Architectural Mechanisms for Dynamic Changes of Behavior Selection Strategies in Behavior-Based Systems

Matthias Scheutz and Virgil Andronache
Artificial Intelligence and Robotics Laboratory
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
{mscheutz,vandrona}@cse.nd.edu

Abstract— **Behavior selection is typically a "built-in" feature of behavior-based architectures and hence not amenable to change. There are, however, circumstances where changing behavior selection strategies is useful and can lead to better performance. In this paper, we demonstrate that such dynamic changes of behavior selection mechanisms are beneficial in several circumstances. We first categorize existing behavior selection mechanisms along three dimensions and then discuss seven possible circumstances where dynamically switching among them can be beneficial. Using the agent architecture framework APOC, we show how instances of all (non-empty) categories can be captured and how additional architectural mechanisms can be added to allow for dynamic switching among them. In particular, we propose a generic architecture for dynamic behavior selection, which can integrate existing behavior selection mechanisms in a unified way. Based on this generic architecture, we then verify that dynamic behavior selection is beneficial in the seven cases by defining architectures for simulated and robotic agents and performing experiments with them. The quantitative and qualitative analyses of the results obtained from extensive simulation studies and experimental runs with robots verify the utility of the proposed mechanisms.**

I. Introduction

Agent architectures are blueprints of control systems of agents. They depict the arrangement of control components (i.e., where and how they are connected), and hence the functional organization of the overall control system. In behavior-based systems, a subset of these components is used to implement *behaviors*, i.e., sequences of actions of an agent that are intended to achieve a certain (sub-)task (such as a `follow wall` behavior in a robot that would make the robot move along a wall at a minimum distance). These components are typically also referred to as "behaviors". Since it is possible that two different behaviors (i.e., the components) require control over the same effectors (such as `turn left` vs. `turn right`, or `go to campus` vs. `go home`), behavior-based architectures need to provide mechanisms to decide which behavior gets to control the agent's effectors at any given time. This is the so-called *action-selection problem* [1], which we will in the following refer to as the *behavior selection problem*, given that behaviors (and not actions) are conceptually the basic elements in behavior-based architectures.[1]

The mechanisms used for behavior selection are typically fixed in behavior-based architectures. And while it may be possible to adjust some of the mechanisms' parameters to make them more adaptive, they cannot be changed altogether. A subsumption-based architecture [2], for example, cannot be changed into a schema-based architecture [3] at the level of the architecture. Switching among different behavior selection strategies, however, may be desirable or even required at times, to increase the system's performance, or to enable the system to achieve a given task in the first place. Looking at biological creatures it seems that many animals have the capability of modifying their behavior selection strategies, typically as a result of some learning process, which then generally leads to better performance at the given task [4]. Furthermore, they seem to be able to switch dynamically among different behavior selection strategies depending on which one leads to the best results. Hence, it would seem natural to allow for a dynamic change of behavior selection strategies in behavior-based systems as well.

There are, however, four problems with dynamic changes of behavior selection strategies: (1) current behavior-based architectures do not support multiple simultaneous behavior selection processes among which the system can switch; (2) it is, therefore, not clear how to structure an architecture to allow for dynamic switching among different behavior selection strategies, nor is it clear (3) which behavior selection strategy to include in the design of the system (among which it will then be able to switch), and (4) it is generally an open question under what circumstances it is beneficial to switch between two conflicting strategies. Note that the first problem is a prerequisite for the other three, since it is not possible to investigate the utility of dynamically switching among behavior selection strategies if they cannot be implemented together within the same architecture.

In this paper, we propose a solution to the first two problems and then show how to tackle the third and fourth. To solve the first two, we use the architecture framework APOC,

[1]Note that since behaviors are typically defined to be sequences of actions, a behavior could be comprised of only one action (and hence be as simple as a single action).

in which we can define and hence study any combination of behavior selection mechanisms.[2] Specifically, we provide translations of standard behavior selection mechanisms into the APOC framework and discuss a general way in which these mechanisms can be combined. To advance on the third and fourth problems, we first isolate seven cases, where dynamically switching among behavior selection strategies might be beneficial, and analyze the circumstances that could trigger these switches. We then verify that switching behavior selection strategies are beneficial in various instances of the seven cases in two simulation studies and three experiments with a robot.

The paper is organized as follows. First, we briefly review different behavior selection strategies and categorize them in terms of whether the behavior selection is achieved by structural features of the architecture or by specialized components. We then discuss seven scenarios in which dynamic changes of behavior selection strategies might be useful for behavior-based systems. After a brief overview of the architecture framework APOC, we show how standard behavior selection mechanism in behavior-based architectures can be formulated within APOC. The previous theoretical discussion of the potential utility of dynamically changing behavior selection strategies is then complemented by experiments with simulated and robotic behavior-based agents that show (1) how dynamic changes in behavior selection strategies can be implemented in behavior-based architectures and (2) that these mechanisms lead to significantly better performance in a variety of tasks compared to non-dynamic strategies.

II. Architectural Mechanisms for Behavior Selection

The "behavior selection" problem is a widely known problem among designers of behavior-based systems. Typically, what is meant by "behavior selection" is the process by which an agent decides what to do next, i.e., what *behavior* to perform. It is important to distinguish two senses of "behavior" in the context of behavior-based systems: (1) "behavior" as that which is observable when an agent performs a task, and (2) "behavior" as represented in an architectural component or mechanism (possibly consisting of multiple components and connections) that brings about or contributes to bringing about a behavior in the first sense. Behaviors in the first sense may or may not correspond to behaviors in the second sense (i.e., to architectural representations). It is, for example, possible that an agent exhibits a behavior in the first sense that is caused by the interplay of several behaviors in the second sense under particular environmental conditions (e.g., "wall following behavior" in a robot might result from the interplay of forward movement and a behavior that moves the robot back and reorients it slightly whenever the front sonars detect an obstacle within a certain distance).[3]

[2]APOC is a general framework for the analysis, comparison, and evaluation of agent architectures and has been developed in our lab over the last several years.

[3]Such behaviors in the first sense that result from the interplay of behaviors in the second sense and environmental conditions are often called "emergent". We will, however, refrain from using this term as it is fraught with conceptual problems [5] and, moreover, is not needed for the explication of this work.

Several terms are used in the literature to refer to "behavior selection", although they often have different connotations depending on when and where they are used. The most commonly used term is "action selection" [1], [6], although this term is less general than "behavior selection", given that not all behaviors might be easily decomposable into sequences of actions in any natural way (e.g., a schema-based implementation of formation maintenance in a team of robots might not lend itself to a decomposition of the coordination behavior in terms of discrete actions [7]). Also, the term "action" is sometimes reserved for simple, atomic processes and the term "behavior" is used to refer to more complex processes composed of simpler ones ([8]). Finally, "behavior selection" is sometimes used synonymously to "behavioral coordination" or "behavior arbitration", although there are cases where all three terms denote different processes (e.g., "behavior coordination" might be what drives "behavior selection" in particular circumstances, and "behavior arbitration" might or might not be involved in "behavior selection").

We will use the term "behavior selection" to refer to the process in a behavior-based architecture by which a set of components implementing behaviors (in sense 2) will lose control of the agent's effectors and another set of components implementing possibly different behaviors (in sense 2) will gain control of the effectors. By "controlling the agent's effector" we mean that the controller (i.e., the component controlling the effectors) and the effectors are connected via an exclusive link that allows for information flow from the controller to the effector. "Losing control", then, means that the information flow is interrupted or inhibited, "gaining control" means that information flow is established or enabled.

This definition of "behavior selection" is an instance of an *architecture-based* definition [9] in that it makes direct reference to architectural features (such as components and connections among them). It has several advantages over other definitions (e.g., [1]): (1) it avoids terminological problems (it is indifferent about what the components controlling the effectors implement, actions or behaviors); (2) it covers the whole process from one component controlling the effectors to another component gaining control (i.e., the selection and arbitration of actions or behaviors); (3) it covers architectures where behavior selection is *implicit* (in that structural features of the architecture determine which behavior is selected at any given time) as well as architectures where behavior selection is accomplished by special components; (4) it allows us to distinguish different mechanisms based on how links between components and effectors are interrupted/inhibited or established/enabled; and (5) it works for architecture specifications at different levels of abstraction (i.e., it does not depend on the degree to which an architecture is *schematic*). Note that this definition only covers "behavior selection" for behaviors in sense 2, and does not extend to "behavior selection" for behaviors in sense 1, where factors extraneous to the architecture might influence the observable behavior of an agent (e.g., a robot that exhibits a turning behavior in sense 1 due to a slanted surface after having driven straight without any architecture internal change to the selected "drive straight" behavior).

Based on this definition, we can now categorize different proposals for behavior selection mechanisms in a systematic way based on whether they are *cooperative* or *competitive*, *implicit* or *explicit*, and *adaptive* or *non-adaptive*, and discuss their different properties.

A. Cooperative versus Competitive Behavior Selection

Following [10], *cooperative behavior selection* requires mechanisms that achieve some sort of "behavior (or command) fusion", integrating information from different sources in the architecture before it is passed on to the effectors to produce the current behavior.

Examples are *voting mechanisms* [11], [12], *superposition techniques* [3], [13], [14], *fuzzy command fusion mechanisms* [15], [16], or *multiple objective behavior coordination* methods [17], [18], [19]. Note that in the extreme case cooperative mechanisms never actually "select" behaviors, because the set of components implementing the agent's basic behaviors is permanently connected to the integration component, which in turn is permanently connected to the effectors (as in simple schema-based architectures).

Competitive behavior selection mechanisms, on the other hand, require the selection of a behavior based on the result of some competition process among different components, possibly followed by the arbitration of the current behavior (if a behavior different from the current one was selected during competition). Examples are *priority-based* [2], *state-based* [20], [21], [22], [23], and *winner-take-all* competition mechanisms [6], [24], [25].

Competitive and cooperative behavior selection mechanisms are mutually exclusive in that the same set of behaviors cannot use a cooperative and competitive mechanism at the same time. However, it is still possible to use them together in the same architecture as long as there is a way to decide which selection mechanism gets to select behaviors at any given time (e.g., a hybrid architecture may consist of a cooperative behavior selection mechanism in the reactive layer, and a competitive mechanism in the deliberative layer, or vice versa).

We will present an example of an architecture with competitive and cooperative behavior selection in Section VI.

B. Implicit versus Explicit Behavior Selection

Behavior-based architectures can also be distinguished based upon how behavior selection is accomplished, i.e., whether it is *implicit* or *explicit*. Implicit behavior selection uses structural features of the architecture to select behaviors (e.g., through a hierarchical arrangement of control components as in the competitive behavior selection of subsumption architectures [2], or through the relative strengths of inhibitory and excitatory connections among components as in the cooperative example of Braitenberg vehicles [26]), while explicit behavior selection uses specialized components (e.g., as the summation component in schema-based architectures for cooperative behavior selection [3] or the global algorithm which chooses a module in the ANA architecture [6], [27] for competitive behavior selection).

Implicit and explicit behavior selection mechanisms are also mutually exclusive analogous to competitive and cooperative mechanisms. Yet, as with competitive and cooperative mechanisms, they can coexist in one architecture. A specialized component for behavior selection, call it D, for example, may be part of a hierarchical structure that determines the order in which behaviors get to control the effectors (e.g., based on inputs from the environment). If the behavior implemented in D is selected, then behavior selection proceeds according to the policy implement by D, otherwise it proceeds according to the hierarchical structure. Note that in this case behavior selection determined by the hierarchy has precedence over behavior selection determined by D. We will present an example of an architecture that combines both implicit and explicit mechanisms in Section VII.

C. Adaptive versus Non-Adaptive Behavior Selection

Since behavior selection is typically a *built-in* feature of behavior-based architectures (especially in architectures with implicit behavior selection like subsumption), it is not amenable to change in most architectures (e.g., [2], [6], [1], [28] and many others). We shall call behavior selection strategies that cannot be modified throughout the lifetime of an architecture instance *non-adaptive*.

Some architectures, however, allow for the adjustment or adaptation of behavior selection strategies [29], [30], [31]. We shall call such architectures *adaptive*. Adaptation in these systems generally consists of either modifying internal parameters which affect the choice of the behavior or modifying the set of available behaviors from which a choice can be made based on a triggering condition. Thus, any modification in the behavior selection strategy in these architectures is achieved within the context of a fixed strategy (and typically only parameters of a specialized component are modified).

One adaptive approach with explicit behavior selection proposed a "hybrid cooperative-competitive" behavior selection strategy [32]. Here, adaptation occurs through reinforcement learning, during which fusion parameters of decision components that integrate the outputs of two behaviors are learnt. Analogous to the above mechanisms, modifications of behavior selection strategies are achieved by varying two parameters of specialized integration components.[4]

Finally, there are also proposals for implicit adaptive mechanisms (e.g., [33]), where a variant of Hebbian learning in a neural network is used to learn the fusion parameters (i.e., the weights on connections from different behaviors that are combined to yield the overall motor output).

While the previous two dimensions are concerned with properties of the architecture layout, the distinction between adaptive and non-adaptive behavior selection is pertinent to the run-time instance of an architecture. Table I summarizes the proposed categorization of behavior selection mechanisms in common architectures along all three dimensions.

[4]Since the employed fusion mechanism only gives rise to competitive behavior selection in the limiting case (analogous to schema-based approaches), it will be classified as "cooperative," see also Section V-F.

TABLE I
EXAMPLES OF BEHAVIOR SELECTION STRATEGIES CLASSIFIED ALONG THE THREE PROPOSED DIMENSIONS: COMPETITIVE VS. COOPERATIVE, EXPLICIT VS. IMPLICIT, AND ADAPTIVE VS. NON-ADAPTIVE.

	Non-Adaptive	
	Competitive	Cooperative
Explicit	Agent Network [6] Bayesian Decision Analysis [20] Foka [22], Probabilistic methods [21]	Lorenz [34], Schema [3], DAMN [12], Balch [13], Jenkins [14] Multiple Objective Behavior Coordination [17], [18], [19] Fuzzy fusion [15], [16], Action Voting [35]
Implicit	Subsumption [2], Baerends [36]	Braitenberg [26]
	Adaptive	
	Competitive	Cooperative
Explicit	Alliance [30], Yamada [29]	Hybrid Coordination[32], BeCA [31]
Implicit		DAC [33]

III. WHY DYNAMIC CHANGES OF BEHAVIOR SELECTION STRATEGIES?

What is common to architectures implementing behavior selection strategies from any of the above categories is that they use each strategy *exclusively*, i.e., only one strategy is employed in any given architecture. And although architectures with adaptive behavior selection strategies can modify parameters (of their behavior selection strategy), thus changing the strategy over time so as to improve the performance of an agent in a given environment, they cannot use different parameter settings for different circumstances (e.g., they cannot learn a good strategy for one environment, and then learn a different strategy for another environment without "unlearning" the previous one). However, there are many situations, where switching among different behavior selection strategies, which we will refer to as *dynamic behavior selection*, can be advantageous for behavior-based agents (in particular, robots).

In the following, we consider seven different cases, in which dynamic behavior selection can be beneficial. Later, we will verify for all of these cases the utility of dynamic behavior selection in experiments with simulated and robotic agents (see Sections VI and VII).

A. Case 1: Selection of Sensory Information

Selecting a subset of all available sensory information for perceptual processing can be useful to reduce overall processing and lead to better performance, especially in robots, where sensory input is often unreliable. Sonar sensors, for example, often produce erroneous readings, cameras fail in dark environments, shaft encoder counts do not reflect the actual distance traveled on slippery surfaces, etc. Fortunately, many of these errors can be detected (at least to some degree) by means specific to each sensory modality (e.g., frequent wide variability in sonar values, low brightness levels in camera images, discrepancy between motor encoder counts and sonar readings relative to landmarks, etc). Behavior selection can then be dynamically adjusted so that it will be based only on the (more) reliable sensory inputs by automatically eliminating sensory input that is not reliable, either temporarily or permanently (e.g., in the case of a broken sonar sensor that always returns the same reading). By the same token, redundant or irrelevant sensor information, if detectable, can be ignored.

B. Case 2: Emergency Responses

Embodied agents will typically need fast mechanisms to deal with emergency situations. Such "global alarm systems" [37] have to be connected to the sensors and effectors in such a way that they can interrupt any behavior and take control of the agent's effectors. In other words, behavior selection in a system with alarms might be competitive at the level of the alarm mechanisms, but could be cooperative for other behaviors (i.e., when the alarm is not activated), thus retaining advantages of cooperative behavior selection. While alarms are typically directly associated with emergency behaviors, it is also possible to use alarms only to change the behavior selection strategy temporarily until the state that triggered the alarm has changed (e.g., a particular urgent goal has been accomplished).

C. Case 3: Infeasible Behaviors

Combining behaviors based on sensory inputs *via* a fixed mechanism can at times prompt the agent to attempt infeasible behaviors. A simple example is illustrated in Figure 1, where an agent (*A*) is trying to reach an item (black circle) in the environment, which is partially blocked by obstacles (white circles). While moving down and moving to the left are both feasible behaviors, their straight-forward combination (e.g., as it might occur in a schema-based system) produces a behavior (diagonal move) which is not feasible given the current state of the environment.

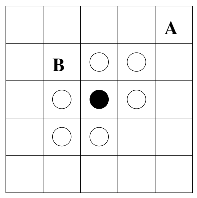

Fig. 1. Examples of an infeasible combination of feasible behaviors (`go left` and `go down` for agent A) and a feasible behavior from combination of infeasible behaviors (`go right` and `go down` for agent B) in order to get to the goal state (black circle). White circles denote obstacles.

A possible (and simple) solution to the combination problem is to temporarily change the behavior selection strategy: instead of combining behaviors, individual behaviors are given priority.

D. Case 4: Extending the Behavioral Repertoire

The converse problem to "infeasible behaviors" is a situation where no sequence of individual behaviors can accomplish an agent's task (e.g., as in the case of agent B in Figure 1, whose basic behaviors are limited to `move-forward`, `move-backwards`, `turn-left`, and `turn-right`). In that case, temporary combinations of behaviors (e.g., *via* behavior-fusion) might permit the agent to achieve the task (e.g., B would combine `move-forward` and `turn-right` to perform a diagonal movement).

E. Case 5: Mappings between Context and Behavior Selection Strategies

The last two cases illustrated that it is sometimes beneficial to switch between cooperative and competitive behavior selection. In general, such switches will be context dependent, and can, hence, be defined by a mapping between context and a set of behavior selection strategies (or parameters of behavior selection strategies), where the context will generally include states internal to the architecture in addition to sensory information. Specifically, instead of only having a single state variable that determines which one of two behavior selection strategy should become active, a mapping from a subset of all agent state variables (including sensory states) to behavior selection strategies generalizes this idea by allowing the agent to decide at any given moment in time which behavior selection strategy to use. For example, a mapping from sonar sensor activations and an inner state variable for determining the robot's current activity (e.g., wandering or target following mode) could be used to implement better obstacle avoidance by making the behavior selection strategy (e.g., cooperative or competitive behavior selection) dependent on the different sensory readings in wandering mode (e.g., competitive behavior selection could be used for any individually low sensor reading, but competitive behavior selection could be used whenever more than two sensors on different sides of the robot are below a certain threshold).

Such mappings between contexts and behavior selection strategies might be a way to represent solutions for whole classes of problems in a very compact way. These mappings could be either fixed or learned through experience, and might improve the agent's overall level of adaptivity without the need for a deliberative layer (in which a planner operates on a representation of the environment in order to find a solution to a particular problem, i.e., finding a plan representing a sequence of actions or behaviors to achieve the task).

In general, mappings from context to behavior selection strategy could benefit many applications. For example, a context-dependent preference mechanism could be implemented based on such a mapping if several behavioral sequences exist to achieve a particular task (e.g., a robot assembling cars might perform actions sequentially if run in "demonstration mode", even though these actions could be performed in parallel). Or a learning mechanism could try to establish the best behavior selection strategy for a given agent state either by systematically trying out all available behaviors (which would allow the agent in the above cases to associate particular environmental setups with behavior selection strategies) or by systematically varying the parameters of a behavior selection strategy in the given state until the best strategy is found (e.g., based on gradient ascent over some performance measure).

F. Case 6: Attentional Mechanisms

Attentional mechanisms in animals and humans serve the purpose of channeling sensory information and focusing processing on salient or important stimuli. They can also redirect processing resources dependent on the "focus of attention", for which they often require control of the agent's effectors as well (e.g., to be able to look in the direction of a loud noise).

Attentional mechanisms can be integrated into behavior-based architectures by allowing them to control dynamic switches among behavior selection strategies (which behavior selection strategy they switch to can either be fixed or learned).

G. Case 7: Learning Behaviors

Often an agent can determine how well it is doing at a given task based on some observable internal or external state (e.g., how much energy is left that can be used for locomotion or how close an agent is to a given goal state based on sensory input). In general, if a performance measure is available within the agent's architecture, it is possible to define a simple unsupervised, reinforcement learning component that can learn a mapping from contexts to behavior selection strategies based on the performance measure.[5] In the simplest case, the component could automatically switch at random among different behavior selection mechanisms during a "learning phase", recording context performance pairs, and then eventually always pick the best behavior selection mechanism based on the learnt associations for the given context. Alternatively, learning could be triggered by other components (or by repeated failure at achieving a task) and proceed in a systematic fashion (e.g., low cost behaviors are always tried first).

IV. The APOC Framework

The examples presented in the previous section are only a few from a much larger set of possible useful applications of dynamically changing behavior selection strategies, which can improve an agent's performance (as measured by quantifiable variables dependent on the task at hand).

Prerequisite to employing dynamically changing behavior selection strategies in an architecture, however, is a behavior-based architecture that allows for multiple behavior selection mechanisms to be present at the same time (during the lifetime of the architecture instance). None of the behavior-based architectures described in section 2 allow for the simultaneous presence of multiple behavior selection mechanisms at the architecture level. And while it may be possible to implement different selection mechanisms on top of existing ones (e.g., an explicit cooperative scheme such as a voting scheme on

[5]The performance measure can be implicit in the architecture or explicitly represented (e.g., as a numeric value).

top of an implicit competitive one such as subsumption), such implementations do not add new architectural mechanisms to an architecture, nor do they integrate existing mechanisms in a systematic and efficient way–both of which are goals of this paper. To be able to study different mechanisms within one architecture, we will use the agent architecture framework APOC, which provides a general formalism for the definition of agent architectures. Within this formalism, we can express all of the above discussed behavior selection schemes and define new architectures that combine any number of them within one architecture. Moreover, APOC provides a formalism to deal with run-time modification of an architecture (i.e., the instantiation of new and termination of existing components), which is crucial for the implementation of the full range of possible dynamic behavior selection strategies that we discussed previously.[6].

In the following subsections, we will provide an informal summary of the relevant properties of the APOC architecture framework. Section 5 will then show how behavior selection mechanisms from each of the seven non-empty categories in Table I can be expressed in APOC.

A. APOC *Components*

The APOC framework views agent architectures as networks of (typically heterogeneous) computational *components* connected by four types of *links* called *A*ctivation, *P*riority, *O*bserver, and *C*omponent links (hence the name "APOC"). Each *component* in APOC is an autonomous computational unit with activation and priority levels. It can receive inputs from and send outputs to other components via its links. Inputs (from incoming links) are processed and outputs (to outgoing links) are produced according to an *update function F*, which determines the functionality that the component implements, i.e., the mapping from inputs and internal component states (e.g., the current activation and priority values) to outputs and updated internal component states (e.g., new activation and priority values). The update function thus provides the specification for a computational process that, in an instantiated component, continuously updates the component's overall state. The particular algorithm for implementing the update function F has to be defined separately for each component type employed in an architecture.

Different from components in other formalisms such as schemas in *RS* [45] or the "augmented finite state machines" (AFSMs) in the subsumption architecture [2], an APOC component can also have an "associated process" (in addition to the computational process updating the state of the component), which it can *start*, *interrupt*, *resume*, or *terminate*. The associated process can either be a physical process external to the architecture (e.g., a process controlling the motors in a wheeled robot), or a self-contained computational process that takes inputs from the component and delivers outputs to it (e.g., a process implementing a blob detection algorithm that takes an image and returns all blobs of a particular color).

[6]More details about APOC can be found in [38], [39], [40], [41], [42], [43], [44]

One way of utilizing the APOC component model is to view it (i.e., the process updating the state of a component based on the definition in the update function F) as a *process manager* of the component's associated process. This construal makes it possible to separate information concerned with architecture-internal processing, i.e., *control information*, from other information (e.g., sensory information that is processed in various stages). For behavior-based architectures, this separation is particularly interesting as it makes the control flow involved in behavior selection mechanisms explicit. Figure 2 summarizes the basic structure of an APOC component.

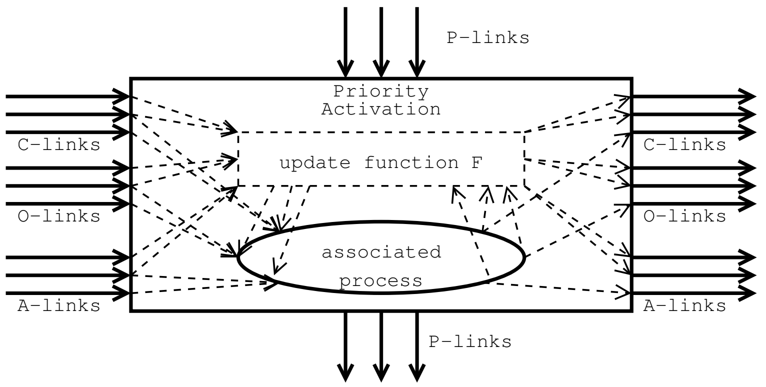

Fig. 2. The basic structure of an APOC component showing bundles of incoming and outgoing links of each of the four types as well as the priority and activation levels together with the update function F and the associated process.

Depending on how behaviors are modelled in APOC, i.e., whether they are expressed explicitly as part of the update function F or implicitly as implemented in the associated process, different architectural mechanisms in APOC can be used to model the behavior selection mechanisms employed in the literature. Typically, there are many ways in which any given mechanism can be expressed in APOC. Since we are interested in general mechanisms for dynamically modifiable behavior selection strategies, we will not make any assumptions about a particular rendition of a behavior in APOC (e.g., whether it is implemented in the associated process), but rather aim at general mechanisms that make use of only a minimal set of APOC mechanisms. As a result, we will exclude the priority mechanism, the involved P-links, and the associated process, and, hence, only briefly the describe the remaining three link types, which are used in the rest of the paper.

B. The A*-link*

A-links are activation passing links, such as those used for communication among components in many behavior-based architectures (e.g., [1], [6]). The function of an A-link is to transfer one unit of information (e.g., the activation of a component) from a source component to a destination component within a particular time interval (this "delay factor" can be specified separately for each A-link).

As part of this transfer, an A-link can perform an (optional) operation on the data (e.g., a weighting of a numerical value, or a filter operation). This allows not only for the transduction of data, but also for several learning mechanisms at the architecture level. For example, by modifying the operator on a link, neural-network-based learning (e.g., Hebbian and back-propagation) can be implemented. By allowing modifications to the delay factor on an A-link, associations can be learned and priming can be obtained for information retrieval as in the ACT-R cognitive architecture [46] (e.g., by decreasing transfer time between two components which are active at the same time).[7] Finally, many learning mechanisms (such as chunking, associative learning, etc.) can be implemented through run-time instantiation and deletion of architectural components in a generic way, as will be demonstrated for trial-and-error learning in Section 7.

C. The O-Link

The O-link provides a "non-intrusive" way for components to observe the state of other components.[8] Analogous to A-links, O-links have a delay factor associated with them. Different from A-links, information flow does not depend on the *observed component* (i.e., whether the component initiates an information transfer), but solely depends on the *observing component*.

D. The C-Link

Architectures in the APOC formalism are specified in terms of type relationships among APOC components, which provides a direct way of specifying abstract structures and modules. In an implemented architecture, instances of these types need to be created in the running virtual machine. To be able to model the relationship between component types and their instances, C-links are used to formally express the ability of a component to instantiate another component or link, or to remove an already existing instance of another component or link *at run-time*. Figure 3 shows a simple example of type specification and run-time component instantiation.

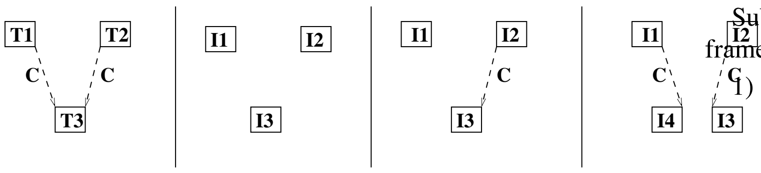

Fig. 3. From left to right: the type diagram of an architecture in APOC with three components, its initial instantiation, the state after an instantiation request by component I2 through a C-link, and its state after an instantiation request by component I1 through a C-link.

T1, T2, and T3 represent three component types, with types T1 and T2 utilizing the functionality implemented by type T3. Components I1 and I2 are instances of types T1 and T2 respectively, while components I3 and I4 are instances of type T3. When the architecture is first instantiated, components I1, I2, and I3 are instantiated. After the first explicit request for execution from I2, I2 is connected to the existing instance I3. The next request from I1 then results in the instantiation of I4.

V. Behavior Selection Mechanisms in APOC

There are many ways in which behavior-based architectures can be expressed in the APOC framework. A sequential architecture (with a single computational process), for example, could be viewed as a single APOC component, where the architecture's functionality is entirely encoded in the update function F. However, this is clearly not an interesting use of the APOC formalism. The utility of APOC lies in its ability to capture the interaction among multiple components that operate in parallel, as is typically the case in most behavior-based architectures. For such architectures, behaviors in sense (2) (i.e., as architectural components) can be represented as APOC components, and APOC links can be used to implement the information and control flow among these components (possibly involving additional components that implement behavior selection policies or behavior arbitration mechanisms).

In the following we will show how architectures with implicit and explicit, cooperative and competitive, adaptive and non-adaptive behavior selection strategies can be cast in the APOC formalism in a unified way. Figure 4 shows how each of the mechanisms described in this section can be expressed in APOC.

A. Non-Adaptive Implicit Competitive Behavior Selection: Subsumption

The *subsumption architecture* [2] is a layered system, in which individual layers work on individual goals concurrently and asynchronously. Layers consist of components, each component being the representation of a behavior. Each behavior is implemented as an *augmented finite state machine* (AFSM).

Subsumption architectures can be translated into the APOC framework in straightforward manner:

1) Augmented finite state machines (AFSM)–the basic components of subsumption architectures–are mapped onto APOC components by directly incorporating the input-output mapping defined by the state table of the AFSM into the update function F of the APOC component
2) The position of an AFSM in a subsumption architecture (i.e., the layer in which it resides) is modelled by activation values of APOC components
3) Message passing links (including environmental/data inputs) are modelled as A-links
4) Inhibitor connections are mapped onto A-links together with simple, specialized APOC components implementing AND gates to decide whether to pass on information or whether to block it
5) Reset connections are modeled as special A-link inputs to an APOC component (F ensures that the component resets its state when these lines are active)

[7] Several other learning mechanisms, e.g., *ART* network-based learning [47], or Q-learning can be defined within the APOC framework, see also [43].

[8] Components are observed to the extent that their state is observable–at the very least the activation and priority values can be observed in all components, but additional states might be observable dependent on the component's specification.

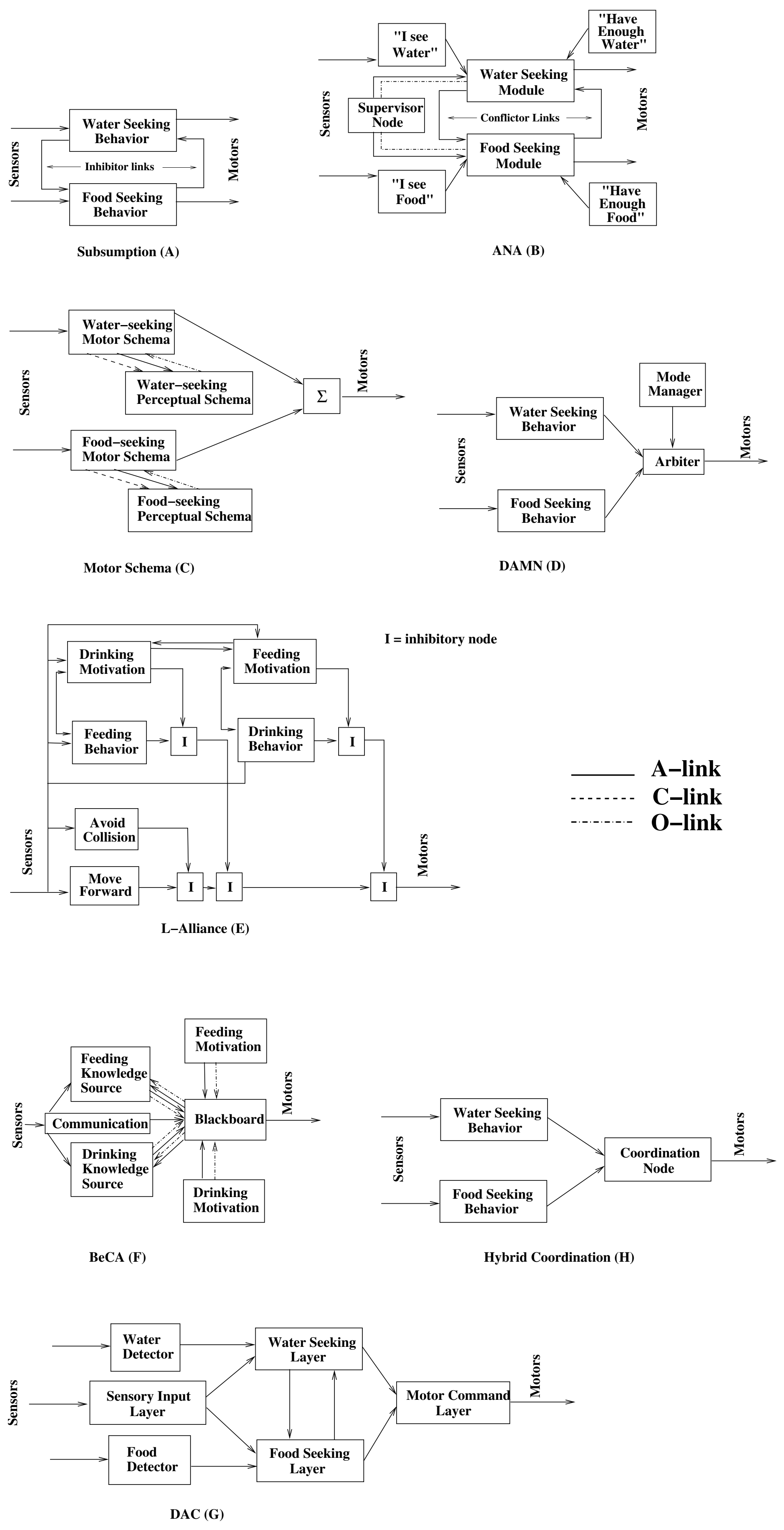

Fig. 4. Sample architectures with food and water seeking behaviors from each of the seven occupied categories in Table I modelled in APOC as discussed in Sections V:A through V:H.

6) Suppressor connections are also implemented as additional A-link inputs, which carry their originating component's activation value (i.e., position in the hierarchy)– according to this value the APOC component decides whether its outputs need to be suppressed

B. Non-Adaptive Explicit Competitive Behavior Selection: Action Network

Based on Minsky's "Society of Mind" [48], the *Agent Network Architecture* (ANA) is viewed as a set of *competence modules* [6]. Competence modules are connected through three types of links: successor, predecessor, and conflictor links. Two general conditions are imposed on the architecture: (1) all components in the network have the same activation threshold, and (2) if no active components are found in the network, the activation threshold is lowered by 10%.

Each component in the ANA architecture is modelled by a corresponding APOC component. A unique activation threshold has to be chosen for the entire network. The activation threshold modification function is then set to bring about a 10% decrease in threshold every time no active components are found.

A special component is implemented with observer and activation links to all other components. The function of this component is threefold: (1) to compute the average activation after each time-step, (2) send this activation back to each component, which can then perform its own normalization, and (3) observe active components within the network and decide if a lowering of the activation threshold is necessary.

Incoming activation links are then structured to form seven categories of ANA links: predecessor links (excitatory), successor links (excitatory), conflictor links (inhibitory), sensor links / activation by state (excitatory), goal links (excitatory), protected goal links (inhibitory), average activation link.

In addition, several global parameters of the ANA architecture need to be fixed:

1) π, the mean activation value after each timestep
2) Θ, the initial value of the global threshold,
3) Φ, the constant determining the weighting of environmental sensor inputs and successor links,
4) γ, the constant determining the weighting of goal inputs and predecessor links, and
5) δ, the constant determining the weighting of protected goal inputs and conflictor links

C. Non-Adaptive Explicit Cooperative Behavior Selection: Motor Schema-based Systems

In schema-based approaches [7], [49], motor schemas operate as "concurrent, asynchronous processes each of which instantiates a behavioral 'intention"' [50]. This principle can be directly expressed in APOC:

1) Each perceptual and motor schema is modelled by an APOC component, with the computation that defines the schema in the update function F of the component (or the associated process).
2) Motor and perceptual schemas are connected by A-, O-, and C-links.
3) Sensors trigger, *via* C-links, the instantiation of the respective perceptual-motor schema combination.
4) A component (the "summation component") performing the fusion part of all schemas for the effector output is needed, which is always instantiated and also connected to the output of motor schema components *via* A-links

D. Non-Adaptive Implicit Cooperative Behavior Selection: DAMN

DAMN [12] is a voting scheme in which a central arbiter tallies votes from all behaviors to select the best behavior. Modelling DAMN in APOC requires the following steps:

1) Each behavior is modelled by an APOC component, with the computation that defines the behavior in the update function, F of the component or its associated process, based on the complexity of the behavior.
2) Each motor controller is modelled by an APOC component, with the motor control being part of the update function, F. Thus, for each motor controller the update function will send commands to the controlled effector.
3) The arbiter is implemented in a specialized APOC component.
4) Votes are passed to the arbiter through incoming A-links.
5) Commands are passed to motor controllers through A-links.

E. Adaptive Explicit Competitive Behavior Selection: L-Alliance

In L-Alliance [30], adaptation is provided through the variation of several parameters during two phases: a "learning phase" and an "adaptive phase". In the learning phase, agents learn about their capabilities and those of their teammates without concern for task completion. In the adaptive phase, agents are still able to modify their behavior selection strategy, but the changes are directed strictly towards goal achievement.

One global parameter, the activation threshold, needs to be fixed for all components. All other parameters are local to each component (and can, therefore, be computed as part of the update function F):

1) *Sensory feedback:* a binary parameter indicating whether the behavior associated with the component is applicable to the current sensory configuration
2) *Inter-robot communication:* a binary parameter indicating whether another agent has sent a message regarding the behavior associated with this component.
3) *Suppression from active behavior sets:* a binary parameter indicating whether another behavior is currently active in the agent
4) *Learned robot influence:* a binary parameter whose value is based on a threshold function. In the active learning phase it indicates whether another agent is attempting the behavior associated with the current component. In the adaptive learning phase, it indicates that the agent's "boredom level" is above a threshold or that the agent believes that it can achieve a task in less time than another agent currently attempting that task.

5) *Robot impatience:* a real-valued parameter indicating the time the agent is willing to allow another agent's messages to influence its own motivation.
6) *Robot acquiescence:* two real-valued parameters indicating the time before the agent yields the task to another and the time before the agent gives up on the task when left on its own.

All the above parameters are then combined according to the update rule of the component (following the definitions of the L-Alliance architecture) to compute a motivational factor. Behaviors associated with components whose motivations exceed the threshold will then compete for execution based on a predefined behavior selection policy (such as "shortest job first" or "pick behavior at random").

F. Adaptive Explicit Cooperative Behavior Selection: Behavior Column Architecture

The Behavior Column Architecture (BeCA) is an architecture that consists of a network of blackboard-based cognitive and motivational components [31]. Each BeCA component is made up of subcomponents: internal (or "elemental") behaviors (which are knowledge sources), a blackboard, activity state registers of the internal behaviors, the interface/communication mechanisms, and the competition/control mechanism.

Either a whole BeCA component or each of the individual subcomponents can be modelled by an APOC component. In the latter case, internal behaviors are condition-action rules which map in a straightforward manner to update functions in APOC components. Internal behaviors connect to the blackboard via O-links (to observe the current state of the problem solving process) and A-links to place new solution elements to the blackboard. The activity state registers are connected to internal behaviors through O-links and observe their state. Environmental and system information is fed to a BeCA component through A-link connections to the communication component, which, in turn, is connected through A-links to the blackboard (to store environmental information) and to other components in the system. The competition/control mechanism is connected to each internal behavior through an O-link which observes the activation of the component, and an A-link, which can activate or inhibit the behavior.

The adaptive part of its behavior selection mechanism comes from parameter modifications in internal behaviors. Two quantities can be modified: the strength of connections between internal behaviors and conditions (the efficacy with which a behavior can satisfy internal perceptions, external perceptions, and drives), and a combination factor between internal and external signals. Both types of modification are performed as part of the update function F of the APOC component representing an internal behavior.

The limiting case of cooperative behavior selection is competitive behavior selection as described in the *hybrid coordination* scheme [32], [51], [52], which collapses the competitive-cooperative distinction.[9] In this approach, behaviors are grouped in sets of two, with one behavior being designated as *dominant*. The two behaviors send their outputs (consisting of a directional vector and an activation value) to a "hybrid component", which combines them in a manner preferential to the dominant behavior. A similar process is applied recursively to hybrid components until an overall unique output vector is generated as the overall behavior of the system.

[9]Note that viewing competition as a special case of cooperation is true of any schema-based approach, where gain values of all but one schema are (possibly only temporarily) set to 0.

The hybrid coordination architecture can be directly expressed in APOC: both behaviors and hybrid decision components are modelled as APOC components, where both desired action vectors and activation values are passed through A-links.

An important contribution of this approach is that the hybridization process is applied "on top" of an existing architecture. We will pursue a similar direction in the next section, where we introduce a generic architecture for the dynamic modification of behavior selection strategies. However, whereas hybrid coordination requires communication to consist of an activation value and a (directional) vector (thus making it difficult, if not impossible, to use with behavior selection mechanisms which use different information for communication among components or are not limited to vectors and activation values such as subsumption components), our approach will be designed to work with any kind of behavior representation.

G. Adaptive Implicit Cooperative Behavior Selection: DAC

The DAC system [33] uses a neural-network based approach to implement controllers for a robotic agent. In DAC, each behavior is implemented through a neural network. When any node in a layer has an activation value of 1, it automatically triggers a motor response. In the original system presented, the two behaviors are connected through an inhibitory element which gives preference to the avoidance behavior over the approach one.

Due to the use of neural networks for control, the DAC architecture exhibits implicit behavior selection. The adaptive component of DAC is also implicit, as DAC architectures adapt by modifying neural network weights as a result of environmental interaction. The changes are made through a modified Hebbian learning algorithm.

DAC can be modelled in APOC as follows:

1) Each sensor maps onto an APOC component.
2) The Target Detector and Collision Detector each map onto an APOC component.
3) Each neuron maps onto an APOC component.
4) All connections among nodes are performed through A-links, where links within a neural network implement, in their operator part, the learning mechanism.

VI. Dynamic Behavior Selection Mechanisms in Simulated Agents

The above examples illustrate how existing models of adaptive and non-adaptive behavior selection can be expressed within the APOC formalism in a unified way. As can be seen from these descriptions, a common characteristic of current architectures with adaptive behavior selection is that the adaptation process does not involve structural changes of the agent architecture. Rather, parameters of specialized components that function as arbiters are modified to change behavior selection strategies. However, architectural mechanisms that modify the layout and connectivity of an agent architecture allow for a wider class of adaptive mechanisms, and in particular, for switching among different behavior selection mechanisms, i.e., for dynamic behavior selection, as we will demonstrate in this section. Specifically, we will provide a mechanism for switching among different behavior selection strategies that works both for implicit and explicit behavior selection mechanisms, and thus allows for any of the eight possible kinds of behavior selection mechanisms we distinguished in Section II to occur at any place in the architecture.[10]

First, we motivate a mechanism for dynamic behavior selection that can switch between explicit cooperative and competitive behavior selection. We then show the utility of this mechanism in several simulation experiments with virtual agents that have to perform a two-resource foraging task in a simulated environment. Specifically, we compare agents that can switch between competitive and cooperative behavior selection strategies to agents that use a fixed strategy (either cooperative or competitive). Finally, we distil a generic architecture from this mechanism that allows for switching among multiple implicit and explicit behavior selection strategies.

A. Switching between Cooperative and Competitive Behavior Selection

Consider two behavior-based architectures with explicit behavior selection for the same task. The first, call it A_{coop}, uses j behaviors implemented in components $B_1, .., B_i, .., B_j$ and a cooperative behavior selection policy implemented in component P_{coop} (left in Figure 5), while the second, call it A_{comp}, uses $n - i$ behaviors implemented in components $B_i, .., B_j, .., B_n$ and a competitive behavior selection policy P_{comp} (on the right in Figure 5)–without loss of generality we assume that they have the $j - i + 1$ behavior components $B_i, .., B_j$ in common. Behavior components receive their input from the environment and their states are observed (via an O-link) by the policy components (P_{coop} and P_{comp}, respectively). The policy components then configure the fusion components Σ (via an A-link) according to the observed states of the behavior components based on the employed policy: in the case of cooperative behavior selection, a special signal is sent to the fusion component (indicating that it should combine inputs from all lines); in the case of competitive behavior selection, the input line number which is connected to the output of the selected behavior is sent.[11] The fusion components receive as input all outputs from the behavior nodes and either combine them all (according to the employed fusion method in the case of cooperative behavior selection), or select one based on the input from the policy component (for competitive behavior selection).

It is easy to see how a particular architecture with cooperative behavior selection, a schema-based architecture, for example, could be specified as an instance of A_{coop}: the A-links between the behavior components and the fusion component contain the gains (with which all behavioral outputs will be scaled) and the policy component configures the fusion component to always combine all outputs from the behavior components (alternatively, the gains can be stored in the fusion component or the behavior components as well).

Similarly, an architecture with competitive behavior selection like the subsumption-based architecture, for example, can be specified straightforwardly as an instance of A_{coop}: all components reside in the same layer, and the fusion component simply selects the output of the behavior component based on the policy component (which always selects only one of the behavior components, e.g., based on internal states of the behavior components).

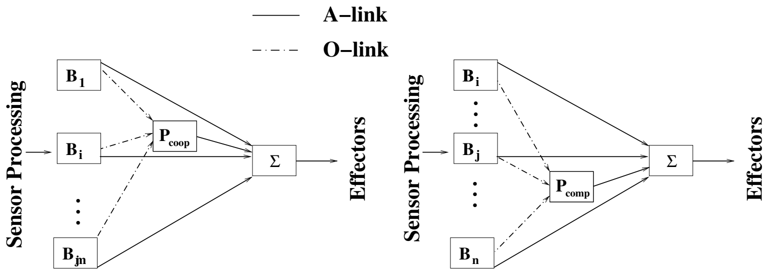

Fig. 5. Architectures with cooperative behavior selection (left) and competitive behavior selection (right).

Given both architectures, it is now possible to *merge* them into a combined architecture that allows for switching between the two policies (see Figure 6). The combined architecture contains all behavior components $B_1, .., B_n$, one fusion component Σ, the two policy components P_{comp} and P_{coop}, and an additional decision component D, which is used to decide when to switch between the two policies. Note that D can take environmental inputs as well as inputs from internal sensors to decide when to switch behavioral strategies (depending on the kind of employed dynamic behavior selection strategy as discussed in Section III).

We will now more closely examine two cases of switching between competitive and cooperative strategies in the context of a two-resource foraging task, where agents are in constant need of two resources *food* and *water*, which they have to find and consume regularly in order to survive. This kind of task has been used by several researchers to study and compare behavior selection mechanisms (e.g., [1]). For foraging then, we assume that all agents have behavioral components for FOOD-SEEKING (B_1) and WATER-SEEKING (B_2).

B. Switching from Cooperative to Competitive Behavior Selection

Consider a type of foraging agent with a combined architecture $Switching_1$, which by default uses cooperative behavior

[10]Note that only 7 of these eight kinds, as depicted in Table I seem to have been used in the literature so far.

[11]Note that other configurations are possible based on the policy, e.g., a subset of input lines could be selected.

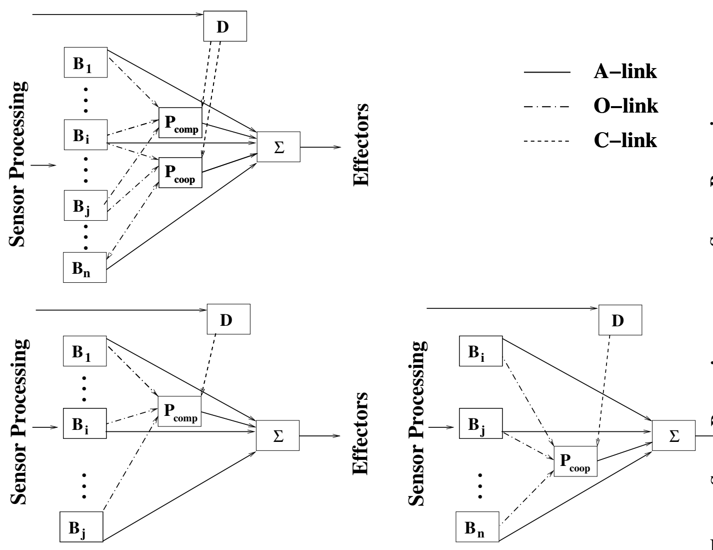

Fig. 6. An architecture for switching between cooperative and competitive behavior selection strategies.

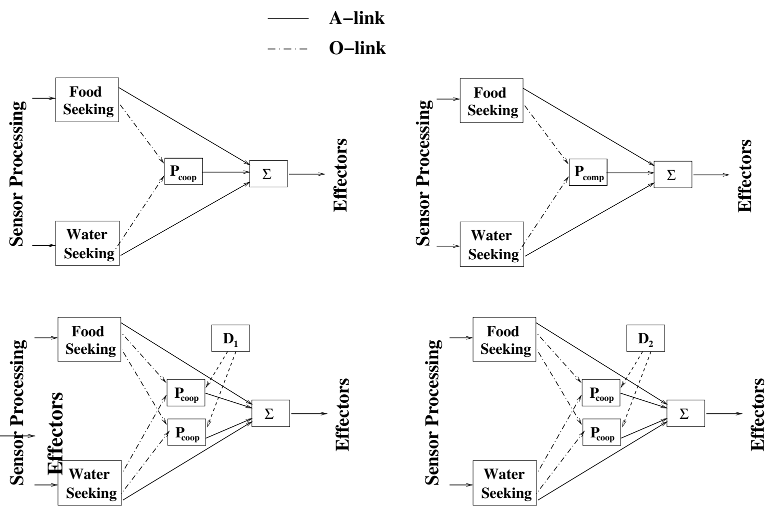

Fig. 7. Architectures for food- and water seeking agents with non-adaptive cooperative behavior selection $Cooperative$ (top left) and competitive behavior selection $Competitive$ (top right), and with dynamic switching from default cooperative to competitive $Switching_1$ (bottom left), and from default competitive to cooperative $Switching_2$ (bottom right).

selection P_{coop} as described above (e.g., using a schema-based approach), where the relative contribution of each behavior depends on its activation. The activation, in turn, reflects the agent's proximity to resources associated with the behavior (e.g., "food" for FOOD-SEEKING, see also [27]). Now consider the case, in which an agent's water level drops below a certain critical threshold. In this circumstance, a competitive behavior selection strategy P_{comp}, which ignores food and focuses entirely on water, is better than the default cooperative strategy. Hence, D_1 could implement an alarm mechanism (case (2) in Section III), which switches to competitive behavior selection, whenever internal energy or water levels drop below critical values. Depending on which level drops below the threshold first, P_{comp} will send a signal to Σ to use either the input from B_1 or B_2–if both drop at the same time, P_{comp} could pick one at random, or suppress the one that seems farther away based on environmental input (the architecture is shown on the bottom left in Figure 7).

C. Switching from Competitive to Cooperative

Now consider an agent that uses a combined architecture $Switching_2$, which by default is competitive (e.g., using a subsumption-style approach). Depending on which resource the agent needs more, food or water, either the FOOD-SEEKING or the WATER-SEEKING component will be selected by P_{comp}. Component D_2, in this case, does not have to monitor internal energy and water levels (as with the $Switching_1$ architecture), because behavior selection already guarantees that the agent will go after the resource it needs most (in terms of the $Switching_1$ architecture, behavior selection in $Switching_2$ achieves a "permanent alarm mode"). However, if on the way to a resource that is needed another close-by resource can be picked up (i.e., without a big detour), then it might be beneficial to do so. To accomplish this, component D implements a decision mechanism that allows the agent to turn on cooperative behavior selection if the activation of the behavior representing the most needed resource is much lower than the activation of the behavior representing the not-so-much needed resource (e.g., in the spirit of case (5) in Section III). For example, the influence of the WATER-SEEKING component is taken into account, even if food is needed, but the activation of the water-seeking behavior is over twice that of the food-seeking behavior, indicating that resources *might be* in the vicinity (the architecture is shown on the bottom right in Figure 7).[12]

D. Simulation Environment and Setup

To test the utility of switching between competitive and cooperative behavior selection mechanisms, we defined and implemented a basic agent model, which we then equipped with four different architectures. Agent types A_1 and A_2 have the two dynamic architectures $Switching_1$ and $Switching_2$, respectively, which are combinations obtained from merging the following non-adaptive architectures (as described in Section VI-B): agent type A_3 has a schema-based architecture with non-adaptive explicit cooperative behavior selection P_{coop} (top left in Figure 7), and agent type A_4 has a subsumption architecture with non-adaptive implicit competitive behavior selection (top right in Figure 7).

All agent types process sensory information, which comes from their vision sensors, by computing "force vectors" from the agent's current position to the perceived resources. For each resource type, i.e., *food* and *water*, a force vector is computed (F_{food} and F_{water}, respectively), which is the sum, scaled by $1/|v|^2$, of all vectors v from the agent to the objects of each type within sensory range (set to 300). The activation of each of the two behavior components is then determined in terms of the length of F (i.e., $|F_{food}|$ and $|F_{water}|$).

[12]Depending on the perceptual system this indication of the proximity of water or food may or may not be accurate.

In the schema-based architecture, the force vectors are mapped into motor space in the Σ component by the transformation function $T(x) = g_{food} \cdot F_{food} + g_{water} \cdot F_{water}$, where each g_{food} and g_{water} are the gain values determining the relative contribution of each behavior component to the overall agent's behavior. In the subsumption architecture, Σ suppresses the output of the behavior, which corresponds to the resource type that is less needed depending on internal energy and water levels.[13]

As environment we used an unlimited continuous surface, which can be populated with the four agent kinds as well as food and water sources, which pop up within a 800 unit by 800 unit area and disappear after a predetermined period of time, if not consumed by agents. Agents are in constant need of food and water as moving consumes energy and water proportional to their speed–even if they do not move, they will still consume a certain amount of both. When the energy/water level of an agent drops below a certain threshold ω, agents "die" and are removed from the simulation (they also die and are removed if they run into other agents). After a certain age $\alpha = 250$ (measured in terms of simulation cycles), agents reach maturity and can produce a variable number of offspring depending on their current energy and water levels (from 0 to 4).

All architectures for the simulation experiments were implemented using the *SimAgent* toolkit [53].

E. Experiments and Results

We performed several experiments, which we can categorize into four main sets depending on whether new resources appeared at a low or a high frequency at random locations (i.e., with a probability of 0.25 for food and 0.15 for water, or a probability of 0.5 for food and 0.3 for water per cycle, respectively), and whether environment were homogeneous or heterogeneous (i.e., consisted of one or more kind of agent).

In the first two sets, we examined the viability of each agent kind in low and high resource environments, respectively. For each of the 8 experiments part of these two sets, we ran 40 simulations of 10000 cycles each with 10 agents (of a given kind), 15 food and 15 water sources placed initially at random locations in the environment. As a measure of an agent type's performance, we used the number of surviving agents at the end of each simulation run, averaged over the 40 runs (to eliminate effects of initial positions). As can be seen from the results in Table II, each agent kind can survive on its own in both low and high resource environments, although the average survival rate for agent kinds other than A_1 is very low in low resource environments. While the performance difference among A_2, A_3, and A_4 is not statistically significant, the difference between A_1 and the other agents is highly significant ($p<0.001$ using a two-tailed t-test). Therefore, the dynamic behavior selection strategy implemented by A_1 is by far better than those implemented by the other agent kinds (including the dynamic architecture implemented by A_2 agents).

[13]Note that A_3 agents do not use their internal energy and water levels for behavior selection or any processing for that matter.

Sets 3 and 4, then investigated the potential of dynamic behavior selection in competitive environments. For this purpose, we ran again 8 experiments with environments initially containing two agents kinds of 10 agents each randomly placed in the environment, one with dynamic behavior selection, the other without (again, we used 15 randomly placed food and 15 randomly placed water sources). As in the homogeneous case, we averaged the number of survivors of each agent kind after 10000 simulation cycles over 40 runs. But different from the previous two sets, each set now contains 8 experiments, comparing each dynamic to each non-dynamic agent for low and high resource environments.

Table III shows the results for A_1 agents, Table IV shows the results for A_2 agents. As expected from the individual experiments, A_1 agents outperform both kinds with non-dynamic behavior selection in both low and high resource environments ($p<0.001$ using a two-tailed t-test). This time A_2 is also significantly better than A_3 and A_4 ($p \leq 0.05$) in all cases, which indicates that dynamic switching from competitive to cooperative behavior selection as implemented by A_2 agents is relatively better than any of its constituent non-dynamic behavior selection strategies (as there is a significant difference between them in heterogeneous environments while there is no significant difference in homogeneous environments).

Finally, it should be mentioned that the thresholds for switching from cooperative to competitive behavior selection in A_1 and for switching from competitive to cooperative behavior selection in A_2 could be learnt (e.g., through an evolutionary method like mutation across generations or through reinforcement learning within the life-time of each agent, where the expected future benefit obtained from switching behavior selection strategies is used to lower or raise the threshold).

F. A Generic Architecture for Dynamic Behavior Selection

The results of the simulation experiments demonstrate that switching between cooperative and competitive behavior selection can be highly beneficial. We can now generalize the above approach to a generic architecture for dynamic behavior selection, which accommodates implicit and explicit competitive and cooperative behavior selection mechanisms. In a first step, we allow for an arbitrary number of policy components in the architecture in Figure 6. In a second step, we add C-links from policy components to behaviors, allowing policy components to terminate and (re-)instantiate links from behavior components to the Σ component. This way implicit behavior selection obviates the need to reconfigure Σ (via an A-link from a policy node) and, moreover, reduces the resource requirements at the architectural level (by reducing the number of links that need to be updated). In some circumstances, it will be possible to allow policy components to even terminate (temporarily) the behavior component itself (e.g., in the case of A_1 agents when one resource drops below the critical level). The advantage of this move is again the reduction of the required computational resources in the running virtual machine (e.g., fewer memory requirements and shorter update times of the architectures, both of which are essential if computational resources are limited as is the case in embedded behavior-based systems like autonomous robots). Figure 8 shows a schematic of the generic architecture.

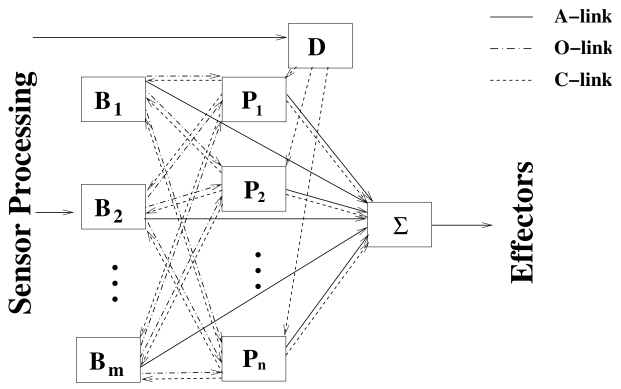

Fig. 8. A generic architecture for dynamic changes among multiple behavior selection strategies.

It is worth noting that this generic architecture necessarily leaves open the functional specification of the decision component D, which is an important strength of the proposed architectural approach to dynamic modifications of behavior selection strategies: there is no unique decision method that equally applies to all tasks and moreover fixes the best strategy for switching among behavior selection mechanisms. This can be best seen from the experiments with A_2 agents where dynamic behavior selection is not better than non-dynamic behavior selection in homogeneous environments, but is significantly better than non-dynamic behavior selection in heterogeneous environments. In other words, whether and to what extent a decision mechanism for switching among different behavior selection strategies is successful, does in general not depend on the architectural mechanism alone, but may also depend on external factors, such as the makeup of the environment, in which the task is to be performed: while there are cases (e.g., the A_1 agents) where an architectural mechanism X is "absolutely better" (i.e., in all considered environments), there are other cases (e.g., the A_2 agents) where mechanisms X is only better in a subset of the considered environments. Consequently, it is crucial for a generic architecture for dynamic behavior selection mechanisms (such as the one proposed here) to allow for different kinds of decision components depending on the given task and environment as otherwise fixing the functional specification of D will limit its generality.

VII. Dynamic Behavior Selection Mechanisms in Robots

Since behavior-based architectures are primarily intended for robotic agents, where real-time performance is critical, we demonstrate the utility of dynamic behavior selection for combinations of the seven cases (discussed in Section III) on a robot in three different sets of experiments:

1) *Experiment 1:* the robot learns to constrain the set of behaviors used for behavior selection based on relevant sensory inputs for a given environment (demonstrating *Cases 1,2, and 7*)
2) *Experiments 2:* the robot changes combinations and sequences of behaviors to overcome deadlocks (demonstrating *Cases 4 and 5*)
3) *Experiment 3:* the robot uses attentional mechanisms to detect a lack of progress towards its goal and changes its behavior selection strategy to overcome the problems (demonstrating *Cases 3 and 6*)

The task is the same for all three sets: the robot needs to locate and reach a target location ("orange ball") as quickly as possible in an obstacle environment without bumping into obstacles (i.e., boxes and chairs, see Figure 9).

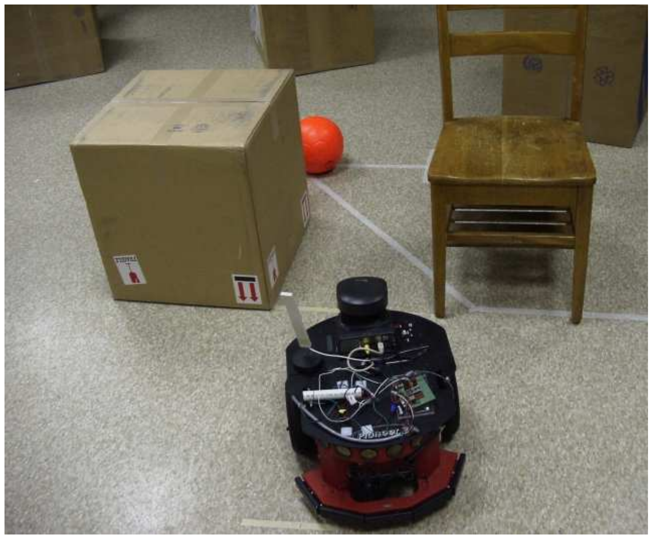

Fig. 9. The robot on its way to the target location (orange ball).

A. Experimental Setup

All robotic experiments were performed on a Pioneer 2DXE robot from ActivMedia, which was equipped with a SONY PTZ camera, an onboard framegrabber, and an onboard PC running Linux. The generic architecture used in all experiments (left in Figure 10) consists of a $Sensory\ Processing$, an $Action\ Processing$, and a $Supervisory\ Decision\ System$ part. The $Sensory\ Processing$ part processes sonar, visual, and bumper sensor information (received from the robot's 16 sonar, camera, and 10 bumper sensors, respectively) in components $b1$ through $b16$, bv, and bb and makes the information available to other parts of the system. For visual images, for example, the centroids of color blobs are computed and converted into directional vectors. The $Action\ Processing$ part implements all behaviors as well as the employed behavior selection strategies (a schema-based cooperative mechanism, by default), while the $Supervisory\ Decision\ System$ implements additional control methods dependent on the specific functionality to be tested in each of the three experiments. All architectures were implemented in *ADE*, a JAVA-based APOC development environment under development in our lab, in which

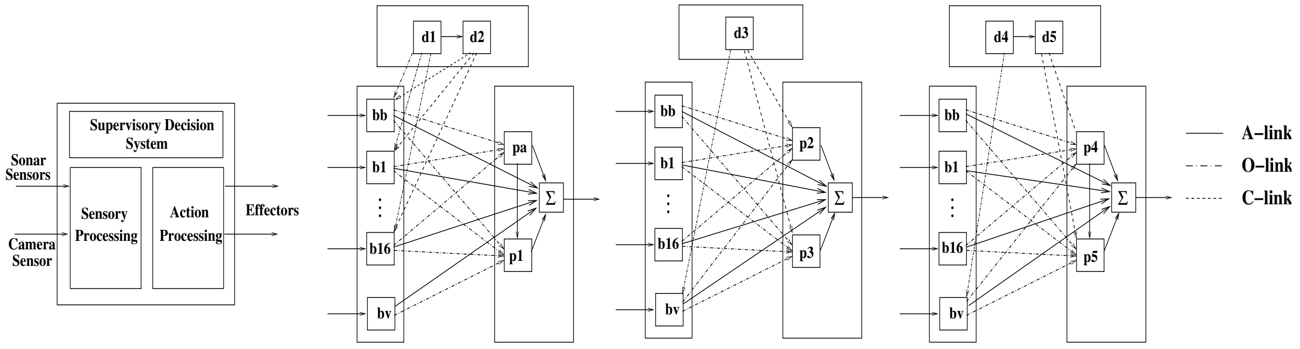

Fig. 10. Architectural description for the robotic experiments. From left to right: the generic architectural set-up and the individual architectures for experiments 1, 2, and 3.

architectures specified in the APOC formalism can be directly implemented [38], [39], [40].

B. Experiment 1

In this experiment, the robot repeatedly moves through the obstacle environment starting in the same area in an effort to learn how to traverse it as quickly as possible. On each run, components processing the largest sonar sensor values are systematically deleted for a fixed period of time in order to determine whether their absence makes a difference in the robot's progress towards the target location (as measured in terms of the size of the identifiable target location in the visual image). If no difference is detected, the sensor component gets permanently deleted for the particular route (*Case 1*), otherwise the component is re-instantiated. An alarm mechanism connected to the bumper sensors ensures that the robot will not inflict any damage if relevant sensors get temporarily deleted (*Case 2*). Over time, the robot learns to remove irrelevant sensor components for a given route, which frees up system resources and leads to better performance at traversing the environment (*Case 7*).

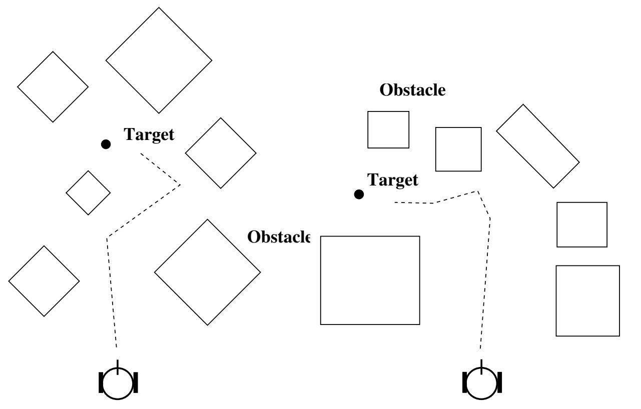

Fig. 11. The two scenarios used for the first robot traversing experiments.

The architecture implementing the cooperative behavior selection strategy that dynamically restricts and widens the pool of contenders is shown in Figure 10 (second from the left). The $Supervisory\ Decision\ System$ structure consists of two APOC components, $d1$ and $d2$. $d1$ collects sonar information and forms a composite "picture" of the environment. $d2$ analyzes the environment, decides which components are expendable and deletes them from the architecture. The $Action\ Processing$ consists of two components: $p1$, which sets up Σ to compute an overall direction towards the goal, and an alarm mechanism pa, whose function is to prevent the robot from hitting obstacles and which, therefore, overrides $p1$ when the robot is in a perilous situation. It should be noted that although sensor components are deleted for the duration of a particular route, they can be re-instantiated if another task requires them. Thus, the experiment illustrates one way of learning fixed action patterns (such as the egg retrieval of the greylag goose) at the architecture level, where only the minimal sensory information necessary to perform the action pattern is processed.

Figure VII-B shows two typical scenarios for this experiment. For each of these two scenarios, we first performed 10 runs without learning to get the baseline results for the traversal times. Then we performed 10 runs each allowing the system to delete at most one sensor component and another set of 10 runs each, in which the system was allowed to delete up to two components. For all runs, we measured the elapsed time between starting the traversal and arriving at the target location. Table V summarizes the results.

TABLE V
THE AVERAGE TIMES TO TARGET AND CONFIDENCE INTERVALS FOR EACH OF THE 10 RUNS WITH 0, 1, AND 2 COMPONENTS DELETED FOR THE TWO TRAVERSING SCENARIOS OF ROBOT EXPERIMENT 1.

	Scenario 1		Scenario 2	
	Mean	Conf.Int.	Mean	Conf.Int.
Baseline	37.06	(30.50,43.61)	21.27	(17.84,24.70)
1 deleted	24.38	(21.68,27.07)	17.64	(16.98,18.29)
2 deleted	20.55	(20.33,20.76)	18.92	(18.00,19.84)

As can be seen from the results, the dynamic architecture modification is beneficial in both scenarios. The traversal times in the runs with one component deleted are in both cases significantly lower than in the two baseline runs ($p<0.05$ using a t-test). Hence, both task performance and computational resource utilization are improved by dynamical behavior selection (as fewer components need to be instantiated in memory and updated). Moreover, in the first scenario, the run with two deleted components is also significantly better than the run with one deleted component, thus further improving both performance and reducing resource requirements. However, the increased speed and reduced number of components can also lead to collisions that otherwise would not have occurred. The robot bumped in 6 out of the 10 runs with two deleted components into obstacles in the first scenario (triggering the alarm mechanisms connected to the bumper sensors to back it up). In the second scenario, on the other hand, all runs were completed without collisions. And even though the difference between the baseline runs and the runs with two deleted components is not significantly different (in fact, the traversal time is slightly higher in the case of two deleted components), there is again a reduction of the computational requirements. Therefore, depending on the structure of the environment and safety concerns, it might be necessary to determine the largest number of components that can be deleted without risking collisions, which can be done by comparing the outcome of several "trial-and-error" runs for different numbers of deleted components.

It is worth pointing out that in the first scenario only 6 out of the 10 baseline traversals completed. In the other 4 runs, the robot got stuck in a "local minimum", which is quite common for schema-based cooperative behavior selection approaches. One effective solution that was proposed to overcome these minima is to add an "avoid-past schema" to the architecture [54]. This schema effectively keeps track of all recent locations the robot has visited and adds a repulsive force to the overall force vector for each of these locations, which eventually causes the robot to be repelled by them, thus overcoming local minima. Unfortunately, the "avoid-past" schema is computationally and architecturally demanding in that past locations have to be stored and their representations need to be updated relative to the robot's movements, which requires at the very least additional (typically proprioceptive) input to determine (approximately) the robot's movement between two architecture update cycles. The proposed dynamic behavior selection mechanism, on the other hand, does not need architectural extensions to achieve the same effect in the above scenarios. On the contrary, it *reduces* the instantiated architecture by (temporarily) deleting those components that are not necessary for and potentially interfere with the completion of the traversal.

C. Experiment 2

In this experiment, the robot eventually encounters a narrow passage on its way to the target location, which it has to pass (see Figure 12). If the target is placed in the center after the passage (left in Figure 12), the robot will typically be able to go through using its standard cooperative behavior selection mechanism. If, however, the target is placed off to one side (partly hidden behind an obstacle as in the right two figures in Figure 12), the robot will either fail to go through the passage or lose track of the ball. Various factors (such as wheel slippage, noise in the sonar values when it gets too close to an obstacle, and others) cause the robot to be unable to center in on the ball directly. By dynamically reconfiguring the relative contributions of different behaviors (in this case motor schemas) based on the context, i.e., whether or not the robot sees the target (*Case 5*), it is possible to extend the robot's behavioral repertoire and recombine behaviors (*Case 4*) to overcome the deadlock.

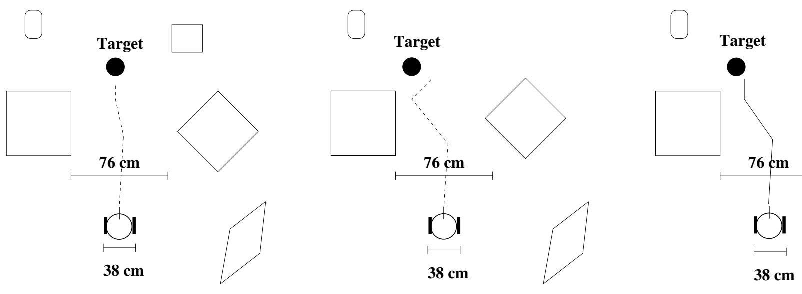

Fig. 12. The robot's trajectory for different setups for robot experiment 2: with centered target behind the passage way (left) and the target off to one side, without (middle) and with supervisory control (right).

The architecture implementing the context-dependent dynamic modifications of the relative contributions of each behavior in the cooperative behavior selection strategy is the second subfigure from the right in Figure 10. The $Supervisory\ Decision\ System$ consists of a single component, $d3$, which processes the visual information available to the system and chooses which of two decision components will be instantiated in the architecture. The $Action\ Processing$ section thus consists of two decision components: one component, $p2$, configures Σ to use a constant attraction to the ball, while the other, $p3$, increases the attraction of the ball over time (for a given time interval, after which it is reset to its original value). The later implements an adaptive behavior selection strategy, as it will change the relative contribution of ball attraction and obstacle repulsion until either the robot can advance through the passage or the time limit of the adaptation process is reached (i.e., the point at which it is assumed that adaptation is not able to resolve the impasse).

Again, we performed 10 baseline runs with the non-dynamic architecture, where the ball was placed off-center on the other side of the passage and the robot failed to traverse the passage in all 10 runs. With dynamic behavior selection, the robot only failed to traverse the passage in two out of 10 trials. This shows that the addition of components $d3$ and $p3$, which establish the context-mapping and allow for dynamic switches among configurations of gain values for motor schemas, is another effective way to overcome local minima. Note that this benefit can be achieved using minimal computational and architectural requirements. Moreover, the dynamic behavior selection mechanisms can be added to an existing schema-based architecture with only minor modifications.

D. Experiment 3

The third experiment uses the setup of experiment 2, except that the opening between the two obstacles is now too small for the robot to be able to pass through safely. On a typical run, the robot will proceed towards the ball using the standard cooperative mechanism. Once the obstacles are reached, the robot will be locked into oscillatory behavior regardless of where the ball is placed on the other side (i.e., it will move towards and away from the obstacles without making any progress towards the target location). In this case, even the dynamic behavior selection mechanism from the above experiment that helped align the robot properly for traversal will fail, as the passage is too narrow for traversal and the repulsive forces exerted by the obstacles consequently cause the robot to turn away.

This is were an attentional mechanism proves useful (*Case 6*): by *noticing* that no progress is made towards the target (comparing different perceptions of the target and robot movements over time), the mechanism switches behavior selection to competitive mode to avoid impossible actions (*Case 3*) and turns off visual input, which effectively leads the robot to move around the obstacle. Once the target reappears, the supervisory mechanism re-enables visual input and full cooperative behavior selection.

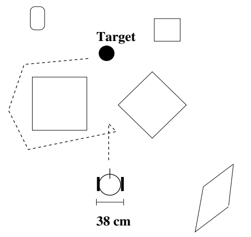

Fig. 13. The environment for robot experiment 3 showing the robot's path with dynamic behavior selection.

The architecture implementing the performance-based attentional control, which can switch among different behavior selection strategies, is shown in the rightmost subfigure of Figure 10. Here, the $Supervisory\ Decision\ System$ consists of two components: $d4$, which observes the vision sensor and identifies the contents of the image seen by the robot, and $d5$, which monitors the progress made by the robot towards achieving its goal. If the progress is unsatisfactory, $d5$ interrupts the decision component which is currently active ($p4$) and starts the inactive one ($p5$).[14] Thus, $Action\ Processing$ consists of two components with different characteristics: $p4$ is a component which uses cooperative decision making to choose a behavior, whereas $p5$ is a competitive decision maker.

As with the other experiments, we performed 10 baseline runs without dynamic behavior selection for both the basic schema-based architecture and the architecture augmented with the mechanism from experiment 2. In both cases, the robot got stuck at the entrance to the passage without ever getting around the obstacles. When dynamic behavior selection was used, the robot was able to move around the obstacle in all 10 runs, showing that attentional mechanisms that can detect deadlocks are useful for controlling dynamic changes of behavior selection strategies.[15]

VIII. Discussion

Currently, there are no systematic studies available that investigate the potential of dynamic modifications of behavior selection mechanisms, which may be partly due to the fact that behavior-based architectures do not easily or not at all support the coexistence of multiple behavior selection mechanisms. This was the main reason for using the APOC framework, in which the proposed mechanisms for behavior-based architectures with different behavior selection strategies can be described and discussed in a unified way.

More importantly, the APOC formalism allows us to define new *efficient integration methods* of behavior selection strategies within one architecture that are not possible in any of the discussed design methodologies (e.g., a subsumption-based architecture does not support the run-time instantiation of schemas based on sensory information; conversely, a schema-based architecture does not support the hierarchical layering possible in subsumption architectures).

"Efficient integration" here is a crucial factor, as it may be possible in some cases to achieve dynamic changes among behavior selection mechanisms simply by running several architectures (each of which employs a particular behavior selection strategy) in parallel and using another architecture solely to decide which of these architectures gets to control the agent's effectors.

Consider, for example, an agent that is part of a group box-pushing task in an environment in which it needs to feed and avoid predators. This is a combination of the tasks in [1] and [29]. The two systems, a modified free flow hierarchy and a case-based system, can operate independently in an agent, requiring merely an additional component to decide which behavior (as recommended by the two subsystems) to use. Figure 14 shows a schematic of such a system, which could be implemented by running three independent architectures in parallel:

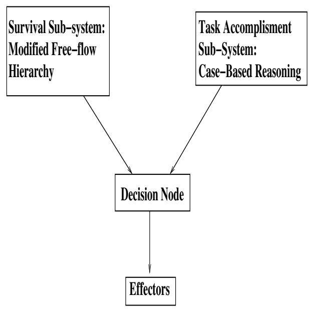
Survival Sub-system: Modified Free-flow Hierarchy
Task Accomplisment Sub-System: Case-Based Reasoning
Decision Node
Effectors

Fig. 14. An architecture with two different behavior selection mechanisms obtained by combining two independent behavior selection architectures via a separate decision system, thus duplicating behavior representations and behavior processing.

[14]Note that while in the present setup the threshold for switching from cooperative to competitive behavior selection is fixed, it could, as in the simulation experiments, be learnt using reinforcement learning.

[15]Note that while the robot will typically be able to move around obstacles in the environments used for our tests, there are environments where it will get stuck, e.g., because moving around the blocking obstacle is not possible. In that case, the supervisory component will consider the attempt to surround the obstacle a failure and switch back to the standard behavior selection strategy, thus effectively restarting the target localization process.

Obviously, such a setup is less than ideal: it will not only be resource-intense in terms of computational time and space (and may, therefore, not be feasible on an autonomous robot), but it will also reduce the flexibility of adjusting architectural parameters across subarchitectures and increase the complexity of the architectural design if multiple behavior selection strategies are to be employed at different places in the architecture at the same time (e.g., competitive selection at the lowest motor control layer, cooperative selection at an intermediate behavior control layer, and a mixture of both at an even higher reflective layer). By implementing the systems within one architecture, it is possible to reuse all components implementing behaviors (without duplicating them as in Figure 14), while only adding the structural mechanisms required for each individual behavior selection strategy together with the decision component that switches between behavior selection mechanisms (as proposed in Section 6). By overlaying several structural features, each of which implements a particular behavior selection strategy, it is possible to build complex systems that can achieve a high degree of adaptivity along several dimensions at the same time. For example, by integrating the structural features of the behavior selection mechanisms of the three robotic architectures presented above it is possible to define a combined architecture that inherits all properties of the individual systems (e.g., the robot can learn to reduce sensory information, while being able to drive through narrow passages, and giving up attempts to pass them if they are too narrow in favor of driving around the obstacle).

IX. Conclusion

In this paper we introduced a taxonomy for behavior selection mechanisms that allows for a distinction between adaptive or non-adaptive, explicit or implicit, cooperative or competitive behavior selection strategies, each of which seems advantageous in different circumstances. We argued that behavior-based architectures should allow for a combination of different behavior selection strategies to be able to study and compare the properties of different mechanisms and furthermore to allow agents to choose (or learn to choose) at run-time the strategy that works best in a given situation. Specifically, we isolated seven possible scenarios for the application of dynamic modifications of behavior selection strategies and demonstrated the utility of architectures with dynamic behavior selection for all seven cases in experiments with simulated and robotic behavior-based agents.

These demonstrations are intended to lead the way for the investigation of additional, more complex architectural mechanisms for dynamic behavior selection to explore and pinpoint the kinds of situations, in which dynamically modifiable behavior selection mechanisms play out their strengths to the fullest. We believe that such studies will ultimately have to involve investigations of the tradeoffs between computationally expensive, highly versatile deliberative systems, which can implement a large class of behavior selection strategies by virtue of general reasoning and planning mechanisms based on explicit representations of the task domain, and much cheaper, architectural mechanisms that are by far not as general as deliberative ones, but sufficient for the given tasks and, more importantly, computationally much less demanding.

References

[1] T. Tyrrell, "Computational mechanisms for action selection," Ph.D. dissertation, University of Edinburgh, 1993.

[2] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, vol. 2, no. 1, pp. 14–23, 1986.

[3] R. C. Arkin, "Motor schema-based mobile robot navigation," *International Journal of Robotic Research*, vol. 8, no. 4, pp. 92–112, 1989.

[4] K. Lorenz and P. Leyhausen, *Motivation and Animal Behavior: An Ethological View*. New York: Van Nostrand Co., 1973.

[5] W. Wimsatt, "Aggregativity: Reductive heuristics for finding emergence," *Philosophy of Science*, vol. 64 (supplement), no. 4, pp. 372–384, 1997.

[6] P. Maes, "How to do the right thing," *Connection Science Journal*, vol. 1, pp. 291–323, 1989.

[7] R. C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.

[8] R. Cooper and T. Shallice, "Contention scheduling and the control of routine activities," *Cognitive Neuropsychology*, vol. 17, no. 4, pp. 297–338, 2000.

[9] A. Sloman and P. Terrier, "Interview on human-like agents and affect," *EACE Quarterly (European Association for Cognitive Ergonomics)*, vol. 3, pp. 23–30, August 1999, 2.

[10] P. Pirjanian, "Behavior coordination mechanisms - state-of-the-art," Institute for Robotics and Intelligent Systems, School of Engineering, University of Southern California, Tech. Rep. IRIS-99-375, 1999.

[11] K. Rosenblatt and D. Payton, "A fine-grained alternative to the subsumption architecture for mobile robot control," in *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks*, 1989.

[12] J. K. Rosenblatt, "DAMN: A distributed architecture for mobile navigation," in *Proc. of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents*, Stanford, CA, 1995. [Online]. Available: CITESEER.NJ.NEC.COM/ARTICLE/ROSENBLATT97DAMN.HTML

[13] T. Balch and M. Hybinette, "Social potentials for scalable multirobot formations," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2000)*, San Francisco, CA, 2000. [Online]. Available: CITESEER.NJ.NEC.COM/BALCH00SOCIAL.HTML

[14] O. Jenkins, M. Mataric, and S. Weber, "Primitive-based movement classification for humanoid imitation," in *Proceedings, First IEEE-RAS International Conference on Humanoid Robotics (Humanoids-2000)*, Cambridge, MA, September 2000. [Online]. Available: CITESEER.NJ.NEC.COM/JENKINS00PRIMITIVEBASED.HTML

[15] J. Yen and N. Pfluger, "A fuzzy logic based extension to payton and rosenblatt's command fusion method for mobile robot navigation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 6, pp. 971–978, June 1995.

[16] A. Abreu and L. Correia, "Fuzzy behaviors and behavior arbitration in autonomous vehicles," in *Portuguese Conference on Artificial Intelligence*, 1999, pp. 237–251. [Online]. Available: CITESEER.NJ.NEC.COM/ABREU99FUZZY.HTML

[17] M. K. Starr and M. Zeleny, "Mcdm-state and tuture of the arts," in *Multiple Criteria Decision Making*, ser. Studies in the management sciences, M. K. Starr and M. Zeleny, Eds. North-Holland Publising Company, 1977, vol. 6, pp. 5–30.

[18] T. Huntsberger, H. Aghazarian, E. Baumgartner, and P. Schenker, "Behavior-based control systems for planetary autonomous robot outposts," in *Proceedings of Aerospace 2000*, Albuquerque, NM, 2000. [Online]. Available: CITESEER.NJ.NEC.COM/HUNTSBERGER00BEHAVIORBASED.HTML

[19] P. Pirjanian, T. Huntsberger, A. Trebi-Ollennu, H. Aghazarian, H. Das, S. Joshi, and P. Schenker, "Campout: a control architecture for multirobot planetary outposts," in *Proceedings of the SPIE Conference on Sensor Fusion and Decentralized Control in Robotic Systems III*, Boston, MA, November 2000. [Online]. Available: CITESEER.NJ.NEC.COM/PIRJANIAN00CAMPOUT.HTML

[20] S. Kristensen, "Sensor planning with bayesian decision theory," Ph.D. dissertation, Aalborg University, 1996.

[21] S. Thrun, "Probabilistic algorithms in robotics," *AI Magazine*, vol. 21, no. 4, pp. 93–109, 2000. [Online]. Available: CITESEER.NJ.NEC.COM/THRUN00PROBABILISTIC.HTML

[22] A. F. Foka and P. E. Trahanias, "Predictive autonomous robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002. [Online]. Available: citeseer.nj.nec.com/536649.html

[23] D. Goldberg and M. J. Mataric, "Reward maximization in a non-stationary mobile robot environment," in *Proceedings of the Fourth International Conference on Autonomous Agents*, C. Sierra, M. Gini, and J. S. Rosenschein, Eds. Barcelona, Catalonia, Spain: ACM Press, 2000, pp. 92–99. [Online]. Available: citeseer.nj.nec.com/goldberg00reward.html

[24] M. N. Nicolescu and M. J. Mataric, "Extending behavior-based systems capabilities using an abstract behavior representation," in *Working Notes of the AAAI Fall Symposium on Parallel Cognition*, North Falmouth, MA, November 2000. [Online]. Available: citeseer.nj.nec.com/nicolescu00extending.html

[25] M. Nicolescu and M. J. Mataric, "A hierarchical architecture for behavior-based robots," in *Proceedings, First International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Bologna, ITALY, July 2002.

[26] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge, Mass: The MIT Press, 1984.

[27] P. Maes, "A bottom-up mechanism for behavior selection in an artificial creature," in *Proceedings of the First International Conference on Simulation of Adaptive Behavior*, 1991, pp. 238–246.

[28] B. Blumberg, "Action selection in hamsterdam: Lessons from ethology," in *Proceedings of the 3rd international conference on the simulation of Adaptive behaviour*, Brighton, UK, 1994. [Online]. Available: citeseer.nj.nec.com/article/blumberg94actionselection.html

[29] S. Yamada and J. Saito, "Adaptive action selection without explicit communication for multi-robot box-pushing," *IEEE Transactions on Systems, Man and Cybernetics, Part C*, pp. 398–404, August 2001.

[30] L. E. Parker, "L-alliance: Task-oriented multi-robot learning in behavior-based systems," *Advanced Robotics, Special Issue on Selected Papers from IROS '96*, vol. 11, no. 4, pp. 305–322, 1997.

[31] C. Gershenson and P. P. Gonzalez, "Dynamic adjustment of the motivation degree in an action selection mechanism," in *Proceedings of ISA '2000*, Wollongong, Australia, 2000.

[32] M. Carreras, J. Yuh, and J. Batlle, "An hybrid methodology for rl-based behavior coordination in a target following mission with an auv," in *OCEANS 2001 MTS/IEEE Conference*, 2001.

[33] R. Pfeifer and P. Verschure, "Distributed adaptive control: a paradigm for designing autonomous agents," in [55]. Cambridge, MA: MIT Press, 1992, pp. 21–30.

[34] K. Z. Lorenz, *The Foundations of Ethology*. Springer-Verlag, New York, 1981.

[35] J. Hoff and G. Bekey, "An architecture for behavior coordination learning," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. 5, Perth, Australia, November 1995, pp. 2375–2380. [Online]. Available: citeseer.nj.nec.com/hoff95architecture.html

[36] G. Baerends, "The functional organisation of behaviour," *Animal Behaviour*, vol. 24, pp. 726–735, 1976.

[37] A. Sloman, "Damasio, Descartes, alarms and meta-management," in *Proceedings International Conference on Systems, Man, and Cybernetics (SMC98), San Diego*. IEEE, 1998, pp. 2652–7.

[38] V. Andronache and M. Scheutz, "Ade - an architecture development environment for virtual and robotic agents," to appear in the International Journal of Artificial Intelligence Tools.

[39] ——, "Integration theory and practice: The agent architecture framework APOC and its development environment ADE," in *Proceedings of the AAMAS*, 2004.

[40] ——, "Ade - a tool for the development of distributed architectures for virtual and robotic agents," in *Proceedings of the Fourth International Symposium "From Agent Theory to Agent Implementation"*, 2004.

[41] M. Scheutz, "Apoc - an architecture for the analysis and design of complex agents," in *Visions of Mind*, D. Davis, Ed. Idea Group Inc., 2004, p. forthcoming.

[42] M. Scheutz and V. Andronache, "Growing agents - an investigation of architectural mechanisms for the specification of "developing" agent architectures," in *Proceedings of the 16th International FLAIRS Conference*, R. Weber, Ed. AAAI Press, 2003.

[43] V. Andronache and M. Scheutz, "APOC - a framework for complex agents," in *Proceedings of the AAAI Spring Symposium*. AAAI Press, 2003.

[44] ——, "Contention scheduling: A viable action-selection mechanism for robotics?" in *Proceedings of the Thirteenth Midwest Artificial Intelligence and Cognitive Science Conference, MAICS 2002*, S. Conlon, Ed. Chicago, Illinois: AAAI Press, April 2002, pp. 122–129.

[45] D. M. Lyons and M. A. Arbib, "A formal model of computation for sensory-based robotics," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 3, pp. 280–293, June 1989.

[46] J. R. Anderson, D. Bothell, B. M. D., and C. Lebiere, "An integrated theory of the mind," to appear in Psychological Review.

[47] G. Carpenter and S. Grossberg, "The art of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, pp. 77–88, March 1988.

[48] M. Minsky, *The Society of Mind*. New York, NY: Simon & Schuster, 1985.

[49] R. Murphy, *Introduction to AI Robotics*. MIT Press, 2000.

[50] R. Pfeifer and C. Scheier, *Understanding Intelligence*. Cambridge, Massachusetts: MIT Press, 1999.

[51] M. Carreras, J. Batlle, and P. Ridao, "Hybrid coordination of reinforcement learning-based behaviors for auv control," in *EEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[52] M. Carreras, J. Yuh, and J. Batlle, "High-level control of autonomous robots using a behavior-based scheme and reinforcement learning," in *15th IFAC World Congress on Automatic Control*, 2002.

[53] A. Sloman, "Sim_agent help file," 1999.

[54] T. Balch and R. Arkin, "Avoiding the past: a simple but effective strategy for reactive navigation," in *Proceedings of the 1993 IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993, pp. 678–685.

[55] F. J. Varela and P. Bourgine, Eds., *Towards a Pratice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*. Cambridge, MA: MIT Press, 1992, the ECAL 91 took place in Paris on December 11-13, 1991.

TABLE II

RESULTS OF THE INDIVIDUAL PERFORMANCE OF ALL FOUR AGENT KINDS IN LOW AND HIGH RESOURCE ENVIRONMENTS.

Resources	Switching 1		Switching 2		Cooperative		Competitive	
	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.
Low	7.25	(6.15,8.35)	0.45	(-0.05,0.95)	0.50	(0.01,0.99)	0.58	(0.15,1.00)
High	13.33	(12.43,14.22)	5.42	(4.79,6.06)	4.20	(2.70,5.70)	3.75	(2.60,4.90)

TABLE III

RESULTS OF THE COMPETITION OF A_1 AGENTS WITH A_3 AND A_4 AGENTS, RESPECTIVELY.

Resources	Switching 1		Cooperative		Switching 1		Competitive	
	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.
Low	5.80	(4.49,7.11)	0.00	(0.00,0.00)	5.33	(4.16,6.49)	0.05	(-0.05,0.15)
High	11.92	(10.76,13.09)	0.00	(0.00,0.00)	11.42	(10.46,12.39)	0.25	(-0.26,0.76)

TABLE IV

RESULTS OF THE COMPETITION OF A_2 AGENTS WITH A_3 AND A_4 AGENTS, RESPECTIVELY.

Resources	Switching 2		Cooperative		Switching 2		Competitive	
	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.	Mean	Conf.Int.
Low	1.03	(0.43,1.62)	0.00	(0.00,0.00)	1.65	(0.87,2.43)	0.38	(0.09,0.66)
High	3.07	(2.01,4.14)	1.45	(0.58,2.32)	5.02	(3.36,6.69)	2.60	(1.63,3.57)