Behavioral Programming with Hierarchy and Parallelism in the DARPA Urban Challenge and RoboCup

Jesse G. Hurdus and Dennis W. Hong

Abstract Research in mobile robotics, unmanned systems, and autonomous man-portable vehicles has grown rapidly over the last decade. This push has taken the problems of robot cognition and behavioral control out of the lab and into the field. In such situations, completing complex, sophisticated tasks in a dynamic, partially observable and unpredictable environment is necessary. The use of a Hierarchical State Machine (HSM) for the construction, organization, and selection of behaviors can give a robot the ability to exhibit *contextual intelligence*. Such ability is important for maintaining situational awareness while pursuing important goals, sub-goals, and sub-sub goals. Using the approach presented in this paper, an assemblage of behaviors is activated with the possibility of competing behaviors being selected. Competing behaviors are then combined using known mechanisms to produce the appropriate *emergent behavior*. By combining *hierarchy* with *parallelism* we present an approach to behavior design that balances complexity and scalability with the practical demands of developing behavioral systems for use in the real-world. The effectiveness of merging our hierarchical arbitration scheme with parallel fusion mechanisms has been verified in two very important landmark challenges, the DARPA Urban Challenge autonomous vehicle race and the International RoboCup robot soccer competition.

Keywords Action Selection · Hybrid Architecture · DARPA Urban Challenge · RoboCup

1 Introduction

The problem of high-level behavioral programming is defined primarily by its position within a greater Hybrid Deliberative-Reactive control architecture such as [1–6]. Traditionally, behavior-based software agents are responsible for low-level

J.G. Hurdus (✉)
TORC Technologies, LLC 2200 Kraft Dr. Suite 1325, Blacksburg, Va 24060, USA
e-mail: hurdus@torctech.com

S. Lee et al. (eds.), *Multisensor Fusion and Integration for Intelligent Systems*,
Lecture Notes in Electrical Engineering 35, DOI 10.1007/978-3-540-89859-7_18,

Fig. 1 Inputs and outputs for a behavioral software module

reflexes and direct actuator control while deliberative agents are used for more cognitive, high-level functions. With the rapid growth of computing technology, however, there has been a re-emergence of deliberative methods for low-level motion planning [6–8]. Such methods provide the important traits of predictability and optimality, which are extremely useful from an engineering point of view. This trend, along with the need for robots capable of handling more and more complex problems, has resulted in a shift in scope and responsibility for behavior-based software agents within Hybrid control architectures .

The need now exists for a behavioral control component capable of bridging the gap between high-level mission planning and low-level motion control. This behavioral module must be capable of abstract decision making in order to complete complex, multi-faceted, temporal problems. This reactive, behavior-based software agent receives perception information about the world through *virtual sensors* and dictates desired high-level action through *virtual actuators.* Virtual sensors use sensor independent perception messages to provide a filtered view of the world and virtual actuators specify abstract motion commands to a deliberative motion planner.

The responsibility of this behavioral module is to provide two important aspects of embodied A.I., *contextual intelligence* and *emergent behavior*. Contextual intelligence provides the robot with a mechanism for understanding the current situation. This situation is dependent on both the current goals of the robot, as defined by the mission planner, as well as the current environment, as defined by the relevant objects present in the world model. Such insight is important for performance monitoring, self-awareness, and the ability to balance multiple goals and sub-goals. Emergent behavior is a very important trait of biological intelligence which is understood to be necessary for the success of living organisms in the real world. It allows for the emergence of complex behavior from the combination of simpler behaviors, which is important not only for individual intelligence, but cooperative intelligence in multi-agent systems as well.

This paper presents a novel formulation of a Hierarchical State Machine (HSM) for providing *contextual intelligence* within a behavioral agent. A generalized description of the behavioral HSM is described here for use on mobile robots with complex applications. This behavioral HSM allows for a subset of behaviors of varying levels of abstraction be activated and deactivated in real-time. Once a context dependent set of behaviors are activated, it is expected that conflicting behavioral outputs be resolved in a manner most appropriate for the specific robot application.

2 Background

2.1 The Action Selection Problem

The central focus of behavioral programming is determining at any given moment what type of action should be performed. Jim Albus, of the National Institute for Standards and Technology, defines mobile robot intelligence as the ability to "act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of *behavioral* goals" [9].

The process of deducing the most "appropriate" action is known as the *Action Selection Problem* (ASP). Unfortunately, the ability to evaluate "appropriateness" is a very complex problem and one that causes even many humans trouble. While choosing the absolutely rational, or optimal action is often impossible without seeing into the future, we can hope to select "good enough" or *satisficing* actions, as defined in [10]. According to Maes, the following requirements are needed of any Action Selection Mechanism (ASM) to produce "good enough" behavior [11].

- **Goal-orientedness** – the favoring of actions that contribute to one or several goals
- **Situatedness** – the favoring of actions that are relevant to the current situation
- **Persistence** – the favoring of actions that contribute to the ongoing goal
- **Planning** – the ability to avoid hazardous situations by looking ahead
- **Robustness** – the ability to degrade gracefully
- **Reactivity** – the ability to provide fast, timely response to surprise

In [10], the following requirements for an ASM capable of producing satisficing behavior were added.

- **Compromise** – the favoring of actions that are best for a collection of behaviors, rather than for individual behaviors
- **Opportunism** – the favoring of actions that interrupt the ongoing goal and pursue a new one

From our own experiences developing ASMs for both the Urban Challenge and RoboCup, a capable ASM should also take into account:

- **Temporal Sequencing** – the ability to define a necessary order for tasks and sub-tasks
- **Uncertainty Handling** – the ability to not react poorly to perception noise

It is very important to note that some of these many requirements conflict with each other. For example, persistence can be in conflict with opportunism and situatedness. Similarly, planning is in conflict with reactivity. It is therefore impossible to create an ASM which meets *all* of these requirements equally. Instead an ASM must attempt to *trade-off* between these requirements in a way that best fits the given application.

2.2 Existing Action Selection Mechanisms

Taxonomies of existing ASMs are seen in [12–14]. Of these taxonomies, the most complete and comprehensive is by Pirjanian in [14]. Pirjanian breaks down all ASMs as being either in the *arbitration* or *command fusion* class.

Arbitration ASMs allow "one or a set of behaviors at a time to take control for a period of time until another set of behaviors is activated" [14]. Arbitration ASMs are therefore most concerned with determining what behaviors are appropriate given the current situation. Once this has been determined it is guaranteed that there will be no conflict in outputs between the running behaviors and so no method of combination or integration is needed. ASMs within the arbitration category are further broken down into priority-based, state-based, or Winner-take-all subclasses.

Command fusion ASMs, on the other hand, "allow multiple behaviors to contribute to the final control of the robot" [14]. Rather than being concerned with selecting appropriate behaviors, command fusion ASMs let all behaviors run concurrently, then rely on a fusion scheme to filter out insignificant behavioral outputs. Command fusion ASMs are therefore typically described of as being *flat*. Since multiple behaviors can end up desiring the same control, these ASMs present novel methods of collaboration amongst behaviors. This *cooperative* approach, rather than *competitive*, can be extremely useful in situations with multiple, concurrent objectives. For example, in the robot navigation domain, command fusion ASMs are useful for both avoiding an obstacle and proceeding towards a goal at the same time. An arbitration ASM would be constrained to doing one or the other. ASMs within the command fusion category are further broken down into Voting, Superposition, Fuzzy, or Multiple Objective subclasses.

Arbitration mechanisms, on the other hand, are more efficient in their use of system resources. By selecting only one behavior from a group of competing behaviors, processing power and sensor focus can be wholly dedicated to one thing. In a flat, command fusion ASM, all behaviors must be operating at all times in order to vote for the action they prefer. As the complexity of the robot application grows, the number of behaviors needed grows, and so does the *necessary resources* in a command fusion ASM. In a hierarchical arbitration ASM, however, the library of behaviors can grow as much as it wants, but only a subset of those behaviors will ever be needed at any given moment.

Well known examples of arbitration ASMs include the Subsumption Architecture [15] and Activation Networks [11]. Popular examples of command fusion ASMs include Potential Fields [16], Motor Schemas [17], Distributed Architecture for Mobile Navigation (DAMN) [18], and Fuzzy DAMN [19].

In this paper, a method of merging these two different classes of ASMs is presented. In doing so, the strengths of *both* arbitration and command fusion mechanisms hope to be preserved. **This is possible by placing an arbitration ASM in sequence with a command fusion ASM.** The result, in essence, is the ability to select a subset of behaviors given the current situation. Then, if multiple behaviors competing for the same output are activated, they can still be cooperatively combined using a method of command fusion. Specifically, a state-based, hierarchical, arbitration ASM is used for behavior coordination. This method utilizes a

hierarchical network of Finite State Automata (FSA), which can be referred to as a Hierarchical State Machine (HSM). To integrate the outputs of the activated behaviors, almost any known method of command fusion may be used. However, the chosen method should exhibit the qualities most conducive to the specific robotic application.

2.3 Hierarchy with Parallelism

It has been shown in [20] that the major bottleneck in developing behavioral intelligence is not selecting the best approach or architecture, but developing the *correct version* of this approach. While complexity is needed for multi-faceted problems, reducing complexity is important for making the robot designer's job simpler. It is not enough that a behavioral system be able to do a lot of things, it is equally important that they do all those things right, and at the right times.

In real-world applications with major repercussions for incorrect behavior, performance predictability can be paramount. The ability to hand code behaviors and ignore certain perceptual triggers at certain times is extremely useful and important for goal-orientedness. Hierarchy takes advantage of *selective attention* to make this hand-coding of behaviors possible and practical. Yet at the same time, complex combinations of behaviors are important for developing higher level intelligence.

Combining hierarchy with parallelism in the method presented in this paper provides important flexibility to the behavioral programmer. Situations in need of predictability can be catered to, while other situations can still take advantage complex, parallel, combination schemes. This approach balances quantity and complexity with design practicality.

3 Behavioral HSMs

Using a hierarchical approach to behavior decomposition is a common practice in *ethology*. It allows for the differentiation of behaviors according to their level of abstraction. According to Minsky in the Society of Mind [21], intelligent beings consist of agents and agencies. All agents are organized in a hierarchy where abstract agents are built upon lower, less abstract agents. Each agent has an individual motive which it pursues by activating and deactivating lower, subordinate agents. Groups of related agents in the hierarchy are viewed as sub-systems, and the hierarchy as a whole is the overall system.

3.1 Hierarchical Structure

A very similar organization has been adapted here, except *agents refer to individual behaviors*. All behaviors are similarly organized in a hierarchy with more abstract behaviors higher in the tree, and more physical behaviors lower in the tree. At any given time a subset of the total number of behaviors in the hierarchy are *activated*

and the rest are *deactivated*. The activated behaviors are considered to be along the *activation path*. Each behavior, or node, in the tree is responsible for determining which of their *sub-behaviors* should be activated. This is determined by each behavior's internal state and is not limited to only one sub-behavior. For example, given behavior *A* in state *X*, two parallel, sub-behaviors may be activated at the same time. The result is a branch in the activation path and can be seen in Fig. 2.

We can also see from Fig. 2 that all behaviors have implied relationships based off of their position within the hierarchy tree. Behaviors can have parent-child relationships or sibling relationships, but it is important to note that these relationships do not necessarily imply importance or priority. While some arbitration ASMs use hierarchy to determine the relevance of a behavioral output [15], this approach uses hierarchy solely as an abstraction method for task decomposition. Simply put, the primary function of the hierarchical tree is to determine what behaviors to run. Using a hierarchy allows us to logically break down a complex task into smaller, more manageable pieces.

Establishing the final output to each *virtual actuator* (VA) is therefore handled by a set of command fusion ASMs. As seen in Fig. 2, two sibling behaviors are collaborating/competing for control of VA_1. VA_2, on the other hand, has a parent-child pair producing command messages. It is also possible for a single behavior to produce more than one VA command if it requires explicit coordination between two or more VAs. However, it is not *required* for every behavior to produce a VA command. Some behaviors, especially higher-level, more abstract behaviors may be used solely as decision nodes in the hierarchy. The internal state of these behaviors is important in determining the activation path and subsequently what lower-level behaviors will run, but do not necessarily request specific action themselves. These behaviors are seen in Fig. 2 as activated, but not having a specific texture.

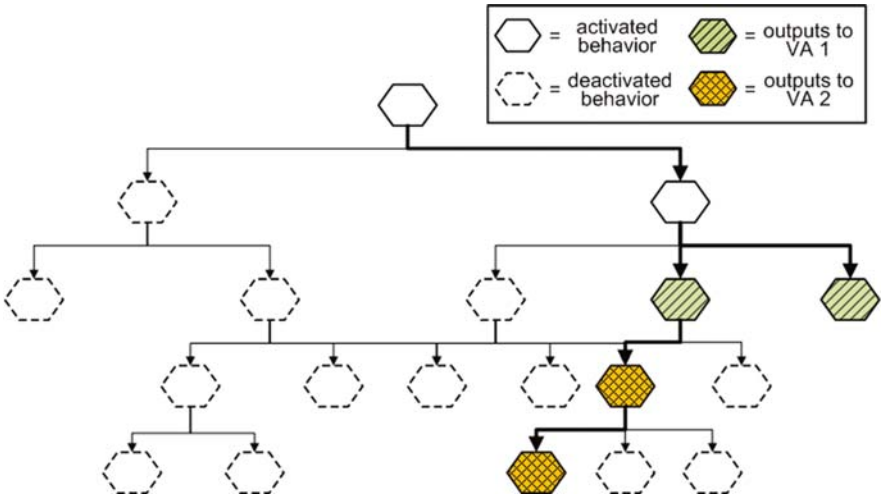

Fig. 2 General example of a behavioral HSM

Any behavior which produces one or more VA commands is classified as a *command behavior*. Any behavior which results in the activation of lower sub-behaviors (i.e., not a leaf node) is classified as a *decision behavior*. These classifications are not mutually exclusive, so it is possible for a behavior to be both a command and decision behavior.

3.2 Behaviors as Finite State Automata

Every behavior is modeled as an individual state machine, or finite state automata (FSA). Individual behaviors can therefore be formally described as consisting of a set of controls states $cs_i \in CS$. Each control state encodes a control policy, π_{va}, which is a function of the robot's internal state and its beliefs about the world (virtual sensor inputs). This policy, π_{va}, determines what action with respect to a specific VA to take when in control state cs_i. All behaviors have available to them the same list of virtual actuators $va_i \in VA$. Furthermore, each control state has hard-coded what sub-behaviors $sb_i \in SB$ to activate when in that state.

Transitions between control states occur as a function of the robot's perceptual beliefs, in the form of virtual sensors, or built-in events, such as an internal timer. While each behavior may have a "begin" and "end" state corresponding to the start and completion of a specific task, a single behavior, or state machine, cannot terminate itself. The higher, calling behavior always specifies what sub-behaviors should be running. Should a sub-behavior complete its state sequence and have nothing to do, it will remain in an idle state and not compete for control of any VA.

A simple example of an abstract behavior used for robot soccer is shown Fig. 3. The *Field Player – Attacker* behavior shown here is just one behavior within the

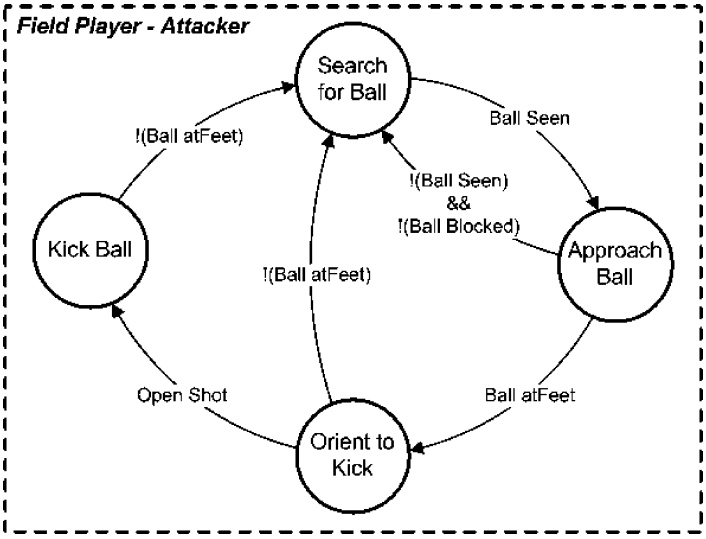

Fig. 3 A behavioral state machine for robot soccer

overall behavior hierarchy needed for a generic soccer playing robot. It is a *decision behavior* with four control states and a multitude of transitions for moving between these control states. While all transitions in this example are based off of perceptual occurrences, some may require a combination of virtual sensor inputs before being evaluated to true. For example, *Open Shot* may require perceiving the goal as being in front of the robot as well as perceiving the presence of no other robots before triggering.

Of course this individual behavior is only one within a hierarchy of other more, and less, abstract behaviors. A higher-level behavior might determine the role of the robot based off of the game situation or user inputs. For example, if the team is winning significantly it might be desired to have attacking players transition to a defender role, at which point the behavior shown in Fig. 3 might no longer be called. On the other side, each control state shown has a selection of sub-behaviors which are activated when in that control state. Let the *Field Player – Attacker* behavior be in $cs_{ApproachBall}$, it is possible then that $sb_{WalkToBall}$, $sb_{TrackBall}$, and $sb_{AvoidObstacle}$ are activated, each with their own state machine and corresponding sub-behaviors. Since the behavior shown here is a decision behavior and not a cmand behavior, $cs_{ApproachBall}$ has no control policy with respect to a virtual actuator. Instead, the primary function of this behavior is to determine what sub-behaviors to run given the current situation.

From these examples we see how a HSM, and particularly the current activation path within that hierarchy, are representative of the robot's current situation. This situation is a function of the robot's environment, the goals of the robot, and the *internal states* of the robot. In total, proper construction of the HSM will result in providing *contextual intelligence* to the robot. Producing *emergent behavior*, however, is left to the Command Fusion mechanism.

3.3 Application Specific Command Fusion

As stated earlier, the hierarchical relationship between behaviors has no relevance to the likelihood of that behavior's effect on a specific VA. Once all the behaviors along the *activation path* have been defined by the arbitration mechanism described previously, their hierarchy is thrown out and they are put in a 'flat' structure. Their individual outputs are then combined by a series of command fusion ASMs, with each instance corresponding to a single virtual actuator. The specific mechanism used for command fusion is not specified in this approach, and instead should be determined by the designer according to the robot application and specific virtual actuator. It is therefore possible to have one command fusion method for VA_1 of robot *X*, and a separate command fusion method for VA_2 and VA_3 of the same robot. This general approach to command fusion is seen in Fig. 4.

Returning to the robot soccer example presented in the previous section, let VA_1 be a vector which defines the direction and speed of a walking gait. Based on the current activation path in the HSM, the *walkToBall* behavior and the *avoidObstacle*

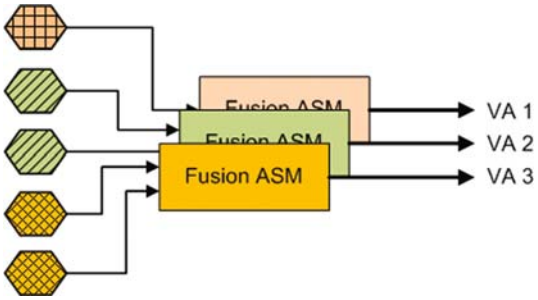

Fig. 4 Layered command fusion mechanisms

behavior are outputting desired gait vectors. It therefore makes sense, in this robot navigation example, to use a superposition mechanism of command fusion, such as *potential fields* or *motor schemas*. This would be the simplest way of producing the desired emergent behavior of approaching the ball while avoiding other robots along the way. Take now the situation where the robot is attempting to kick the ball into the opposing goal. Let VA_2 be a set of discrete kick types, *leftFoot_forward*, *leftFoot_backward*, *rightFoot_forward*, *rightFoot_backward*, etc. Just the fact that there are only a set number of discrete kick types makes a superposition-based ASM inappropriate. Instead a voting-based ASM would be much more applicable, where each behavior would vote for one type of kick, and the kick with the most votes would be selected. Taking yet another, further example, examine the behavior needed to select lanes when driving down in urban street in an autonomous vehicle. In this situation, one behavior desiring to stay in the right lane for an upcoming turn is running concurrently with a behavior desiring to pass a slow moving vehicle by moving to the left lane. Let the VA be the desired lane, and again we see that a superposition ASM is not appropriate. In this robot application, driving in between two lanes is unacceptable. Instead, a single lane should be chosen, either the left *or* the right.

We see from these examples the result of selecting different fusion ASMs. Depending on the exact mechanism chosen, completely different emergent behavior can result. This provides the robot designer with the flexibility to pick and choose the most appropriate method for the desired emergent behavior.

4 Real-World Application

The ultimate goal of action selection and behavior-based decision making research within mobile robotics is to build a physically embedded system that can exist autonomously in the real world. Action selection mechanisms that work in virtual environments are often unsatisfactory when transported to agents that must deal with real world uncertainty. It is therefore desirable to inspect the performance of any approach to behavioral programming on real robots performing real tasks.

The use of behavioral HSMs as described in this paper has been verified in two very important examples, the DARPA Urban Challenge and the International RoboCup soccer competition. At first glance, these two real-world robotic applications are extremely different. The DARPA Urban Challenge is concerned with building a full-sized autonomous ground vehicle capable of driving in an urban environment. RoboCup, on the other hand, is focused on creating a team of fully-autonomous humanoid robots capable of playing soccer. Across these two applications, the base platform is drastically different; from a 1.8 ton, 4-wheel, differentially steered vehicle to a bi-pedal, 2 foot tall humanoid robot. The goals of each robot are significantly different as well, from urban driving to goal scoring. In both of these landmark challenges, however, the core problem of a behavioral control structure is the same. Both robots must somehow balance dynamically changing desires while trying to achieve mission objectives in a real and unpredictable environment.

4.1 DARPA Urban Challenge – Team VictorTango

In November 2007, the Defense Advanced Research Projects Agency (DARPA) hosted the Urban Challenge, an autonomous ground vehicle race through an urban environment. In order to complete the course, the fully autonomous vehicle had to traverse 60 miles under 6 h while negotiating traffic (both human and robotic), through roads, intersections, and parking lots. Out of an original field of hundreds of teams from across the globe, only 35 were invited to the National Qualifying Event (NQE) in Victorville, California. After rigorous testing, only 11 teams were selected to participate in the Urban Challenge Event (UCE). Of these 11, only 6 teams managed to finish the course, with the top three places going to Carnegie Mellon University, Stanford University, and Team VictorTango of Virginia Tech.

In order to complete the challenge, vehicles had to contend with complex situations in crowded, unpredictable environments. A behavioral system capable of obeying California state driving laws in merging situations, stop sign intersections, multi-lane roads, and parking lots was needed. While a vehicle did not need to actively sense signs or signals such as traffic lights, right-of-way rules had to be followed as well as precedence-order at predefined intersections. This required the sensing, classification, and tracking of both static and dynamic obstacles at speeds up to 30 mph. To be successful, the vehicle had to balance goals of dynamically changing importance, traversing the course as quickly as possible while remaining a safe and "defensive" driver. The software module utilized by Team VictorTango to attack this problem employed a behavioral HSM for arbitration and a voting-based method for conflict resolution.

This implementation was able to produce an excellent performance at the Urban Challenge Final Event. Team VictorTango placed third overall, completing the course and all of its rigorous tests well within the 6 h limit and only minutes behind the leaders. After post-processing all the recorded data from the final race and exam-

Fig. 5 Odin, Team Victor Tango's entry in the DARPA Urban Challenge (Credit: Dr. Al Wicks, Mechanical Engineering Department, Virginia Tech)

ining hours of video, it was determined that the behavioral component made no incorrect decisions throughout the entire course of the race.

4.2 RoboCup – Team VT DARwIn

The landmark challenge presented by RoboCup is to develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team by the year 2050. While it is unlikely that this will be accomplished in any near term, the idea of soccer as a standard arena for mobile robots has been widely accepted. It is estimated that more than 500 teams consisting of 3,000 scientists from 40 countries will participate in RoboCup 2008 in Suzhou, China, making it the largest competition in the project's history.

The Robotics and Mechanisms Laboratory (RoMeLa) of Virginia Tech has developed a team of fully autonomous humanoid robots for entry in the kid-size humanoid division [23, 24]. In this division a team of 3 fully autonomous humanoid robots must play the game of soccer against another team of robots. All sensing and processing must be performed on-board, and wireless transmission may be used only for communication amongst individual players. All sensing must be roughly equivalent to the capabilities of a human, prohibiting the use of active sensors that emit light, sound, or electromagnetic waves. In order to qualify for competition, robots must be able to localize an unknown ball position, walk to the ball while maintaining stability, localize a goal and position around the ball for kicking, kick the ball towards the goal, and autonomously detect and recover from a fall. To perform well in competition, robots must also be able to defend against other teams attacks, dive to block kicks if designated as a goalie, avoid contact with other robots, and work strategically as a team.

Fig. 6 DARwIn IIa and IIb competing in RoboCup 2007 (Credit: Dr. Dennis Hong, RoMeLa, Virginia Tech <www.me.vt.edu/romela/RoMeLa/Meda.html>)

Like the Urban Challenge, each individual robot must be able to handle complex situations in an unpredictable and noisy environment. A behavioral system is needed that can balance dynamic goals such as scoring, defending, and maneuvering. Therefore, a method for providing contextual intelligence and the ability to produce emergent behavior are again required for successful operation. For RoboCup, a software module built around a behavioral HSM was used. By developing this implementation and comparing it with the Urban Challenge implementation, the portability of high-level behavioral programming across drastically different platforms and functionality requirements can be seen.

5 Discussion and Conclusion

The arbitration ASM presented in this paper is a novel variant of existing state-based ASMs and utilizes a Hierarchical State Machine for task decomposition and behavior selection. The mechanism proposed in this paper provides the robot with *contextual intelligence* by maintaining a subset of activated behaviors with internal states that represent the robot's *current situation*. With environmental changes or the completion of sub-tasks and sub-sub-tasks, the activation path within the behavioral HSM will reflect the new situation.

In the case of multiple behaviors competing for control of a virtual actuator, the specific command fusion ASM is not specified and should be chosen based on the robot application. The organization of ASMs in this approach allows many typical and well known command fusion ASMs to be implemented. The selection and implementation of these command fusion mechanisms will result in the selected subset of behaviors producing the appropriate *emergent behavior*. In total, the use of behavioral HSMs addresses many important problems with existing ASMs, but like any solution, there are some important benefits and drawbacks which should be identified.

5.1 Benefits

Task Decomposition – The organization of behaviors in a hierarchical tree according to their level of abstraction is extremely useful for breaking down a task into manageable sub-tasks, and sub-sub-tasks that can be solved as independent solutions. Due to the fact that robotic behaviors still need to be largely hand-coded, a logical method for decomposition is very helpful in this process.

Temporal Sequencing – Through the use of state machines in each behavior, the robot designer can easily imply when the order of tasks is important and when it is not. Every behavior uses a state machine to define which sub-behaviors are activated. This designer can therefore use state transitions to imply order in the completion of those lower sub-behaviors.

Behavior Reuse – By taking a "divide-and-conquer" approach to behavioral problem solving, it is possible to reuse lower-level behaviors for similar problems. A sub-behavior for control state i of behavior x, can also be a sub-behavior for control state j of behavior y.

Behavior Commonalities – In conventional state machines, there are many commonalities amongst different states. In the behavioral programming example, it is possible that many different behaviors would encode the same policy for a specific VA. By using a hierarchical state machine, encoding this policy in every behavior is unnecessary. Instead, a higher-level behavior allows us to define common policies only once.

Perception Requirements – From a systems engineering perspective, the use of state machines is very useful because state transitions define all perception and virtual sensor requirements. By building the behavioral HSM first, a robot designer is aware of what information needs to be pulled from the environment.

Uncertainty Handling – A unique property of state-based behaviors is that they can be made robust to perception noise. This is possible because state transitions are directional. The requirements for transitioning from control state A to control state B can be different then the requirements for transitioning from B to A. If there is noise in the perception data (which there usually is), defining these transitions properly can prevent flip-flopping between states.

5.2 Drawbacks

Preprogrammed vs. Learned – Individual behaviors and their relationships within the greater hierarchy must be hand-coded. As a result, determining the control policies and parameters built into each state of each behavior is a time consuming and error prone process. Testing, both in simulation and on the actual robot, is absolutely essential but not always possible. It is desirable to

automatically generate or learn behaviors, or at least autonomously modify parameters and control policies based off of the robots actual experience. Such learning methods are not addressed in our approach but are being researched elsewhere [22].

Performance Measurement – There exists no formal method for measuring and comparing the performance of the presented approach against other existing approaches. While "good enough" behavior defines important functional requirements, there is no quantitative method of comparison for "goal-orientedness," for example. Qualitative observations are the only major source of comparison which is generally insufficient. Performance comparison of ASMs can be done in a standard simulation environment [10] or even better in real-world competitions such as the DARPA Urban Challenge. However, with non-standardized platforms, sensors, and technology, the overall performance of any team is not a good indication of the smaller behavioral programming problem. Furthermore, since the behavior hierarchy is hand-coded, different implementations of the same approach can have very different results. The overall performance, therefore, is still dependent more on the designer than the approach itself.

Acknowledgments This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) through the Track A – Urban Challenge development grant, Science Applications International Corporation (SAIC), and the National Science Foundation (NSF) under grant no. OISE 0730206.

References

1. R. C. Arkin, E. M. Riseman, and A. Hansen, AuRA: an architecture for vision-based robot navigation, *Proceedings of the DARPA Image Understanding Workshop,* pp. 414–417, Los Angeles, CA, 1987.
2. R. Murphy and A. Mali, Lessons learned in integrating sensing into autonomous mobile robot architectures, *Journal of Experimental and Theoretical Artificial Intelligence special issue on Software Architectures for Hardware Agents*, 9(2), 191–209, 1997.
3. E. Gat, Three-layer architectures, in *Artificial Intelligence and Mobile Robots*, D. Kortenkamp, R. Bonasson, and R. Murphy, editors. Cambridge, MA: MIT Press, 1998.
4. R. Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. O'Sullivan, A layered architecture for office delivery robots, *Proceedings Autonomous Agents 97*, Marina del Rey, CA: ACM pp. 245–252, 1997.
5. K. Konolige and K. Myers, The saphira architecture for autonomous mobile robots, in *Artificial Intelligence and Mobile Robots*, D. Kortenkamp, R. Bonasson, and R. Murphy, editors. Cambridge, MA: MIT Press, 1998.
6. A. Bacha et al., Odin: Team VictorTango's Entry in the DARPA Urban Challenge, *Journal of Field Robotics*, 25(8), 467–492, 2008.
7. S. Thrun, M. Montemerlo, et al., Stanley: the robot that won the DARPA Grand Challenge: research articles, *Journal of Field Robotics*, 23(9), 661–692, September 2006.
8. C. Urmson, et al., A robust approach to high-speed navigation for unrehearsed desert terrain, *Journal of Field Robotics*, 23(8), 467, August 2006.

9. J. S. Albus, Outline for a theory of intelligence, *IEEE Transactions On Systems, Man, and Cybernetics*, 21(3), May/June 1991.
10. H. A. Simon, *The New Science of Management Decision*. New York: Harper and Row, 1960.
11. P. Maes, How to do the right thing, *Technical Report NE-43-836*, Cambridge, MA: AI Laboratory, MIT, 1989.
12. D. Mackenzie, R. Arkin, and J. Cameron, Specification and execution of multiagent missions, *Autonomous Robots*, 4(1), 29–52, 1997.
13. A. Saffiotti, The uses of fuzzy logic in autonomous robot navigation: a catalogue raisonne, *Technical Report 2.1*, IRIDA, Universite Libre de Bruxelles, 50 av. F. Roosevelt, CP 194/6, B-1050 Brussels, Belgium, 1997.
14. P. Pirjanian, Behavior coordination mechanisms – state-of-the-art," *Technical Report IRIS-99-375*, Institute for Robotics and Intelligent Systems, University of Southern California, Los Angeles, CA, 1999.
15. R. A. Brooks, A robust layered control system for a mobile robot, *IEEE Journal of Robotics and Automation*, 2(1), 14–23, 1986.
16. O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots, *The International Journal of Robotics Research*, 5(1), 90–98, 1986.
17. R. C. Arkin, Motor schema based navigation for a mobile robot: an approach to programming by behavior, in *IEEE International Conference on Robotics and Automation*, pp. 264–271, 1987.
18. J. Rosenblatt, DAMN: a distributed architecture for mobile navigation, in *AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents*, Menlo Park, CA: AAAI Press, 1995.
19. J. Yen and N. Pfluger, A fuzzy logic based extension to Payton and Rosenblatt's command fusion method for mobile robot navigation, *IEEE Transactions on Systems, Man, and Cybernetics*, 25(6), 971–978, 1995.
20. J. J. Bryson, Hierarchy and sequence vs. full parallelism in reactive action selection architectures, in *From Animals to Animats 6 (SAB00)*, pp. 147–156. Cambridge, MA: MIT Press, 2000.
21. M. Minsky. *The Society of Mind.* New York, NY: Simon and Schuster, 1985.
22. B. Argall, B. Browning, and M. Veloso, Learning to select state machines using expert advice on an autonomous robot, in *IEEE International Conference on Robotics and Automation*, pp. 2124–2129, 2007.
23. K. Muecke and D. W. Hong, The synergistic combination of research, education, and international robot competitions through the development of a humanoid robot, 32nd ASME Mechanisms and Robotics Conference, New York City, NY, August 2008.
24. K. Muecke and D. W. Hong, DARwIn's evolution: development of a humanoid robot, *IEEE International Conference on Intelligent Robotics and Systems*, October 29–November 2, 2007.