J Intell Robot Syst manuscript No.
(will be inserted by the editor)

Dynamic Behavior Sequencing for Hybrid Robot Architectures

Gilbert L. Peterson · Jeffrey P. Duffy · Daylond J. Hooper

Received: 17 May 2010 / Accepted: 27 December 2010

Abstract Hybrid robot control architectures separate planning, coordination, and sensing and acting into separate processing layers to provide autonomous robots both deliberative and reactive functionality. This approach results in systems that perform well in goal-oriented and dynamic environments. Often, the interfaces and intents of each functional layer are tightly coupled and hand coded so any system change requires several changes in the other layers. This work presents the dynamic behavior hierarchy generation (DBHG) algorithm, which uses an abstract behavior representation to automatically build a behavior hierarchy for meeting a task goal. The generation of the behavior hierarchy occurs without knowledge of the low-level implementation or the high-level goals the behaviors achieve. The algorithm's ability to automate the behavior hierarchy generation is demonstrated on a robot task of target search, identification, and extraction. An additional simulated experiment in which deliberation identifies which sensors to use to conserve power shows that no system modification or predefined task structures is required for the DBHG to dynamically build different behavior hierarchies.

This work was supported in part through AFRL/RYRN Lab Task 06SN02COR from the Air Force Office of Scientific research. The views expressed in this article are those of the author and do not reflect the official policy or position of the United State Air Force, Department of Defense, or the U.S. Government.

Air Force Institute of Technology
2950 Hobson Way, WPAFB OH 45433
E-mail: daylond.hooper.ctr@wpafb.af.mil

G.L. Peterson
E-mail: gilbert.peterson@afit.edu

J.P. Duffy
E-mail: gilbert.peterson@afit.edu

Published Online: 13 January 2011

1 Introduction

Many successful autonomous system architectures follow a layered approach, decoupling immediate responses from longer-term objective reasoning [Urmson, et al.(2008)]. These systems receive a high-level tasking and attempt to complete the task autonomously using a combination of deliberative planning and reactive execution. However, many of these systems require code restructuring when the environment or functional specifications change. The single purpose design often makes the reactive behaviors and their hierarchies specific to the robot and task [Scheutz and Andronache(2004)], and the connections between different layers of the architecture [Gat(1997)] are tightly coupled (where small changes in one layer may result in significant changes in the others). Because of this, a change in purpose of the robot requires significant redevelopment because the hierarchies are handcoded [Hurdus and Hong(2008)]

In linking deliberation with execution two processes exist, task execution and sensor management. Task execution transitions deliberation into action, abstract concepts become motor commands. Alternatively, sensor management deals with abstracting low level sensor data into concepts for deliberation. Sensor management tasks include identifying task completion and performing fault diagnosis[Murphy and Hershberger(1999), Parker and Kannan(2006)]. The dynamic behavior hierarchy generation (DBHG) algorithm focuses on task execution, and rather than make use of user supplied scripts, task networks, or behavior hierarchies is designed to be build these and be modular for use in conjunction with existing sensor management work.

The majority of hybrid architectures link planning to execution using task-level control languages [Simmons and Apfelbaum(1998)]. These languages require that each behavior hierarchy is expressed explicitly by the syntax of the language. The constructs of the language limits the implementation, and requires the robot software designer to adjust the behavior hierarchy (or task network) when a behavior is rewritten or when the system goals change. We present an alternative representation that generically describes each behavior for sequencing. This representation is used with the DBHG algorithm that translates a high-level tasks into a behavior hierarchy (or task network) that can complete the task. This aids in decoupling the layers by uniformly representing the behaviors. The behavior representation and DBHG algorithm's implementation within the proposed architecture design enables a modular, robust system that requires no code development in the task sequencer during a significant system modification.

The DBHG algorithm is demonstrated as the task sequencer component of a hybrid architecture on a Pioneer P2AT8 robot, and in a Player/Stage simulation. The experiments show that the DBHG algorithm can use the behavior representation to dynamically generate an arbitrated behavior hierarchy for accomplishing desired goals without *a priori* knowledge of system capabilities and behavior functionalities. Using the behavior representation without knowledge of the underlying implementation shows that the DBHG algorithm operates independent of behavior implementation. Additionally, the behavior representation acts as the defined mechanism for linking the planning of the behavior hierarchies to their execution.

2 Related Work

Hybrid robot control architectures have separate functional components (or layers) for plans, coordination, and actions. These layers can be generalized into three composite layers based on increasing levels of abstraction and temporal complexity: a reactive feedback control mechanism (Controller), a deliberative planner (Deliberator), and a sequencing mechanism that connects the first two components (Sequencer) [Gat(1997)]. This design approach promotes systems that perform well in goal directed and dynamic environments at the expense of system complexity. Additionally, implementations of the architecture typically couple the connections between these layers tightly, so changes within one layer require modifications in other layers.

All layered architecture creators describe the layers in detail [Bonasso et al(1997), Connell(1992), Konolige et al(1997), Simmons et al(1997)] and have shown that a layered architecture accomplishes the merging of reactive execution with deliberative planning. Many of these architectures make use of sophisticated plan execution systems that continuously interleave planning and execution [Infantes et al(2006)], referred to as task-level control languages.

Task-level control languages link the Sequencer to the Controller. These languages require that each behavior (e.g. task-net [Firby(1989)]) is expressed explicitly by the syntax of the language. Although the expressed behaviors are limited to the constructs of the language, they demonstrate the benefit a behavior representation can have toward automated behavior sequencing. The common plan execution languages implemented in various hybrid architectures include Reactive Action Packages (RAPs) [Firby(1989)], Procedural Reasoning System (PRS) [Ingrand et al(1996)], Reactive Plan Language (RPL) [McDermott(1991)], Executive Support Language (ESL) [Gat(1997)], Propice-Plan [Despouys and Ingrand(2000)], and Reactive Model-Based Programming Language (RMPL) [Williams et al(2003)].

The RAP (Reactive Action Package) [Firby(1989)] representation groups together all known ways to accomplish a task in various situations. Each RAP is a sequenced collection of methods for accomplishing a particular task. The RAP selected for execution is the one that most effectively accomplishes the task for a given environment at that time. The three major components of a RAP are: index, success clause, and the methods that accomplish the task for different situations. The index is the task that a particular RAP achieves. The success clause identifies the test used to indicates when the task is completed, and the methods describe the behavior sequencing to accomplish the task for different environments. A drawback of RAP-based systems is that the programmer must know all of the system's potential tasks and develop task-net descriptions to accomplish these tasks for every possible situation the system may encounter. This hard-coded approach limits the ability to apply these systems to dynamic and diverse environments.

Reactive Plan Language (RPL) is descended from the RAP notation with a few differences [McDermott(1991)]: RPL plans look like Lisp programs, explicit constructs exist for high-level concepts (interrupts and monitors), and world state is not maintained by the interpreter. An RPL plan describes an environment driven behavior governed by temporal changes (fluents). Fluents are conditional statements that detail a condition that must be true for an action to be executed. These fluents are controlled by state model variables or direct sensor input. RPL uses projection mode to anticipate whether the current plan can achieve the desired goals. Dynamic environments make this internal simulation difficult in that the computation time alone may require more time than the changing environment allows. This approach is also hard-coded to the environment, thereby limiting its more general applicability for dynamic and complex environments.

Executive Support Language (ESL) is a language for encoding execution knowledge in embedded autonomous agents [Gat(1997)]. Unlike RAPs and RPL, ESL is not intended for automated reasoning or formal analysis. Instead, ESL uses a cognizant failure concept to handle errors and failures. ESL assumes failures are inevitable and can recover from errors including infinite loops. The behaviors in ESL are identified by the goals that they achieve and the conditions that make the behavior's method appropriate. ESL supports multiple concurrent behavior execution by allowing behaviors, or tasks, to wait for events. However, a detailed description of the behavior's execution is required, along with *a priori* knowledge of the working environment. These limit the applicability of ESL for more sophisticated environments.

Similar to ESL, several architectures exist which include components (sensor managers) that monitor sensors for failures. In the case of Sensor Fusion Effects-Exception Handling (SFX-EH)[Murphy and Mali(1997),Murphy and Hershberger(1999)], it reallocates a sensor, tries to reinitialize the sensor, or throws a behavior fault which is caught by the task manager. The task manager includes a schema of the individual behavior to execute given the current sensor failure. SFX has been expanded to function across multiple cooperative robots in DFRA[Long et al(2005)]. The generation and testing components of the sensor manager exists in DFRA, however, another robot's sensor may be used to complete the task. In Learning-based Fault diagnosis (LeaF)[Parker and Kannan(2006)], case-based reasoning on a pre-specified partial causal model is used to learn cases for an extended SFX-EH. All of these architectures focus on identifying a failure and the correct response, where the correction is encoded as a script[Murphy and Mali(1997)], not on building a behavior hierarchy.

The Procedural Reasoning System (PRS) uses behavior libraries that map out low-level behavior activations to represent and execute procedures, scripts, and plans in dynamic environments [Ingrand et al(1996)]. PRS maintains a world model with derived beliefs or user entered static beliefs [Ingrand et al(1992)]. The behaviors of the robot system are described by the goals in PRS. Knowledge for accomplishing these goals are stored in knowledge areas (KAs). KAs are declarative procedure specifications that describe the conditions for which the KA is useful and steps for accomplishing the goals of the KA. Each KA can be viewed as a task-tree that requires certain goals to be true in order to activate the next step. The plans in the library are not combined to create other plans and thus PRS does not promote behavior or plan reuse.

Despouys and Ingrand [Despouys and Ingrand(2000)] propose Propice-Plan which couples planning with plan execution using an operational plan. Each operational plan uses PRS to describe the behavior execution, and includes the goal and initial conditions required for execution. The LAAS architecture uses the temporal planner ($I_X T_E T$) with Propice as a procedural executive to close the loop between the levels of the architecture [Lemai and Ingrand(2004)]. Propice-Plan does not interpret the plan built but identifies, given the world state and the desired goal, which operational plan to execute. The operational plans contain a procedural context describing the behavior execution, thus limiting robustness and flexibility since the specifics of the behavior execution must be known.

The Reactive Model-Based Programming Language (RMPL) [Williams et al(2003)] is an object-oriented, constraint-based language that follows a model-based programming approach. The constructs of RMPL provide behaviors with conditional branching, preemption, iteration, and concurrent and sequential composition. These constructs enable behaviors that range from simple, reactive behaviors to complex, multi-task achieving behaviors. Therefore, RMPL can offer some of the goal-directed tasking and monitoring capabilities that RAPs and ESL offer. RMPL employs Titan, a dedicated executive for controlling robot behavior. Titan uses a reactive control loop to monitor the current state for failures and transition from the current state to the desired goals. In RMPL, much of the low-level details, often coded within a behavior, are deduced and analyzed within the deductive controller. This convention does not promote a reactive sensor-to-action pair behavior implementation since the behavior is decomposed further in the Controller. Furthermore, the decomposition of the behavior model may limit a behavior's reactivity since decomposition is performed every cycle.

The primary weakness in using task control languages is that they do not follow modular software coding practices and rely heavily on language understanding and syntax. The tasks and associated planning are programmed using these language constructs, which dictates behavior implementation. This dictation prevents application of an abstract representation, thus restricting the implementation. This results in systems which cannot change a behavior's implementation without changing its description. Another weakness is their tight coupling and interleaving of the sequencing and control layers' functional components. Often, the plan execution languages perform the functions of both the Sequencer and Controller without a clean division between the two layers. Therefore, changing the intended environment or system capability requires an almost complete rewrite of the sequencing layer. Thus, we present a representation and DBHG algorithm which provides a departure from this tight coupling.

3 Sequencer Control Logic

To automate the link between the Sequencer and the Controller, we create a control algorithm in the Sequencer that generates a behavior hierarchy. The control loop begins by receiving, from the Deliberator, a goal-set (or objectives plan (OP)) that describes the tasks and order in which they are to be met. The DBHG algorithm at the core of the Sequencer searches through the library of behaviors and generates a behavior hierarchy. The behavior hierarchy is the collection and organization of the behaviors and arbiters that accomplishes the objectives set forth by the Deliberator. This section presents the Sequencer design, the representation used for the behaviors, and the representations used in the DBHG algorithm.

3.1 Behavior Representation

Behaviors accomplish a specific set of tasks or subtasks in a specific environmental context. When the robot is in the context, the behavior responds with an action and a positive vote. If the robot is not in the situation, the behavior responds with a null action and a zero vote. The characteristics of a behavior are abstractly described, enabling the use of a mechanism for searching and selecting appropriate behavior activations and deactivations to accomplish desired objectives. This uniform handling of abstract behavior descriptions improves the robustness of the Sequencer.

A simple behavior's description contains a set of outputs triggered by a set of input conditions. The input conditions dictate specific output conditions, though a behavior may also affect other conditions. Behaviors control various settings, accomplish different abstract goals, and suggest a confidence value (vote) for its action recommendations. All of these characteristics play a role in choosing combinations of behaviors to accomplish high-level tasks. Each behavior representation reflects a single functionality of the behavior and is represented as the tuple $\{I, P, D, G, C, v\}$, where I and P are the initial and post conditions, D is the sensor data required, G is the set of high-level abstract goals, C is the set of system controls (e.g., motor outputs) that the behavior sets, and v is the behavior's vote, generated when it delivers an action recommendation. An example behavior performing obstacle avoidance using a LiDAR is shown in Table 1.

Table 1: Example behavior *LaserAroundObstacle*.

LaserAroundObstacle	
Initial Conditions (I) - Active - Passive	 not-carrying threshold-min
Post Conditions (P) - Add - Delete	 avoid-obstacle-target
Required Data (D)	Laser
Goals Achieved (G)	AVOID_OBSTACLE_TARGET
Action Settings (C)	Velocity, RotationVelocity
Vote (v)	0 (if nothing is close), 7 (if an obstacle is close)

3.1.1 Initial Conditions (I)

The initial conditions of a behavior represent the environment variables that, when true, generate an action recommendation and vote from the behavior. Additionally, I represents the conditions required for the behavior to produce the post condition (P). There are two types of initial conditions: active and passive. Active conditions are the initial conditions that are actively pursued to activate the behavior. Passive conditions cause the behavior to activate but are not actively pursued for task completion. For example, the *LaserAroundObstacle* behavior, which does not vote to control movement until it reads that there is an obstacle within its projected path (threshold-min), does not lead to completion of a sub-task. However, the behavior itself is necessary whenever the robot is moving to ensure it does not run into obstacles. Note that this representation of the initial conditions is an example and does not dictate the manner of detecting and avoiding obstacles.

3.1.2 Post Conditions (P)

The post condition represents the set of environment effects that the behavior intends to achieve. This intent is based on action recommendations for the behavior at the given initial condition I. These post conditions may invalidate other goals, so a plan may require multiple steps or multiple simultaneous behaviors to be accomplished.

3.1.3 Required Data (D)

Since a behavior is a tight coupling of sensor readings to motor commands, D represents the set of sensors (or data) required for the behavior to function properly. This data also includes computed data that is not directly from a sensor. For example, the LaserAroundObstacle behavior requires only laser data, but a more advanced obstacle avoidance behavior may use a virtual field histogram to avoid obstacles.

3.1.4 Abstract Goals (G)

The behaviors in a system exist to accomplish a specific task. For example, the *LaserAroundObstacle* behavior in Table 1 is used to complete the AVOID_OBSTACLE_TARGET_GOAL task. The representation of the abstract goal is in the same language used by the deliberator to generate the OP. Behaviors may meet multiple goals, such as HAS_ITEM, GRIPPER_UP, and GRIPPER_CLOSED. Since there are circumstances where a gripper should be up independent of having an item, the DBHG algorithm satisfies the task in the OP using the subset of the abstract goals as needed.

3.1.5 Control Settings (C)

Behaviors are written to affect settings for specialized controls. Most commonly, these are motor controls. Dependent upon which controls are set, the control loop determines the most appropriate arbiter for use with a set of behaviors. The controls that the behavior affects are denoted by the set C. Table 1 shows the sets of control settings (velocity and rotational velocity) for the *LaserAroundObstacle* behavior.

3.1.6 Vote (v)

The value v represents the vote for that behavior when in a state in which it acts. This value is a user-determined value that relays the strength of the action recommendation. A behavior may vote 0 when the robot is not in a state where the behavior would be useful and some other value when the state is different. This situational variability on the vote is dependent upon the behavior itself and its own evaluation of the state. The issued vote is used by the UBF [Woolley and Peterson(2009)] determine the action output, which is also dependent upon the arbitration technique applied.

3.2 Architecture Design

The Sequencer selects the behaviors the controller uses to accomplish long-term objectives. The Sequencer is designed such that after initial implementation, it requires minimal software maintenance and modifications for system changes. This is accomplished using the behavior representation and DBHG algorithm. The Sequencer uses the behavior representation as an abstract interface for sequencing the behaviors without knowledge of each behavior's concrete implementation. The transition from the Sequencer to the Controller is the passing of a composite behavior module that represents an arbitrated hierarchy of behaviors that, when executed, accomplishes high-level objectives.

The Sequencer contains a number of components that perform specialized tasks. Figure 1 shows the Sequencer's functional decomposition. These components are: Behavior Library, Resource Manager, Behavior Planner, and Behavior Executive. The interaction of these components is shown in the timing diagram (Figure 2).

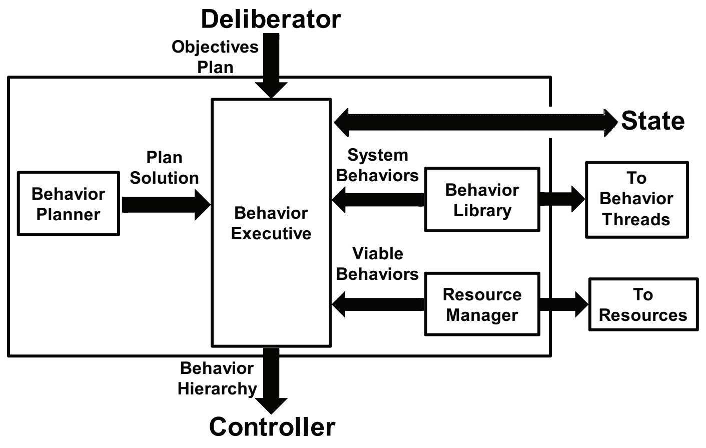

Fig. 1: Sequencer Component Breakout. The Behavior library contains all behaviors and the resource manager provides a subset of usable behaviors. The planner uses the state and this subset to generate behavior hierarchies which satisfy the objectives plan.

The Behavior Library maintains the implementations and representations of the behaviors contained within the system. The library initializes the behaviors and supplies the set of available behavior representations to the Sequencer. This allows the behavior planner to evaluate a behavior in the context of the behavior's required data, abstract goals, and pre and post conditions, regardless of the behavior's specific implementation. Only the Behavior Library requires knowledge of the addition or removal of behaviors, thus making the implementation transparent to the other components. This minimizes change throughout the system when adapting the robot to a new environment. The behaviors are defined in an abstract representation to facilitate this transparency (Section 3.1).

The Resource Manager monitors system resources (hardware and data) and optimizes based on planned objectives and power management in relation to current tasks. The Resource Manager also answers queries about the prospects for a behavior's activation based on resource availability. This allows the system to dynamically respond to low battery life, failure of sensors, and addition of new sensors. This availability provides a trimming of the set of behavior candidates used for task execution, thus reducing the plan space.

The Behavior Planner generates a set of behaviors that satisfy the OP. It uses the behavior representation to generate plans that are composed of a set of behaviors and the ordering constraints necessary to accomplish the OP. This functionality is currently implemented using a simplified version of RePOP (Reviving Partial Order Planning) [Nguyen and Kambhampati(2001)], a variant of Partial-Order-Planning (POP). However, due to the modular design of this framework, the BP may use any planning technique modified to generate a set of behaviors and a set of ordering constraints. Since the Behavior Executive generates a hierarchy of behaviors based on the planner's solution, a partial plan is sufficient for describing behavior interactions.

The Behavior Executive receives the OP from the Deliberator, then enters a hierarchy generation loop for translating the OP to an arbitrated hierarchy of behaviors that accomplish the objectives set forth by the Deliberator. The Behavior Executive

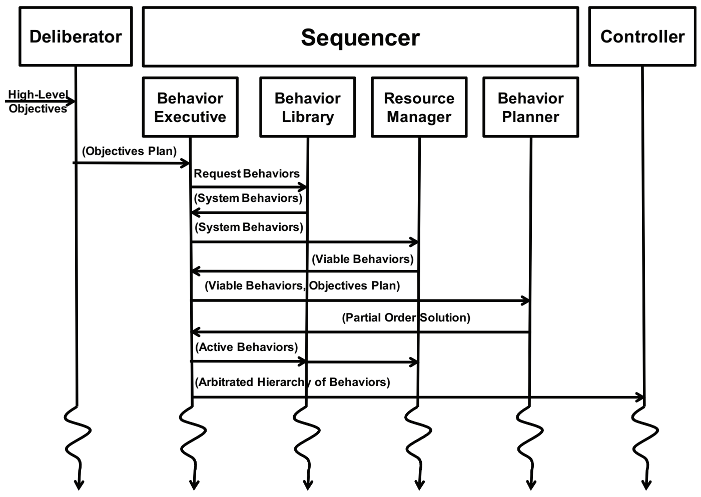

Fig. 2: Timing diagram for translation of an objective plan (OP) to the arbitrated hierarchy of behaviors.

requests a set of available system behavior representations from the Behavior Library. It then sends the available behaviors to the Resource Manager for analysis. The Resource Manager responds with the subset of behaviors that require only those resources (or data) which the system can currently supply. This subset of behaviors is sent to the Behavior Planner, which returns a partial order plan containing the subset of behaviors and required ordering constraints. The Behavior Executive creates an arbitrated hierarchy of behaviors, which for each OP is sent to the Controller, Behavior Library, and Resource Manager for execution. The Behavior Library activates only the behaviors of the hierarchy and deactivates the remaining behaviors. The Resource Manager manages the resources that the current hierarchy requires.

3.3 Dynamic Behavior Hierarchy Generation

The DBHG algorithm uses the behavior representation with the flexibility of the Unified Behavior Framework (UBF) controller [Woolley and Peterson(2009)]. The hierarchy generation uses the UBF's ability to support multiple arbitration processes, an issue identified in [Scheutz and Andronache(2004)] that has prevented this type of system in the past. It receives an objectives plan from the Deliberator and generates a sequence of arbitrated behavior hierarchies that accomplish the desired goals. By constructing the algorithm around the behavior representation, it creates a robust system that dynamically sequences behaviors based on goal requirements, resource availability, and behavior descriptions. The hierarchy generation algorithm performs the following control loop:

1. Receive objectives plan (OP) from Deliberator
2. Identify behaviors that require only data that the robot can provide
3. Generate a solution plan to the partial plan

4. Generate and validate an arbitration that accomplishes objectives and satisfies the solution plan
5. Generate arbitrated hierarchy of behaviors and send to controller
6. Monitor for progress, hardware changes, and new OPs

The DBHG algorithm receives an objectives plan providing the sequence and priority of task completion from the Deliberator. Passive tasks, like obstacle avoidance, are included in this set. Each task in the plan has an assigned sequence number and activation priority. The sequence number dictates the order in which the tasks must be completed. If two goals have the same sequence number, they are accomplished before the next sequence but in no particular order. However, when behaviors compete for resources, the behavior with the higher activation priority takes precedence.

The DBHG algorithm begins by placing the behaviors B that require only available data D into a library of viable behaviors for each sequence step of the OP. This process enables the use of a Resource Manager that conserves energy by deactivating sensors during critical times or due to failure.

3.3.1 Solution Generation

Since the initial conditions I and the post conditions P do not represent the atomic actions that Partial-Order-Planning (POP) expects [Russell and Norvig(2003)], a modified RePOP planner is used to incorporate the complexity of the concurrent taskings that each behavior may encounter.

The planner uses the initial conditions to determine the conditions that must be met for behavior activation. However, if the initial condition is a passive behavior, the planner ignores the conditions and assumes that the post conditions are met when necessary. This reduces the search space but also introduces new challenges to the planner since the initial conditions are not linked to the outputs of a previous behavior, but previous outputs could dictate the passive activation. This scenario is not handled until arbiter selection and validation (step 4).

The difference between this planning approach and the POP approach is that POP solves for a plan from an initial state to a specific goal condition. Here, the resultant plan is a set of actions that have priorities and sequence numbers so that hierarchical reactive behaviors can accomplish them. Furthermore, the response here is not scripted, so complex, pre-generated responses to sensor failures need not be present. This enables the incorporation of new sensors with little overhead.

3.3.2 Arbitration Selection and Validation

Appropriate selection of an arbiter ensures proper fusion, priority activation, and ordering of behavior action recommendations. The arbitration selection is based on the controls the behaviors affect (C) and how the behaviors vote for each branch and what that branch affects. For example, a garbage collection robot has a *ScanForTrash* behavior which controls only the camera and a *WallFollow* behavior which controls only the velocities. Since these two behaviors control different motors, they can operate concurrently. A Utility Fusion arbiter [Arkin(1998)] behaviors both control the forward and rotational velocities and compete for motor usage. A Highest Activation arbiter is best for this combination.

Currently, arbitration selection is chosen by a rule based method that chooses between a Highest Activation arbiter and a Utility Fusion arbiter [Arkin(1998)]. The decision between which arbiter to use is based on the controls the behaviors effect and the sequence of the OP. Since the sequence of behaviors is dictated by the OP, the behaviors that meet each sequence are selected by highest activation arbiters. If a behavior competes with another behavior for affective control, a highest activation arbiter is used. However, if the behaviors do not compete for affective control, a utility fusion arbiter is selected. If the selected hierarchy does not satisfy the initial OP and the generated solution, then the planning fails and exits the control loop.

The validation process of the arbitrated hierarchy of behaviors ensures the hierarchy satisfies the initial OP and the generated solution. Since a hierarchy is described as a composite behavior, it generates a behavior representation based on its output to every possible initial condition combination. By cycling through the possible environment conditions after each behavior's activation, a sequential ordering is generated to compare to the initial plan and the generated solution. If the sequential plan satisfies the initial OP and the generated solution, then the activation priorities of the OP are valid.

3.3.3 Progress Monitoring

The Sequencer also monitors the state for anticipated changes and dispatches the appropriate hierarchies at appropriate times. To accomplish this, the Sequencer has a state monitoring convention, which waits for the post conditions of the executing hierarchy to indicate goal completion. When monitoring for the post conditions of a hierarchy, the Sequencer identifies whether all of the adders are present and all of the deleters are absent. When these conditions are met, the Sequencer dispatches a new hierarchy to achieve the next task or, for the final task, indicates achievement of the entire plan.

The monitoring process also includes monitoring for conditions that potentially affect the currently planned hierarchies. An example of this is when the Resource Manager declares that a hardware change has occurred. Since the behaviors are initially selected based on the sensor data (or hardware) that it expects, the current (and subsequent) plans may contain behaviors that use hardware that is no longer available. This situation results in the replanning of the currently running hierarchy and any hierarchies that have been scheduled for later.

4 Results

The Sequencer presented uses a behavior representation as an abstract interface and a modified planning algorithm to build behavior and arbiter hierarchies to meet plan goals. The experiments demonstrate the robust architectural design by accomplishing high-level taskings using the same software implementation of the architecture on systems with different capabilities and active behaviors. They also demonstrate dynamic sequencing that occurs when the system and deliberator are concerned with power management as well as completing the tasks.

4.1 Physical Robot Experiment

The extensibility of the DBHG algorithm is demonstrated on an ActivMedia Pioneer 2AT robot (named Gollum). Gollum initially has 24 behaviors, each of which fulfills certain goals, applies action settings, and contains pre and post conditions. Among these behaviors are *GoToXY*, *AroundObstacle*, and *LaserAroundObstacle*. The specifics of these behaviors are shown in Table 2, with the exception of *LaserAroundObstacle*, which is shown in the example behavior description previously described in Table 1.

Table 2: Two initial behaviors.

GotoXY	
Initial Conditions (I)	N/A
Post Conditions (P) - Add - Delete	 desired-x-location,desired-y-location,all-stop
Required Data (D)	Position, Odometry
Goals Achieved (G)	XY_LOCATION
Action Settings (C)	Velocity, RotationVelocity
Vote (v)	3 (constant)

AroundObstacle	
Initial Conditions (I) - Active - Passive	 not-carrying threshold-min
Post Conditions (P) - Add - Delete	 avoid-obstacle-target
Required Data (D)	Sonar
Goals Achieved (G)	AVOID_OBSTACLE_TARGET
Action Settings (C)	Velocity, RotationVelocity
Vote (v)	0 (if nothing is close), 4 (if an obstacle is close)

These behaviors are selected for a plan if the OP has the goals XY_LOCATION and AVOID_OBSTACLE_TARGET. A utility fusion arbiter is selected for use by the UBF, so for behaviors that have the same action settings, it is a winner-take-all behavior execution. Thus, if both the Laser and the Sonar are available, the *LaserAroundObstacle* behavior dictates the motor output when an obstacle is close. Otherwise, the *GoToXY* behavior dictates the motor output. If the resource manager were to indicate a failure or unavailability of the Laser, the *AroundObstacle* would handle the obstacle avoidance aspects of the task execution. The *GoToXY* behavior operates by first rotating the robot to the appropriate angle, then moving the robot to the indicated position. The obstacle avoidance behavior activates when an obstacle is close enough for the behavior to vote higher than 3. It then takes over, moves to an area where the obstacles are sufficiently far away as to be uninteresting, then votes 0, allowing the *GoToXY* behavior to take over once more. In areas where there are large or many obstacles, the robot may encounter a “see-saw” effect, switching from the *GoToXY* behavior to the *LaserAroundObstacle* behavior and back multiple times.

To address this effect, a Vector Field Histogram-based behavior was developed [Borenstein and Koren(1991)]. This behavior is designed to proceed to the target location via the widest available area within a certain span of the laser. Instead of swapping

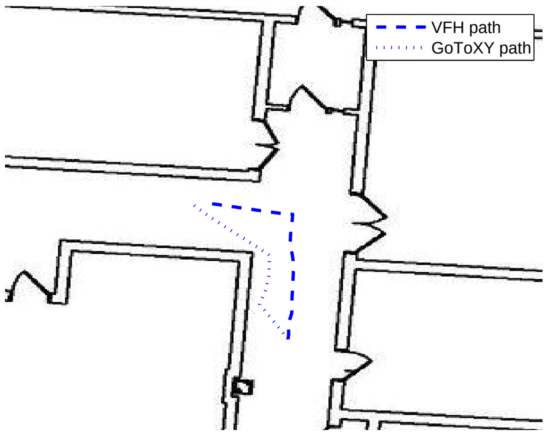

Fig. 3: The path Gollum traversed with and without the VFH behavior.

between two behaviors, it often proceeds in a more straightforward manner to the target position. The specification of the behavior is shown in Table 3.

Table 3: *VFH* Behavior.

VFH	
Initial Conditions (I)	N/A
Post Conditions (P) - Add - Delete	desired-x-location,desired-y-location,all-stop
Required Data (D)	Position, Laser, Odometry
Goals Achieved (G)	XY_LOCATION, AVOID_OBSTACLES_TARGET
Action Settings (C)	Velocity, RotationVelocity
Vote (v)	6 (constant)

The VFH behavior with the parameters as shown in Table 3 is added to the behavior library, and no further changes are made. The activity of the robot both with and without the VFH behavior are tracked on a task to traverse a hallway. The goal is to arrive at a location of (5, 4) from the initial position of (0, 0) in a hallway.

Figure 3 shows the different paths Gollum followed. Before the addition of the VFH behavior, Gollum followed the left wall and attempted multiple times to turn towards the wall and move through it. After crossing the threshold at which *LaserAroundObstacle* votes higher, it turned away from the wall and moved a short distance. The numerous attempts to turn back into the wall caused odometry drift, thus causing Gollum's final position to be off by 0.2 meters. However, the VFH behavior allowed Gollum to travel roughly down the center of the hall, turning only once and traveling to the target position. The final position is off by less than 0.05 meters. The addition of the VFH behavior to the robot enabled the robot to enter the target area with smaller error and in a shorter time (12 seconds vs. 28 seconds). The advantage of the DBHG framework in this context is that, besides the programming of the behavior itself, the time and effort it requires to integrate a new behavior into the robot's control architecture is minimal. The behavior is added to the behavior library, and the appropriate hardware requirements and goals are incorporated into the behavior. The system requires no additional overhead to incorporate the VFH (or some other) behavior into the architecture. The rapid deployment of behaviors allows the robot's overall architecture to be more compartmentalized, thus requiring minimal adjustment, reprogramming, or tuning of other architecture components.

4.2 Power Simulation Results

To demonstrate the DBHG algorithm building different behavior hierarchies for the same task under different constraints, we use the Player/Stage simulation platform [Gerkey et al(2003)] and have the robot locate and deliver targets to a specified location under different sensor restrictions and power management strategies. In this experiment, there is one robot, and it must repeatedly find the targets and deliver them to the trash can.

The power management strategies focus on allowing the use of LiDAR, sonar or both. The strategies include: a) no power management, all sensors are available, b) strict, only sonar is available, c) lenient, both LiDAR and sonar are available until a critical power threshold (15 percent) is passed at which point only sonar is available, and d) predictive, both sensors are available and power management is handled by the deliberator. The predictive method uses the Sequential Planning Using Decision Diagrams (SPUDD) [Hoey et al(1999)] Partially Observable Markov Decision Problem (POMDP) planning algorithm as the deliberator.

In the environment, an assumption is made that the first target is in an environment which can be traversed using sonar. The second target however is in a more complex environment in which the robot has a high probability for getting stuck if only using sonar (0.9 probability of success using LiDAR, 0.3 using sonar). The predictive method outputs a set of OPs that suggest using the sonar for the first target and the LiDAR for the second. Table 4 shows the modeled power requirements for each sensor. The costs in SPUDD are set to 4.0 for the sonar, and 1.0 for the LiDAR.

Table 4: The amount of power consumed for each device.

Device	Power (watts)	Discharge (units/ms)	Error (%)
Laser	20	0.1667	±20
Blobfinder	12	0.1000	±20
Gripper	10	0.0833	±20
Sonar	0.7	0.0058	±10
Bumpers	0.25	0.0021	±10
Motors	0.19-13.29	0.0016-0.1108	-
Controller/PC	12.6-19.6	0.1050-0.1633	-

The behaviors used in this experiment include *GotoXY, GotoXYT, AroundObstacle, Release, Grab, AllStop, WallFollow, ApproachTarget* and *LocateTarget* behaviors. There are two *WallFollow, ApproachTarget,* and two *AroundObstacle* behaviors, one using the LiDAR and the other the sonar. The LiDAR requires more power but is more accurate, enabling the robot to traverse more difficult environments.

To accomplish the high level tasking, there are four OPs: find the target, pick it up, bring it to the designated area, and drop the target off. These OPs, including the goals, sequence numbers, activation priority (or rank), and goal parameters, are shown in Table 5. The table represents one cycle of finding one target and disposing of it. When these OPs are repeatedly sent to the Behavior Executive, the result is that every target is placed in the designated area.

Table 5: Objectives Plans for collecting targets and delivering to the destination area.

OP	Goal	Seq	Rank	Params
Find/Get	EXPLORE	1	1	N/A
	VISUAL_TRACK_OBJECT	1	1	Yellow
	GRAB_OBJECT	2	1	N/A
Deliver	AVOID_OBSTACLE_TARGET	1	2	N/A
	GOTO_XYT	1	1	x = 5.5; y = -5.5; $\theta = 0.0$
Drop	RELEASE_OBJECT	1	1	N/A

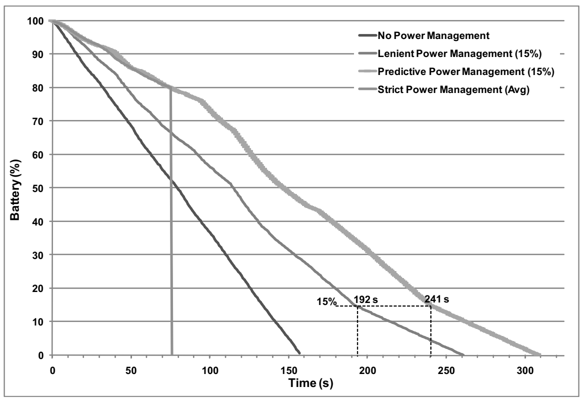

Fig. 4: The results from four trials of a test sequence of goals (showing only the 15% thresholds).

When the power management scheme allows for the use of the LiDAR, the behavior hierarchy generated includes the WallFollow and AroundObstacle make use of it. When the power management scheme only allows for sonar use DBGT builds a hierarchy with the sonar versions of WallFollow and AroundObstacle. The resultant behavior hierarchies for each OP and sensor availability are shown in Table 6. Although the use of a sensor is dictated by the deliberation layer, a similar result would result if the DBHG were included in SFX-EH [Murphy and Hershberger(1999)] in place of the scripted responses. The sensor manager would remove the sensor and trigger the DBHG to rebuild using an alternative sensor. If no behavior could be generated the deliberator would then be called to generate a new plan.

Table 6 shows the generated behavior hierarchies after a resource change that effects the previous solutions to the OP. As can be seen, the DBHG generates behavior hierarchies which use the behaviors that are available. For the Find/Get (a) and Deliver (b) OPs hierarchy generation fails because there are no behaviors that are capable of accomplishing the goal of the OP due to resource unavailability. When hierarchy generation fails, the dispatch item for that OP is marked as a plan failure and awaits for a new replanning trigger to re-evaluate the OP. If the Behavior Executive reaches this item in the dispatch queue, then the Behavior Executive dispatches the default behavior (*AllStop*) and waits until replanning occurs, or the Sequencer resets the dispatch queue and sends in new OPs.

Sensor	Both	Laser	Neither	Sonar
Arbiter	*Utility Fusion*	N/A	N/A	*Utility Fusion*
Behaviors	*WallFollow* *LocateTarget* *Grab* *SonarApproachTarget* *Release* *LaserApproachTarget*	Failed	Failed	*WallFollow* *LocateTarget* *Grab* *SonarApproachTarget* *Release*

(a) Find/Get

Sensor	Both	Laser	Neither	Sonar
Arbiter	*Highest Activation*	*Highest Activation*	N/A	*Highest Activation*
Behaviors	*GoToXY* *SonarAroundObstacle* *LaserAroundObstacle*	*GoToXY* *LaserAroundObstacle*	Failed	*GoToXY* *SonarAroundObstacle*

(b) Deliver

Sensor	Both	Laser	Neither	Sonar
Arbiter	*Highest Activation*	*Highest Activation*	*Highest Activation*	*Highest Activation*
Behaviors	*Release*	*Release*	*Release*	*Release*

(c) Drop

Table 6: Resultant behavior hierarchies for resource availability reconfiguration for each OP.

Except for the strict power management, which gets stuck trying to get to the second target, the simulated robots completed the set of goals within approximately 150 seconds. After 150 seconds, the robots have found and delivered both targets and begin wandering (looking for targets that are not found) which creates a power consumption curve that is very linear and not noteworthy. Figure 4 shows that using predictive power management yields, on average, a 17% increase in remaining battery charge after task completion over lenient power management, and an average increase of 46% over no power management. This is achieved by using the predictive power plan that the MDP power planner produces. In this domain, maximum utility is found by using the sonar in the first collection and the laser in the second.

5 Conclusions

Layered, hybrid robot architectures combine deliberative planning with reactive behavior execution. This article presents the use of an abstract behavior representation with the dynamic behavior hierarchy generation algorithm to dynamically sequence system behaviors for accomplishing high-level tasks in a robust and modular implementation. The abstract behavior representation provides an interface for the DBHG algorithm within the Sequencer to dynamically build behavior hierarchies without requiring the user to handcode a behavior script or be concerned with a behavior's low-level implementation. This creates the defining entity that allows the Sequencer and Controller to seamlessly pass the planned behavior hierarchy between layers, but still enables the robustness and modularity of the components. The experiments show the behavior representation and DBHG algorithm successfully implemented within a hybrid architecture, and that the system can build arbitrary hierarchies based on variations in the tasks needing to be executed and sensor availability.

A future extension that will increase the system capabilities with minimal changes to the current modular entities include the creation of an abstract arbiter representation, rather than the set of rules currently used. The goal would be a representation that can be used like the behavior representation to describe different arbiters with a vote-weighting reasoning capability.

References

[Arkin(1998)] Arkin RC (1998) Behavior-Based Robotics, MIT Press

[Bonasso et al(1997)] Bonasso RP, Firby J, Gat E, David K, Miller DP, Slack MG (1997) Experiences with an architecture for intelligent, reactive agents. Journal of Experimental and Theoretical Artificial Intelligence 9(2/3):237–256

[Borenstein and Koren(1991)] Borenstein J, Koren Y (1991) The vector field histogram-fast obstacle avoidance for mobile robots. IEEE Transactions on Robotics and Automation 7(3):278 –288, DOI 10.1109/70.88137

[Connell(1992)] Connell JH (1992) SSS: a hybrid architecture applied to robot navigation. In: Proceedings of the 1992 IEEE International Conference on Robotics and Automation, pp 2719–2724

[Despouys and Ingrand(2000)] Despouys O, Ingrand FF (2000) Propice-plan: Toward a unified framework for planning and execution. In: Proceedings of the 5th European Conference on Planning, Springer-Verlag, London, UK, pp 278–293

[Firby(1989)] Firby RJ (1989) Adaptive execution in complex dynamic worlds. Tech. Rep. YALEU/CSD/RR #672, Yale University

[Gat(1997)] Gat E (1997) ESL: A language for supporting robust plan execution in embedded autonomous agents. In: Proceedings of the IEEE Aerospace Conference, vol 1, pp 319–324

[Gerkey et al(2003)] Gerkey BP, Vaughan RT, Howard A (2003) The Player/Stage project: Tools for multi-robot and distributed sensor systems. pp 317–323

[Hoey et al(1999)] Hoey J, Staubin R, Hu A, Boutilier C (1999) Spudd: Stochastic planning using decision diagrams. In: Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, pp 279–288

[Hurdus and Hong(2008)] Hurdus JG, Hong DW (2008) Behavioral Programming with Hierarchy and Parallelism in the DARPA Urban Challenge and Robocup, In: Proceedings of IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp 3193–3200.

[Infantes et al(2006)] Infantes G, Ingrand F, Ghallab M (2006) Learning behaviors models for robot execution control

[Ingrand et al(1992)] Ingrand FF, Georgeff MP, Rao AS (1992) An architecture for real-time reasoning and system control. IEEE Expert: Intelligent Systems and Their Applications 7(6):34–44

[Ingrand et al(1996)] Ingrand FF, Chatila R, Alami R, Robert F (1996) PRS: A high level supervision and control language for autonomous mobile robots. In: Proceedings of the 1996 IEEE International Conference on Robotics and Automation, Minneapolis, pp 43–49

[Konolige et al(1997)] Konolige K, Myers K, Ruspini E, Saffiotti A (1997) The Saphira Architecture: A Design for Autonomy. Journal of Experimental & Theoretical Aritifical Intelligence 9(2/3):215–235.

[Lemai and Ingrand(2004)] Lemai S, Ingrand FF (2004) Interleaving temporal planning and execution in robotics domains. In: Proceedings of the 19th National Conference on Artifical Intelligence, pp 617–622

[Long et al(2005)] Long M., Gage A, Murphy RR, Valvanis K (2005) Application of the Distributed Field Robot Architecture to a Simulated Demining Task. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp 3193–3200.

[McDermott(1991)] McDermott D (1991) A reactive plan language. Tech. Rep. YALE/DCS/TR-864

[Murphy and Hershberger(1999)] Murphy RR, Hershberger D (1999) Handling Sensing Failures in Autonomous Mobile Robots. The International Journal of Robotics Research 18(4):382–400.

[Murphy and Mali(1997)] Murphy RR, Mali A (1997) Lessons learned in integrating sensing into autonomous mobile robot architectures. Journal of Experimental & Theoretical Artificial Intelligence 9(2/3):191–209.

[Nguyen and Kambhampati(2001)] Nguyen X, Kambhampati S (2001) Reviving partial order planning. In: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pp 459–466

[Parker and Kannan(2006)] Parker LE, Kannan B (2006) Adaptive Causal Models for Fault Diagnosis and Recovering in Multi-Robot Teams. In: Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 2703–2710.

[Russell and Norvig(2003)] Russell S, Norvig P (2003) Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs, NJ

[Scheutz and Andronache(2004)] Scheutz M, Andronache V (2004) Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. IEEE Transactions on Systems, Man and Cybernetics, Part B 34(6):2377–2395

[Simmons et al(1997)] Simmons R, Goodwin R, Haigh KZ, Koenig S, O'Sullivan J (1997) A Layered Architecture for Office Delivery Robots. In: First International Conference on Autonomous Agents, pp 235–242

[Simmons and Apfelbaum(1998)] Simmons R, Apfelbaum D (1998) A task description language for robot control. In: Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp 1931–1937

[Urmson, et al.(2008)] Urmson C, et al (2008) Autonomous driving in urban environments: Boss and the urban challenge. Journal of Field Robotics 25(8):425–466

[Williams et al(2003)] Williams BC, Ingham MD, Chung SH, Elliott PH (2003) Model-based programming of intelligent embedded systems and robotic space explorers. Proceedings of the IEEE 91(1):212–237

[Woolley and Peterson(2009)] Woolley B, Peterson G (2009) Unified behavior framework for reactive robot control. Journal of Intelligent and Robotic Systems 55(2):155–176