCA; while in LIDA (see Section 2.9), all of three components are encapsulated together: a data structure called a scheme includes context, action, and result, corresponding to BECCA's cause, action, and effect respectively.

Another issue is that the action execution process is outside the standard BECCA architecture. The commands for controlling actuators are not generated by the BECCA agent, but rather by the preprocessing of a world module. Also, the world module maintains the domain knowledge; this allows the BECCA agent to remain unchanged between many different domains (tasks). A similar strategy is implemented in Soar as well: its operator application doesn't really perform the action on actuators but an external program is always necessary to handle the final execution (performance). In contrast, some architectures do involve the action execution process in their architecture, such as ACT-R and LIDA.

2.4 CERA-CRANIUM

We cite a paper (Arrabales, Ledezma, and Sanchis, 2009) for this entire subsection of CERA-CRANIUM. Other citations or quotations will be explicitly noted in the text.

The Conscious and Emotional Reasoning Architecture (CERA) is a cognitive architecture structured in layers, providing a flexible framework with which to integrate different cognitive models of consciousness. CERA offers a basic hypothesis that conscious contents emerge as a result of competition and collaboration between specialized processors (functions). The Cognitive Robotics Architecture Neurologically Inspired Underlying Manager (CRANIUM) provides services through which CERA can execute thousands of asynchronous but coordinated concurrent processes.

CERA is structured in four layers, which are as follows:

1) The sensory-motor services layer comprises a set of communication services that provide a uniform access interface for the agent's physical sensors and actuators.

5. A BECCA agent is not autonomous because its action decision is not driven by an agenda or motivation inside but by artificial rewards created outside of the agent.

2) The physical layer is responsible for the low-level representations and preparations of the agent's sensors and actuators. Actuator commands are finally regulated at this level.

3) The mission-specific layer maintains complex perceptions and behaviors that are combined from and decomposed to the single sensory-motor content. Complex behaviors represent the agent's missions; one mission typically involves several goals.

4) The core layer includes a set of modules that perform higher cognitive functions. CERA is designed to allow customized core modules, such as attention, preconscious management, memory management, and self-coordination.

A CRANIUM workspace implements a set of specialized processors and a shared access working memory for the processors. Each of these processors is designed to perform a specific function, cooperating and competing with other functions. A CERA agent has two hierarchically arranged CRANIUM workspaces. The low-level workspace is located in the CERA physical layer and the high-level is located in the CERA mission specific layer. Based on the two CRANIUM workspaces, the perception flows are organized bottom-up in packages called single percepts, complex percepts, and mission percepts. Meanwhile in the same workspaces, a top-down action[6] flow includes mission behaviors, simple behaviors, and single actions; behaviors are iteratively decomposed until a sequence of atomic actions is obtained.

The core layer's operations are problem domain dependent. The operations are directed by the problem's interests, meta-goals, instead of mission-specific goals coming from lower layers. Meta-goals shape the overall resulting behaviors. At any given time, a number of possible behaviors are generated in the CRANIUM workspaces; however, only those behaviors that are directed to the same locations as the represented meta-goals are likely to be selected and finally executed.

There are three types of processors related to the action process implemented in the CRANIUM workspaces.

1) Action planners transform the input behavior into the corresponding sequence of atomic actions that are submitted for eventual execution, so as to achieve the behavior's missions.

2) Action preprocessors prepare the atomic actions generated from action planners. Action preprocessors build so-called "single action constructs" to provide specific contextual data for actions. "Proprioceptive sensory data is also included in order to adapt actions to the current position of the actuators" (Arrabales et al., 2009).

3) Reactive processors are typically located in the CERA physical layer. They provide a quick response to stimuli that are considered harmful or highly undesired for the agent. These processors build simple behaviors to diminish or prevent negative consequences when unsafe or undesired situations are detected, without the participation of upper cognitive processes.

6. The term "action" is an abstract concept; it refers to a class of action-kind concepts organized in different levels, including mission behaviors, simple behaviors, and single actions.

In summary, the CERA-CRANIUM action process supports both action selection—the selection between behaviors driven by the domain independent meta-goals—and the action execution—the decomposition from a behavior to a sequence of atomic actions, implemented by the action planners and the final execution preparation in the physical layer. CERA-CRANIUM also establishes the interface between the agent and its environment in the sensory-motor services layer, providing the agent with the necessary environmental specifications.

2.5 CLARION

CLARION stands for Connectionist Learning with Adaptive Rule Induction ON-line. The purpose of this architecture is to capture all the essential cognitive processes within an individual cognitive agent (Sun, 2003, 2006). CLARION consists of a number of subsystems, including the action-centered subsystem (the ACS), the non-action-centered subsystem (the NACS), the motivational subsystem (the MS), and the metacognitive subsystem (the MCS). The ACS implements the action decision making of an individual cognitive agent (Sun, 2003). The MS motivates an agent to choose its actions by means of the rewards or gains which the agent seeks to maximize. The MS influences the working of the ACS by providing the context in which the goal and the rewards of the ACS are set (Sun, 2006).

The ACS consists of two levels of representation: the top level for explicit and the bottom level for implicit knowledge. The implicit knowledge generally does not have associated semantic labels, and is less accessible. Accessibility refers to the direct and immediate availability of mental content to the major operations that act on it. The bottom level is a direct mapping from perceptual information to actions, implemented in backpropagation neural networks[7] involving distributed representations, whose representational units in the hidden layer are capable of accomplishing tasks, but are generally not individually meaningful (Sun, 2003). Furthermore, the backpropagation neural network in the ACS has the potential for multiple instances, and a selection process is proposed for the backpropagation neural network. In contrast, the explicit knowledge is more accessible, manipulable, and has conceptual meaning (Sun, 2006). At the top level, a number of explicit action rules are stored, which are usually in the following form: current-state-condition $\rightarrow$ action (Sun, 2003). An agent can select an action in a given state by choosing an applicable rule. The output of a rule is an action recommendation, which is similar to the output from the bottom level (Sun, 2003).

The two levels implemented in CLARION's ACS operate independently: each of them makes action decisions based on the current state in parallel. The action sent out from both top and bottom levels are all performable. The final output action of the ACS is a combination of the output actions from the top and bottom levels. On the other hand, CLARION also models an interaction between the top and bottom levels, as well as between explicit and implicit knowledge. The input state or the output action to the bottom level is structured using a number of input or action dimensions; each of the dimensions has a number of possible values. At CLARION's top level, an action rule's condition or action is represented as a –high-level node which is connected to all the specified dimensional values of the inputs or actions at the bottom level (Sun, 2003).

The overall algorithm of CLARION's action decision making consists of a structure that goes from perception to actions, and ties them together through the top and bottom levels of the ACS's

7. Learning of implicit knowledge (the backpropagation network) transpires at the bottom level. "In this learning setting, there is no need for external teachers providing desired input/output mappings. This (implicit) learning method may be cognitively justified" (Sun, 2006).

cognitive processes as follows: “Observing the current state of the world, the two levels of processes within the ACS (implicit and explicit) make their separate decisions in accordance with their own knowledge, and their outcomes are somehow ‘combined’. Thus, a final selection of an action is made and the action is then performed” (Sun, 2003). This decision making mechanism covers both action selection (preparation) and action execution. The top level of the mechanism provides the agent’s internal goal for action execution, and the bottom level provides real-time environmental information. Note that the specifics of the agent’s actuators are not involved in the representation of output actions (motor commands). This means the output performable actions of CLARION are independent of the motors of the robot’s actuators; this is different than the executable motor commands mentioned in the introduction.

Additionally, learning has been applied in CLARION’s ACS in three distinct ways: 1) the learning of implicit knowledge at the bottom level; 2) bottom-up learning, or learning explicit knowledge at the top level by utilizing implicit knowledge acquired in the bottom level; and 3) top-down learning, the assimilation at the bottom level of explicit knowledge from the top level (which must have previously been established through bottom-up learning) (Sun, 2003).

2.6 EPIC

This review of EPIC mainly relies on an EPIC overview paper (Kieras and Meyer, 1997). We cite the paper for this entire subsection, unless explicit citations or quotations are claimed in the text.

Executive Process-Interactive Control (EPIC) is a cognitive architecture created for modeling human task performance. EPIC has three processing modules: 1) sensory processors, 2) motor processors, and 3) a cognitive processor that represents a general procedure as a set of production rules to perform a complex multimodal task. During the execution of a procedure, EPIC specifies both the production-rule programming for the cognitive processor as well as the relevant perceptual and motor processing parameters.

Specifically, there are visual and auditory processors that accept multimodal stimuli as perceptual input. This input is stored in the corresponding working memory located in the cognitive processor.

The motor processors produce a variety of simulated movements for the hands, eyes, and vocal organs. From the cognitive processor, an action’s command is sent to a motor processor that consists of the movement type (name) and certain parameters.

There are two steps for a complete movement: a preparation and an execution. In the preparation, the motor processor transforms the movement type (name) into a set of movement features and generates them. “The time to generate the features depends on how many features can be reused from the previous movements (repeated movements can be initiated sooner), and how many features have been generated in advance” (Kieras and Meyer, 1997). The ensuing execution step begins with an initiation phase, followed by the actual physical movement. In addition to reusing the features remaining from previously executed movements, the movement features may be prepared in advance. “If the task permits the movement to be anticipated, the cognitive processor can command the motor processor to prepare the movement in advance by generating all of the required features and saving them in motor memory” (Kieras and Meyer, 1997).

Rather than the voluntary movements produced by various motor processors, as mentioned above, the oculomotor processor may produce the involuntary (reflexive) eye movements, either saccades or small smooth adjustments in response to the visual situation.

In the cognitive processor, there is a set of production rules that specify which actions are performed in certain situations to accomplish a task. The format for a rule is <rule-name> IF <condition> THEN <action>. The rule condition will test the contents of the production system working memory. The rule action will then add or remove information from the working memory, or send a command to the motor processors. Motor working memory stores information about the current state of the motor processors.

The cognitive processor operates cyclically. During each cycle, the contents of working memory are first updated with the perceptual input information and the previous cycle's modifications; then the contents of the production system working memory are updated based on the rules that fire, and the action commands of the firing rules are sent to the motor processors. A unique feature of EPIC is that it will fire all rules in which conditions match the contents of working memory and will execute all of the corresponding actions; in other words, the EPIC cognitive processor allows parallel cognitive processing.

The EPIC model builder should provide 1) the task environment, either physical or simulated, which includes the characteristics of relevant objects external to an EPIC agent, 2) a set of tasks which specify the environmental events, 3) task-specific sensory data encodings (representations), and 4) the task procedures represented as production-rules.

In summary, regarding the action process in EPIC, action selection and action execution have been implemented by production rules firing in the cognitive processor, and movement preparation and execution in the motor processors separately.

2.7 GLAIR

We cite a paper (Shapiro and Bona, 2010) for this subsection, unless other citations or quotations are explicitly mentioned in the text.

Grounded Layered Architecture with Integrated Reasoning (GLAIR) is a multi-layered cognitive architecture for embodied agents. In GLAIR, the highest layer is the Knowledge Layer (KL), which contains the agent's beliefs, and performs reasoning and selects acts. The middle layer is the Perceptuo-Motor Layer (PML), which grounds the KL symbols in perceptual structures and primitive actions. The lowest layer is the Sensori-Actuator Layer (SAL), which contains the controllers of the sensors and actuators of the hardware or software agent.

The KL contains the beliefs of the agent; with respect to action, it includes 1) plans for carrying out complex acts and for achieving goals, 2) beliefs about the preconditions and effects of acts, and 3) policies about when, and under what circumstances, acts should be performed.

The PML is responsible for the communication between the KL and the SAL by three top-down sub layers: the PMLa, the PMLb, and the PMLc. The PMLa grounds the KL symbols, providing primitive actions; the PMLc abstracts the sensors and actuators into basic behavioral repertoire of the robot. The PMLb translates and communicates between the PMLa and the PMLc.

GLAIR agents execute a sense-reason-act cycle. The original focus of the GLAIR design is on reasoning, but not problem solving or goal-achievement such as ACT-R. Its basic driver is based on reasoning: either thinking about some perceptual input, or answering some question. If the input (typically a natural language utterance) is a statement or a question, the GLAIR agent will output the proposition of the statement or the answer to the question respectively. A later added acting component allows a GLAIR agent to obey a command, to perform an act and to achieve a goal. When the input is a command, the agent will perform the indicated act, implementing an action process that is the focus of this review.

An act consists of an action and one or more arguments. For an example, “the term find (Bill) denotes the act of finding Bill (by looking around in a room for him), composed of the action find and the object Bill” (Shapiro and Bona, 2010). Acts may be classified as either external, mental, or control. External acts affect the outside world. Mental acts affect the agent’s beliefs and policies. Control acts are the control structures used to support ground computational processes such as inference operations so as to maintain the GLAIR acting system.

GLAIR acts may also be classified as primitive, defined, or composite. Primitive acts are the basic acts predefined in the PMLa. Composite acts consist of primitive acts. A defined act is the abstracted identifier of a plan; if a GLAIR agent is to perform a defined act, it “deduces it” to a plan and performs it. Such a plan is an act, which can be either a primitive, composite, or defined. It is assumed that a plan is “closer” to primitive acts than a defined act. A defined act may have different plans depending on circumstances. The use of conditional plans has allowed a GLAIR agent to select among alternative procedures to perform.

The procedure for performing an act consists of several steps:

1) To attempt to achieve the preconditions of the act and, if it is a defined act, to prepare a set of candidate plans that can be used to perform the act.

2) If the act is a defined act, only its most suitable plan is tried, after which the agent will automatically consider it successful.

3) Effects of the act are derived before the act is performed; and after that, the agent will consider all the effects of the act to hold.

In GLAIR, the act is represented in the same formalism as other declarative knowledge such as the agent’s beliefs. However, the declarative knowledge and the act are maintained by the KL and the PMLa layers separately. In this way, the declarative and procedure knowledge are represented with the same formalism but operated in different levels.

The PMLa layer grounds the KL by providing primitive actions, transforming high-level actions to low-level. Although the actions maintained in PMLa are primitive, they are independent of the implementation of the agent’s body. It is the PMLc layer which directly abstracts the actuators[8] of the robot into the basic behavioral repertoire of the robot body. The primitive actions in the PMLa are translated to these basic behavioral repertoires—the basic execution units of GLAIR—through PMLb.

Two steps occur during the process of action execution: 1) actions are initially selected in the KL driven by reasoning results and translated into their primitive format in the PMLa layer; and 2) the primitive actions are translated to actuator-dependent basic behavioral repertoires in PMLc through PMLb, and then those basic units are sent to SAL for execution.

2.8 ICARUS

We cite a paper (Langley and Choi, 2006) for this entire subsection. Other citations or quotations will be notated explicitly.

ICARUS is a cognitive architecture for physical agents that has been influenced by results from cognitive psychology. ICARUS’s most basic mechanism, conceptual inference, operates by

8. In the original paper (Shapiro and Bona, 2010), the authors use the term “effectors” instead of “actuators” as we do here.

matching long-term conceptual structures against short-term perceptual data and beliefs. Based on the inference, ICARUS operates processes for goal selection and skill execution.

In order to perceive the states of the external environment, ICARUS incorporates a perceptual buffer (short-term memory) that describes aspects of the environment. The element stored in this memory responds to a particular object, and characterizes the object's specifications at the current time step. ICARUS also includes a conceptual memory, which contains long-term structures that are the classifications of the environmental state. During each cycle of conceptual inference, objects are perceived first into the perceptual buffer, where they begin to match against long-term conceptual classifications. The system updates its belief memory based on the results of this matching. The elements in the belief memory describe relations among objects. ICARUS repeats this inference process, updating its beliefs about the environment over time.

In order to take action in the environment, ICARUS has a performance mechanism that concerns goals the agent wants to achieve, skills the agent can execute to reach them, and intentions about which skills to pursue.

Specifically, ICARUS includes a goal memory that contains a list of agent's objectives. The goal is a set of concept instances that the agent wants to achieve, and the goal memory takes the same form as belief memory. An agent needs to select only one goal at a time among multiple elements in goal memory. On each time, it chooses the goal with the highest priority that is not yet achieved.

ICARUS has a long-term skill memory that contains skills it can execute in the environment and use to accomplish goals. Each skill has a head and a body. The head states the skill's objective, and the body specifies the necessary perceptual data and beliefs of a skill. Multiple skills may have the same head; they provide different ways to achieve the same goal under different conditions. Once the agent has chosen a goal, it selects a skill to achieve the goal based on a matching between the skill body's specifications and the agent's current perceptual data and beliefs[9].

The skills are organized hierarchically. "Primitive skills" refers to actions that the agent can execute directly in the environment, while non-primitive skills are goals/sub-goals that the agent might seek to achieve. Primitive skills correspond to the executable motor commands mentioned in the introduction. During the execution of a non-primitive skill, the agent must find a path downward from its goal to one or more terminal primitive skills in the hierarchy. Once the agent has selected a skill path for execution, it invokes the actions referred to the primitive skill or skills in the path.

If the applicable skill is not found, an impasse appears, and the agent invokes its problem solver for achieving the goal. The agent decomposes the goal into sub-goals iteratively until find the skills for achieving them. The skills applicable for all sub-goals are finally selected and then executed to achieve the goal. A new skill is learned for the goal by structuring the applicable skills for those sub-goals. The similar impasse resolving mechanism has been implemented in Soar (Section 2.10) as well.

2.9 LIDA

For historical reasons LIDA stands for Learning Intelligent Distribution Agent. The LIDA Model (Franklin et al., 2014) is a conceptual, systems-level model of human mental processes, used to

9. In LIDA (see Section 2.9), a similar matching occurs in the process of recruiting schemes. Schemes are selected based on a matching between the agent's conscious contents, the most salient current situation, and the scheme's context and result contents.

develop biologically-inspired intelligent software agents and robots. It implements and fleshes out a number of psychological and neuropsychological theories, but is primarily based on Global Workspace Theory (Baars, 1988, 2002).

The LIDA model is grounded in the LIDA cognitive cycle. Each cognitive cycle consists of three phases: 1) the LIDA agent first senses the environment, recognizes objects, and builds its understanding of the current situation; 2) by a competitive process, as specified by Global Workspace Theory (Baars, 1988), it then decides what portion of the represented situation should be attended to, and broadcast to the rest of the system; 3) finally, the broadcast portion of the situation supplies information allowing the agent to choose an appropriate action to execute, and modulates learning (Franklin et al., 2014). The simulated human mind can be viewed as functioning via a continual, overlapping sequence of these cycles.

The dual aspects of action are represented in the LIDA Model as the distinct processes of action selection and action execution. Specifically, the sensory data retrieved in LIDA influences the action process at two "levels". At one level, sensory data is filtered through the understanding and attention phases, and then helps recruit appropriate actions in the action selection process; the selected result is used to initiate certain processes operating in the concomitant action execution process, ultimately generating executable low-level actions. At the other level, the sensory data is sent through a dorsal stream channel[10] directly to the action execution process for assisting the execution.

In LIDA's action selection process, one or more schemes are recruited first based on the most salient current situation—a scheme is a data structure representing the procedure knowledge stored in LIDA's Procedural Memory. It is comprised of three components: a context, an action[11], and a result. With some reliability, the result is expected to occur when the action is taken in its context—and then, the schemes' context and result components are bound with additional information of the current situation, so that the recruited schemes are instantiated into behaviors. A behavior has a data structure similar to a scheme, but the components of context and result have been instantiated with concrete values. Finally, a behavior is selected based on the agent's motivation and its understanding of the current situation.

The process of action execution has been recently added to LIDA, modeled by the Sensory Motor System (SMS) (Dong and Franklin, 2014). Two other LIDA modules, Action Selection and Sensory Memory, provide relevant information—a selected behavior and the sensory data through a dorsal stream channel, respectively–as inputs to the SMS. The SMS sends out motor commands to an agent's actuators to execute its selected action in the environment. Within the SMS, three data structure types have been proposed—the motor command (MC), the motor plan (MP), and the motor plan template (MPT)—and three types of processes have been modeled: online control, specification, and MPT selection.

A motor command (MC) is applied to an agent's actuator. Every MC has two components: a motor name, and a command value. The motor name indicates which motor of an actuator the MC specifically controls, while the command value of a MC encodes the extent of the command applied to the motor.

An MP acts like a MC generator that generates MCs based on the sensory data transmitted via the dorsal stream. An MP is implemented based on the principles of the subsumption

10. In LIDA, the dorsal stream channel directly passes sensory data from the sensory memory to the action execution process.

11. In this context, the term "action" refers to a component of a scheme. This differs from the general usage, such as in the phrase "action execution". In this paper, we use "action" in the general sense, while "action of a scheme" refers to a particular component of that scheme.

architecture (Brooks, 1991), a reactive structure. In the subsumption architecture, 1) the sensory data is linked to directly thus determining the selection of motor commands that drive the actuators; 2) it decomposes a robot's control architecture into a set of task-achieving behaviors; and 3) it does not maintain any internal model of the world[12], and is without any explicit representations. The MP generates motor commands as the output of the SMS to the environment (using actuators), while environmental data directly influence the generation process through the dorsal stream channel from Sensory Memory. These cyclically occurring processes are called the online control process of the SMS.

An MPT is an abstract MP that resides in an agent's long-term memory (Sensory Motor Memory in LIDA). It has a set of motor commands (MCs) that are not yet bound with the command values, whereas after a specification process, the motor commands are bound with specific values, instantiating the MPT into a concrete MP. Both sensory data from the dorsal stream and the selected behavior determine the specification process (Dong and Franklin, 2014). MPTs and MPs have very similar structures, so they are designed with nearly the same data structure. Their major differences are 1) an MPT is persistently stored in a long-term memory, while an MP is short-term, and created anew each time it is used; and 2) typically an MP's command values have been specified, while those of an MPT have not.

As the SMS's initial process, A MPT selection acts to select and initiate a MPT by an incoming selected behavior before the MPT is specified into a concrete motor plan. MPT selection chooses one MPT from the set of those associated with the selected behavior. It connects action selection to action execution. Currently this selection is built in by the agent designer (Dong and Franklin, 2014).

2.10 Soar

Soar is a cognitive architecture in pursuit of general intelligent agents (J. E. Laird, 2008). "The design of Soar is based on the hypothesis that all deliberate goal-oriented behavior can be cast as the selection and application of operators to a state. A state is a representation of the current problem-solving situation; an operator transforms a state (makes changes to the representation); and a goal is a desired outcome of the problem-solving activity" (J. E. Laird, Congdon, Coulter, Derbinsky, and Xu, 2012).

Soar has separate memories for descriptions of its current situation and its long-term knowledge. It represents the current situation in its working memory, which is Soar's short-term memory and maintains the sensory data, results of intermediate inferences, active goals, and active operators (J. E. Laird et al., 2012). The long-term knowledge specifies how to respond to different situations in the working memory so as to solve a specific problem.

All of Soar's long-term knowledge is organized around the functions of operator selection and operator application, which are organized as a processing cycle as described below (J. E. Laird, 2008; J. E. Laird et al., 2012).

1) Elaboration. Knowledge with which to compute entailments of short-term memory, creating new descriptions of the current situation that can affect operator selection and application indirectly.

12. Although no central world state is one of the essences of the subsumption architecture, implicit understanding and expectation of the environment has been built into the architecture by its layered structure.

2) Operator Proposal. Knowledge with which to propose operators that are appropriate to the current situation based on features of the situation tested in the condition of the production rules.

3) Operator Comparison (Evaluation). Knowledge of how to compare candidate operators, to create preferences for some proposed operators based on the current situation and goal.

4) Operator Selection. Knowledge with which to select an operator based on the comparisons. "If the preferences are insufficient for making a decision, an impasse arises and Soar automatically creates a substate in which the goal is to resolve that impasse. … The impasses and resulting substates provide a mechanism for Soar to deliberately perform any of the functions (elaboration, proposal, evaluation, application) that are performed automatically/reactively with rules." (J. E. Laird, 2008)

5) Operator Application. Knowledge of how the actions of an operator are performed on the environment, to modify the state.

Four of the above functions require retrieving long-term knowledge that is relevant to the current situation: Elaborating, Operator Proposal, Operator Comparison, and Operator Application. These functions are driven by the knowledge represented as production rules (J. E. Laird et al., 2012). A production rule has a set of conditions and a set of actions. The production's actions are performed if its conditions match working memory; that is, the production fires (J. E. Laird et al., 2012). The other function, Operator Selection, is performed by Soar's decision procedure, which is a fixed procedure that makes a decision upon the knowledge that has been retrieved (J. E. Laird et al., 2012).

An operator contains preconditions and actions; its action differs from a production rule's action. The operator action is an output for the agent to its internal or external environment, while actions of a production rule generally either create preferences for operator selection, or create/remove working memory elements (J. E. Laird et al., 2012).

When Soar interacts with the environment, it must make use of a mechanism that allow it to effect changes in that environment; the mechanism provided in Soar is called output functions (J. E. Laird et al., 2012). During the operator application process, Soar productions could respond to an operator by creating a structure on the output link, a substructure which represents motor commands for manipulating output. Then, an output function would look for specific motor command in this output link and translate this into the format required by the external program that controls the agent's actuators (J. E. Laird et al., 2012). "[In the external program,] functions that execute motor commands in the environment use the values on the output links to determine when and how they should execute an action" (J. E. Laird et al., 2012). This means that it is the Soar's external program, not its output functions, that specifies how to execute the action in detail (when and how).

In the case of Soar's output, motor commands, which cannot be directly performed (executed) on the external world, an external program is always necessary to handle the final "real" execution (performance) for Soar. Soar does not cover the representation of environmental information related to action. This allows it to maintain generality with a clear standard, without the necessity of considering every possible domain that the Soar agent might live in. Note the term "motor commands" in Soar expresses completely different concepts than in other architectures, such as LIDA, although it is used to represent the final output data in both cases. In

LIDA, motor commands are executable, while in Soar they are not. By saying that motor commands are "executable", we mean that these commands 1) are able to be applied to the agent's actuators directly, and 2) are maintained in an order appropriate to both the agent's internal goal and the current environment's dynamics.

3. Conclusions

We realize that the action execution processes implemented in the above cognitive architectures have many similar representations and procedures though they use different structures. We conclude with some general observations regarding the nature of these representations and procedures, followed by a summary.

3.1 Inside Action Execution

Each cognitive architecture having a representation at an explicit level of knowledge, typically also needs a process that transforms high-level knowledge into motor-level commands; that is, action execution. For example, a task (sub task) is transformed into task commands in 4D/RCS, a production rule's action into movements in ACT-R and EPIC, a behavior into atomic actions in CERA-CRANIUM, an act into basic behavioral repertoires (the basic execution units) in GLAIR, a non-primitive skill into primitive skills in ICARUS, and a behavior into a sequence of motor commands in LIDA. Some other architectures, such as BECCA and Soar, prepare the actions for external programs to finish the action execution. These architectures accomplish only the initial phase of the action execution process. CLARION has a unique action decision mechanism in its two levels of representation. As we have discussed in Section 2.5, this mechanism covers both action selection (preparation) and action execution (performance), though its action performance does not maintain the specifics of the actuators within the representations of the output actions.

3.2 The Cooperation between Action Selection and Execution

A goal-directed action resulting from action selection concomitantly initiates the initial action execution process. This process may be implemented in two different ways. One possibility is to decompose the selected goal-directed action into primitive actions, in which case the action's data structure is gradually broken down from high-level to low-level without qualitative changes, such as tasks in 4D/RCS, behaviors and atomic actions in CERA-CRANIUM, actions in CLARION, skills in ICARUS, and operators in Soar. The other option is to map the selected goal-directed action to another type of action representation—the action's data structure has been qualitatively changed—that enables the generation of the ensuing low-level actions, such as a production rule's action mapping to a movement style in ACT-R and EPIC, a transition mapping to an action in BECCA, and a behavior mapping to a Motor Plan Template (MPT) in LIDA. GLAIR combines the natures of these two options: it first decomposes acts into primitive ones, and then translates them into actuator-dependent basic behavioral repertoires.

3.3 Environmental Information for Action Execution

During the action execution process, additional environmental information is usually supplied to specify and adjust the final command values for execution. For example, in both ACT-R and EPIC, a preparation phase operates during the action execution process to build and specify a list of "features"; the features include the movement's style—the name of a low-level action's identifier—and the values of its parameters. In CERA-CRANIUM's action execution, action preprocessors provide specific contextual data for preparing atomic actions. In LIDA, the sensory data sensed through its dorsal stream channel is sent directly to the action execution process, so that a Motor Plan Template (MPT) is instantiated into a Motor Plan (MP) that generates the final motor commands.

This additional information might be directly sensed from the environment through sensory processes; in this case, a direct communication between the perceptual and motor modules is implemented to assist the action execution. For example, in CLARION's bottom level, perceptual information directly maps to actions, and in LIDA, sensory data may be sent to the motor system directly through a dorsal stream channel. On the other hand, this environmental information might come from high-level cognitive modules that store the current state of the environment. For example in ACT-R, the data passed to the motor module comes from high-level declarative or production memory.

3.4 Summary

Based on the above review, we can identify certain common characteristics for action execution. It provides an elementary cognitive layer by which a cognitive architecture couples the environment with its high-level cognitive modules, including action selection. Action execution finalizes an agent's intention, so that it generates a cognitive architecture's output. Action execution involves domain specifications—including environmental specifications and those of the agent's actuators—sufficient to enable execution to be actuated in the environment. However, it does not contain so much abstract knowledge, which is the province of high-level cognitive modules.

On the other hand, action execution may vary in concrete implementations, with respect to its representations, procedures, and the means of cooperation with high-level cognitive modules. Also, action execution itself may or may not be considered a standard module of a cognitive architecture, so that architectures may differ in their degree of completion.

References

ACT-R 6.0 Tutorial. 2012. Unpublished manuscript. Retrieved from http://act-r.psy.cmu.edu/wordpress/wp-content/themes/ACT-R/actr6/actr6.zip

Albus, J. S., and Barbera, A. J. 2005. RCS: A cognitive architecture for intelligent multi-agent systems. *Annual Reviews in Control, 29*(1):87-99.

Anderson, J. 2007. *How can the human mind occur in the physical universe?* : Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., and Qin, Y. 2004. An integrated theory of the mind. *Psychological Review, 111*(4):1036-1060.

Arrabales, R., Ledezma, A., and Sanchis, A. 2009. *CERA-CRANIUM: A test bed for machine consciousness research*. International Workshop on Machine Consciousness.

Baars, B. J. 1988. *A cognitive theory of consciousness*. New York: Cambridge University Press.

Baars, B. J. 2002. The conscious access hypothesis: origins and recent evidence. *Trends in cognitive sciences, 6*(1):47-52.

Bothell, D. n.d. *ACT-R 6.0 Reference Manual (Working Draft)*. Retrieved from http://act-r.psy.cmu.edu/wordpress/wp-content/themes/ACT-R/actr6/reference-manual.pdf

Brooks, R. A. 1991. How to build complete creatures rather than isolated cognitive simulators. *Architectures for Intelligence: The Twenty-second Carnegie Mellon Symposium on Cognition*:225-239.

Budiu, R. 2013. ACT-R Website. from http://act-r.psy.cmu.edu/

Byrne, M. D., and Anderson, J. R. 2001. Serial modules in parallel: The psychological refractory period and perfect time-sharing. *Psychological Review, 108*(4):847-869.

Castiello, U. 2005. The neuroscience of grasping. *Nature Reviews Neuroscience, 6*(9):726-736.

Dickmanns, E. D. 1992. A general dynamic vision architecture for UGV and UAV. *Applied Intelligence, 2*(3):251-270.

Dickmanns, E. D. 2000. *An expectation-based, multi-focal, saccadic (EMS) vision system for vehicle guidance.* Paper presented at the International Symposium of Robotics Research (ISRR'99), 421-430, Snowbird Utah, USA.

Dong, D., and Franklin, S. 2014. *Sensory Motor System: Modeling the process of action execution.* Paper presented at the Proceedings of the 36th Annual Conference of the Cognitive Science Society, 2145-2150, Austin TX, USA.

Duch, W., Oentaryo, R. J., and Pasquier, M. 2008. *Cognitive Architectures: Where do we go from here?* Paper presented at the AGI, 122-136, Memphis TN, USA.

Franklin, S., and Graesser, A. 1997. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents *Intelligent agents III agent theories, architectures, and languages*, 21-35. London, UK: Springer-Verlag.

Franklin, S., Madl, T., D'Mello, S., and Snaider, J. 2014. LIDA: A Systems-level Architecture for Cognition, Emotion, and Learning. *IEEE Transactions on Autonomous Mental Development, 6*(1):19-41. doi: 10.1109/TAMD.2013.2277589

Goertzel, B., Lian, R., Arel, I., De Garis, H., and Chen, S. 2010. A world survey of artificial brain projects, Part II: Biologically inspired cognitive architectures. *Neurocomputing, 74*(1):30-49.

Goodale, M. A., and Milner, A. D. 1992. Separate visual pathways for perception and action. *Trends in neurosciences, 15*(1):20-25.

Grafton, S. T. 2010. The cognitive neuroscience of prehension: recent developments. *Experimental brain research, 204*(4):475-491.

Jeannerod, M. 2006. *Motor cognition: What actions tell the self.* Oxford, UK: Oxford University Press.

Kieras, D. E., and Meyer, D. E. 1996. The EPIC architecture: Principles of operation. *Unpublished manuscript from ftp://ftp.eecs.umich.edu/people/kieras/EPICarch.ps*.

Kieras, D. E., and Meyer, D. E. 1997. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-computer interaction, 12*(4):391-438.

Laird, J. 2012. *The Soar cognitive architecture*: MIT Press.

Laird, J. E. 2008. *Extending the Soar cognitive architecture.* Paper presented at the Artificial General Intelligence 2008, 224-235, Memphis TN, USA.

Laird, J. E., Congdon, C. B., Coulter, K. J., Derbinsky, N., and Xu, J. 2012. *The Soar User's Manual Version 9.3.2*. Computer Science and Engineering Department. University of Michigan. Unpublished manuscript.

Langley, P., and Choi, D. 2006. *A unified cognitive architecture for physical agents.* Paper presented at the Proceedings of the National Conference on Artificial Intelligence, 1469-1474, Boston MA, USA.

Langley, P., Laird, J. E., and Rogers, S. 2009. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research, 10*(2):141-160.

Milner, D., and Goodale, M. A. 2008. Two visual systems re-viewed. *Neuropsychologia, 46*(3):774-785.

Rohrer, B. 2012. *BECCA: Reintegrating AI for natural world interaction.* Paper presented at the AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI, Stanford California, USA.

Russell, S. J., and Norvig, P. 2009. *Artificial intelligence: a modern approach* (Third ed.): Prentice hall.

Searle, J. R. 1983. *Intentionality: An essay in the philosophy of mind*: Cambridge University Press.

Shapiro, S. C., and Bona, J. P. 2010. The GLAIR cognitive architecture. *International Journal of Machine Consciousness, 2*(2):307-332.

Sun, R. 2003. *A tutorial on CLARION 5.0*. Unpublished manuscript.

Sun, R. 2006. The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In R. Sun (Ed.), *Cognition and multi-agent interaction*, 79-99. New York: Cambridge University Press.