Cognitive Architectures And Their Potential For Applied Cognitive Psychology

Tuuli Pöllänen

Typically, papers on cognitive architectures appear to begin with a short account of their history, and the accounts for the history of cognitive architectures typically approach the theme from the first computational cognitive models (often the GPS, by Newell and Simon, 1963 – already ten years before Allen Newell's paper, 1973, discussed in detail in the following paragraphs), then forwarding the theme by explaining how newer models were based on the first ones, how they became more elaborate over time, introducing their applications and domains. Often, Allen Newell's 1973 paper is mentioned shortly, possibly even as a side note. However, I find that if the audience of the paper is not well-acquainted with the concept and uses of computational cognitive models (including cognitive architectures), Newell's (1973) critical vantage point is a very good start point for approaching the subject, as it explains what shortcomings of the experimental paradigm the first computational models attempted to mitigate.

In 1973, Allen Newell was given a task to comment on every paper appearing in W.G. Chase's (Ed.) symposium on visual information processing. Rather ambitiously, Newell decided to not only comment on every paper, but also to attempt to draw a conclusion that would give a holistic picture of visual processing – a decision he claims he soon regretted (Newell, 1973). After realizing the impossibility of the task, Newell (1973) titled his paper (which is now considered a well-read classic and which is quite an entertaining read – highly recommended) »You can't play twenty questions with nature and win«, and dove into the problems of cognitive psychology of his time (and largely what is still one of the key issues in cognitive psychology as well as AI in the more recent years).

Newell (1973) expressed his despair over how nearly all experimentalists of his time were dealing with their own little areas, performing what he thought were absolutely magnificent experiments and concluding interesting results, but driven by mutually exclusive binary distinctions (nature *or* nurture, parallel *or* serial, central *or* peripheral) and with very little interest in drawing together with other studies to create a thorough, holistic picture of human cognition, where processes could happen simultaneously on multiple systems level on several different ways, influenced and modified by several different factors. He pointed out that there were a few strands of theory that throve for some quantitative explanation over a class of phenomena, but they were not common

enough to »quiet his concern«. He argued that if something was not to change soon to provide the field a »path to clarity«, the situation was to get muddier and muddier. His key issue was stated as follows:

> »Science advances in playing twenty questions with nature. The *proper* tactic is to frame a general question, hopefully binary, that can be attacked experimentally. Having settled that bits-worth, one can proceed to the next. The policy appears optimal – one never risks much, there is feedback from nature at every step, and progress is inevitable. Unfortunately, the questions never seem to be really answered, the strategy does not seem to work.« (Newell, 1973, p. 290).

In his paper, Newell diagnoses the problems he found in an attempt to make a synthesis on the paper, and writes down two injunctions on how to approach cognitive experimentation:

1. Know the method your subject is using to perform the experimental task. Behavior is programmable. This means that it is under the control of the subject as means to his own ends. To predict behavior, you must know his goals, the structure of the task environment, and the invariant structure of his processing mechanisms. From these, you can predict what methods are available to the subject.
2. Never average over methods. Uncertainty of which method the subject uses takes substantial space in discussion of experimental results, but averaging methods leads to either “garbage, or, even worse: spurious regularity” (Newell, 1973, p. 295).

Newell (1973) argued that the task in psychology is first to discover the structure of the subject, which is fixed and invariant so that we can theoretically infer the method it would choose for solving a task (this, I suspect, is also why cognitive architectures prefer to model subject characteristics that are relatively stable over time, as I will explain later). Then, once we know the subject’s goal and the task environment, we generate a small collection of methods that are likely to be used by him in the task, and with careful experimental design or by suitable post hoc analyses, we can find out what method he selected. He explains that the reason for numerous correctional experiments in cognitive psychology is likely the fact that subjects opted for new methods to approach the problem

– by predicting likely methods beforehand, it would be possible to exit the loop of repeating correctional experiments. Newell (1973) argued that where experimentalists in cognitive psychology often did attempt to pick a couple of methods their subject could have used to approach the task, or portrayed the entire set of processing stages in a flow diagram (a novelty at his time), they often left open the total method being used, and did not operate within a frame, a *control structure*, which would *constrain* what other methods might also be evoked to perform the same task. This brings us to computational cognitive models, as Newell argues that the control structures are best illustrated by programming languages. A rather simplified account of Newell's explanations is that programmed algebraic structures work well for making natural constraints (or control structures) in that they create a space where different methods can be tested *if* they conform to the limitations of space and time.

In order to remedy the problems he pointed out, Newell (1973) suggested three approaches, or paradigms, to experimentation:

1. Complete processing models: Newell (1973) suggested that scientists should attempt to construct complete processing models, rather than partial ones as was the tradition. Newell also argued that the control processes employed here should be psychologically relevant.
2. Analyzing complex tasks: Focusing on a single complex task instead of designing specific small experiments to try and settle specific small questions would, and making sure that experiments employing those complex tasks could be sown together and each contribute its own detail to a bigger picture.
3. One program for many tasks: A third solution would be to construct a single system to perform a diverse collection of small experimental tasks. This suggestion contains the concept of both cognitive architectures, as well as psychometric artificial intelligence, as Newell proposes that an alternative mold would be to construct a single program that could take a standard intelligence test.

Although computational cognitive models were certainly already around before Newell's critique of cognitive psychological experiments in 1973, I believe Newell's contribution helped raise

them more towards the main streams of cognitive research, where in the sixties, computational cognitive models were the subject of only a handful of scientists (including Newell himself, rather unsurprisingly). Langley, Laird, Rogers and Sun (2008) refer to Newell's (1973) text as a "call to arms", which elicited a response in the scientific community and triggered new lines of research and modeling, mainly the specific class of architectures called *product systems*. Over the past decades, a variety of other architectural classes has emerged, some less concerned with human behavior than others. In the nineties, the movement transferred beyond basic research into the commercial sector with applications such as creating believable agents for simulated training environments, computer tutoring systems and interactive computer games (Langley et al., 2008).

Cognitive models and cognitive architectures

Cognitive models are simply models of human cognition, typically presented in the form of a more or less automatic cognition-simulating computer model. This allows us to inspect the behavior of the agent of the model, to predict it and its outcomes, observe the path of its unfolding. Cognitive models provide means for applying knowledge gathered by psychology along the years to a range of different domains, such as designing instruction (e.g. Merriboer, Paas & Sweller, 1998), designing and simulating functional behavior in non-human intelligent agents (e.g. the ADAPT-architecture for robotics; Benjamin, Lonsdale & Lyon, 2004), and the design and optimization of user interfaces (e.g. Ritter & Young, 2001).

Contemporary cognitive models are often created in the form of cognitive architectures, which differ in some ways from just any cognitive model. Cognitive architectures are much like foundation blocks to a building, so that part of the content of the created model (or the finalized house in our analogy) belongs to the outlying architecture, and part of it is supplied by the analyst, who adds to the *generic* architecture in order to construct a *specific* model. All models that are constructed around the same architecture share features of the architecture, as the architecture provides *constraints* to the model. What distinguishes the different models is information that has been added to the architecture in order to build them (Ritter & Young, 2001).

Dutch, Oentaryo and Pasquier (2008) define the concept of a cognitive architecture as a set of rules, for an intelligent agent, by prescribing computational processes with the intention of modeling a given cognitive system the architecture is designed for, most often a person or some other entity that can be considered intelligent under some definition. Although their definition is certainly extensive, Dutch, Oentaryo and Pasquire (2008) miss the central notion that the key idea of a cognitive architecture is to simulate *human* performance, in a human-like way (Byrne, 2001); very different from the engineering-approach to artificial intelligence, where the intention is to model the process with »any means necessary« using any available technology that would fit as means to an end. As an example, Byrne (2001) describes the Deep Blue chess program, which would not qualify as a cognitive architecture, because its problem solving is not human-like, but rather, based on massive searches of the entire game space. Byrne describes cognitive architectures as attempts to describe the overall structure and arrangement of the human cognitive system – a broad theory of human cognition, based on extensive experimental data transformed into a running computer simulation program. Ritter and Young (2001) define a cognitive architecture as an embodiment of »a scientific hypothesis about those aspects of human cognition that are relatively constant over time and relatively independent of the task«, and Langley, Laird, Rogers and Sun (2008) add that the term »architecture« implies that the intention is not only to model behavior or performance, but that a key notion is modeling the entire outlying infrastructure of the system.

Kieras (2007) explains that one of the first intentions of creation of cognitive architectures was to provide a balance to the so-called box-system that had been popular in cognitive psychology. Such models attempted to illustrate flow of information and storage mechanisms in different processing stages. Kieras (2007) argues that the problem with the typical box model was that sensory and motor organs were rarely included, and if they were, they were not presented in sufficient detail. He goes on to explain that since most of these models were created in service of different memory paradigms, it made sense that storage systems were emphasized, but that also limits their use for other applications. Additionally, the box models did not provide a perfect model even for the memory paradigms, for which they were created, since the contents of the storage systems were loosely specified in terms of

»features«, the processes of information exchange between different storage systems was rarely elaborated in sufficient detail, and often it was assumed that there was some kind of a vague executive process or component somehow responsible for overseeing, maintaining and managing function of the other components (Kieras, 2007). The components and processes in these box models were extracted from simple probability theory models for each individual box (such as what the probability might be for an item to be lost over time or information to be successfully transferred from one system to another (Kieras, 2007). Since each component was subsumed by a different probability model, interaction between the individual components may not have been modeled sufficiently or at all.

If we compare a more modern cognitive architecture model to one of the simpler box-models, we notice that they attempt to approach the modeled agent with a more holistic, dynamic approach. They typically still contain the box-model as an integral part of the architecture, but also – depending on the intention of the architecture and the modeling agent – other factors and processes, such as sensory-motor peripherals and separated memory and processing systems (Kieras, 2007). Where the "executive" component pulls together the box diagram, its use is avoided in a cognitive architecture by distributing its functions into different, more clearly defined processes between the different components, and by subsuming the entire architecture under a "cognitive processor", which in addition to performing cognition also controls the system as a whole, including itself (Kieras, 2007), (although in my opinion, one can question whether the cognitive processor and the executive component in the box systems really are that different from one another, and whether the cognitive processor could not be considered just a more elaborate notion of the executive).

Cognitive architecture research supports the creation and understanding of synthetic agents that support the same capabilities as humans; a central goal of artificial intelligence and cognitive science (Langley et al., 2008). But there is variability between cognitive architectures themselves, depending on numerous different factors. For example, architectures are created for different intentions, such as making an as-good-as-possible summary of quantitative data from cognitive psychology, or by focusing more on functional behavior and focusing on modeling functional behavior while not following experimental data in such a conscientious manner. Other models focus on detailed

accounts of all variables influencing behavior, while others aim at simplicity. Some include perceptional and motor components, where others only attempt to model memory and learning (compare e.g. EPIC and and ACT-R). Also the programming language the architecture is written in creates some variation (Kieras, 2005), as does the agent they attempt to model. For instance, one architecture may focus on modeling inherent aspects of human cognition, where another may be an attempt to create an effective path to building intelligent agents or optimizing their performance (such as the ADAPT architecture for robotics; Benjamin, Lonsdale & Lyons, 2004).

A recent trend has been to add actual extensions to the models built on an architecture (such as simulated eyes or hands, depending on what is relevant to the task), enabling construction of embodied models. Embodied models have the benefit of interacting directly with interfaces, allowing direct evaluation of the interface as well as better predictions and explanations for user behavior (Ritter & Young, 2001).

Capabilities of cognitive architectures

Not just any cognitive model qualifies as a cognitive architecture. I explained earlier that cognitive architectures function as blueprints for modeling performance and behavior of some intelligent agent. A cognitive architecture specifies the underlying infrastructure of an intelligent system, including aspects that remain relatively *constant over time* and across *different application domains*, including (Langley, Laird, Rogers & Sun, 2008):

a) The short-term and long-term memories that contain the agent's beliefs, goals and knowledge
b) The elements contained in short- and long-term memory and their organization into mental structures on a larger scale
c) The functional processes operating on these structures, including performance mechanisms that utilize them, and learning mechanisms that alter them

The memory contents of a cognitive architecture have a certain alterability to them: because the contents of a modeled' agents memories can change over time, different knowledge bases and beliefs can be interpreted by the same architecture. Langley et al. (2008) draw an analogy from an apartment, where there are structural parts that remain the same, such as walls and foundations, and as

well as furniture and appliances that can be both moved and replaced. Architectures differ in their specific assumptions about the alterability and use of memory contents, similarly as to how some apartment have built-in bookshelves or have appliances fixed to the structures, whereas others have freely movable furniture.

A few other abilities, such as recognition and decision-making, are a necessary requirement to support a well-defined architecture, but cognitive architectures frequently employ other functions, too, depending on their domain and intention of use (Langley, Laird, Rogers & Sun, 2008). A central issue is how to design the architecture so that the agent can access different sources of knowledge (Langley et al., 2008). For example, the system needs some kind of a perception function to gain knowledge about the environment, and planning, reasoning and prediction are required to know the implications of the current situation. Knowledge from other agents comes through communication, and knowledge about past situations requires memory and learning (Langley et al., 2008). The more of these abilities the architecture supports, the more sources of knowledge it can access to guide its behavior.

Another thing to define is whether the architecture should support these abilities directly with an embedded process, or if it should provide ways to implement them. The decision influences what the agent can learn from experience, what the designer can select to optimize for their model, and what specialized mechanisms are available for which functions (Langley et al., 2008).

Next I will go through some of the most common and fundamental capabilities of different cognitive architectures, abstracted largely from Langley et al. (2008), who provide an excellent account on the subject.

Recognition and categorization

Recognition serves as a contact between the agent's environment and its knowledge by allowing recognition of situations or events as reproductions of familiar patterns. Recognition can be *static* (e.g. recognizing letters that make up a word) or *dynamic* (e.g. recognizing a sequence of movements). Recognition is closely related to *categorization*; assigning objects, situations and events

to known concepts and categories. Recognition and categorization are tightly linked to *perception*, as they operate on output from the perceptual system.

To support recognition and categorization, a cognitive architecture needs to provide a way to represent patterns and situations in memory (Langley et al., 2008). It also needs some recognition process that lets it identify when (and to what degree) a particular stimulus matches a stored category. An architecture should also involve some mechanism for learning new categories from instruction or experience, and for modifying existing categories when necessary.

Decision making and choice

The system needs the ability to make decisions and discriminate among alternatives to operate in its environment. Decisions are typically associated with recognition of a situation or a pattern and then behaving accordingly, which is why cognitive architectures typically combine the two mechanisms into a *recognize-act cycle* underlying all cognitive behavior (Langley, Laird, Rogers & Sun, 2008). Architectures also enable models with more complex decision-making than the recognize-act cycles performed even on the architectural level, allowing modeling of complex, intelligent decision-making on different domains.

In order to support decision making, the system needs to provide some way of representing alternative choices or actions, and then some process for selecting among the alternatives – most architectures separate this step into two parts, where the system first determines whether a given choice is allowable (often by comparing it with some pattern and using it only if a match is found), and then selecting among different allowable alternatives, typically by computing them some numerical score and choosing one with a preferable score. This is called *conflict resolution* and it takes a very different form in different architectures (Langley et al., 2008).

Finally, a good architecture also incorporates some way to learn better decision making (Langley et al., 2008). This can involve learning new alternatives, but most typically means revising the conditions under which an existing action is considered allowable or by altering the numeric functions used to define scores for different action patterns during the conflict resolution.

Perception and situation assessment

A cognitive architecture also must provide some way for the modeled agent to sense, perceive and interpret its environment. Different modalities may be modeled, though the sensors range from simple devices like a thermometer to more complex mechanisms like stereoscopic vision or sonar generating a depth map in the area of the visual field. Perception can also integrate the results from different modalities into a single description of the environmental situation, which the architecture can represent then utilize by other cognitive processes (Langley, Laird, Rogers & Sun, 2008).

Perception covers many types of processing, from automatically supported simple ones to ones that require limited resources and need to be consciously invoked (Langley, Laird, Rogers & Sun, 2008). For example, our visual system detects peripheral movements automatically, but the fovea only extracts details from the small area where it is pointed. Actively directing the fovea to a point of interest calls for *attention*, which also needs to be included in a cognitive architecture for it to be able to direct its limited perceptual resources to detect relevant information in a complex environment (Langley et al., 2008). An architecture acquires and improves its perceptual knowledge about what sensors to invoke and where to focus them by learning from previous experiences, thus making it more efficient in a noisy, dynamic environment.

It's also not sufficient for a model to perceive isolated objects and events, but it needs to be programmed to understand and interpret the broader environmental situation. This calls for the agent to be able to combine perceptual information about many entities and events, from many sources, to compose a large-scale model of its current surroundings and internal states. This process relies on recognition and categorization and on *inferential* mechanisms.

Prediction and monitoring

One intention of cognitive architectures is to make *predictions* about future situations and events, based on the time and experiences they have had in an on-line state. Making a prediction requires some internal model of the environment and of how actions influence it, and a representation of this in the agent's memory. One approach involves mapping a description of the original situation

and the decided action, to a description of the resulting situation. Another approach is to encode the effects different actions have in changing the environment. Both cases require some mechanism that uses knowledge structures to predict future situations, e.g. by recognizing a class of situations where a certain action will have a certain effect. Ideally an architecture should also be able to learn predictive models from experience, and to optimize them over time. (Langley, Laird, Rogers & Sun, 2008).

An architecture with a mechanism for making prediction can also use it to monitor the environment. However, this poses an issue for architectures with limited perceptual resources (such as the ACT-R), because monitoring relates perception to prediction (Langley et al., 2008).

Problem solving and planning

Cognitive architectures must be able to generate plans and solve problems, because their modeled agents need to achieve their goals in new situations. Planning is only possible, if the agent has an environmental model that predicts the outcomes of its actions. Additionally, the architecture must be able to represent a plan as a sequence of actions, their expected outcomes, and what new actions their execution will enable. The plan structure can also include *conditional actions* (“in the form of “perform x only if y=1”) and different *branches* that depend on the outcome of earlier events (“if x leads to z, perform y, otherwise perform c”).

Planning can take advantage of numerous systems embedded in the architecture, such as low-level motor and sensory actions, but always require some sort of memory, problem solving and search. The challenge is to perform actions through a sequence of problem states, where at each step the system needs to consider applicable operators, select one or more, and apply it, leading to a new problem state. This is then repeated until a plan is found acceptable, or the system gives up. Being able to predict several sequences ahead makes problem solving easier, but how many steps ahead the system can think depends on its working memory and overall capacity. Even primitive potential for machine learning is of great use, since it allows the system to detect planning and problem solving patterns that have proven useful previously. There are different ways to achieve this. An architecture can learn from direct instruction, from the results of a problem-space search, by observing another

agent's behavior – also called *behavioral cloning* (Sammut, 1996, as cited in Laird, Langley, Rogers & Sun, 2008), or from delayed rewards via *reinforcement learning* (Sutton & Barto, 1998, as cited in Laird et al., 2008). There are two ways learning helps problem solving: by reducing the branching factor of search, either by utilizing heuristic conditions or by refining a numeric evaluation function to score different options, in order to guide the agent's choice. The system can also store plans which makes a search faster since it allows it to take larger steps in the problem space, essentially skipping multiple problem stages.

Sometimes it is necessary to modify an existing plan in response to unexpected changes in the environment, for example, when a situational variable changes so that the action plan is no longer applicable, or when a new situation brings some more beneficial way of performing the action sequence. Changing action in this case is possible only for a system capable of monitoring its own performance.

Reasoning and belief maintenance

Reasoning is the process of drawing mental conclusions from other beliefs or assumptions that are known to hold (Langley, Laird, Rogers & Sun, 2008). In order to support such a process, an architecture must be able to represent relationships among beliefs. Typically these relationships are coded in first-order logic, but other systems have been applied, such as neural networks, production rules and Bayesian networks (Langley et al., 2008). Reasoning can also be heuristic or approximate, not necessarily logical, and still prove useful for an intelligent agent – this is also sometimes closer to the reality of human reasoning and in some cases may thus even be preferable to other types of inferences. Deductive reasoning is often used as a mechanism to draw inferences using knowledge structures. Inductive reasoning is also a possibility, as is abductive inference. Selecting which method of inference to apply happens similarly as plan making or problem solving, in a sequence of evaluative steps.

Cognitive architectures infer new beliefs as well as decide whether to maintain old ones. Certain beliefs depend on others, and the agent needs to track the latter to determine whether it should

continue to believe in the former, abandon it, or alter its confidence. This process is called *belief maintenance*, and it is especially important in dynamic environments with unexpected changes. It also necessitates a dependency structure that connects beliefs in the system's memory.

Execution and action

Since cognition occurs to support and drive activity in the environment, a cognitive architecture must be able to represent and store motor skills enabling such activities. These can range from primitive, or component actions, to complex multi-step procedures (the latter is related to planning action sequences), however, a high-quality architecture should support anything from fully reactive and simple reactions to automatized, open-loop behaviors, as the capability of performing both is an inherent characteristic of nearly any intelligent system.

Actions and their sequences should also be learnable, by behavioral cloning, reactive learning or successfully achieving goals. Knowledge for action selection as well as skill complexity are also learnable in an optimal cognitive architecture.

Interaction and communication

Agents exist in environments in interaction with other agents, and communication allows one agent to obtain knowledge from another. The modality in which communication occurs can vary, but any communication allows for conveying any of the cognitive activities discussed so far (categories, decision making, perception, actions, predictions, plans, inferences, problem solving…) An architecture should support a mechanism for transforming this knowledge into a communicative form, the most common being spoken or written language, but in any case, in some natural or artificial language understood by both entities despite of different mental structures. Translating into an external format requires planning and execution, whereas language comprehension requires inference and reasoning. Conversational dialogues pose a specific challenge by requiring both generation and understanding of natural language, as well as coordination with the other agent in the form of turn taking.

Remembering, reflection and learning

These are so-called *meta-management mechanisms* (Sloman, 2001, as cited in Langley, Laird, Rogers & Sun, 2008), not strictly required for an intelligent agent, but their inclusion significantly increases flexibility and robustness.

Remembering is the ability to encode and store results of cognitive processing in the system's memory, for retrieval or access later on. To remember a cognitive activity, the architecture stores the cognitive structures created during that activity, indices them in its memory and retrieves them when needed. The resulting content is referred to as *episodic memories* (Langley et al., 2008). *Reflection* is the processing of either recently activated mental structures or older structures retrieved from the episodic memory. *Explanation* of an agent's cognitive actions (such as inferences, plans, decisions) is one type of reflective activity, another one is *meta-reasoning* about other cognitive activities – a process where the agent explains the generation of its cognitive activities (e.g. its process of making inferences). Reflective processes leave their own traces and so may themselves be subject to reflection.

The final meta-management mechanism is related to learning, which is typically an automated function in nearly any given cognitive architecture, not subject to the agent's inspection or conscious control. With more elaborate learning systems, however, the agent gains the ability to transfer learned information either between similar tasks, within the same domain, and even to tasks within related but distinct domains. Data for learning can come from many sources, such as observation, problem-solving, practice or known skills, but always involves the processing of memory structures with an intention to improve the agent's capabilities (Langley, Laird, Rogers & Sun, 2008).

Properties of cognitive architectures

The internal properties of cognitive architectures largely define their capabilities. The properties can be split to the architecture's knowledge representation, the organization of its knowledge, the manner in which knowledge is utilized, and the mechanisms for acquisition and revision of knowledge. I will discuss these shortly and with an attempt to avoid technicalities.

Knowledge representation

Knowledge itself is not built into the architecture, as it can change across domains over time, and the architecture is there to model characteristics that stay stable over time. However, how the agent encodes its knowledge constitutes a central aspect of a cognitive architecture (Langley, Laird, Rogers & Sun, 2008), and different representational choices come with different advantages and disadvantages. The primary representational question is whether the system should use a single notation for encoding all its knowledge, or employ a mixture of formalisms. Selecting only one notation brings elegance and works well for abilities that must operate only on one type of structure (such as learning or reflection), but focusing on one framework forces the system to an unpractical approach towards certain problems (Langley et al., 2008). Langley et al. (2008) present distinguishing *declarative* and *procedural* representation as a common tradition, and explain that architectures typically include some of both. Declarative encodings are such that can be manipulated by cognitive mechanisms independent of their content, supporting very flexible use, but leading to inefficient processing. Procedural formalisms encode knowledge about how to complete tasks, are often represented as production rules, and allow efficient but inflexible application of knowledge.

Some architecture also support *meta-knowledge* about the agent's own capabilities. This type of knowledge supports meta-reasoning, and enables good understanding of the cognitive steps taken during an agent's activities and the reasons for them.

Tulving (1972, as cited in Langley et al., 2008) differentiates between *semantic* and *episodic* memory. Most architectures focus on semantic memory, whereas episodic memory appears better for retrieving specific facts and occurrences. In architectures, both can fundamentally serve the same purposes, so focusing on one does not necessarily limit the agent's possibilities (Langley et al., 2008).

Through the time, knowledge representation has happened through different representational formalisms (Langley et al., 2008). *Semantic networks* encode both generic and specific knowledge in a declarative format consisting of nodes (for concepts or entities) and links (relationships between them). *Conceptual spaces* (Gärdenfors, 2004) consist of a number of "quality dimensions" often

derived from perceptual mechanisms, capable of representing various kinds of information as well as to describe concept learning. *First-order logic* encodes knowledge as logical expression in terms of predicates and arguments, along wih logical statements about the relationships of the expressions such as conjunction, disjunction, negation and implication. Langley, Laird, Rogers and Sun (2008) also briefly describe *frames* and *schemas* and *plans*, a structured framework for encoding courses of action.

Organization of knowledge

Another important question is how the knowledge should be organized in the memory of the architecture. The first point to consider is whether the knowledge representation scheme should support hierarchical structures (Langley, Laird, Rogers & Sun, 2008). The decision influences whether or not task hierarchies will be easily executed by the system, but different types of semi-hierarchical structures are also possible (Langley et al., 2008). The second point to think about considers the granularity of the knowledge stored in memory. For example, production systems and first-order logic constitute fine-grained knowledge, and so an architecture using them for encoding needs an interpreter to compose them in order to achieve complex behavior. Coarse-grained structures store entire plans and macro-operators, which puts less burden on the interpreter but leads to less flexibility and generality of the applied knowledge. Langley et al. (2008) suggest a structured framework, which describes coarse elements in terms of fine-grained ones, giving access to both.

The final important point about knowledge organization is the number of distinct memories the architecture supports, and how they are connected to one another. The modeled agent needs to have some kind of long-term memory (LTM); the contents of the LTM should be relatively stable over time, but accessible to alteration through instruction and learning. The agent also needs short term memory (STM) to contain more dynamic and short-lived beliefs and goals. In most production system architectures, LTM and STM are structurally distinct but related through a process comparing the conditions of the LTM production rules to the STM structures (Langley et al., 2008). Some architectures have distinct types of short- and long term memory, e.g. procedural, conceptual and episodic structures, as they appear in humans.

Utilization of knowledge

The knowledge lying in the different memories must of course be utilized for the architecture to execute any cognitive activities. One question is whether problem solving should rely on heuristic search through the problem space, or on retrieving solutions or plans from the long-term memory (Langley et al., 2008). These approaches are not mutually exclusive, but an architecture typically focuses on one over the other.

Approaches for architectures supporting multi-step problem solving and inference are *forward chaining*, *backward chaining* and *means-ends analysis*. In forward chaining, the system applies relevant operators and inference rules to the current problem state and beliefs to produce new states and beliefs. Backwards chaining generates new sub goals from current goals, progressing from some goal state towards current states or beliefs. Means-ends analysis combines backwards and forwards analysis by selecting operators through backward chaining, but executing them whenever their preconditions are satisfied (Langley et al., 2008). It's important to differentiate between problem-solving techniques supported by the architecture and ones implemented by knowledge. Different types of knowledge utilization can occur in architectures that don't inherently contain them, if the conditions are met.

Another central issue for knowledge utilization concerns the relationship between cognitive activities and action. Systems can be split to *deliberate* and *reactive*, in that deliberate architectures first plan or reason an action plan and after that begin its execution, whereas reactive architecture selects its action on each decision cycle based on its comprehension of the current situation. Deliberation has advantages in predictable environments, whereas reactive faire best in unpredictable circumstances. There is a similar point between perception and action. A *closed-loop* control system senses the environment on each cycle, providing an opportunity to respond to recent changes. And *open-loop* system runs a long action sequence over multiple cycles without sensing the environment.

Acquisition and refinement of knowledge

Knowledge is acquired from instruction or experience, and both elicitation and execution of learning behaviors are handled at the architectural level. Some architectures support many such mechanisms, whereas others rely on a single learning process interacting with knowledge and experience to achieve different outcomes (Langley, Laird, Rogers & Sun, 2008). Some processes learn entirely new knowledge structures, whereas others only modify existing functions. Existing structures can also be revised by adding or removing parts. Another common distinction is to differentiate *analytical* and *empirical* methods – where analytical knowledge acquisition relies on reasoning and is more exploratory in nature, empirical methods rely on inductive operations, detecting regularities and using them to transforming experience into usable knowledge of descriptive nature. Architectures can utilize hybrid methods of incorporating ideas from both frameworks as well as combine them through different mechanisms.

The fourth issue concerns at what point the acquired knowledge should be incorporated into the system. Most frameworks are of a so-called *eager* approach, forming generalized knowledge structures from experience as soon as it enters memory, whereas those with the *lazy* approach store experiences untransformed, then perform implicit generalizations at the time of retrieval (Langley et al., 2008).

Finally, learning occurs either in an incremental or non-incremental manner. Incremental learning incorporates training cases one at a time, with limited memory for previous cases, so that the knowledge base is updated after processing every single experience. Non-incremental methods process all cases in a single step. This is not as natural as incremental learning, because agents exist over time and accumulate knowledge gradually, but non-incremental learning avoids the drawback of order of presentation influencing behavior, as it does for incremental learning utilized by most architectural systems.

The use of cognitive architectures; what are they good for?

Constructing a user model within a cognitive architecture offers various advantages to the modeler. The architecture provides a framework to models that can be created on it by posing constraints. One of the consequences is that the constrained models allow identification of information, or knowledge, needed in order to perform the task being analyzed, as well as to differentiate what information is gained from the environment, and what of it is inherent to the modeled agent. Additionally, since models built around a common architecture share constraints and assumptions contained in the architecture itself, they can be relatively easily integrated into a coherent, single model of a broader scope. This allows for the development of libraries of »behavior idioms« and partial models, which eases the task of constructing new models (Ritter & Young, 2001). For example, such a library already exists for the ACT-R architecture (Kieras, 2005).

Cognitive architectures contain cognitive mechanisms and resources that are relevant to the task at hand, allowing the analyst to recognize their meaning in successful performance or error generation. For example, for interface design such measures can include the working memory load, the time taken to learn an interface, what gets learned and the causes and types of errors in using it (Ritter & Young, 2001). They also provide a basis for teaching designers about users. Designers can refer to the architecture and its behavior for answers to both general and specific questions about users (Ritter & Young, 2001).

Overall, cognitive architectures provide a very useful tool for contemporary cognitive psychology. With the interplay of cognitive science, artificial intelligence and cognitive psychology, they can be considered a nearly inseparable part of the field, working on both the theoretical aspect of cognitive psychology by helping with the construction of complete models of human cognition, as with the practical aspect, modeling holistic performance and events within the system of the intelligent agent, representing an interesting supplement to task optimization efforts using only expert systems.

Considering the birth of computational cognitive models as a remedy to integrate findings from experimental cognitive psychology with, among others, the intention to increase their usability,

and the fact that the entire idea of cognitive architectures is to *apply* vast quantities of empirical data from cognitive experiments to artificial intelligent systems with human-like problem-solving methods and qualities, one could argue that in a way, cognitive architectures are by nature the subject of applied cognitive psychology. Their applicative nature makes them very useful for some practical intentions. The uses for cognitive architectures vary greatly by domain of application, but considering the range of different available architectures and the possibility of always developing new architectures, essentially only imagination is the limit, assuming that we're considering an application which would actually benefit from the use of a computational cognitive model (which, I believe, is not always the case – a point discussed more in the later section of critique). A few examples of the applications of different architectures include designing instruction (e.g. Merriboer, Paas & Sweller, 1998), designing and simulating functional behavior in non-human intelligent agents (e.g. the ADAPT-architecture for robotics; Benjamin, Lonsdale & Lyon, 2004), and the design and optimization of user interfaces (e.g. Ritter & Young, 2001). The applications can also be very specific, such as planning automatized pilots with realistic AI for fighter flight simulators (Jones, Laird, Nielsen, Coulter, Kenny & Koss, 1999).

We will discuss some individual cognitive architectures and their applications in the next section. Unfortunately going through all possible interesting architectures is absolutely out of the scope of this seminar, but I will nevertheless attempt to gather a list that would be extensive enough to cover some of the most prevalent architectures today. At this point, I recommend the reader to note how some architectures are more loyal to the notion of modeling »as-human-as-possible« behavior than others. All the architectures examined have some theoretical commitment to parallelism, specifically in memory retrieval, and also employ one or a few decision modules. Connectionist approaches demonstrate weaker performance in the broad functionality cognitive architectures offer at best, and although they sometimes serve as significant components in larger architectures, such as in CLARION (Sun, Merril & Peterson, 2001), I will not discuss them in detail for the reason.

I will start by discussing three related production systems, Soar, EPIC, and ACT-R/PM. They were originally developed to model slightly different aspects of human cognition. However, as they develop, there appears to be more convergence than divergence. (Byrne, 2001).

Soar

Soar (Laird, Newell & Rosenbloom, 1987, as cited in Langley, Laird, Rogers & Sun, 2008) is a cognitive architecture that has been developed since the beginning 1980's. It contains procedural long-term knowledge in the form of production rules, organized in terms of operators associated with different problem spaces. The operators range from those describing simple, primitive actions with the intention to modify the agent's internal state or to generate primitive external actions, to others describing more abstract activities (Langley et al., 2008). Soar used to represent all long-term memory in this manner, but has recently had additions of episodic and semantic memories; the episodic memory holding a history of previous states, whereas the semantic memory contains previously known facts.

All of Soar's tasks are attempts to pursue a given goal. The knowledge components are used to dynamically determine the selection and application of basic deliberate acts performed by operators. The system moves through one decision at the time, repeatedly suggesting, selecting and applying different operations. Soar also contains a learning component to determine which operator to apply when an abstract operator cannot be implemented, by creating a new goal to determine which operator should be implemented and how. The nature of this process leads to the generation of a goal hierarchy, where the problems are decomposed into sub problems when necessary, so that the current state of a specific goal includes the features of its super ordinate goals and the state of any additional cognitive structures bound to pursuing the goal. Information obtained by sensors about the external environment is available to both sub- and superordinate goals. Then the blockage generating a sub goal is solved, that goal disappears, along with its related sub-goals.

Soar includes different learning mechanisms for different types of knowledge: chunking and reinforcement learning for procedural knowledge (reinforcement learning adjusts numeric values associated with rules that help select operators), and semantic and episodic learning for declarative knowledge. Chunking is the learning process, where one or multiple results are produced in a sub goal, and represented as a new production rule as a summarization of the process leading to the results – a process similar to that in ACT-R. Once the system has learned a chunk, it will be applied as a possible

action pattern in new situations similar along relevant dimensions to the situation where it was learned, attempting to avoid the blockage that lead to its formation.

Soar has been used to develop a variety of sophisticated agents demonstrating impressive functionality (Langley et al., 2008), for example, the TAC-Air-Soar, modeling fighter pilots in air combat scenarios. Soar has also been used in the realization in numerous intelligent non-player characters in interactive computer games (Haunt 2, on the UT game engine; Laird, Magerko, Kerfoot & Stokes, 2004). Other applications involve modeling details of language processing (Lewis, 1993, as cited in Langley et al., 2008), categorization (Miller & Laird, 1996, as cited in Langley et al. 2008), and other aspects of cognition, but overall Soar has been developed towards high-level functionality, rather than as-good-as-possible fit to experimental data.

EPIC

EPIC (Meyer & Kieras, 1997) is a production system cognitive processor, which also includes perceptual and motor processors. It encodes long-term knowledge as production rules and organizes them as methods for accomplishing goals which match elements in a variety of its short-term memories, including visual, auditory and tactile buffers. The system selects matched rules and applies them in parallel to control eyes, hands, or to alter the memory contents. EPIC focuses strongly on achieving quantitative fits to human behavior, and as a multimodal system it has a great emphasis in human-like problem solving and performance on tasks involving interaction with complex devices. Due to the sensory-motor approach of the system, it is excellent in predicting strategies, performance and typical errors in a task employing multiple sensory systems. It is also relatively simple to program, as the underlying architecture is relatively simple, however as it is primarily an academic research system, it has limited support and poor commercial availability (Kieras, 2005). Additionally, it offers no coverage of intermediate to longer-term memory phenomena (and as such is not very adequate for modeling learning in a multimodal task environment). The perceptual/motor systems offer many details and parameters to be specified to fit the task of interest, but this also leads to the modeler having to confront more details than she might really like to.

EPIC-Soar

EPIC-Soar is just what it sounds like, a combination of EPIC and Soar, two complementary cognitive architectures, with functions and features of both combined into one. Since I've already discussed features of both, I will only discuss the special features of their combination. An unusual thing about the architecture is that it has no declarative knowledge representation (Kieras, 2005). The system is well-established in the AI community and offers good software support and training courses. Additionally, it is included in several commercial-grade applications and easily accessible through a website. The combination of EPIC and Soar costs some money, whereas Soar itself is available free of charge. Unfortunately, the psychological basis of learning and reasoning is not as well developed as in some other cognitive architectures, and the program is said to be very difficult to program (Kieras, 2005).

ACT-R/RP

ACT-R is one of the latest families of cognitive architectures (even though it has been around since 1970's, concerned primarily with modeling human behavior (Langley, Laird, Rogers & Sun, 2008). It is a hybrid production system architecture, with neural-net like activation mechanisms, its infrastructure is related to components from cognitive neuroscience, and the system has a heavy emphasis on activation and learning mechanisms (Kieras, 2005). The ACT-R community aims at building a system that can perform the full range of human cognitive tasks and describe in detail the mechanisms underlying perception, thinking, and action (Dutch, Oentaryo & Pasquier, 2008). ACT-R 6 includes sensory modules for visual processing, motor modules for action (adopted from EPIC; Kieras, 2005), an intentional module for goals and a declarative module for long-term declarative knowledge (Langley et al. 2008). Each module has a buffer, which all together comprise ACT-R's short-term memory. It also has a long-term memory of production rules to coordinate processing of the modules.

ACT-R functions by determining which productions match against the contents of the short-term memory and then computing the utility for different productions by calculating the difference

between its expected benefit (the goal's desirability times its probability of success) and its expected cost. The system selects the production with the highest utility and executes its action. The situation changes, leading to the evaluation of new possible productions and actions.

ACT-R learns on both structural and statistical levels. The base activation for declarative chunks increases when they're associated with selected productions and decays when they're unused for a long time, and the cost and success probability for productions goes through updates as the system gains "experience" by observing its own behavior. The architecture is capable of learning entirely new rules by analyzing rule firings, replacing constants with variables and combining them into new conditions and actions (Taatgen, 2005; as cited in Langley, Laird, Rogers & Sun, 2008).

The ACT-R has been used to model numerous phenomena from experimental psychology, such as memory, attention, reasoning, problem solving and language processing, and there are several publications reporting accurate fits to quantitative data about human reaction times and error rates (Langley, Laird, Rogers & Sun, 2008). Anderson, (2007, as cited in Langley et al, 2008) has related ACT-R modules to different brain areas and developed models that match results from brain imaging studies. On an applied front, the architecture has played a central role in a widely used school tutoring system (Koedinger et al. 1997, as cited in Langley et al., 2008), and it has been used to control mobile robots that interact with humans (Trafkon et al., 2005, as cited in Langley et al., 2008). ACT represents one of the most adequate types of cognitive architectures at this time. It has a very active user community consisting primarily of psychologists – it is the best known of current psychology-based systems (Kieras, 2005). The team offers training programs, good client support and commitment to user-friendly software, however since the ACT is primarily an academic research system, its value for large scale applied problems is not clear. Additionally, it lacks perceptual and motor representations.

ICARUS

Icarus is a recently developed architecture (Langley, Cummings & Shapiro, 2004, as cited in Langley, Laird, Rogers & Sun, 2008). It stores two distinct forms of knowledge – concepts, which

describe classes of situations by comparing them to other previously known concepts and percepts, and skills, specifying goal-achievement behaviors and decomposing them to subordinate goals.

ICARUS functions in a somewhat different way from the older cognitive architectures presented earlier. The system basically searches for visible objects and deposits their descriptions into a perceptual buffer. It then compares the "perceptions" to primitive concepts and adds matched instances to its STM as "beliefs". These activate matching higher-level concepts, allowing the system to deductively search for all other implied beliefs. After this, the system finds a path downward through its skill hierarchy, where each sub skill has satisfied conditions but an unsatisfied goal. The path terminates in a primitive skill with executable actions, which the system applies to influence the environment. This process leads to new percepts through the perceptual buffers, changes in beliefs, and the execution of other skill paths to achieve the agent's goals as a response to the first action. When the system fails to find any applicable path through the skill hierarchy in order to achieve its top-level goal, it applies a kind of means-ends analysis to solve the problem. This process backs begins acting around either a skill that would typically achieve the current goal, or attempts to redefine the goal so that it resolves the problem a step at the time, executing necessary actions as they become applicable. Whenever this problem solving achieves a goal, the system creates a new skill indexed by that goal, which it then executes if it finds itself in a similar position.

ICARUS has been specifically useful for tasks requiring a combination of inference, execution, problem solving and learning, across multiple domains, ranging from computerized solitaire, through simulating drivers in an urban virtual environment, to planning large scale logistics. ICARUS is also linked to physical robots carrying out interactive or joint task activities with humans.

Critique and problems in applicability

Although cognitive architectures provide a useful tool for practical applications as well as a theoretical base for research, they have some questionable aspects and shortcomings of theoretical, philosophical and practical nature. I will dedicate this section to discuss some of my own questions

about the nature of cognitive architectures, as well as those raised by researchers who are obviously more qualified in the field and, thus, likely bring out more relevant points.

My first argument lies on Newell's (1973) second injunction on cognitive experimentation, where he warned against averaging methods, as it leads to either poorly deviated results or regularity. If Newell (1973) considered averaging experimental results a bad idea, then on what results should a programmer base their cognitive model? If a cognitive architecture must be based on extensive experimental results across the relevant domain, then do the results not need to be averaged *some* way to be useful for building a single architecture? In this way, is a cognitive architecture not an expression of the average human's average performance, completely ignoring individual differences, which we have reason to believe should distribute normally; and if a cognitive architecture attempts to take into account individual differences in predicting performance, then how is that practically taken into account? This is obviously not really a problem in using an architecture to create a human-like intelligent agent, but does pose a problem when using an architecture-based model in a phenomenon greatly influenced by individual differences.

This leads me to another problem; applying the architectures in a practical situation where the idea is to model individuals that have been *proven* to somehow vary from the normative population the architecture is based on. For example, cognitive architectures have been used to model fighter pilot performance (e.g. Jones, Laird, Nielsen, Coulter, Kenny, & Koss, 1999). Since the pilots are selected with extensive psychometric and physical testing and then go through extensive training before allowed to fly their vehicles, is it realistic to assume that an architecture based on an average individual would provide an adequate model of, and perform similar errors, as an actual fighter pilot? Additionally, would the pilot's affective state in an actual combat situation bring performance variance that would be difficult to capture with an architecture that includes no conative component? Jones et al. (1999, as cited in Byrne, 2001) explicitly wished to use the cognitive architecture model in their research on fighter pilots in order to »eliminate the need for expert participants«, which in my opinion was a mistake, since it is – again, in my opinion – realistic to assume that an expert participant would be significantly different from the agent of the architecture. The idea was especially strange, since the

research was carried out by the U.S. military, which, I imagine, would have easy access to interview expert fighter pilots.

Another question is whether or not a cognitive architecture provides anything more than their programmers and the experts participating in the process can provide without any kind of a computational model. Or, in other words, if the computer model is really a necessity if the same solution can be acquired with the same development team, using other means – in the most extreme case, only pens and papers. If a cognitive architecture is used to supplement information gained from an expert system, and the architecture is also specified after an expert in the specific field, does it really provide any new information? This is an important question, if time and resources are limited – especially if the person interested in the information would have to outsource personnel for the cognitive model.

Another issue with the direction computational computer modeling has taken on is that Newell's critique is again becoming more and more topical, but when it was earlier directed at experimental cognitive psychology, it could now be directed at computational psychology and artificial intelligence. As AI and cognitive science have matured, they have fragmented into several well-defined sub-disciplines, with different interests, goals and independent criteria for evaluation (Langley, Laird, Rogers & Sun, 2008), despite the fact that commercial and governmental applications require increasingly integrated intelligent systems, not just improved system components. Newell's critique against setting binary distinction is also relevant to the field, as different schools disagree on whether processing should be parallel to model neurons and brain function as a base for every application, or if it is arbitrary how the system is constructed, as long as it is a good-enough model of its intelligent agent. In a way, computational modeling just changed the platform of Newell's arguments from experimental cognitive psychology to cognitive modeling.

Human behavior is variable across different domains, with different variables causing different responses depending on environmental parameters. Responses also vary from individual to another due to individual differences. Theoretically it would be necessary to know all parameters and all

individual characteristics to build and specify a cognitive model that would model one individual person, and so we'd need an infinite amount of models to display performance of each individual person with their individual differences, and we would still not have perfect predictive power. In my opinion, this displays how the power of cognitive models is in some ways limited to either the creation of intelligent agents or to modeling behaviors where there is relatively little individual variation. Granted, judging by the literature I've reviewed for this seminar, they appear to provide solid performance when used for this purpose.

References

Benjamin, D. P., Lonsdale, D. & Lyons, D. (2004). ADAPT: A Cognitive Architecture for Robotics. *International Conference on Cognitive Modeling,* Pittsburgh PA, July 2004.

Duch, W., Oentaryo, R.J. & Pasquier, M. (2008). Cognitive architectures: where do we go from here? *Frontiers in Artificial Intelligence and Applications, 171*, 122-136

Gärdenfors, P. (2004). Conceptual spaces as a framework for knowledge representation. *Mind and Matter (2)*2, 9-27.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999). Automated intelligent pilots for combat flight simulation. *AI Magazine, 20(1),* 27-41.

Kieras, D. (2005). *A survey of cognitive architectures – pros and cons of existing architecture applications*. University of Michigan.

Kieras, D.E. (2007). The control of cognition. In W. Gray(Ed.), *Integrated models of cognitive systems*. Oxford University Press.

Langley, P., Laird, J.E., Rogers, S. & Sun, R. (2008). Cognitive architectures: research issues and challenges. *Cognitive Systems Research,* doi:10.1016/j.cogsys.2006.07.004

Magerko, B., Laird, J. E., Assanie, M., Kerfoot, A., & Stokes, D. (2004). AI characters and directors for interactive computer games. In Proceedings of the sixteenth innovative applications of artificial intelligence conference (pp. 877–884). San Jose, CA: AAAI Press.

Meyer, M., & Kieras, D. (1997). A computational theory of executive control processes and human multiple-task performance. Part 1: Basic mechanisms. Psychological Review, 104, 3–65.

Newell, A. (1973). You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. In W. G. Chase (Ed.), *Visual information processing* (pp.283-308). New York: Academic Press

Ritter, F. E. & Young, M. R. (2001). Embodied models as simulated users: introduction to the special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies, 55*, 1-14

Sammut, C. (1996). Automatic construction of reactive control systems using symbolic machine learning. *Knowledge Engineering Review*, *11*, 27–42.

Sun, R., Merrill, E., & Peterson, T. (2001). From implicit skills to explicit knowledge: A bottom–up model of skill learning. *Cognitive Science, 25*, 203–244.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Van Merrienboer, J. J. G., Paas, F.G.W.C, & Sweller, J. (1998). Cognitive architecture and Instructional Design. *Educational psychology review, 10*(3), 251-296