Abstract

We compare Action Selection and Schema mechanisms for robotic control, focusing mainly on the reactive vs. anticipatory distinction. We present AKIRA, an agent-based hybrid architecture, focusing on its capabilities to design fuzzy-based Schema models. We implement in AKIRA reactive and anticipatory mechanisms, and we compare them in an experimental set-up in the Visual Search domain.

1 Introduction

Action Selection is the task for a cognitive system of deciding what to do next: it is a central problem in behavior-based robotics, where normally the control is not centralized but distributed between semi-independent modules, called behaviors. It was soon evident that performing complex operations such as planning, and even maintaining internal representations, was problematic in real time. Thus, many authors avoided internal representations and only used reactive behaviors, following the slogan “intelligence without representation”. Behaviors are like stimulus-response rules; their organization in a connectionist network also permits reactive planning.

Schema mechanisms, inspired by neurobiological evidences, are quite popular in cognitive robotics, focusing on anticipation [6, 20, 5, 1, 16]. Schemas exploit internal representations, that are not “objective” models of the phenomena (such as logical definitions, topological or metric maps) but “interactive” ones: predictions about actions consequences, produced by internal (forward) models. Schemas are thus three parts rules: stimulus-response-expectation.

In Section 2 we compare the strengths and limitations of Action Selection and Schema mechanisms. In Section 3 we introduce AKIRA [14, 21], an hybrid symbolic-connectionist, agent-based framework; and its fuzzy-based Schema mechanism. In Section 4 we compare reactive and anticipatory mechanisms in a Visual Search task.

2 Action Selection vs. Schema mechanisms

Action Selection and Schema mechanisms are both inspired by the situated cognition approach, postulating that a cognitive system is “attuned” to its environment by the means of its sensorimotors interactions. Here we focus on a crucial distinction between them: the former exploits reactive mechanisms, mainly without internal representations; the latter has instead also anticipatory representations.

2.1 Reactive Action Selection

Action Selection depends on many factors: urgency of behaviors; priority of drives; availability of opportunities in the environment; commitment to an already started sequence of actions. The crucial point is that all this is realized in a reactive way, only by the means of connected structures linking sensors to effectors (with few intermediate nodes representing e.g. drives): in [11, 4] perception and action realize a continuum that is not mediated by representations and does not involve reasoning. Here we only point out some relevant points for the further comparison; for an extensive review see [18].

Subsumption Architecture and Behavior Networks. The first behavior-based mechanism, the Subsumption Architecture [4], puts “in stack” a set of independent modules (with separate sensors and effectors) responsible for different drives (e.g. move, eat). Arbitration follows a predefined sequence, since the modules are prioritized.

The Behavior Networks (BNs) [11] represent instead goals, facts (perceptions) and competence modules (actions) as nodes in a connectionist network. Actions and goals are in concurrence and there is activation flow between them (from goals to actions, from perceptions to ac-
tions, etc.). The BNs have not a predefined arbitration sequence; on the contrary, depending on the goals as well as on the bottom-up pressures of active facts, the more contextually appropriate module is selected for activation. In this way planning is never predetermined, but is on-line and context sensitive. Subgoaling is implicit: if a module has a false fact as a precondition, it spreads activation to another module that is able to make it true. In order to be more adaptive in dynamic environments, the BNs do not store or pass representations (e.g. variables in modules), but each action is totally specialized. This feature seriously weakens their scalability; recently some extensions have been proposed allowing to pass between the modules deictic representations (a single variable, the “attention focus”) or variables [13], thus realizing a symbolic/connectionist hybrid.

Behavior-Based Robotic Control. In the field of autonomous robot navigation there are many examples of behavior-based, reactive mechanisms, using e.g. potential fields [2] or fuzzy logic [17]. In [17] two main problems are pointed out: behavior selection and command fusion. The first problem is how to select the right behavior(s) to apply; the second one is how to integrate the (converging or diverging) commands from all the active behaviors. Both problems are addressed by using fuzzy logic, that permits to integrate in a native way many commands and to weight them according to the “desiderability” of the behaviors, also dealing well with uncertainty and measurement errors. It is worth to notice that the system integrates preferences rather than decisions (i.e. fuzzy sets rather than crisp values): this makes a difference, mainly in cases where the “average” of different commands leads to erroneous consequences (e.g. in the task of avoiding an obstacle, going on the left and going on the right can be averaged as going straight). The architecture is hierarchical: high level behaviors (that can represent goals) can activate low level ones. A behavior is thus activated both for its preconditions (that are matched in a reactive way) and for top-down, goal driven influences.

2.2 Anticipatory Schema Mechanisms

In cognitive robotics Schema mechanisms are widely used [6, 5, 20, 1, 16]. Schemas play roughly the role of behaviors, but they also have anticipatory capabilities, furnished by internal models. For example, Schemas in [20] include a forward model predicting actions consequences, that is used for schema control, learning and selection; and an inverse model, that is used to plan the next action.

Exploiting Forward Models. Using forward models for control is not new; in fact, they are often used in control theory. For example, Kalman Filters have an internal model that receives an efferent copy of the motor commands and generates feedback for the system in the form of a “mock” sensor information. This allows to fine-tune the motor commands if there are errors in the model and sensor signals; to learn (the internal model); to fuse model and sensor information depending on their reliability, and also to substitute sensors if they are absent (e.g. a robot enters a dark room). There are also applications of Kalman Filters in cognitive modeling, such as the emulation theory of [7], where anticipatory mechanisms are used for action control and learning; what is new in [20] is their use for Schemas selection.

While in reactive mechanisms behavior selection depends only on previous successes, in anticipatory selection preconditions are only used for the initial selection, while it is the success of prediction that determines if a schema continues to be used or it is substituted. During use, the internal model of the schema generates predictions that are matched against actual perceptions; if there is a significant mismatch the schema is discarded and a new one, generating more accurate prediction, is activated. The rationale is that if a schema is erroneously selected e.g. by visual information (say: grasp a cup instead of grasp a pencil), after an actual grasp the predicted consequences mismatch with observations. In this case schema control (for fine tuning) and selection (if there is significant mismatch) are a continuum; success of prediction determines schema tuning, switching and also blending, if we allow many Schemas to send motor commands, that have to be “fused”.

As a further consequence, the same Schemas are used both for acting and categorizing; this assumption is becoming a mainstream in situated approaches to cognitive science [3], at least after the discovery of mirror neurons [15], that gave new momentum to the simulative approaches.

Exploiting Inverse Models. While in the preceding paragraphs we only considered bottom-up, stimulus-based behavior selection, here we focus on top-down, goal-based selection and on planning by the means of expectations.

It should be noticed that even in the BNs there are “implicit” expectations: the post-conditions of the behaviors. However, there are some differences with the “explicit” expectations in schema mechanisms: they are neither used for action control, nor for explicit action selection. It is more appropriate to say that the structure of the BNs involves anticipation, since goal nodes are connected to behaviors realizing them in their post-conditions, and spread them energy: but these are anticipatory mechanisms, not representations, since they are not available for other processes. This architecture does not allow a true selection “because of the goal” (distinct from “because of the precondition/stimulus”); nor they endorse planning (as in inverse modeling). The power is in the topology of the network: this is the strong and weak point of action selection.

In Action Selection mechanisms goals can thus activate
actions only by the fact that there is a link between them (either by learning or by design, as in the BNs). On the contrary, in Schema mechanisms there are explicit expectations that can be exploited for goal-driven schema selection. Sequences of actions can also be selected: for example, Schemas in [20] have also an inverse model allowing to plan a sequence of actions leading from the current state to the goal state.

The Role of Expectations. Reactive mechanisms, once advocated for their quickness and adaptivity, are in the recent years challenged for their poor behavior, in particular in the presence of noisy or delayed sensors, and in uncertain or rapidly changing environments (exactly the case of robotics). Despite the success of some frameworks [17], it is becoming increasingly important to understand which are the limits of reactive mechanisms that are intrinsically due to their representational limitations (lack of internal states and in particular expectations). There are in fact many possible uses of expectations inside the Schema mechanisms.

Expectations allow a step-to-step control of action; the mismatch between an expectation and an observation can be used for fine-tuning and for schema shifting. Moreover, they allow goal-directed Schemas selection, distinguishing the fact that they realize a goal from other contingent conditions, such as matched preconditions or contextual opportunities. This is indispensable for truly purposive behavior. Forward models can be run (online or offline) simulating, “imagining” courses of actions that are not actual (and perhaps will never become true); this not only endorses deliberate (and not only reactive) planning, but also counterfactual reasoning, etc. There are also more subtle uses of this productive activity, such as recovering from failure. For example, by using the grasp cup schema for a pencil, the system loads it too heavily; other Schemas (better dealing with low weights) that run in simulation and do not have jet the control of action, can immediately recover from failure. Moreover, if a the first schema fails, the system can know that the expectation mismatch is about weight; even without having a ready schema it can recover by replanning. On the contrary, a failed reactive behavior does not give any cue for recovering or replanning; the system somewhat automatically changes behavior, because some conditions are changed; but with much less possibilities to do the right thing now, as highlighted in Section 4.

Hierarchies of Schemas. The problem of organizing Schemas hierarchically (from the more abstract to concrete level, e.g. from “take” to “grasp”) is poorly investigated. In fact, many authors focus only on behaviors at the sensorimotor level, thus a “flat” level of Schemas and compositionality (“approach_x” + “grasp_x”) are sufficient. But if we want to build Schemas hierarchies we have to address another difficulty: the granularity of the predictions. While predictions at the sensorimotor format can be directly compared with perceptions [20, 7], it is not clear how to compare predictions at different granularities and time scales (e.g. objects and situations with perceptions).

3 AKIRA

AKIRA [14, 21] is an open source, C++ multithread platform, inspired by the Society of Mind [12]. AKIRA is an hybrid symbolic-connectionist, as shown in Figure 1. According to the Pandemonium metaphor [8], the framework is called Pandemonium and the agents, called Daemons, have a variable amount of activation. Within the Pandemonium, Daemons are not isolated but related each other via an Energetic Network and to a central resource called the Energy Pool; both these structures are carriers of energy. Energy is very relevant: more energy for a Daemon means more computational resources, i.e. more priority to its thread. Thus, at the connectionist level, each Daemon is like a node in a network, having its activation and spreading it via edges. At the multi agent level, activation becomes the priority of the thread of each Daemon: thus, more active Daemons have more computational resources. In AKIRA complex operations can be performed by distributing the operations between many simple, interacting processes, each one carried on by a Daemon. Due to the energetic dynamics, Daemons become more or less active in a context dependent way, thus the “power and influence” of the processes they carry on change dynamically during the computation; the Energetic Model introduced later ensures that activation reflects relevance. For example, a successful behavior (or a Schema producing good predictions) is very relevant at a given moment and rewarded with more energy (and thus more control of action); or a more active Goal has much priority and is executed before less relevant ones.

The AKIRA Energetic Model. The energetic dynamics of the system [14] are designed for realizing the functionalities of Action Selection and Schema mechanisms. We assume that each Daemon is the “carrier” of a Schema; they are like nodes in the BNs, but they can perform more complex operations, store and pass any kind of variable or object via a Blackboard (as described in [14]).

A centralized pool of resources, the Energy Pool, sets an upper bound to the resources that the Daemons can tap. If a Daemon taps some resources, these are not available to the others until they are released; Daemons thus compete

---

\[2\] (6) is constructivist: it only assumes objects (synthetic items) if they are built up via interactions. Objects retain a perceptual nature, so expectations about them can always be matched against observation, like “virtual sensors”. [17] also uses “perceptual signatures” for anchoring.

\[3\] This feature is called the Energetic Metaphor in [9].
for energy (access to the Energy Pool) and resources (access to the effectors). An active Daemon thus inhibits the concurrent ones, preventing oscillations between conflicting behaviors. Spreading activation has a special meaning: it means losing it in favor of the receiver agent, as in the BNs, and permits similar dynamics: for example, a Daemon can spread activation to another one that is able to fulfill one of its needs, in order to successively exploit its results (a Schema can fuel another one realizing one of its PreConditions). However, AKIRA Daemons run in parallel and there is no centralized cycle of control (as in the BNs\(^4\)). The priority of the Daemons, according to the Energetic Metaphor, is directly translated into computational resources: this means that the most active Daemon(s) have not to be centrally selected, but run directly.

Since each task takes (real) time, more active Daemons can perform more operations; in the case of multiple Schemas sending commands to a Controller (as in the case study in Section 4) a more active Daemon can fire with an higher fire rate; this avoids “weighting” the commands.

Performing a task has a cost in energy, that has to be paid before executing and depends on its complexity: Daemons have to accumulate energy before performing their tasks; after execution they release energy that comes back to the Energy Pool and is ready to be tapped by other Daemons. Partially activated Schemas, even if have not the control of action, introduce an energetic pressure over the computation (and can run their forward models, as in Chapter 4); this mechanism however prevents too many Schemas to fire in parallel, or the same one to fire again and again.

Daemons also evolve links: when a Daemon succeeds in its task it sends the request to be linked, while when it fails it creates/reinforces a link to a random requesting Daemon. More active Daemons will thus have more incoming links; in the long run, Daemons that are active in the same span of time become more mutually linked, in an hebbian way.

The AKIRA Logic Model. The AKIRA Logic Model, even if currently specialized for fuzzy algebra, is not committed to any specific logical domain. There is in fact a meta-model algebraic description of the logical framework, having the following components:

- **Variable**: An association between a symbolic name and a set of possible values in an universe \(U\).

- **Functional**: A function describing the semantic of (a portion of) the universe of discourse, associating a descriptive value to each instantiated variable (e.g. in the fuzzy context, the Membership Function \(U \rightarrow [0, 1]\)).

- **Term**: The operational abstraction of the Variable. It is an association between a variable, a current instantiated value for that variable and two operators able to convert and re-convert unknown values into the admissible ones for the current universe. (e.g. in the fuzzy context, the Fuzzification and Defuzzification operators).

- **Operator**: A function from one or more Term to one or more Term. (e.g. fuzzy TNorm: \(\text{Term} \times \text{Term} \rightarrow \text{Term}\)

- **Rule**: An association between Terms. It creates an inference path between 2 Terms in the form of : \(\text{Term(precondition)} \rightarrow \text{Term(postcondition)}\).

- **RuleEngine**: A collection of rules. It answers to queries about the state and the values of the stored variables.

Daemons based on fuzzy rules and algebra can be exploited for implementing Schemas; this formalism makes it possible to store and pass fuzzy Terms and not only crisp values: preferences vs. decisions, as in [17].

Implementing Schemas in AKIRA. Each Daemon embeds one Schema (as shown in Figure 3). Schemas are organized in a hierarchical way and exploit fuzzy rules. They also produce explicit expectations and exploit them in many ways. Expectations, as shown before, can be used for action selection, by selecting schemas because their expected results satisfy one or more Goals. This use can be extended for planning (inverse modeling), too. Expectations can also be used as a measure of success: for a Schema, success is success of prediction. In AKIRA a successfull Daemon becomes more active and more linked by the other ones; and its commands have more impact (similar to [20]) either by assigning them a higher “weight” or by exploiting its faster fire rate. In Section 4 we compare the effects of basing schemas success on action success or on prediction success. The fuzzy rules can also include defining predicates about future states; for example: \(IF \: \text{x will be y } \THEN \text{do z}\).

Our approach to command fusion is similar to [17]: Schemas produce fuzzy commands (preferences) that are integrated by a controller, as shown in Section 4. The main differences are that the process is asynchronous, since the Schemas run in parallel; and the “weights” of their commands depend not on a “desiderability” parameter but on Daemons’ fire rate, reflecting their priority and saliency.

---

\(^4\)In the BNs for each cycle the more active module, if over a fixed threshold, is executed; otherwise the threshold is lowered and the cycle restarts.
4 A Case Study in Visual Search

Here we compare reactive and anticipatory strategies in a Visual Search task [19]. The goal is to find the red “T” in a picture containing also many distractors (green “Ts” and red “Ls”). The task is performed by many fuzzy-based Daemons (Schemas) operating concurrently and competing for resources in the Energy Pool. Daemons with different capabilities reside in different layers; some of them are sensitive to the environment, some others to the activity of the Daemons in the lower layers; they communicate via the Blackboard. The only visible part of the picture is the content of a movable spotlight, consisting in three concentric spaces having good, mild and bad resolution (a simplified model of human fovea). The Daemons are divided into five layers:

1. **Full Points Detectors** monitor one points of the spotlight each, e.g. the left corner. Daemons of the inner spotlight are more numerous and have more resources; the number and resources are lower in the central and outer spotlight. They ask to the environment: *is this point full or empty?* and notify the result.

2. **Color Recognizers** monitor the activity of the Full Points Detectors; if they find a “full” point, they ask e.g. *is this point red? or is this point green?* and notify the result.

3. **Line Recognizers** recognize sequences of points having the same color as lines. They concatenate on-the-fly (without a permanent memory) two or more consecutive points of the same color and notify their position.

4. **Letter Recognizers** use the information provided by the Line Recognizers for assembling “Ls” or “Ts” (even having different orientations) and notify their position.

5. **The Spotlight Mover** is a single Daemon receiving commands from all the other ones (e.g. move to the left) and consequently moving the center of the spotlight (and thus the area of influence of the Full Points Detectors).

The left part of Figure 2 shows the Daemons involved into the simulation; the layers are numbered. The simulation starts by setting a *Goal Daemon*, representing e.g. “Find the Red T” that spreads activation to the “Red Recognizer” and the “T Recognizer” (the arrows represent the links); it introduces a strong goal directed pressure: at the beginning of the task some Daemons are more active than others (dark and white circles). The dotted lines represent instead the monitoring activities performed by the Daemons: if a Daemon successfully matches, it notifies it to the Blackboard and some Daemons in the higher layers can exploit its activity. During the simulation, as the scenario changes, there will be more or less active Daemons influencing the overall process. Successful Daemons send commands to the Spotlight Mover in the form of fuzzy Terms; it dynamically blends them (there is also a certain inertia), and the spotlight traces a trajectory (starting from the center), as illustrated in the right part of Figure 2.

Each Daemon tries to move the spotlight where it anticipates there is something relevant for its (successive) matching operation. For example, if the Red Recognizer matches (or anticipates) something relevant to its task in a certain point, it tries to move there the spotlight; the Green Recognizer does the contrary (but with much less energy, because it does not receive any activation from the Goal Daemon). The Line and Letter Recognizers try to move the spotlight in the surroundings of already matched points in order to verify if there is a complete line or letter and its position. Here we only used built-in reactive planning: some fuzzy rules indicate which is the next interesting point to move on (e.g. *If two vertical points THEN move_in_vertical*). In a weak sense the mechanism is anticipatory, since it indicates the implicit expectation to find something interesting there; but there are not explicit anticipatory representations.

The simulation runs according to the AKIRA Energetic Model; Daemons exchange activation with the Energy Pool and the priority of their symbolic operations depend on their activation; they also evolve temporary links. Daemons in the upper layers have more default activation at the begin-
Does Anticipation Make any Difference? In the case study we mainly used reactive mechanisms; the only anticipatory element is in the command sent to the Spotlight Mover. Here we compare reactive and anticipatory strategies. We included in the Schemas a very simple forward model, realized by using Fuzzy Cognitive Maps [10], producing an explicit expectation about future perceptions, e.g., IF go \leftarrow THEN red point; or: IF go \leftarrow THEN Daemon \_x active. In the preceding (reactive) example Schemas are rewarded and gain control if they succeed in action (matching); now we reward them if they succeed in predicting: the rationale is that Schemas that predict better are more appropriate, as in [20].

Consider the example shown in Figure 3. In the picture, the shapes of a “T” and a reversed “L” differ for the pixel (0.0). If the spotlight moves from (3.1) to (0.1), both a \_T Recognizer and an \_L Recognizer recognize a line and send commands to the spotlight. Supposing that the \_T Recognizer is more active, its command is selected and the spotlight moves to (0.0). Now both schemas try to match (find a red point) and fail, since (0.0) is empty. If the criterion of selection is success of matching, both are punished. However, the \_T Recognizer was expecting to succeed, while the \_L Recognizer was expecting to fail: if the criterion is success of prediction, the \_L Recognizer is rewarded and gains action control.

In the new model Daemons in the same layer can have different forward models and become specialized: e.g. Letter Recognizers will specialize for the different orientations of “Ls” or “Ts”. Daemons in the different layers have different kinds of expectations. Expectations of the lower level Daemons have a perceptual nature and can be directly matched with perceptions; at the higher levels expectations are more “conceptual”, representing compound objects (there is a line there), abstract (this zone is empty or Daemon \_x is active) or vague ones (a letter is quite close); they are not directly matched with perceptions, but only with the results of lower level Daemons.

The anticipatory model is quicker and more accurate than the reactive one. The use of explicit expectations has many advantages: Schema selection is better, because success of prediction reflects well Schemas saliency. Schemas blending and shifting are smoother, too; and it is easier for the system to recover from failures. There are also advantages in computational terms, since Schemas can avoid really operating until their “simulated” activity indicates them as good candidates for action control. While in the previous example many Schemas run (and try to match) concurrently, now it is possible to test many Schemas operations in “simulated”, by comparing their expectations with perceptions. Schemas that have not the control of action can in fact (only) run their forward models; if they succeed in predicting they are rewarded with more energy and have the opportunity to take the control of action. As a result, the system is significantly faster in achieving the goal. An analysis of the paths (mean distance from the red T during time) shows also that very seldom the system “gets lost”. The computational requirements are lower, too, since the prediction criterion is
not only more accurate, but even more restrictive, and so only a very limited number of Daemons send messages to the Spotlight Mover.

5 Conclusions and Future Work

It is becoming popular the view that mind is an anticipatory rather than storing device. Replying to the slogan “intelligence without representation”, our experiments indicate that if representations are anticipatory, they are useful. Our work tries to synthesize the advantages of fuzzy logic and hierarchical structure of [17] and the anticipatory capabilities of [20]. Schema selection is anticipatory, based on success of prediction; command fusion and hierarchical control are based on fuzzy logic. Fuzzy logic is the ideal tool for a number of reasons: it simplifies Schemas design by providing linguistic primitives; it permits to use the same representation format for low-level tasks (e.g. pattern matching), high-level ones (e.g. reasoning requiring compositionality), and in the messaging; it addresses in a native way command fusion; it is robust in dynamic and uncertain domains.

Schemas’ anticipatory capabilities can be used for many other purposes, e.g. learning; step-to-step control of action; planning by inverse modeling. For these purposes, probably the case study provided here is too simple; the next step will be applying this architecture to real robotic domains, in order to verify if it scales up.

References

[21] www.akira-project.org