7-2009

Unified Behavior Framework for Reactive Robot Control

Brian G. Woolley
Air Force Institute of Technology

Gilbert L. Peterson
Air Force Institute of Technology

Recommended Citation

Woolley, B. G., & Peterson, G. L. (2009). Unified Behavior Framework for Reactive Robot Control. Journal of Intelligent and Robotic Systyems, 55(2–3), 155–176. DOI: 10.1007/s10846-008-9299-1

Woolley, B., Peterson, G.L., "Unified Behavior Framework for Reactive Robot Control," Journal of Intelligent and Robotic Systems, vol. 55, no. 2, pp.155-176, 2009. DOI 10.1007/s10846-008-9299-1

Unified Behavior Framework for Reactive Robot Control

Brian G. Woolley and Gilbert L. Peterson
Air Force Institute of Technology
Department of Electrical and Computer Engineering
2950 Hobson Way, Bldg 640
Wright-Patterson AFB, OH 4543
(937) 255-3636

brian.woolley@ieee.org, gilbert.peterson@afit.edu

ABSTRACT
Behavior-based systems form the basis of autonomous control for many robots. In this article, we demonstrate that a single software framework can be used to represent many existing behavior based approaches. The unified behavior framework presented, incorporates the critical ideas and concepts of the existing reactive controllers. Additionally, the modular design of the behavior framework: 1) simplifies development and testing; 2) promotes the reuse of code; 3) supports designs that scale easily into large hierarchies while restricting code complexity; and 4) allows the behavior based system developer the freedom to use the behavior system they feel will function the best. When a hybrid or three layer control architecture includes the unified behavior framework, a common interface is shared by all behaviors, leaving the higher order planning and sequencing elements free to interchange behaviors during execution to achieve high level goals and plans. The framework's ability to compose structures from independent elements encourages experimentation and reuse while isolating the scope of troubleshooting to the behavior composition. The ability to use elemental components to build and evaluate behavior structures is demonstrated using the Robocode simulation environment. Additionally, the ability of a reactive controller to change its active behavior during execution is shown in a goal seeking robot implementation.

Categories
-Artificial Intelligence/Expert Systems/Knowledge−Based Systems Applications

-Intelligent Systems/Intelligent Control/Fuzzy Control/Prosthetics/Robot Motion Planning

Keywords
Behavior-based robotics, reactive control architecture, software design patterns, software frameworks.

1. INTRODUCTION
Mobile robotics applications place vehicles into environments where uncertainty is normal. Environments like homes, offices, and public areas are inherently unconstrained, and the use of cost effective motors and sensors that are inaccurate and prone to failure adds additional uncertainty about the environment. As intelligent vehicles are expected to handle increasingly more complex endeavors, their design, implementation and testing requirements grow. Symbolic approaches to world modeling are quickly overwhelmed trying to represent detail of an environment, both in computational time and memory requirements. The alternative, reactive behavior-based controller implementations maintain responsiveness by abandoning the goal of optimality.

Behavior-based controllers employ a small number of behaviors with an arbitration element that performs action selection. This design provides robust low-level control but is customized for specific environments. This customization makes reuse of the control structures and behaviors for different scenarios or reuse on different robots rare. Complexity issues also plague reactive approaches when they attempt to scale up to support additional system responsibilities and deal with insurmountable obstacles, a capability ceiling. Reactive controllers face a capability ceiling because they lack a mechanism for managing their complexity [18].

To combat the capability ceiling, and because the deliberative qualities of symbolic approaches are as important as the responsiveness of reactive approaches, layered architectures combine deliberative and reactive controllers [17]. Layered architectures gain the benefits of both goal seeking (deliberation) and responsiveness (reaction) [14]. Since sequences of specialized behaviors provide a means of achieving higher order goals and plans, then a mechanism must exist that allows individual behaviors to be called on or activated interchangeably.

Software engineering has identified that encapsulation via a well defined interface is effective for allowing independent components to be used interchangeably. This has the added benefit that modular components become reusable. By establishing behaviors as reusable atomic elements, one can quickly form new collections of behaviors as simple adaptations or constructions of existing elements. By incorporating existing behavior modules, new designs capitalize on reuse as a way of reducing their design complexity, allowing specific system attributes to be developed and tested independently. To meet this need, a unified

behavior framework is introduced that enforces a generalized interface that supports: modularity, functional abstraction, and reuse. Further, because the framework is modeled after the composite pattern [16], new behaviors can be formed as arbitrated hierarchies using any combination of atomic behaviors or existing hierarchical structures. This abstraction establishes functional boundaries that create a modular environment in which new structures are easily formed as arrangements of existing behaviors and hierarchies. An added benefit of the abstraction is that developers are not tied to one type of behavior-based controller and can switch and combine concepts from all of them.

This paper explores the development of a reactive control system and its use within a layered architecture. A unified behavior framework is defined for a system's reactive control element that abstracts the implementation details of specific behaviors and permits behavior reconfiguration during execution. It demonstrates how popular behavior architectures are established within the context of the unifying framework, making them modular and interchangeable and concludes with two implementations of the framework, one in simulation and the other in hardware as a goal seeking robot.

2. BACKGROUND

Initial research efforts in robotics focused on planning and world modeling [17] in an attempt to develop completely rational mobile robots [33]. The sense-plan-act approach, used in Shakey [29] and Rover [28] (Figure 1a), proved inadequate in dynamic and unpredictable environments, where the robot finds itself in trouble when its internal state loses sync with the reality it is intended to represent [2]. This is caused by the amount of time and memory required to maintain and develop plans, that the state of the environment changes before the actions can be carried out [8].

The planning bottleneck was alleviated through tasks being decomposed into collections of low-level primitive behaviors, reactive planning [17]. The ideas behind reactive planning stem from arguments such as Braitenberg's, who argues that the complex behavior of natural organisms may be the result of combinations of simple behaviors that generates more complex behavior [6]. In equivalent research Brooks claims that for many tasks, robots do not need traditional reasoning, only a tight coupling of sensing to action. This is demonstrated with the Subsumption architecture [7,9].

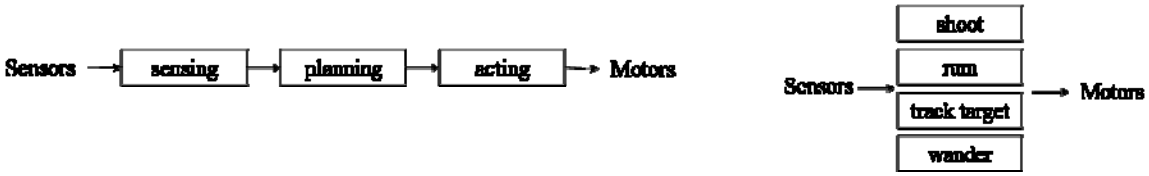

Figure 1: Two organization decompositions for robot control (A) Sequential execution of functional modules; (B) A task-based decomposition into parallel execution modules.

The Subsumption architecture advocates for a layered control system based on task decomposition, an approach that is radically different from previous research. Figure 1 shows the linear sense-plan-act architecture (Figure 1a) in comparison with the horizontal structure of Subsumption (Figure 1b). This parallel organization promotes concurrent and asynchronous responses to sensor input. Each individual layer works to achieve its particular goal. Coordination between layers is achieved when complex actions (or higher layers) subsume simpler actions, or when low-level behaviors inhibit higher layers.

Other reactive architectures emerge as effective robot control structures, each retaining the original ideas of reactive planning. Specifically, that the system: is responsive to the environment, includes a tight coupling of sensing to action (keeping little or no state representation), and should be robust, able to perform in the face of unanticipated circumstances or sensor failure. Additionally, the system needs to be modular, using incremental development to add capabilities, and the system must have an execution that embraces parallelism and concurrency [2]. In addition to Subsumption [7,9], this article uses five reactive architectures for discussion: Motor Schema [1], Circuit Architecture [22], Action-Selection [26], Colony Architecture [12], and Utility Fusion [32,33]. They are briefly introduced here and revisited in Section 4, including a description of how they map into the unified behavior framework.

Unlike the priority ordered layers of Subsumption, the motor schema architecture emerged as a cooperative control approach, allowing for the simultaneous pursuit of multiple goals. This approach captures behavioral primitives such as vector fields that support specific perception tasks (e.g. obstacle avoid, move-to-goal, stay-on-path, etc.) which are arbitrated as a normalized vector summation to form a continuous potential field. This approach is useful in navigation tasks, but is subject to local minima and cyclical paths [1].

Circuit architecture is a hybridization that allows reactive behavior elements and logical formalisms to be bundled into arbitrated collections. Because priority arbitration occurs at each level of abstraction, developers can bundle unlike approaches into mediated hierarchies that are combinations of reactive approaches, logical formalisms, and situated automata [22].

Action-selection is an architecture that uses activation levels as a dynamic mechanism of behavior selection. Individual behaviors are grouped as competence modules that respond when predefined conditions are detected. Activation levels are used to indicate a level of confidence that can persist while specific conditions exist or may decay over time. Action coordination is achieved by

selecting the competence module with the highest activation level. When used in dynamic environments, action-selection gives the global behavior an emergent quality because there is no predefined layering or order of execution. Based on events in the environment, a robot may suddenly begin to display radically different attributes [26].

Colony architecture is a direct descendant of Subsumption, allowing higher layers to suppress lower layers but eliminating the ability of lower layers to inhibit higher ones. As a result of enacting the suppression only approach, the colony architecture breaks away from the total ordering of layers found in Subsumption and permits the formation of priority based behavior hierarchies [12].

The utility fusion architecture is an expansion of DAMN (Distributed Architecture for Mobile Navigation) [32]. Under this architecture, action selection is coordinated via an evaluation of the utility that would result from taking a particular action from a discrete set of actions. In DAMN, the arbiter is central to the architecture, taking on the unique kinematics of the specific robot, allowing the evaluation behaviors to remain platform independent and reusable. Behaviors use their own criteria to assess the utility of a proposed future state. The action that collects the highest overall utility is enacted by the arbiter. Although actions are enacted in a winner-takes-all fashion, the utility fusion approach is considered to be cooperative because it selects the action that best serves the global goals of the system [33].

Current research uses behavior based systems in three layer architectures [17], or hybrid architectures [4]. This research focuses on refining connections between the behavior based system and other components such as the deliberator. Extensions include adding stochasticity to the deliberator [31,3], temporal components [25], and alternative fuzzy representations [21]. Additionally, behavior based systems have seen application in games [19] and on airborne platforms [13].

In addition to the arrangement of the behaviors and arbitration components, much research has been done on having the robot learn behaviors and sets of behaviors. This includes using artificial neural networks in combination with other components such as fuzzy control [34], Q-learning [10]. Or alternative ANN representations such as CTRANN [15,27], GasNets [27], and Convolutional NN [24]. Other approaches include evolutionary algorithms that evolve fuzzy based behaviors [20], or augmented neural topologies [35].

While reactive architectures that are organized as task based decompositions are responsive and able to operate in dynamic environments, they forfeit the ability to make plans and pursue goals. Driven by requirements that systems must not only be responsive in dynamic environments, but rational and deliberative as well, a three layer architecture merges these two goals by incorporating reactive planning as a low-level control element and a deliberator at the highest level. The three layer architecture is a common paradigm for designing these hybrid autonomous robot control architectures [16]. Under this architecture, the structure of the software system consists of three components: a reactive feedback control mechanism (the controller), a slow deliberative planner (the deliberator), and a sequencing mechanism that connects the first two components (the sequencer). Each layer of the architecture provides additional environment and sensor abstraction over the previous, and focuses on larger reasoning and goal time scales. Despite the previous work on reactive behaviors, the controller remains the most time consuming segment to design and implement.

3. UNIFIED BEHAVIOR FRAMEWORK

Often, a mobile robot design implements a single behavior architecture customized for the intended application, thus binding its performance to the strengths and weaknesses of that architecture. Further, the ability to change or expand the base behavior is made difficult because it is developed as an integral part of the controller, making one indistinguishable from the other. By making a clear separation between these two pieces, the controller can use various packages of behavior logic without changing the controller implementation. Additionally, behavior packages are no longer tied to specific platform implementations, encouraging reuse. This section introduces a Unified Behavior Framework (UBF) which allows a robot to seamlessly change between disparate architectures and provides mechanisms that simplify the design, development and implementation of reactive control structures.

The UBF, shown in Figure 2, uses the strategy pattern [16] to provide the controller with the ability to dynamically swap its behavior packages at runtime. An abstract behavior interface is used to define a set of related classes that can be used interchangeably, namely leaf and composite behaviors. The controller, knowing how to use a behavior in its abstract form, is able to use any of the concrete implementations that belong to the family of behaviors in a uniform manner. This frees the low-level controller from being bound to any single behavior architecture. In fact it provides the ability to seamlessly switch between distinct architectures during execution, and promotes the reuse of existing behaviors.

The reuse of subcomponents is also encouraged in the UBF via a mechanism modeled on the composite pattern [16]. The composite pattern allows new control structures to be formed as arbitrated hierarchies of existing behaviors, with the resulting structure being usable as a behavior. The consequence of this is that two or more existing behaviors can be combined (regardless of their being leaves or composites) to form a new behavior structure.

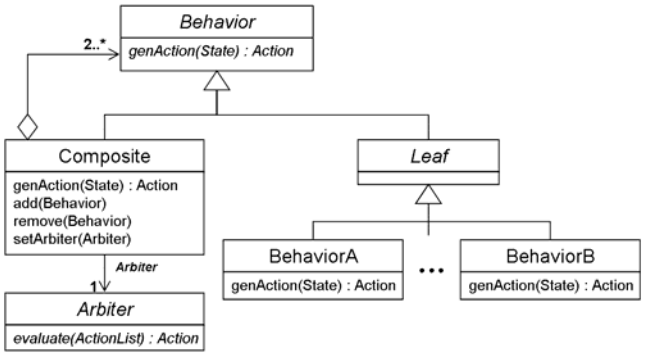

Figure 2: Class diagram for the Unified Behavior Framework.

The software design mechanisms of the strategy and composite patterns encourage a developer to use modular approaches that ease the complexity of designing, testing and implementing a collection of reactive behaviors, while providing the ability to form larger hierarchies of behaviors. This isolates code complexity to the atomic (or leaf) behaviors. The freedom to join existing behaviors as compositions encourages experimentation with various structural arrangements of elemental behaviors, arbitration components, as well as existing behavior structures. Because the operational logic of subcomponents is independently verified, this approach isolates the task of aligning a system's outward behavior to the structural arrangement of control modules. While the UBF does ensure that individual elements are compatible with one another, it makes no assertion about the coherence of the resulting control structure. The results in Section 5.1 demonstrate these concepts as a robot simulation is used to observe the radical differences that arise in the external behavior attributes as various arbiters are used on an identical set of base behaviors.

3.1 Encapsulating Behaviors

To integrate the UBF with the controller, a layer of abstraction is required to make a clear delineation between the controller and the reactive behavior that drives it. This concept is similar to that used in Pyro to provide a standard interface for application to multiple robot platforms [5]. To make this possible, a standardized behavior interface is established that allows an action recommendation to be generated based on the current state. The behavior interface of the abstract behavior class in Figure 2 consists of the genAction method that accepts the state to be evaluated and returns an action recommendation in the form of motor outputs. The creation of a concrete behavior that sub-classes the abstract behavior has two distinct advantages over standalone implementations. The first is that polymorphism requires the presence of a genAction method. The second is that all such behaviors are interchangeable because, notionally, they are all behaviors.

The interface for sensor and state information is administered by the reactive controller. The reactive controller uses a generic sensor interface referred to as the Perceived State [23] to send the current perceived state to each behavior. Each behavior use the perceived state to determine what action it wants to take, send out as a motor command. The reactive controller then issues a motor commands based on the final arbitrated recommendation of the behavior hierarchy.

In order to discuss how the controller might employ this construct, assume that fully implemented behaviors are available. From the controller's perspective, a three-step process is enacted as a continuous loop shown in Figure 3. First, the state is updated with the robot sensor information to represent the current conditions. Second, the behavior is asked to generate a recommended action by invoking the genAction method. The behaviors then get the state information and output a set of motor commands as an action object. The reactive controller receives a unified set of actions from the behaviors and identifies the motor commands that are then issued directly to the motors via the execute method.

This consistent interface enables seamlessly changing the active behaviors at runtime and provides a responsive and flexible basis of control. The ability to change behaviors during execution allows the temporal sequencing of specialized behaviors to pursue higher order goals without the reactive behaviors requiring information about the plans that they are used to achieve. This provides a means to in a hybrid architecture have a component similar to reactive action packages (RAP) [14] convert planning goals to behavior hierarchies. From an implementation perspective, specialized behaviors have limited complexity and are easier to design and test over monolithic behaviors that attempt to address all possible world conditions. Instead a system that relies on a collection of specialized behaviors with a common interface is able to observe the environmental conditions and apply a particular behavior when it is most effective.

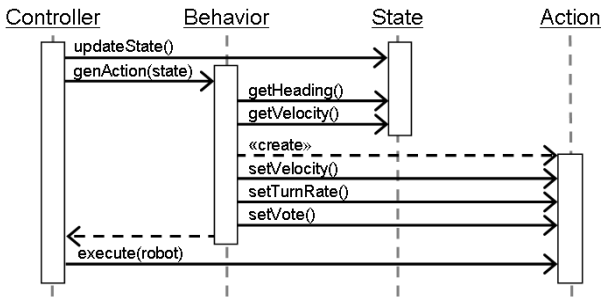

Figure 3: Sequence diagram of a controller using its behavior.

The encapsulation of behavior logic is intended to allow the controller to use disparate behavior based systems (e.g. Subsumption, action-selection, or utility fusion) as interchangeable elements of a behavior library. Because a significant goal of the UBF is to encourage the development of reactive control structures that are reusable, interchangeable, and scalable [36], the following subsection continues the discussion about how existing behavior modules can be used in the development of more complex behaviors.

3.2 Constructing Behaviors

To support software reuse, the developer must be free to call on existing behavior modules "as-is" and without modification to form a new reactive controller. The UBF supports this by allowing behaviors to be joined via an arbitration node. Modeled on the composite pattern [16], the arbiter provides the UBF the ability to form hierarchical structures of behavior collections. Thus a developer is free to reuse the functionality of an existing behavior and incorporate it as a part of a new structure, regardless of its underlying implementation. Not only can a controller switch between behavior architectures at will, it can also use a hierarchy that is a composition of disparate behavior based systems.

Figure 2 shows the leaf and arbitrated composite behaviors. The leaf behavior is the atomic building block behavior for more complex structures, and is the smallest behavior component.

The composite behavior is a joining element, allowing many behaviors to be formed into an arbitrated hierarchy. Its functional purpose is to maintain a set of sub-behaviors, $\boldsymbol{B} = \{b_1, .. , b_n\}$, and when asked to generate an action it builds a corresponding set of proposed actions, $\boldsymbol{A} = \{a_1, .. , a_n\}$, which are collected by invoking the genAction method for each member of $\boldsymbol{B}$. Since a composite extends the abstract Behavior class, it returns a single action object. Thus, an arbitration unit is used to determine a single action, $\boldsymbol{a'}$, from the set $\boldsymbol{A}$. Because many arbitration techniques exist, the UBF does not attempt to embed a fixed approach into the composite behavior, rather it encapsulates the arbitration task as an associated arbiter module. This separation of the arbiter from the composite behavior allows the arbitration technique to be changed at any time by invoking the setArbiter method. In addition to setArbiter, the composite class has other helper methods to manage the structure of a composite behavior.

Outwardly, the responsibility of an arbiter's evaluate method is to accept the set of actions $\boldsymbol{A}$ and return a single action recommendation, $\boldsymbol{a'}$. Individual arbitration algorithms are established by extending the abstract arbiter class. By doing this, all arbiters belong to the same family of arbitration algorithms and can be applied interchangeably. Internally, an arbiter can use a behavior's action vote value and an associated behavior weight to generate the ultimate recommendation, $\boldsymbol{a'}$. The set of weights, $\boldsymbol{W} = \{w_1, .. , w_n\}$, is a normalized set used to scale the affect or selection of a behavior in relation to the other behaviors in the set $\boldsymbol{B}$.

For example, a vector summation arbiter, which is used in the motor-schema behavioral system, scales all the motor command recommendations made by the elements in $\boldsymbol{A}$ by the related weights $\boldsymbol{W}$. The resulting values are summed to form the action recommendation $\boldsymbol{a'}$, which is sent to the next higher level or the motors themselves. The vector summation arbiter allows the weights, $\boldsymbol{W}$, to function as a means of tuning the contributions of individual behaviors to achieve the desired effect.

4. BUILDING BEHAVIOR STRUCTURES

The UBF is a structural guide that applies standard software engineering approaches to simplify development and testing of reactive behavior modules for autonomous robots. At the highest level, it uses a strategy pattern [16] to establish a family of interchangeable behaviors. Additionally, the capabilities of the UBF allow control structures to be formed as arbitrated hierarchies of existing behaviors. The use of the composite pattern [16] then ensures that the resulting structures are scaleable and belong to the existing family of behaviors. This approach allows independent development teams the freedom to use the behavior system they feel will best achieve their design goals, and due to the ease with which components can be formed into stable structures encourages reuse and experimentation. Additionally, since the scope of each behavior is focused, code complexity is reduced which in turn eases testing requirements. Once established, any compatible robot controller can use a behavior as an interchangeable behavior element. Note that the complexity associated with testing the interactions of the behaviors under the arbiters still requires trial and error to achieve the desired environment responses.

This section demonstrates how the Unified Behavior Framework (UBF) is used to construct the typical behavior architectures: Subsumption [7,9], motor schema [1], circuit architecture [22], action selection [26], colony architecture [12], and utility fusion [33]. Each sub-section provides a short summary of architecture and demonstrates its implementation in the context of the UBF with behaviors as defined for the Robocode simulation described in Section 5.1.2. The goal is to demonstrate that each of the architectures can achieve a common interface, which allows autonomous robot system developers the flexibility to adapt to the current environmental conditions by selecting and using the most appropriate behavior structure.

4.1 Subsumption Architecture

The Subsumption architecture [7,9] advocates for a layered control system based on task decomposition rather than function that promotes concurrent and asynchronous responses to sensor input, where each individual layer works to achieve its particular goal. Coordination between layers is achieved when complex actions (or higher layers) subsume simpler behaviors, or the low level behaviors inhibit the higher layers. Fundamentally, Subsumption can be viewed as a competitive architecture using rule-based encodings and priority-based arbitration based on hierarchical priority [2].

To establish the Subsumption architecture in the context of the UBF, two fundamental intents of Subsumption need to be captured and preserved. The first major concept of Subsumption is the use of rule-based behaviors (or layers) and the second is the suppression/inhibition mechanism that allows layers to subsume or inhibit the outputs of other layers.

The translation is depicted by Figure 4 where the traditional Subsumption architecture is shown in (A) and a corresponding implementation under the UBF is shown in (B). Each task layer is represented directly in the UBF as an individual leaf behavior. A behavior builds an action object with the motor commands that it would execute if given control and when environmental conditions are suitable, it sets the vote field to indicate a desire to be considered for selection. The original suppression mechanism used for coordination exists in the priority arbiter associated with the composite node. By using the priority arbiter as the merging mechanism, a composite behavior that models Subsumption is formed, where the action recommendation of the highest priority behavior that did not abstain is selected and returned by the composite node.

The use of a hierarchical structure to encapsulate each behavior layer preserves Brooks' original intention to be able to implement, test, and debug each layer independently. Additionally, because the behaviors in the hierarchy are independent, they are well suited for concurrent and asynchronous execution.

In the Subsumption architecture, communication is allowed between behaviors represented as Augmented Finite State Machines. Because of this communication Subsumption has received criticism because upper layers interfere with lower ones, which keeps each layer's design from being independent, and prevents the ability to test and debug layers independently [2]. Since the independence of behavioral layers and the arbitration due to priority is fundamental to the concept of Subsumption the behavior communication is not supported. This is justified as often examples of Subsumption (for example [2]) appear similar to the representation in Figure 4a, and the descendant of Subsumption, the Colony Architecture only makes use of subsumption.

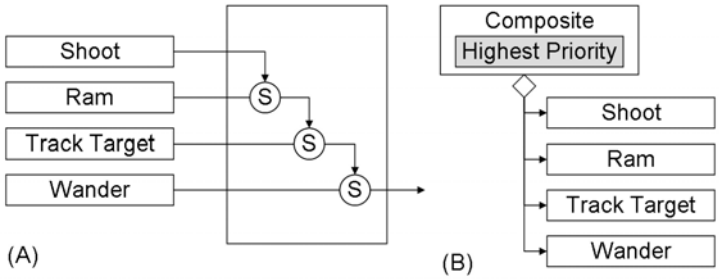

Figure 4: Equivalent implementations of Subsumption (A) Behavioral layer arbitrated via a suppression network; (B) A behavior hierarchy using a priority arbiter.

4.2 Motor Schema Architecture

The motor schema architecture [1] is a cooperative control approach that allows for the simultaneous pursuit of multiple goals. Under this architecture, behavioral primitives are captured as vector fields that support specific perception tasks (e.g. obstacle avoid, move-to-goal, stay-on-path, etc.) which are arbitrated as a vector summation and normalization of a continuous potential field. This approach is useful in navigation tasks where a path to a goal must be discovered and obstacles exit on the direct path to the goal.

To enact the motor schema architecture in the context of the UBF two aspects are captured. The first is the motor schema architecture's use of perception schemas to capture the governing motion effects around goals, obstacles, wall, etc. and the second its use of vector summation and normalization as a means of coordinating motor commands.

The transformation is depicted in Figure 5 where the traditional motor schema architecture is shown as (A) and the corresponding implementation under the UBF is shown as (B). Since perception schemas are already modular and independent of one another,

each one is represented as a leaf behavior that returns its action recommendation. Because the resultant vector field is normalized, each schema can set the vote field of its action if an event that attracts or repels the robot is detected, otherwise it abstains. The original summation and normalization mechanism used for coordination is captured directly as an arbiter that generates a fused command response. By grouping the schemas together under a composite node that uses a command fusion arbiter the motor schema architecture is effectively implemented within the context of the UBF.

In the original implementation of the motor schema architecture, there are mechanisms that allow the effect of individual schemas to be weighted as a means of tuning the global behavior. When implemented under the UBF, this is achieved by weighting the affects of the associated behaviors within the arbiter. Consider a simple schema pair where goals attract and obstacles repel. By increasing the weights associated with the obstacle-avoid schema the robot will swing wide of obstructions and may navigate around cluster of obstacles. By making adjustments to the weights within an arbiter, the individual schema remains generic and reusable. Additionally, this implementation maintains the key strengths of Arkin's original motor schema architecture. The use of modular perception schemas supports parallel and distributed computation, runtime flexibility and delivers reusable components that can be stored and called from behavior libraries [2].

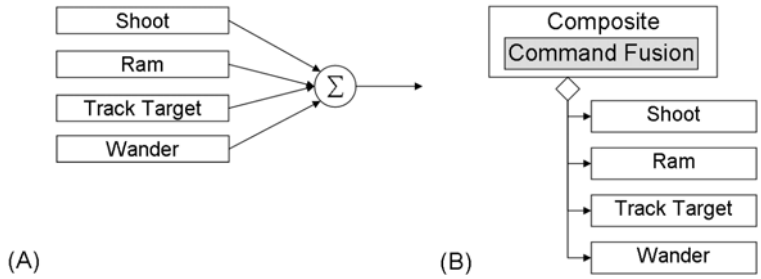

Figure 5: Equivalent implementations Motor Schema (A) Independent motor schemas coordinated via summation and normalization; (B) A behavior hierarchy using a command fusion arbiter.

4.3 Circuit Architecture

The circuit architecture [22] is an abstraction approach that groups reactive behaviors and logical formalisms into arbitrated collections. Because arbitration occurs at each level of abstraction, developers can build hybridized bundles of unlike approaches into mediated hierarchies that are combinations of reactive approaches, logical formalisms, and situated automata [2].

The circuit architecture in the context of the UBF includes the ability to create bundles of either reactive behaviors or logical formalisms, and to use hierarchical mediation. The key components of this architecture translate directly into an implementation using the UBF since leaf behaviors can be implemented as either a reactive behavior or as a logical formalism. Additionally, composite behaviors are used for mediated hierarchies since each has an associate arbiter. The ability for each composite behavior junction to use a different arbitration technique allows the leaf implementations to be arranged into synonymous hierarchical structures.

4.4 Action-Selection

Action-selection [26] uses activation levels as a dynamic mechanism of competitive behavior selection. Individual behaviors are grouped as competence modules that respond when predefined conditions are detected. These requirements can be simple environmental triggers, sequential observations, the existence of higher level goals, previous or potential success, etc. When a module's preconditions are satisfied, an associated action sequence is initiated at a given activation level. Activation values can be instantaneously reported while the trigger condition is present, can persist for a set period of time or can have various decay rates. Action coordination is achieved by selecting the competence module that currently has the highest activation level. Because there is no predefined layering or order of execution when used in dynamic environments, this gives the global behavior a greater emergent quality. By basing priority on events in the environment a robot can suddenly and deliberately respond to unique conditions or changes in the environment.

The UBF implements the modularity and independence of individual competence modules as individual leaf behaviors. Action-selection's use of competing activation signals has no direct equivalent under the UBF. To establish an equivalent ability, the activation level is embedded in the action recommendation using the behavior's vote field to indicate its current activation level. The arbitration scheme used by action-selection is represented in the UBF as a highest activation arbiter associated with the composite behavior that groups related competence modules together. Being a winner-take-all approach, the arbiter evaluates the vote value for each action recommendation and returns the action with the highest value. This approach lends itself to a hierarchical structure of competence modules that can be several layers deep.

4.5 Colony Architecture

The Colony architecture [12] is a descendant of Subsumption, allowing higher layers to suppress lower layers but eliminates the ability of lower layers to inhibit higher ones. As a result of enacting the suppression only approach, the colony architecture breaks away from the complex ordering of layers found in Subsumption and permits hierarchical arrangements of behavioral priorities [2].

This architecture is naturally represented in the context of the UBF, as a priority based hierarchy. By capturing the logical control layers as leaf behaviors, functional branches of control are then formed via composite nodes using highest priority arbitration. Figure 6 depicts a Colony architecture design, grouping the shooting and target tracking behaviors into separate control structures. A traditional suppression network is shown in Figure 6a, and an equivalent representation using the UBF structure is shown in Figure 6b.

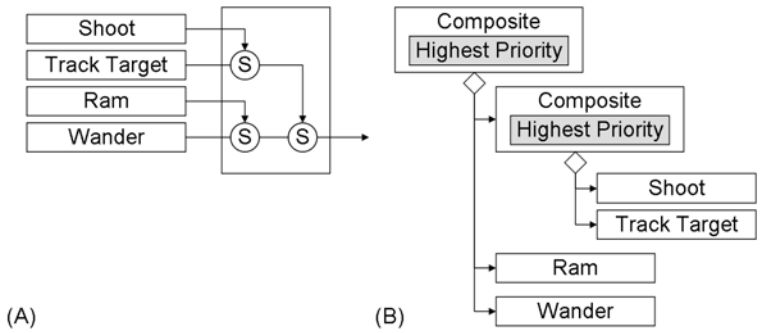

Figure 6: Equivalent implementation of Colony Architecture (A) Priority based behavior hierarchy using a suppression network; (B) An equivalent control structure using the UBF.

4.6 Utility Fusion

The utility fusion architecture [33] is an expansion of the DAMN architecture [32] which distributes action selection via utility instead of priority-based or command-fusion arbitration approaches. Under utility fusion an arbiter collects utility votes for proposed actions from the associated evaluation behaviors. Each behavior uses its own criteria to independently assess the utility of a future state that results from taking a given action. The action that collects the highest overall utility is enacted by the arbiter. This approach provides the arbiter with much richer evaluation information because actions are selected by how they best satisfy the system's overall goals, or how the action best meets the goals of all of the behaviors. For example, if several behaviors assign a modest utility to one action while only one behavior assigns a high utility value to a second action, the utility based arbiter will select the action which accumulated the largest total utility. In most cases the first action will be selected because the overall utility from several moderate votes is greater than a high utility vote from a single behavior. This is an appropriate assessment because the first action simultaneously meets more of the system's overall goals.

Under the utility fusion architecture, Rosenblatt intends that behaviors evaluate candidate future states so that they do not need to know the system kinematics or the specific motor commands to achieve the proposed state. This modular approach binds the abilities of behaviors to the detection of favorable/unfavorable conditions within the environment and divorces them from implementation details. The benefit of this approach is that behaviors become modular and reusable across systems that employ the utility fusion architecture.

Capturing the DAMN and utility fusion architectures under the UBF is a difficult problem because sub-behaviors are being asked to evaluate projected states rather than the shared perceptual space. While the evaluation behaviors can be represented directly as leaf behaviors that indicate their utility assessment using the vote field embedded in the action object returned by the behavior

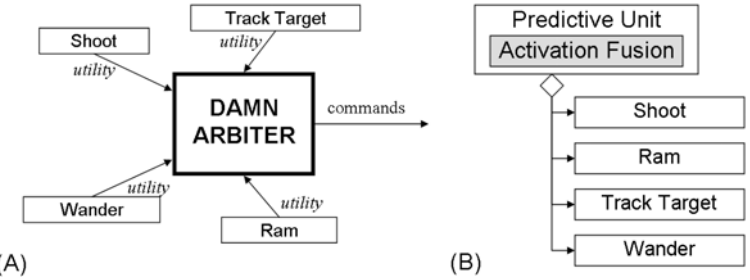

Figure 7: Equivalent implementations of Utility Fusion (A) Evaluation behaviors assess utility; (B) Behavior hierarchy using a predictive unit and a highest-utility arbiter.

(any proposed action values are ignored), a sensor like mechanism that understands the kinematics of the robot must be implemented such that it can generate predictive state representations for the leaf behaviors to assess from the perceived state.

This central predictive mechanism would hold a discrete set of actions such as the six actions: strong-left, weak-left, straight, weak-right, strong-right, and stop, and generates the six future states that would result from the execution of each action. The utility of each future state is assessed as a sum of the utilities returned by the five evaluation behaviors and is embedded into its associated action using the vote field. The enacted action is selected using a simple highest-utility arbiter that evaluates the vote values for each element in the action set. An example DAMN architecture with the predictive element is shown in Figure 7a with the equivalent representation in the UBF structure, shown in Figure 7b.

5. RESULTS

The ability of the UBF to form modular behavior elements into usable low-level control structures is demonstrated in two implementations of the framework, one in simulation and another in hardware. The first uses an adaptation of the Robocode robot battle environment to show that basic behavior/arbiter elements can be used to construct and test a variety of behavior structures. The second, a hardware implementation using a Pioneer P2-AT8 robot demonstrates how basic reactive control modules can be reused to form new control structures and how specialized structures can be employed interchangeably to produce a coherent overall behavior for use in a three layer architecture.

5.1 Robocode Implementation

The six behavior based architectures are implemented using the UBF and tested in the Robocode simulation domain. The implementation and test demonstrates how the framework promotes code reuse (all of the architectures use the same behaviors) and supports the development of hierarchical control structures (the hierarchies and arbiters are all that are altered between implementations. As a byproduct of this demonstration, the importance of selecting an effective arbiter for the behavior tasks chosen is highlighted by the radical changes in the outward attributes of the behavior structures as different arbiters are employed for the same behavior structure.

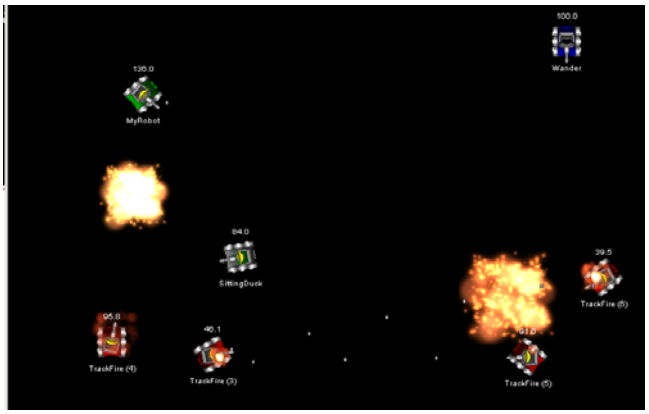

Figure 8: The RoboCode Environment.

The RoboCode environment, shown in Figure 8, was developed as a Problem-based Learning environment to improve retention of programming languages [30]. The objective is to develop control software for a tank like robot that survives a battle for as long as possible while doing as much damage to the other players as it can. However, Robocode is not as useful for experimenting with established robot control architectures because rather than accepting motor commands, commands are discrete requests that set a robot to turn left 90 degrees, or travel a set distance and then stop. This motor interface is atypical of standard robot motor control mechanisms. For this reason, the motor interface of Robocode version 1.0.7 was adapted to allow for a velocity based approach, it now accepts commands that specify the desired velocity and rate of turn for the chassis as well as the turn rate for the gun turret and the radar. Once set, these rate based values persist until changed.

The discussion of this experiment is broken into three sub-sections: a listing of the behavior/arbiter components developed, a description of the behavior structures formed from these components, and the observations of the outward behavior for each structure along with an assessment of the performance gap that emerges.

5.1.1 Description of Elemental Components

Using the UBF interface, five elemental behaviors and six arbiters are developed and tested on the architectures described in Section 5.1.2. The behaviors are:

Ramming—when another robot (with a lower energy level) is detected, this behavior causes our robot to turn towards the robot, charge towards it, attempting to cause damage by hitting it.

Shoot—causes the robot to fire on another when the target is less than three degrees off bore site and is within range. The power committed to the bullet is a function of the target off bore site angle and the distance to the target.

Scan—causes the radar to oscillate in a twenty degree arc around the bore site of the gun. As the gun rotates the active radar scan area tracks with it.

Target Tracking—has two operating modes. When no target is detected, the default mode turns the turret in a clockwise direction. When a target is detected, the target tracking behavior causes the gun turret rotation to slow or reverse its direction in an attempt to continue tracking the target. This behavior sets the turret turn rate to be one-third of the current off bore site angle, as a target's own motion causes the off bore site angle to increase, the turret turn rate increases accordingly in an attempt to track the target.

Wander—is always active, attempting to performs a series of "S" turns across the battlefield with the arc length being randomly selected between 30 and 120 degrees. When a wall is encountered, the polarity of the velocity is flipped.

The arbiters include:

Monte Carlo—is a stochastic arbitration technique that uses fitness proportional random selection to activate one sub-behavior for a period of time. At the end of a period another random selection occurs, activating the chosen sub-behavior in the next period.

Highest Priority—is a winner-take-all arbiter that returns the action set of the highest priority behavior indicating a desire to act, regardless of vote value.

Priority Fusion—is a semi-cooperative arbiter that uses a highest priority selection approach on a per motor command basis. Unlike the highest priority arbiter above, priority fusion builds a new action set that allows the unspecified action fields of higher priority behaviors to be filled by lower priority action requests.

Command Fusion—is a cooperative arbitration approach that uses summation and normalization of proposed motor commands to derive the resultant motor commands. The input of all contributing behaviors are used on a per motor command basis to form the resultant command vector.

Highest Activation— is a winner-take-all arbiter that returns the action set with the highest vote value. This approach provides a dynamic mechanism for competitive selection by allowing behaviors to indicate their urgency for activation. Associated behavior weights are used to internally tune global performance by scaling the votes of behaviors that either over or under vote. The concept of activation levels is synonymous with the concept of utility in market based systems.

Activation Fusion—is a semi-cooperative arbiter that uses a highest activation selection approach on a per motor command basis. Unlike highest activation, activation fusion builds a new action set, allowing the motor commands left unspecified by behavior with high levels of activation to be set using the recommendations of behaviors with lower activation levels. When used with market based systems, this technique is easily referred to as utility fusion, but risks confusion with DAMN's utility fusion [33].

5.1.2 Behavior Structures

The ability of the UBF to easily assemble elemental components into operationally sound structures encourages developers to experiment with different structural arrangements. This approach isolates the task of troubleshooting a system's outward behavior to the structural arrangement of independent control elements that have been previously validated. The experiment in this section demonstrates how structural changes affect the global attributes of a behavior, and that the UBF provides a straightforward means to experiment with changes to structure and arbitration.

To provide a quantitative comparison of the relative effectiveness of one behavior structure to another, the experimental structures are evaluated in relation to the performance of a benchmark architecture. The benchmark's behavior control architecture is shown in Figure 9a, and consists of the wander, ramming behavior, and track-fire behaviors arbitrated by an activation fusion arbiter. The observed behavior of the benchmark has two operating modes, one that wanders the battlefield attempting to track and shoot opponents and another that aggressively charges towards a weaker opponent with guns blazing.

The Subsumption, motor schema, circuit architecture, action selection, and utility fusion behavior based architectures use the structure shown Figure 9b. Each differs in the arbiter used in the composite behavior; the mapping of each architecture to an arbiter is shown in Table 1, with the arbiter noted in the Behavior Structure column. The exceptions are the benchmark, the Monte Carlo arbiter, which has no corresponding existing behavior base architecture, and the colony architecture. The priority-based hierarchy approach specified by the colony architecture [12], is used to demonstrate how hierarchical control structures can be formed using the UBF. Two variations of the colony architecture are introduced, each grouping the three shooting related behavior elements into a fire control branch. The first structure, Colony A shown in Figure 10a, uses strict priority based arbitration at both levels of the hierarchy as would be done in a colony architecture. The second structure, Colony B shown in Figure 10b, is identical but uses priority fusion arbitration instead, and shows how the UBF can extend beyond the discussed behavior based architectures.

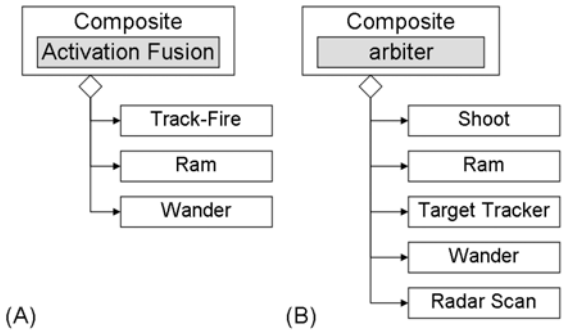

Figure 9: (A) The behavior structure of the benchmark; (B) The standard behavior structure. The arbiter is unspecified to allow each of the six arbitration approaches to be evaluated in-turn.

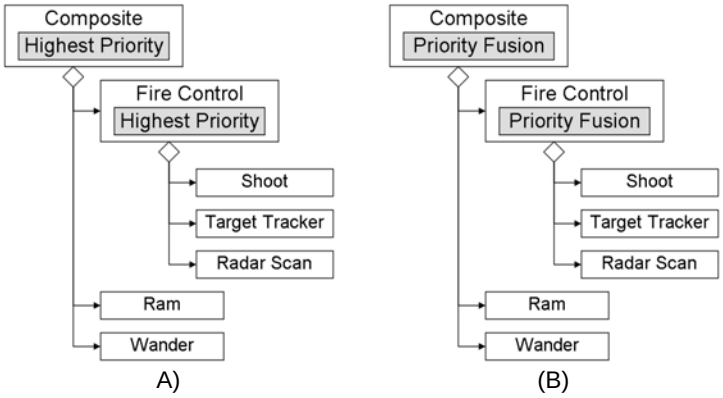

Figure 10: The two colony representation A) Colony A is a traditional colony architecture using highest priority arbiters in a hierarchal structure; and B) Colony B, an adaptation that uses priority fusion to support the concurrent activation of sub-behavior requests

5.1.3 Observations and Assessment

The quantitative measures for each structure's performance, measured relative to the capabilities of the benchmark behavior, are presented in Table 1. The rating of each member is measured using the final score from a fifty round battle against a robot running the benchmark behavior control architecture. Rankings are the average percent difference of the benchmark's score; values above zero indicate superior combat skills while below zero ratings indicate an inferior level of performance. Subjective observations about a behavior's apparent coherence are also made to aid in the discussion of the quantitative results.

Table 1: Relative performance of behavior structures.

Behavior Structure	Architecture Representation	Rating	Action Selection
Colony B	–	58%	Cooperative
Activation Fusion	Utility Fusion	57%	Cooperative
Priority Fusion	Circuit Arch	33%	Cooperative
Command Fusion	Motor Schema	20%	Cooperative
Benchmark	–	0%	Cooperative
Highest Priority	Subsumption	–66%	Competitive
Highest Activation	Action-Selection	–73%	Competitive
Monte Carlo	–	–76%	Competitive
Colony A	Colony Arch	–82%	Competitive

From the empirical results presented in Table 1, the most noticeable grouping is that the cooperative based arbitration techniques (motor schema, circuit, utility fusion, and Colony B), perform better against the benchmark while the competitive arbiters (Subsumption, action-selection, Colony A, Monte Carlo) perform significantly worse against the benchmark.

The reason for the performance gap is rooted in the functional capabilities of the base behaviors. The behaviors made available for this study are atomic in nature and are not intended to be coherent when used alone. Therefore the arbiters that simulate separate control circuits for each motor command allow continuous tasks like target tracking to continue while behaviors that

affect motion, like ram and wander, compete to provide chassis control. This approach is effective because the efforts of target tracking are usable by the shoot and ram behaviors without having to reproduce the control logic. For example, activation and priority fusion arbiters demonstrate a robust blending of independent behaviors, switching between charge and wander behaviors while still tracking and firing on opponents. This produces an effective emergent behavior that is not expressly developed.

The two main limitations plaguing the competitive arbitration techniques are starvation and disruption, which can both be tied back to the functional abilities of base behaviors. Starvation occurs when higher priority or behaviors with higher activation levels are continuously active and never yield to the other capabilities present in the structure, thus limiting the global capabilities of the structure. The competitive arbiters uniformly sufferer from disruption because the goals being pursued by individual behaviors stop when control is handed over to higher order tasks that do not continue the lower order tasks that support them. For example, the Subsumption, action-selection, and Colony A behavior based systems, are all capable of tracking and firing, but tend to not move. When they do move it is because the ram behavior activates for a close target, but because of the priority, the robot ceases tracking and loses the target.

The columns in Table 1 that classify an arbiter as competitive or cooperative highlight the split around the need to allow a blending of the behavior recommendations. Due to the tailoring of the behaviors toward cooperative arbitration, the fact that all of the cooperative arbiters outperform the competitive underscores the interchangeability of the arbitration components and that the choice between cooperative and competitive paradigms is a design choice that must be made when developing the task decomposition that is translated into the behaviors. This is not meant to generalize or suggest that cooperative approaches are always better than competitive ones. In fact, the command fusion arbiter is the only pure cooperative arbiter presented in this study, the others are simply a fine grain implementation of competitive approaches on a per motor command basis. The ranking results only indicate how well each arbiter supports the specific behavior set used in this study and in no way suggest that competitive arbitration techniques are poor in general.

In assessing the time and effort spent during implementation of these approaches. The majority of the effort occurs during the implementation and the testing of the individual behaviors. Once the behaviors are completed, they are then connected using an arbitration hierarchy. Because of the composite pattern in the hierarchy, the connection of the behaviors is completed and alterable very quickly. Changing an arbiter is often accomplished by altering a single line in a configuration file or object factory.

5.2 Pioneer 2AT-8 Implementation

From the success of the UBF in the Robocode simulation domain, demonstration of the UBF in a goal directed setting is performed using a Pioneer P2-AT8 robot running the Player control suite [18]. This implementation demonstrates that the system is able to switch between behavior heirachies based on input signals. For this study the task of the deliberator and sequencing layers is enacted via human input, but demonstrates the cability for UBF to be a reactive component for higher layers to dynamically adjust the low-level control mechanisms to pursue higher order goals and plans. Using the controller design described in Section 3, the system is able to switch between behaviors at runtime without disrupting the robot's low-level execution control loop.

Figure 11 presents the three control behaviors available for the robot controller to use: a halting behavior shown in (A), a random-wander behavior, which randomly moves while avoiding obstacles is shown in (B) and a goal-seeking behavior shown in (C) which is a control structure that incorporates the existing random-wander behavior as a means of obstacle avoidance, with a behavior which directs the robot along a direct route to a specified location. The use of a highest activation arbiter allows the goal seeking component to yield to the random-wander behavior for a period of time in an attempt to bypass the obstacle in the path toward the goal and then out vote it to make another attempt at moving towards the goal.

To demonstrate the ability of the reactive controller to use its available behaviors interchangeably, the robot it is expected to switch from the halting behavior to the goal seeking behavior and move to a partially obscured goal ten meters from its starting position. Upon achieving the goal location the robot must then wander the local area.

The Pioneer used is equipped with 16 sonars to sense obstacles and relies on the dead reckoning navigation capabilities supplied by Player [18]. Keyboard commands over a wireless link are used to simulate instructions from a sequencing element to the controller.

At the start of the experiment the controller switches its active behavior from *halt* (Figure 11a) to goal seeking (Figure 11c), immediately aligning itself on the bearing to the goal. After moving forward approximately five meters, the mid-course obstacle is detected. The influence of the embedded *random-wander* behavior becomes apparent as the *gotoXY* behavior, sensing the obstacle, enters a fifteen second back-off period. The displacement of the robot (from the logged odometry) during this period is shown in Figure 12. When the *gotoXY* behavior resumes it reorients the Pioneer to face the goal coordinate, pursuing what is ultimately an unobstructed path to the goal. Upon reaching the goal, user input is used to set the robot's active behavior to wander (Figure 11b).

This experiment shows the ability of the UBF to dynamically alter its active behavior hierarchy during execution to meet user or long term goals, without affecting performance. It also shows how the ability to use independent and specialized behaviors in a sequential manner can effectively achieve goals without increasing the complexity of the base behaviors. Additionally, the ability

to reuse the existing random-wander behavior in the formation of the goal seeking behavior underscores that the UBF can reduce the time required for development and testing.

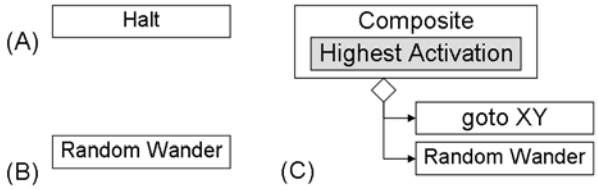

Figure 11: Behavior library for the Pioneer 2AT-8 robot.

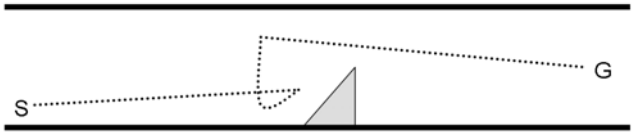

Figure 12: The observed path of the Poneer 2AT-8 robot as it navigates a horizontal hallway from its starting location (S) to the target goal location (G). The shaded triangle represents the mid-course obstacle obscuring the robots path to the goal.

6. CONCLUSIONS

This paper demonstrates that five popular behavior-based control structures can be represented inside a single software framework, the Unified Behavior Framework (UBF). In addition to this capability, the UBF 1) eases the complexity of development and testing; 2) promotes code reuse; 3) supports designs that scale easily into large hierarchies of focused base behaviors, and 4) allows system developers the freedom to use the architectures that they feel will function the best.

Additionally, beyond providing a standardized interface, the critical aspects of existing reactive behavior architectures are evaluated and standard software engineering principles are applied to establish mechanisms that accommodate these critical aspects. This is demonstrated with the UBF being used to represent many of the common reactive architectures as interchangeable modules. The use of the composite pattern [16] allows arbitrated hierarchies to be easily formed from existing behaviors and arbiters.

Simulation results highlights that the UBF ensures the ease of reuse, reduces complexity, and allows freedom of architecture choice. The structures presented, despite being constructed from the same set of base behaviors, split themselves into two performance categories when evaluated against the benchmark. The reason for the performance gap is that the base behaviors deliver tasks that were designed to support each other, thus arbiters that perform action selection on a per motor command basis deliver higher performance than those making a single action selection.

As a suggestion for future work, the independent and modular construction of the unified behavior framework behaviors lends itself to concurrent and parallel implementations. The YARA project [11] demonstrates the importance of using real-time systems to maintain the responsiveness that is critical to safety in robot operations. As the reactive elements of behavior-based systems grow in complexity the responsiveness of the system must be maintained. This suggests that an ability to schedule behaviors at periodic intervals is needed to allow for growth in computational complexity without jeopardizing the requirement that the robot remain responsive. The rate of change in the environment would directly influence such a schedule.

ACKNOWLEDGEMENTS

The authors would like to acknowledge funding through AFRL Lab Task 06SN02COR from the Air Force Office of Scientific Research, Lt Col Scott Wells, program manager. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

REFERENCES

[1] R.C. Arkin, “Behavior-Based Robot Navigation for Extended Domains,” *Adaptive Behavior*, vol. 1, pp. 201-225, 1992.
[2] R.C. Arkin, *Behavior-Based Robotics*. Cambridge, MA: MIT Press, 1998.
[3] A. Atrash, and S. Koenig, Probabilistic Planning for Behavior-Based Robots, *Proceedings of the 14th International FLAIRS Conference (FLAIRS)*, pp. 531-535, 2001.
[4] R. Bischoff, and V. Graefe, Learning from Nature to Build Intelligent Autonomous Robots, *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Beijing, China, pp. 3160-3165, 2006.
[5] D. S. Blank, D. Kumar, L. Meeden, H. Yanco, "Pyro: A Python-based Versatile Programming Environment for Teaching Robotics," *Journal on Educational Resources in Computing (JERIC)*, vol. 4:3, 2004.

[6] V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology*. Cambridge, MA: MIT Press, 1984.
[7] R. A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 14-23, 1986.
[8] R. A. Brooks, "Elephants Don't Play Chess," *Robotics and Autonomous Systems*, vol. 6, pp. 315, 1990.
[9] R. A. Brooks, "New Approaches to Robotics," *Science*, vol. 253, pp. 1227-1232, 1991.
[10] M. Carreras, P. Ridao, J. Batlle, and T. Nicosevici, Efficient learning of reactive robot behaviors with a Neural-Q-Learning approach, *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1020-1025, 2002.
[11] S. Caselli, F. Monica, and M. Reggiani, "YARA: A Software Framework Enhancing Service Robot Dependability," *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference*, pp. 1970- 1976, 2005.
[12] J. Connell, "A Behavior-Based Arm Controller," in *IEEE Transactions on Robotics and Automation*, vol. 5, 1989, pp. 784-791.
[13] M. Dong, and Z. Sun, A behavior-based architecture for unmanned aerial vehicles, *Proceedings of the 2004 IEEE International Symposium on Intelligent Control*, Taipei, Taiwan, pp. 149-155, 2004.
[14] R. J. Firby, "Task Networks for Controlling Continuous Processes," *Proceedings of the Second International conference on AI Planning Systems*, Chicago, IL, 1994.
[15] J. Gallagher, An Evolvable Hardware Layer for Global and Local Learning of Locomotion Control in a Hexapod Robot. International Journal on Artificial Intelligence Tools, vol. 14:6, pp. 999-1017, 2005.
[16] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns*. Boston, MA: Addison-Wesley, 1994.
[17] E. Gat, "On Three-Layer Architectures," in *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*. Cambidge, MA: AAAI Press/MIT Press, pp. 195 – 210, 1998.
[18] B. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323, 2003.
[19] E. Gordon and B. Logan, Managing Goals and Resources in Dynamic Environments, *Visions of Mind: Architectures for Cognition and Affect*, ed. Darryl N. Davis, Hershey, PA: Information Science Publishing, pp. 225-253, 2005.
[20] F. Hoffmann, Evolutionary Algorithms for Fuzzy Control System Design, *Proceedings of the IEEE*, Vol 89:9, 1318-1333, 2001.
[21] R. Huq, G.K.I. Mann, and R.G. Gosine, Behavior-based Robot Control using Fuzzy Discrete Event System, *2006 IEEE International Conference on Fuzzy Systems*, Vancouver, BC, Canada, pp. 1146-1153, 2006.
[22] L. P. Kaelbling, "An Architecture for Intelligent Reactive Systems," in *SRI International Technical Note No. 400*. Menlo Park, CA, 1986.
[23] K. Konolige, "The Saphira Architecture: A Design for Autonomy," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 9, pp. 215-235, 1997.
[24] Y. LeCun, W. Muller, J. Ben, E. Cosatto, and B. Flepp, Off-Road Obstacle Avoidance through End-to-End Learning, *Advances in Neural Information Processing Systems (NIPS 2005)*, MIT Press, 2005.
[25] B. Lussier, M. Gallien, J. Guiochet, F. Ingrand, M.-O. Killijian and D. Powell, Planning with Diversified Models for Fault-Tolerant Robots, *ICAPS 2007 The International Conference on Automated Planning & Scheduling*, Providence, RI, pp. 2-9, 2007.
[26] P. Maes, "Situated Agents Can Have Goals," in *Robotics and Autonomous Systems*, vol. 6, pp. 49 – 70, 1990.
[27] S. Magg, and A. Philippides, GasNets and CTRNNs – A Comparison in Terms of Evolvability, *From Animal to Animats 9*, pp 461-472, 2006.
[28] H. P. Moravec, *Robot rover visual navigation*: UMI Research Press Ann Arbor, Mich, 1981.
[29] N. J. Nilsson, *Shakey the Robot*: SRI International, 1984.
[30] J. O'Kelly, and J.P. Gibson, RoboCode & Problem-Based Learning: A non-prescriptive approach to teaching programming. *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE06,* Bologna, Italy, pp. 217-221, 2006.
[31] G.L. Peterson, and D.J. Cook, Integrating Decision Theoretic Planning in a Robot Architecture, *Robotics and Autonomous Systems*, vol. 42:2, pp. 89-106, 2003.
[32] J. Rosenblatt, "DAMN: A distributed Architecture for Mobile Navigation," presented at the AAAI Spring Symposium on Lessons Learned for Implemented Software Architectures for Physical Agents, Palo Alto, CA, 1995.
[33] J. Rosenblatt, "Utility Fusion: Map-Based Planning in a Behavior-Based System," in *Field and Service Robotics*, 1998, pp. 411 -418.
[34] P. Rusu, E.M. Petriu, T.E. Whalen, A. Cornell, and H.J.W. Spoelder, ANN arrangement of fuzzy controls, *IEEE Transactions on Instrumentation and Measurement,* Vol. 52:4., pp 1335-1340, 2003.
[35] L. Trujillo, G. Olague, E. Lutton, and F.F. de Vega, F.F., Discovering Several Robot Behaviors through Speciation, *Applications of Evolutionary Computing*, Berlin: Springer, pp. 164-174, 2008.
[36] H. Utz, G. Kraetzschmar, G. Mayer, and G. Palm, Hierarchical Behavior Organization, *2005 International Conference on Intelligent Robots and Systems*, Edmonton, Canada: IEEE/RSJ, pp. 2598-2605, 2005.