Department of Computer Science
George Mason University
Technical Report Series

4400 University Drive MS#4A5
Fairfax, VA 22030-4444 USA
http://cs.gmu.edu/
703-993-1530

A Control Architecture for Autonomous Mobile Robotics

Randall Steck
rsteck (at) gmu.edu

Technical Report GMU-CS-TR-2009-004

Abstract

The field of mobile robotics is on the forefront of robotics research around the world. Control architectures for complex autonomous mobile robots have largely settled on hybrid architectures for their suitability at dealing with the opposing forces of planning and reactivity. We present a general, heterogeneous 3-Tier hybrid architecture for control of an autonomous mobile robot and discuss an implementation in the domain of campus navigation. The architecture features a useful organization structure for high-level skills and offers flexible construction options for low-level behavior hierarchies.

1 Introduction

The field of mobile robotics is on the forefront of robotics research around the world. Recent public attention and government dollars have been drawn to the field due largely to the DARPA Grand Challenge (DGC) competitions from 2004–2007 [2, 12]. There is clear interest in robust and useful automomous driving systems for both civilan and military use. One of the most important aspect of such a complex autonomous system is the control architecture making decisions and reacting to the dynamic, real-world environment.

Control of autonomous robots has been a subject of active research since the first mobile robot, Shakey, built by SRI in the early 1970s. Shakey used a deliberative architecture, a schema which came to be called Sense-Plan-Act (SPA). SPA is a closed loop of three parts: the sensor input is processed and a world model developed or updated from the data; a deliberative planner is used to create a sequence of actions; the sequence of actions is attempted until a failure condition is detected. The deliberative planner [8] is a complex and search-intensive operation. Any deviation from the plan causes an expensive re-planning loop which can often exceed the time frame of usefulness. Later researchers addressed this limitation by techniques of partial planning, leading to modern planners which feature dynamic-scope planning [20].

To increase reactivity in a dynamic environment, the behavior-based architecture was introduced by Brooks [3]. Brooks' Subsumption architecture organized actions by layers of competency, with obstacle avoidance at the bottom and progressively complex goal-attainment behaviors in several higher levels. Each behavior is executed asynchronously and is reactive (low-latency and maintains no state). In Brooks' original formulation, the behaviors were encoded as Finite State Machines (FSMs), although many techniques have been used in later implementation. A higher level can inhibit (or *subsume*) the actions of a lower level, thus gaining control of the outputs. The introduction of the behavior-based architecture resulted in an explosion of research [1, 10, 11, 16] that resulted in a diverse ecosystem of related architectures. Although Brooks argued that strict representations of objects and world models were unnecessary for notions of intelligence [4, 5], behavioral architectures have limitations regarding issues of tractability and the complexity of the set of goals a behavior-base architecture can achieve.

The solution to the problems of both architectures was to combine them into hybrid architectures. The reactive behaviors are essential for interacting with a complex and dynamic world, but a planner is appropriate (and even required) for more complex goal sets. Hybrid architectures are typically arranged in a 3-Tier (3T, 3-Layer) organization: behaviors on the bottom, a planner on the top, and an interface layer of various descriptions [14]. Gat et al. [9] argues that 3T hybrids are the natural, even ideal, organization for an architecture for reasons of optimal separation of time and complexity domains.

We propose a 3T hybrid architecture for the general problem of autonomous control of a mobile robot. We introduce a definition of mission domain, a novel organizational structure for high-level behaviors based on disjoint skill domains, and a highly flexible behavior implementation for creating a wide variety of complex reactive behaviors. The architecture is heterogeneous between and within layers. To firmly establish the principles of the architecture, this report also details plans for our prototype implementation using the prototype as a running example. This prototype will be constructed in Spring 2009 for a competition in late April, 2009.

This technical report is organized as follows: Section 2 covers the general mission profile that is the focus of our interest and the robotics platform that will be used for the creation of the prototype. Section 3 introduces and discusses the the proposed general architecture, using the planned prototype as an example implementation. Section 4 offers some concluding thoughts on the both the proposed architecture and prototype.

2 Mission Description

The GMU Applied Robots Club has had a long-standing informal design goal dubbed the "GMU Grand Challenge", inspired by the DGC. The goal of the challenge is to autonomously navigate between the front of the main CS building and any GPS waypoint on campus, driving only on pedestrian pathway, and avoiding all obstacles. These goals are fairly low-hanging fruit, but still, various attempts have only garnered tepid interest over the last several years.

More recently, formal competitions have begun to spring up around a similar set of goals, also likely inspired by the DGC if naming conventions hold. The most notable of these competitions is called the "Mini Grand Challenge" sponsored by Penn State Abington.

We expect that in the near future additional competitions will rise in prominence. With some variation, we anticipate a central core requirement of these competitions to involve navigating an environment principly similar to that of a college campus. We have fused these known and anticipated goals into a problem definition we call the *Autonomous Campus Navigation (with Tour Guide)* (ACN/ACNTG) problem. The primary goals of the problem are:

- Operate on a typical college campus, in an environment assumed devoid of malicious agents, in one of two driving regimes:

  On-Road: Drive on pedestrian pathways only; the robot should not interact with automobile traffic. Should operate smoothly, with respect to other users of the pathway. There will never be more than light human traffic on the roadway, and often none. Pathway may be restricted by orange traffic cones 12–18 inches tall.

  Off-Road: Drive over cleared fields with sparse or no vegetation (e.g. football field or open public space). Obstacles will be generally large in relation to the vehicle, and detectable by horizontally-oriented body-frame-mounted 2D sensors (e.g. laser range finder or sonar ring).

- Navigate an ordered list of GPS waypoints from start position to goal position. The waypoints are guaranteed to be connected by navigable on-road or off-road terrain sections (as defined above). Roadway may not follow shortest distance.

- Drive subject to a flexible set of parameters (max speed, obstacle avoidance policy, pause and entertainment cues, etc) that is provided with the waypoints.

- Avoid local obstacles, subject to a given policy (e.g. swerve around obstacles or stop). Obstacles may be static or moving, and may come from a variety of sources (sensors, map restrictions, etc). Static objects may be known *a priori* or discovered dynamically.

- Practice "Safe Robotics", including no unintended contact with environment, humans always have right-of-way, full safe E-Stop, and mechanisms to ensure self-protection.

- Report technical feedback for diagnostics and research.

- Support a variety of technologies to entertain human companions/travellers/guests; to fulfill competition feedback requirements and eventually for applications in autonomous tours of the campus.

We intend to construct a prototype implementation of the proposed architecture (called the *George Mason University / Universal Touring Guide* or GMUTG) to compete in the 2009 Abington MGC. The Abington MGC competition rules are a subset of the ACNTG problem, but a large contributor to the problem's formulation. This section describes the hardware that has been provided for the project. Virtually all of the hardware was provided by GMU's Applied Robotics Lab (http://cs.gmu.edu/∼eclab/robotlab/pmwiki.php).

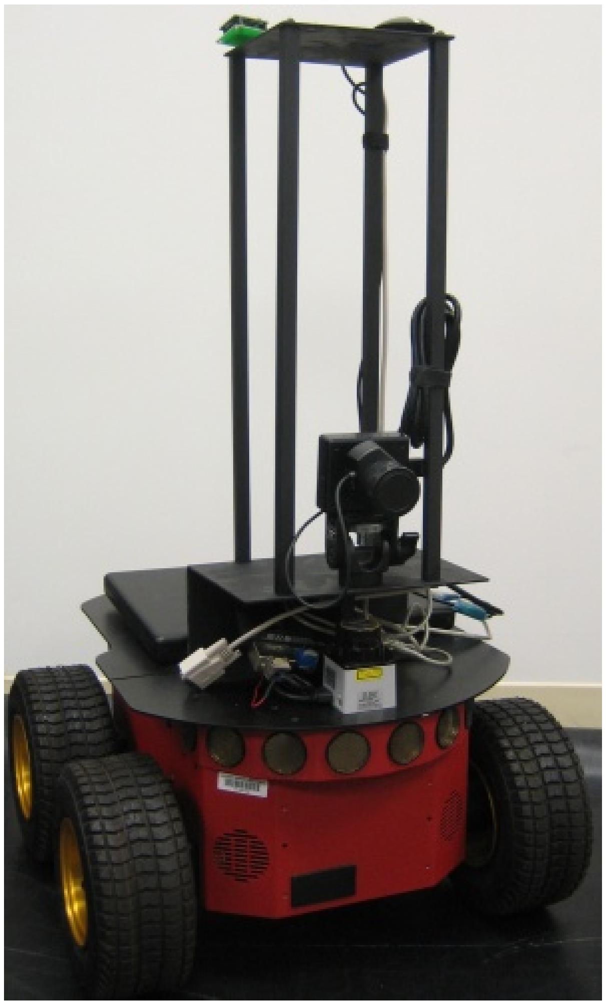

Figure 1: The Prototype Robot Platform

2.1 Mobile Robot

The mobile robot platform will consist of an ActivMedia Pioneer AT mobile robot and attached sensors. The ActivMedia Pioneer is a well-known and popular robotics platform for research and industry. The AT model is an outdoor-only mobile robot with a maximum velocity of 0.6 m/s. The drivetrain is 4 wheel drive from two electric motors configured in a skid-steer arrangement (turns like a tank). It has traversal specifications of maximum 35 degree grade, 10cm vertical step, and 12.5cm horizontal gap. Continuous runtime is 4–8 hours from built-in 12V sealed lead-acid batteries (252 watt-hr). Payload capacity is 30 kg.

The Pioneer robot includes several built-in features that will be utilized. First, the firmware offers status information of the running system which, for purposes of the prototype, will be limited to the battery level. Additionally, there are position encorders with 500-tick precision. These measurements can be integrated over time to construct the robot pose. Finally, there is a built-in sonar ring with 16 sensors, divided 8 front and 8 rear. These are low-resolution compared to other sensors available on the robot. The ring will likely be deactivated for the prototype, but the capability exists.

2.2 Sensor Suite

The following equipment is installed on the Pioneer AT frame and comprises the sensor suite available for the prototype.

2.2.1 Hokuyo Laser Range Finders

Two Laser Range Finder (LRFs), model URG-04LX, for local object detection. The URG is capable of sweeping 240 degrees in $\sim$100ms with a stated accuracy of 1mm. The data result is an array of length 683 at $\sim$0.36 degrees per index. One LRF is mounted horizontally on the front of the Pioneer deck for local obstacle avoidance and the second is mounted vertically on top of the mast for detecting both vertical clearance and dropoff conditions.

The horizontally-mounted LRF is robust for detection of obstacles of concern to the robot body, which is suitable for general traversibility. The vertically-mounted LRF yields accurate data for dropoff detection and height-clearance, but is constrained to a 2D plane of limited usefulness (the centerline of the robot). The 2D plane has consequences for both detection requirements. In dropoff detection, the sensor can fail to detect a pair of unfortunately-placed potholes which are large enough to cause the robot distress. In height detection, we are aided by the narrow dimensions of the mast relative to height, meaning any obstacle with clearance for the top-mounted sensor should have adequate clearance for the entire width of the mast. However, if additional sensors are added to the mast, any increase in width will increase the chances of mast collision.

We are investigating the possibility of mounting the vertical LRF on a swivel, allowing for a simple 3D picture of the space directly ahead and behind the robot. This would solve both of the problems outlined above.

2.2.2 GPS Receiver

A GPS receiver is required, but we have not settled on the specific equipment we will use. We have access to a consumer-grade, non-corrected (10m-resolution) unit, but are actively seeking to acquire a sub-meter unit.

2.2.3 Vision Subsystem

Built around the Point Gray Scorpion camera (640x480 @ 30fps Color or 60fps Mono). The camera requires a

IEEE-1394 connection and 12V @ 1A power. The camera is mounted at the top of the mast for optimal visibility.

A full description of the subsystem is outside the scope of this document. In brief, a color camera is monitoring the area directly in front of the robot. Road detection is accomplished by a novel histogram matching method. The system also performs color matching to detecting orange cones used to signal out-of-bounds (for path control). Relevent interfaces to the prototype will be noted where appropriate (specifically Sensor Behaviors, subsection 3.3.1).

2.3 Computing Platform

The computer used will be an Apple Macbook with the general specifications: Intel dual-core T7200 @2.0GHz, 2GB RAM, 5400RPM drive, 13" screen, running Linux 2.6.2x. Critically, the Macbook features a 6-pin IEEE-1394 (Firewire) connector which is essential for interfacing with the camera. The majority of laptops that include Firewire only provide the 4-pin variant of the plug, necessitating the use of a card-based Firewire adapter.

2.4 Development Environment

The prototype software will be written in C/C++. The language provides the necessary object-oriented features and offers maximum flexibility in interfacing to other subsystems.

We have chosen to use the Player/Stage open source project for robot and sensor interface. The reasons for Player/Stage are four-fold: open source principles, the wide variety of sensor systems with built-in support, the ease of integrating custom sensors, and an active community providing documentation and support.

GMU has an active project underway to investigate the use of the Intel C/C++ compiler instead of GCC. Testing with algorithms of similar complexity to those for both navigation and vision indicated a 10x speedup or more. The Intel compiler produces better optimization results on Intel hardware than the open source, platform independent GCC. There exist problems interfacing the Intel-compiled code with GCC-compiled code (like kernel header files).

3 Proposed Architecture

We propose a hybrid architecture based on the popular 3-Tier (3-Layer) model of design, as shown in Figure 2.

The *Mission Layer* is comprised of one or more planners, which create long-term contingency strategies and short-term concrete plans to accomplish a set of mission

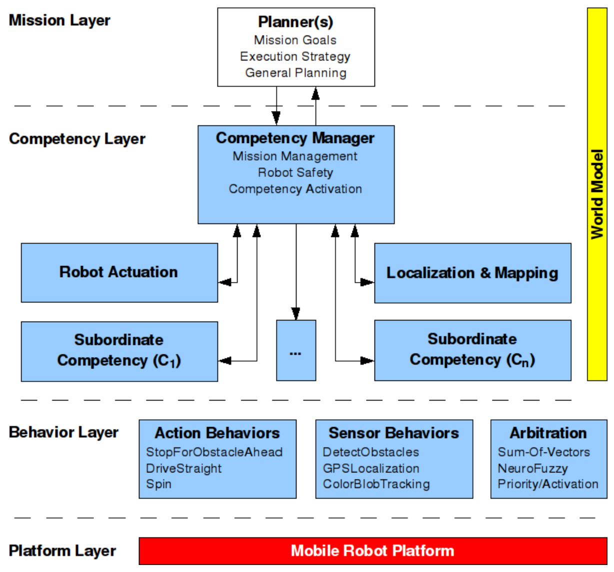

Figure 2: Architecture Overview

goals. A planner operates on very long timescales, maintaining the overall mission goals and monitoring their completion. The planner is less concerned with the mechanics of how a particular action gets done than in how the action is sequenced with other actions to attain a goal. It can be push- or pull-based with respect to assignment of tasks and monitoring the progress of goals.

The *Competency Layer* is responsible for accomplishing tasks assigned by the planner(s). The layer consists of small number of *competencies* which are individually responsible for a specific task domain (e.g. Driving, Localization). Each competency is composed of a set of *skills* (high-level behaviors) which use a mix of low-level behaviors and longer-term state management to monitor progress, handle fault conditions, and interact with other competencies. A skill should be able to robustly handle the execution of a specific complex task, on a persistent (non-reactive) basis.

The *Behavioral Layer* is comprised of behaviors, which are reactive components that manage, solve, or interface atomic actions. Behaviors comprise the basic actions by which the robot interacts with the world, usually holding exclusive access to the hardware inputs and outputs. Behaviors may be composed into hierarchies of complex reactive behaviors through use of Arbitration Operators (Arbitrators).

The division line between the layers has some flexibility subject to design requirements. The decision of whether a module should be a behavior or a competency should hinge on whether the action can be done reactively (in a guaranteed fixed and *very* short time). Behaviors must be reactive; Competencies offer design flexibility, access to the full world model, and adequate CPU cycles for heaftier calculations. The decision between mission and competency should hinge on whether the module needs to use forward-planning or projection of future results.

The architecture assumes the use of a world model, but places no restriction on the implementation details. The world model will generally be developed hand-in-hand with the Mission and Competency layers, and will fit the exact requirements of a given mission domain. In Figure 2 we suggest that the model is only accessible from the Mission and Competency layers. In our prototype, behaviors will have limited access to the model in order to store their raw values in a common, public place. We anticipate that access to those values for testing and tuning will be useful.

3.1 Mission Layer

Academic and research interests aside, a robot serves no useful function if it cannot successfully accomplish a set of predefined goals. The Mission layer is responsible for the the planning required to discover the sequence of actions sufficient to solve a series of goals, and for the overseeing the progressive execution of actions to achieve those goals. We have purposefully chosen the term *mission*, which we argue can be defined as a plan coupled with both the intention and capability to execute. A plan alone is useless without the means to see the goals of the plan achieved, or a waste of time to construct if you have no desire to achieve the goals.

A set of related missions, all achievable by the same agent(s), have a common description language that is familiar to the agent(s) capable of carrying it out. Like any language, it describes the structure (nouns and verbs) used to describe the actions and goals, and by consequence, the domain of problems the language can solve. A language with more complex operators/operations can describe more complex goals. The languages needed to describe the respective mission domains would be quite different for a search-and-rescue robot and a cross-country autonomous truck for military applications.

The mission layer is best defined only after the entire mission domain has been specified and sufficient information is available to make informed decisions. The requirements of the mission are integral to the design of the software required to execute the mission. The mission layer must at minimum include the capability to define new missions for the robot to perform – For the forseeable future, the mission must be defined by an expert human operator.

Our own prototype implementation includes a planner that is simple but well suited for the limited mission domain we have chosen to solve.

The user input to the system will consist of an ordered list of waypoints to achieve, in the form of a plaintext parameter file. The datapoints will include GPS position, the driving policy to follow for the travel segments, and potentially other data. The driving policy will include data on the On-Road/Off-Road conditions, the maximum safe speed, whether to chat up any human travelling companions, whether to stop at the destination, and potentially other data.

The planner will startup in a simple FSM we will call ”Pit Mode”. While in Pit Mode, the robot can be driven via a Wii Remote Controller (hereafter, Wiimote) and the various subsystems can be tested. The robot is driven to the competition starting position under human control. Once at the start, the Wiimote is used to put the robot into ”Run Mode”. The Wiimote becomes innactive (no cheating), and the robot waits for a start signal. Once the start signal is given, the FSM issues the first waypoint to the Captain (the Manager competency) and enters a passive mode.

The Captain goes about attempting to drive to the waypoint. The Captain will periodically poll the planner when waypoint achievement is immenent in order to be prepared for continuous movement. If the Captain detects a path-blocked or path-unreachable condition, the planner will be notified and asked for a new plan. The planner will examine the current map, determine possible back-track locations, and issue new orders or call for failure. At the completion of the mission (success or failure), the Pit Mode is automatically reactivated.

3.2 Competency Layer

Within this architecture, we use the term *Skill* to denote a high-level capability at performing a given task. A skill executes an individual complex task, with as much robustness as is reasonable, and reports success/failure within a reasonable amount of time. A *Competency* is a family of related skills that together solve all the requirements within a task domain. Our use of the term is essentially a logical organization and is simply a shorthand for ”the family of skills that accomplishes . . . ”. For example, the *Airplane Pilot* competency handles all flight operations, and has individual skills such as *Takeoff*, *Cruise*, *Land*, *EmergencyOverWater*, *EmergencyOverLand*, etc.

Competencies are generally disjoint in their task domains and possess exclusive access to the inputs or outputs they are responsible for operating (only the pilot is allowed to fly the plane). A mobile robot will typically have at minimum an Actuation competency and a Localization & Mapping competency, with others as necessary. The Localization & Mapping competency would use sensor behaviors to create data for the world model, which the Actuation competency would use to make decisions about steering and velocity. The Actuation competency would handle tasks such as *GotoXY* and *Stop* by activating relevant skills, which would in turn activate the behaviors necessary to accomplish the specified task.

Skills run at a much lower frequency than the reactive behavior layer. If we use an FSM to implement a skill, each state would establish a set of behaviors as active. These behaviors would execute reactively while the FSM is periodically executed to examine the model and determine if a transition is required.

There must be one Competency Manager and all other Competencies are subordinate to the Manager. The Manager is the primary interface between the Mission and Competency layers. In theory, it is the only Competency that communicates with (or knows about) the Mission layer. The Manager's responsibility is to execute the orders (sequence of tasks) given from the planner, making its primary skill the delegation of tasks to the subordinate competencies. Like any good manager, one must effectively use the talent pool available. Tasks are delegated to competent subordinates and only status messages and task changes need be communicated thereafter.

The Manager can communicate to all subordinates, but subordinates can only communicate to the Manager. Subordinates should exchange status and messages through the world model. At no time may a competency other the Manager issue a task to another competency.

Each competency must expose a list of the tasks it can handle and the information it requires to do so. This is necessary so that the Manager can make proper assignment decisions. The competency may expose meta-commands and use a dispatcher to activate skills according to some internal criteria, or may use a one-to-one mapping, listing all the available skills directly. The choice of how a competency's task assignment is mapped to skill activation has important implications.

For example, an implementation that forced all competencies to expose one-to-one mappings could assign pre- and post-conditions to those skills and manipulate the skills directly in the planner [14]. A competency that retains the authority to choose the skills must be built smarter. For Actuation, the skills might encompass complexities such as *GotoXY-OffRoad* which would ignore vision and place high priority on laser range finder data, *GotoXY-OnRoad* which might use vision data for steering and "rules-of-the-road" logic, and *GotoXY-OnRoadBadWeather* which might force a most-direct path and use audio cues to encourage humans to follow the robot to safety. The smart skill dispatcher would have to classify the input request and make decisions appropriate to the situation.

The architecture is heterogeneous with respect to the methodology used to construct skills. Skills should be built with the methodology that best suits the needs of the tasks. For our prototype, most skills will be built from FSMs (with timers) whose states activate different high-level composite behaviors. Other methods that are identified as potential candidates include petri nets [6] and Hierarchical Behavior Networks (HBNs) [14], although many techniques exist in the literature.

For the prototype, we chose to use the metaphor of a naval ship to delineate and label the competencies. Thus, the Manager competency is called the Captain, and the subordinate competencies are labelled Pilot, Navigator, Engineer, and so forth. The metaphor provides a familiar naming scheme that crosses cultural boundaries and idioms, which aids in making decisions in support of the clear delineation of tasks.

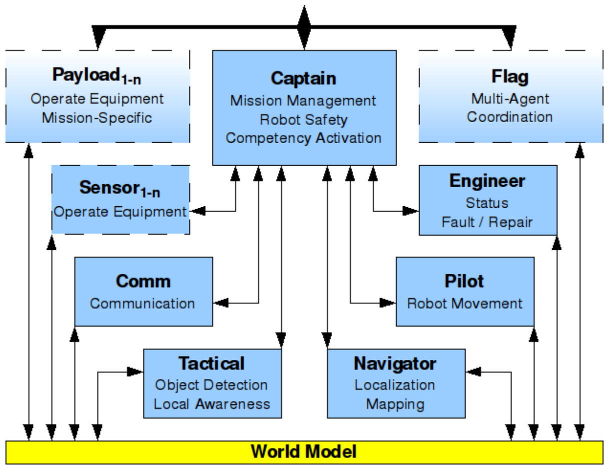

Figure 3: Prototype Competency Layer

Captain

The Captain has the primary responsibilities of the Manager, as described above. The interface to the Mission layer will be described in that subsection. The Captain will assign tasks to competencies based on a lookup table. A monitoring skill (built from an FSM) will check the progress of task completion. An error detection skill (of undefined method) will handle cases such as path blocked (forcing back-tracking) or unreachable path (forcing a request to the planner for new orders).

The Captain will create and monitor the **EStop** behavior and upon receiving an event, will issue suspend orders to all competencies. The Captain will also be designed to accept messages from other competencies regarding health and status, from battery level through traversal error conditions (grade or height restrictions, destination unreachable from path, etc).

Pilot

The Pilot competency is responsible for all actuation of the platform (except sensors). Tasks are accomplished by use of the compound behaviors that fuse data from the model of the local surroundings, localization and the current map. The skills are each designed to drive in a particular regime (based on policy), with several general use behaviors such as **Spin** and **Pause**.

The skills will be constructed from FSMs, one for each driving regime. Each FSM will have 5-10 states that detect faults and recovers from errors gracefully. Arbitration is likely to be a straight-forward sum of vectors with some notion of activation or priority (see Behaviors below).

Navigator

The Navigator competency is responsible for performing localization and mapping, and maintaining the world model with that information. The skills will include processing the various sensors and performing fusion on the data, transforming it to the representations of the model.

The methods for data fusion will be based on simple algorithms that can be completed and testing within the timeframe. More advanced algorithms, such as particle filters or full Simultaneous Localization and Mapping (SLAM) may be added in future upgrades.

For localization, the primary sensors will be the GPS receiver and the Pioneer position encoders (dead-reckoning), fused for mutual correction.

For mapping, the model stores a list of GPS points that define the valid roadway, as detected by the local surroundings. This list is initially empty. At a defined "start state", the current GPS reading is stored as the start point of the robot. This start point is used as reference for normalizing sensors to the local conditions. As GPS waypoints are issued by the planner, the Captain will add these to the list (corrected for GPS start). As the robot travels, the Navigator will periodically add current intermediate GPS locations to the list. Eventually, this list will comprise the entire road network the robot has detected. If the vision subsystem can be made to detect intersections in the roadway, the map would include unfollowed intersections that could be used for back-tracking.

Tactical

The Tactical competency is responsible for detecting and tracking all objects in the local space, and maintaining the world model with that information. The skills will include processing the various sensors and performing fusion on the data, transforming it to the representations of the model.

An example skill for fusion of situational awareness data is the **ConstructOccupancyGrid** skill: Execute the **LRFHorizObstacles** and **LRFVertObstacles** behaviors, each returning an array of length 683, and the **VisionCorridor** behavior, returning a list of lines segments in the body frame of the robot. All behaviors already perform smoothing on the data. Compile the data into an occupancy grid centered about the robot: use the vertical LRF as gospel on objects ahead; supersample the horizontal LRF data to world model grid size; Transform vision data to walls. Compare this grid to the one stored in the model. Add/Update/Remove objects to new grid and update the model.

We are investigating the possibility of mounting the vertical LRF on a swivel, as discussed in Section 2. This would vastly increase the complexity of this competency and the representation capabilities of the model. The actuation of this sensor would be handled by skills within this competency.

Engineer

The Engineer competency is responsible for monitoring the overall health of the robot. Skills include monitoring battery level, fault detection, and detecting e-stop events from button on Pioneer. Implementations will all be simple, for demonstration of capabilities.

Comms (Communications)

The Comms competency is primarily responsible for sending and receiving all messages to and from the robot. It maintains exclusive access to the communications hardware via its behaviors. The prototype requires no off-board communications, so this competency is void of those responsibilities. Later implementations could add wireless connectivity to a base station so that mission updates could be pushed to the autonomous guides.

A secondary function of the competency is the maintainance of a GUI of essential data. This GUI will include the current camera image, data from the Pilot regarding steering vectors and velocities, robot status including battery level warnings, and a console (or several) that contain debug information from the internals of the system.

An additional function of the competency is to handle the communications with humans in regard to voice recognition, speech synthesis and text-to-speech. Certain competitions require this, others award bonus points. How much speech capability that is included in the prototype is an open question. Depending on the degree to which these functions are implemented, all entertainment tasks could be refactored to a new competency, called the *Purser* or *Guide*.

Monitor

The Monitor competency is responsible for all logging and debugging within the system. It includes the capabilities for data collection (continuous or periodic model dumps) for output to logging files on the drive or real-time to a console for output (via the Comms).

If time allows, possible expansions include a full logging system for after-action replay and the capability to save the video of a run to hdd (currently not planned to maximize system reactivity).

Flag (Optional)

The Flag competency is responsible for all coordination within a multi-agent system. This competency is marked optional and will not be developed for the prototype.

The naming of this Role is based on the naval "Flag" officer, usually some kind of Admiral, who is in overall command of the fleet. The Admiral issues orders to all ships, including the one she is on (the "flagship"). The captain of the flagship is still and always responsible for the vessel and crew, subject to the orders of the Admiral, up to and including an order certain to result in destruction.

In a multi-agent expansion of this prototype, the Flag would exist simultaneously in both the Mission and Competency layers. It would need to be a behavior to directly access the Comms competency to transmit messages and would need to be a planner to issue tasks to the Captain. This is perhaps a matter of semantic difference, and we leave the details to those that will implement.

Sensor/Payload (Optional)

The Specialist competency is responsible for all tasks related to the use of an on-board, mission-specific sensor or payload package. The skills would include activation and monitoring of the hardware, actuation (as required), and for data collection, filtering, and transformation to model representations. This competency is marked optional and will not be developed for the prototype.

This Specialist could also span the Mission-Competency layer barrier. It is forseeable that the events from the payload could cause replanning, for example a science goal generated from sensor readings forces a replanning of the travel path. A direct link to the planner could streamline the interactions and representation differences. In a behavior-only implementation, the planner could receive events through the world model.

3.3 Behavior Layer

The behavior layer is a collection of atomic and composite behaviors that serve as the basis for all functionality. Atomic behaviors perform atomic actions (drive straight). Composite behaviors are hierarchies of behaviors that perform complex actions (drive straight and don't hit anything). Behaviors are defined to be reactive components – returning in a finite and *very* short time-frame. Behaviors should only depend on current state information, or a finite and *very* small number of previous steps for filtering purposes. No forward-looking planning should be done.

Behaviors exist in either an active or inactive state. The activation state of a behavior can only be changed by a skill. Active behaviors are executed asynchronously at a parameterized rate (10-50Hz). Activation of a composite behavior may or may not trigger activation of all child behaviors – this is implementation-specific.

We define two general categories of behaviors, for actions and sensors. Sensor behaviors passively gather information. Action behaviors cause the robot to interact with the world.

Methodology for behaviors is heterogeneous. Behaviors should be built with the methodology that best suits the needs of the action. Our research has identified a number of methods which show promise for implementing behaviors [7, 13, 17, 19, 21].

The following behavior descriptions are examples from the prototype.

3.3.1 Sensor Behaviors

Sensor behaviors observe the world. They are primarily input-oriented and generally have read-write access to the world model.

Sensor behaviors provide generic access to the sensors of the robot, often changing the raw data to the various representations stored in the world model. Many are simple "virtual sensors" that wrap the abstract the implementation details of the sensor and follow a Read-Filter-Update (RFU) pattern: Read data from source, perform filtering, update data in model and return data. They can use any filtering technique, including simple window or integer smoothing, advanced Kalman filters, or none. Data filtering must be accomplished within the reactivity requirements of the behavior layer – If too much processing is required, move the functionality to the Competency layer.

EStop: Monitor the status of the Pioneer built-in Stop button. When pressed, the power to the wheel motors is automatically shutoff, stopping the vehicle. This behavior captures that event so the architecture can be made aware of the condition. This behavior is usually created and used by the Captain competency that is responsible for robot safety. Other competencies could receive these notifications directly or indirecly through the actions of the Captain.

RoboStatus: Read the current status from the Pioneer. For the prototype, the only status information we are concerned with is the battery level. This will be critical to monitor on competition days as well as during testing to ensure there are no unplanned recharge delays.

RoboLocalize: Read wheel-watcher data from Pioneer and filter the data using a simple and fast smoothing method (TBD).

GPSLocalize: Read data from GPS receiver and apply a filter. The prototype version will be implemented with Player/Stage and will be able to interface with any receiver using the standard NMEA format. An additional version of this behavior will be written, if time allows, to interface with a GMU-designed sensor suite that includes GPS. Additional versions are possible, such as reading proprietary formats or standard NMEA through direct serial (without player/stage).

LRFHorizObstacles: Read the horizontally-mounted LRF to detect all nearby objects and filter the data using a simple and fast smoothing method (TBD).

LRFVerticalObstacles: Read the vertically-mounted LRF to detect all nearby objects and filter the data using a simple and fast smoothing method (TBD).

MonitorVertical: Monitor the vertically-mounted LRF to detect dropoff conditions, grade restrictions, and height-clearance conditions.

VisionVector: Read the suggested drive vector(s) from the vision subsystem. The result is an array (unknown length, max 8) containing the (x,y) coordinates of the approximate center of the road in the body frame of the robot. As more road is seen ahead, more data points are returned. The multiple return values represent the possibility for analyzing the state of the road ahead (curving or straight).

VisionCorridor: Read the suggested line-obstacles detected by the vision subsystem. These are usually the road edges, but may also include color-based detections, such as orange cones in the roadway. The vision subsystem returns a list of detected line segments. The line segments are provided in the coordinate system of the robot (all transformations completed by the subsystem).

3.3.2 Action Behaviors

Action Behaviors interact with the world. They are primarily output-oriented and generally have read-only access to the world model.

Action behaviors are responsible for all actuation on the robot. Action behaviors typically gather information from the world model and output a set of commands to the robot platform. Action behaviors can be accessed by any Competency, but generally only one Competency has exclusive access to a given output (and all action behaviors that have the same output) at a time. For a synchronous drive robot with a turret, one Competency may be responsible for driving the wheels and a second for aiming the turret. Both would use action behaviors, but always behaviors with clearly disjoint outputs.

Spin: Spin in place a specified number radians at a specified angular velocity.

DriveForward: Drive straight with a specified velocity, subject to a speed governor. Use a PID controller to maintain straight line.

StopForObstacle: If obstacle detected ahead within specified distance D_1, slow down with a PD controller until stopped at specified distance D_2. The Abington MGC rules specify that the robot should stop if it detects an obstacle ahead (no attempt to swerve). This behavior accomplished that. If the robot is expected to attempt to drive around obstacles, a behavior such as **SwerveObstacle** should be used to turn the vehicle away from the object, thus clearing the front sensor to quiet this behavior. This behavior should always be active in order to protect the robot from harm. Practice Safe Robotics!

SwerveObstacle: If obstacle detected ahead within a specified distance, suggest a heading vector to avoid it. Use the presence of obstacles left/right of centerline to aid in the decision. Do not use the vision-based vector(s) in this behavior; Mix the results through an arbitrator. If this behavior is not loaded/active, the robot will not turn in the face of an obstacle; If the **StopForObstacle** behavior is active, the robot will gracefully come to a stop, else, it will crash.

VelocityGovernor: Update the maximum global speed for the robot. This ”set member” method is encapsulated into a behavior for use by competencies with respect to the current state or goal (change max speed to go under a bridge or around a tight corner).

3.3.3 Composite Behaviors

Any instance of multiple active behaviors, whether hierarchical or parallel, must have a means to arbitrate between the opinions of the behaviors. Many theories exist on the best way to accomplish arbitration [15, 17]. All of the methods in the literature are good for a particular subset of needs, assumptions, capabilities, and goals. In the interests of maximizing flexiblity, we implement arbitration using the behavior system. It is hoped that this flexibility will lead to a wide variety of techniques available in a growing behavior library.

The architecture implements a generic type of behavior that accepts as input a set of data from multiple sources and a set of decision parameters (priorities, data from model state, etc). When the behavior is executed, each subordinate behavior is polled for input (executed) and the decision criteria is consulted. The arbitrator performs the work of determining a single set of data to output, including potentially performing some kind of mixing. The signature of the data set returned by an Arbitrator may be identical to the input set or completely different.

Arbitration operators can be constructed to mimick most other arbitration methods. Our prototype uses sum-of-vector and priority-based arbitration for driving and custom fusion arbitrators to take different sensor datasets and normalize their values to one another. A subsumption arbitrator could use an inhibit/suppress input from each behavior to determine the output to use [3]. A fuzzy arbitrator could use fuzzy classification rules [18].

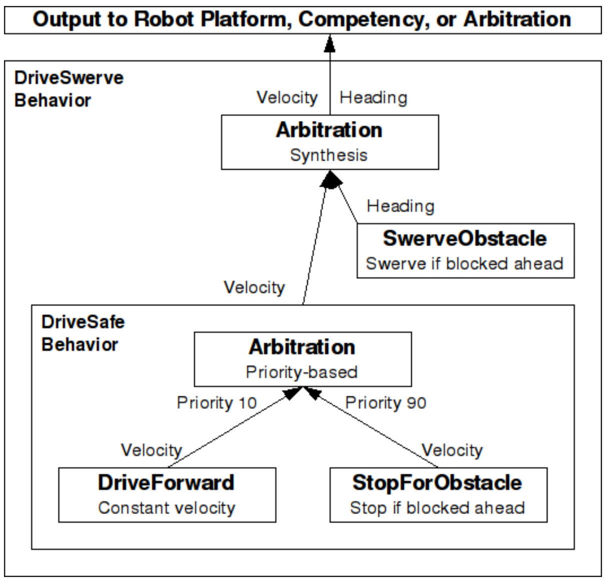

Figure 4: Behavior Composition with Arbitrators

Figure 4 is an example of a multi-level hierarchy constructed to develop increasingly complex driving behaviors.

The **DriveForward** behavior (providing positive velocity) is combined with the **StopForObstacle** behavior by priority arbitration. The **StopForObstacle** behavior is given a higher priority to prevent a collision. The robot will move forward until its path is blocked, whereupon it will slow and stop at a safe distance.

The **DriveSafe** behavior (providing safe positive velocity) is combined with the **SwerveObstacle** behavior (providing a heading) through a Synthesis arbitrator, forming a (velocity, heading) pair that is ready for transmission to the robot. The **SwerveObstacle** behavior must provide enough turning motion to clear the front of the robot before the **DriveSafe** behavior completely stops forward motion. The competency which uses this behavior should detect the case of no forward motion (due to wide blockage, corner, or other anomolous cases) and use the **Spin** behavior to clear the front of the robot and allow forward motion to resume.

4 Conclusion

The proposed architecture is purposefully under-defined, specifically with regard to the composition of the mission layer and the methodologies employed in all layers. Much of the architecture literature is focused on the use of a particular method and building an architecture around that method. We feel the unrestricted nature of this architecture will encourage collaboration and promote reuse of components in a move toward incremental development of complex mission architectures. Experts in algorithms or sensor systems can create behaviors to better manage sensor input, using methods and knowledge sets entirely different from the mission planner (AI, search, reinforcement) or the actuation (reactive motor control).

The mission domain to be completed by the prototype is lacking in sufficient complexity to warrant development of a complex planner and support subsystem. The planned implementation is sufficient to successfully complete both the GMU and Abington MGC competitions and to demonstrate the interaction between the Mission and Competency layers with sufficient depth.

The hard deadlines imposed by the competition limits the scope of the prototype. Additional development time would likely yield a more full-featured robot with better "personality", or an attempt to develop a more proper planner. That said, we fully intend to compete, not embarass ourselves or our school, and hopefully show up with a legitimate shot at winning.

References

[1] R. Arkin. Motor schema based navigation for a mobile robot: An approach to programming by behavior. In *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, volume 4, pages 264–271, 1987.

[2] J. Bohren, T. Foote, J. Keller, A. Kushleyev, D. Lee, A. Stewart, P. Vernaza, J. Derenick, J. Spletzer, and B. Satterfield. Little Ben: The Ben Franklin Racing Team's entry in the 2007 DARPA Urban Challenge. *Journal of Field Robotics. Vol. 25*, 25(9):598–614, 2008. ISSN 1556-4959. Special Issue: The 2007 DARPA Urban Challenge, Part II.

[3] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of [legacy, pre - 1988]*, 2(1):14–23, 1986. ISSN 0882-4967.

[4] R. A. Brooks. Integrated systems based on behaviors. *Stanford University*, 2:46—50, 1991. doi: 10.1.1.53.4578.

[5] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991. doi: 10.1.1.12.1680.

[6] H. Costelha and P. Lima. Modelling, analysis and execution of multi-robot tasks using petri nets. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3*, pages 1187–1190. International Foundation for Autonomous Agents and Multiagent Systems, 2008. ISBN 978-0-9817381-2-X.

[7] B. E. Eskridge and D. F. Hougen. Using priorities to simplify behavior coordination. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–3, 2007. ISBN 978-81-904262-7-5. doi: 10.1145/1329125.1329411.

[8] R. E. Fikes, N. J. Nilsson, and R. B. Raphael. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971. doi: 10.1.1.93.2824.

[9] E. Gat, R. P. Bonnasso, R. Murphy, and A. Press. On three-layer architectures. *Artificial Intelligence and Mobile Robots*, pages 195–210, 1998. doi: 10.1.1.35.2130.

[10] P. Maes. The agent network architecture (ANA). *SIGART Bull.*, 2(4):115–120, 1991. doi: 10.1145/122344.122367.

[11] P. Maes. Situated agents can have goals. *Robotics*, 6(1-2):49–70, 1990. ISSN 0167-8493.

[12] M. Montemerlo, S. Thrun, H. Dahlkamp, and D. Stavens. Winning the DARPA Grand Challenge with an AI robot. *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 17–20, 2006. doi: 10.1.1.94.9364.

[13] A. Nelson, E. Grant, G. Barlow, and M. White. Evolution of complex autonomous robot behaviors using competitive fitness. *KIMAS Proceedings*, 2003. doi: 10.1.1.4.2085.

[14] M. N. Nicolescu and M. J. Matari. A hierarchical architecture for behavior-based robots. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 227–233. ACM, 2002. ISBN 1-58113-480-0. doi: 10.1145/544741.544798.

[15] P. Pirjanian. Behavior coordination mechanisms - state-of-the-art. Technical Report IRIS-99-375, Institute for Robotics and Intelligent Systems, University of Southern California, 1999.

[16] J. K. Rosenblatt. Utility fusion: Map-based planning in a behavior-based system. *in Field and Service Robotics*, pages 411–418, 1998. doi: 10.1.1.52.333.

[17] P. Rusu, E. Petriu, T. Whalen, A. Cornell, and H. Spoelder. Behavior-based neuro-fuzzy controller for mobile robot navigation. *Instrumentation and Measurement, IEEE Transactions on*, 52(4):1335–1340, 2003. ISSN 0018-9456. doi: 10.1109/TIM.2003.816846.

[18] A. Safiotti. Fuzzy logic in autonomous robotics: behavior coordination. In *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*, volume 1, pages 573–578, 1997. doi: 10.1109/FUZZY.1997.616430.

[19] E. Sanchez. A digital artificial brain architecture for mobile autonomous robots. *Proceedings of the Fourth International Symposium on Artificial Life and Robotics AROB'99*, pages 240–243, 1999. doi: 10.1.1.49.1326.

[20] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, and H. Das. The claraty architecture for robotic autonomy. In *Aerospace Conference, 2001, IEEE Proceedings.*, volume 1, pages 121–132, 2001. doi: 10.1109/AERO.2001.931701.

[21] C. Wagner and H. Hagras. A genetic algorithm based architecture for evolving type-2 fuzzy logic controllers for real world autonomous mobile robots. In *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, pages 1–6, 2007. ISBN 1098-7584. doi: 10.1109/FUZZY.2007.4295364.