A Case Study in Behavioral Subsystem Engineering for the Urban Challenge

Christopher R. Baker, *Student Member, IEEE* and John M. Dolan, *Member, IEEE*

***Abstract*—We present a case study in the design of an autonomous robotic software subsystem for managing mission execution and discrete traffic interaction in the 2007 DARPA Urban Challenge. Its role is reviewed in the context of the overall software system that controls "Boss", Tartan Racing's winning entry in the competition. Design criteria are presented, followed by the application of design principles to derive an architecture well suited to the rigors of developing complex robotic software. The design's effectiveness is highlighted through a discussion of its ability to meet the subsystem's requirements while remaining adaptable to the ever-changing needs and capabilities of the surrounding system.**

***Index Terms*—Software Architecture, Tartan Racing, Boss, Robot Behaviors, Urban Challenge**

I. Introduction

The Urban Challenge[4] was an autonomous vehicle competition sponsored by the US Defense Advanced Research Projects Agency (DARPA) in November 2007. Contestant robots were required to autonomously execute a series of navigation missions through a simplified urban environment consisting of roads, intersections, and parking lots while obeying road rules and interacting safely and correctly with other traffic. In contrast to previous DARPA challenges, which focused on rough-terrain navigation[7], [8], this competition required a system capable of complex autonomous behaviors, such as waiting for precedence at an intersection or passing a slow-moving vehicle on a multi-lane road.

These behaviors were managed by a software subsystem called the *Behavioral Executive* in Boss, Tartan Racing's winning entry in the Urban Challenge, shown in Figure 1. The fulfillment of this role required the carefully structured integration and management of many disparate capabilities in a manner that remained highly flexible over the course of the development, accommodating whatever changes were necessary to win the competition amidst a continuously evolving software system. These requirements, among many others, are reflected in the Behavioral Executive's architecture, the flexibility and adaptability of which played an important role in the team's success.

The overall software system that controls Boss is briefly presented in Section II to provide context for discussion of the Behavioral Executive. Section III presents the criteria that contributed to the architecture of this subsystem, which is presented in terms of design patterns and functional decomposition in Section IV. Section V follows with the detailed roles and interactions of the primary functional elements, focusing on the use of input abstraction and clear role definition to meet the design goals. Some of the architecture's key strengths are highlighted in Section VI, emphasizing the simplicity of using existing intermediate data products to fulfill several additional responsibilities. Section VII summarizes the lessons learned over the development of the system and discusses the design's the overall ability meet the needs of the competition.

This work would not have been possible without the dedicated efforts of the Tartan Racing team and the generous support of our sponsors including General Motors, Caterpillar, and Continental. This work was further supported by DARPA under contract HR0011-06-C-0142.

C. Baker and J. Dolan are with Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA {cbaker,jdolan}@andrew.cmu.edu

Fig. 1: Boss: Tartan Racing's entry in the Urban Challenge

II. Total System Architecture

The software system that controls Boss, shown in Figure 2, is comprised of four primary subsystems: *Perception*, *Mission Planning*, *Behavioral Executive*, and *Motion Planning*. These subsystems generally consist of multiple processes running in a distributed computing environment and communicating via message-passing according to the anonymous publish-subscribe[2] pattern. Their individual responsibilities are described here in brief, focusing on their roles relative to the Behavioral Executive. The algorithmic details of these subsystems are beyond the scope of this article and are instead presented in [9].

The *Perception* subsystem processes sensor data from the vehicle and provides a collection of semantically-rich data products to the rest of the system. These include the current pose of the robot, the geometry and connectivity of the road network, a local grid representation of traversable terrain, and the location and nature of various discrete obstacles such as

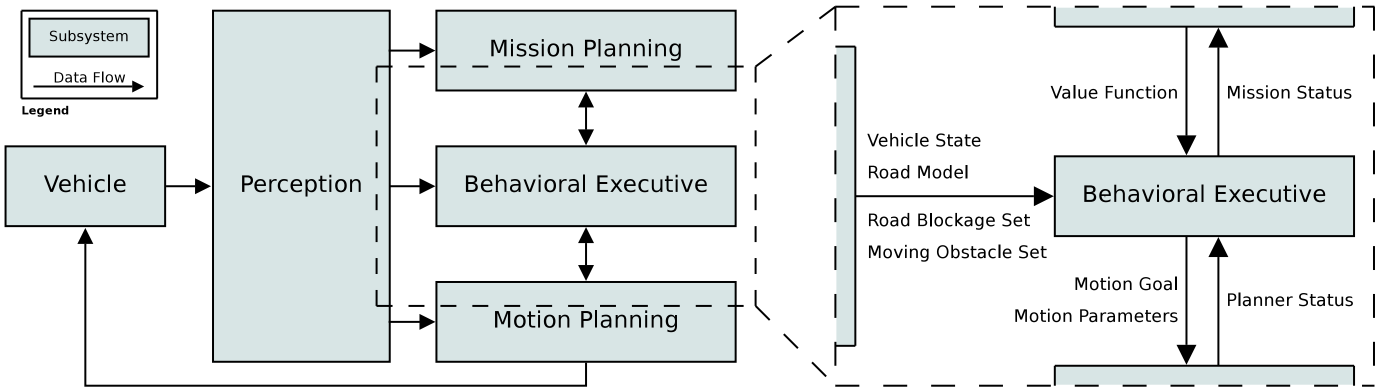


Fig. 2: Total system architecture, showing primary subsystems, data paths, and the detailed interface to the Behavioral Executive.

road blockages and other vehicles. These data products are called the *Vehicle State*, *Road Model*, *Static Obstacle Map*, *Road Blockage Set*, and *Moving Obstacle Set*, respectively. Each other subsystem subscribes to some subset of these messages and interprets the data according to its specific responsibilities. For instance, the Behavioral Executive uses each of these outputs except for the Static Obstacle Map, which is used exclusively by the Motion Planner for collision avoidance. The Road Model is the only data product that is used by all three, and it represents the road network as a directed graph of latitude/longitude waypoints, grouped hierarchically into larger elements such as individual lanes, multi-lane roads, parking spots and obstacle zones. Each element in the Road Model is assigned a unique identifier, called a *World ID*, which may be used to retrieve extended information such as the location and shape of the associated element.

The *Mission Planning* subsystem computes the fastest route to reach the next checkpoint[1] from any point in the road network, incorporating knowledge of road blockages, speed limits, lane geometry, and the nominal time required to make special maneuvers such as lane changes or U-turns. The Mission Planner publishes this information as a *Value Function* that maps each waypoint in the road network to an estimated time to reach the next checkpoint. This allows a simple search among the options at any intersection to follow the optimal path to the checkpoint, while also supporting local reasoning about the cost of taking an alternate route when faced with an aberrant situation such as a blocked road or intersection.

The *Behavioral Executive* combines the Value Function provided by Mission Planning with local traffic and obstacle information provided by Perception to generate a sequence of incremental *Motion Goals* for the Motion Planning subsystem to execute. Typical goals include driving to the end of the current lane or maneuvering to a particular parking spot, and their issuance is predicated on conditions such as precedence at an intersection or the detection of certain anomalous situations. A *Mission Status* message is periodically published that informs the Mission Planner of the progress being made toward the current checkpoint, triggering the issuance of subsequent Value Functions as checkpoints are achieved. When the robot is driving along a road, periodic lane-tracking and speed-government commands, called *Motion Parameters*, are published to enact behaviors such as safety gap maintenance, passing maneuvers and queueing in stop-and-go traffic. The Behavioral Executive attempts to select and govern the actions of the robot such that they are safe beyond a significant margin of the vehicle's dynamic capabilities, deferring the ultimate responsibility for the robot's safety to the subsystem with direct control of its actions.

[1]In the Urban Challenge, navigation missions were described as an ordered series of specially-labeled waypoints, called checkpoints, to visit along the road network.

The *Motion Planning* subsystem is responsible for the safe, timely execution of the incremental goals issued by the Behavioral Executive. Separate path-planning algorithms are used for on-road driving and unstructured zone navigation, and the nature and capabilities of each planner have a strong influence on the overall capabilities of the system[5]. Generally speaking, these algorithms plan and execute a path toward the goal and attempt to avoid both static and dynamic obstacles in a purely reactive fashion, deferring more complex interactions to the Behavioral Executive. The Motion Planner periodically publishes its progress on the current goal, which is used by the Behavioral Executive to cue the selection and publication of subsequent goals as appropriate. In the event that no collision-free path can be found, the Motion Planner reports a tracking error by this same channel, and the Behavioral Executive must determine the next best action to take according to the current context.

The final capabilities of each subsystem could not be completely understood and specified *a priori*, as they each included open research problems in urban navigation. Moreover, the development window was narrow and rigid, with less than 18 months between the competition announcement and the final event. The differences between platforms, sensors, team composition, and the overall nature of the competitions meant that very few software artifacts could be reused from the previous Grand Challenge efforts. The few elements that were reused were very narrow in scope, such as interface software for specific sensors, along with various utility classes, such as geometric primitives and other abstract data representations. Otherwise, it was necessary to develop a completely new software system within the aforementioned constraints, which together represented significant risks to the team's success.

III. Design Criteria

To mitigate these risks, the team followed the *Spiral*[3] development process, wherein incrementally more complex abilities are added to a growing core of functionality. These incremental enhancements are rigorously tested, and the system's design and priorities are periodically revisited to incorporate an evolving understanding of the problem domain. This incremental development process was a significant influence on the design of the Behavioral Executive, as it depended heavily on the competence and capability of each other subsystem while simultaneously being responsible for some of the robot's most outwardly-visible operation. As such, it had to remain highly agile with respect to evolving functionality in the rest of the system while also supporting incremental growth of its own features. This requirement for agility is reflected in many of the design criteria that were applied to the Behavioral Executive.

In terms of functional requirements, the Behavioral Executive had two fundamental responsibilities. First, it had to guarantee the robot's adherence to the Urban Challenge rules regarding logical traffic interaction, including:

- The maintenance of a safe following distance to a leading vehicle when travelling on a road;
- The determination of and adherence to the arrival-based precedence ordering at intersections;
- The safe merging into or across moving traffic at intersections, such as at a typical T-intersection;
- The initiation of tactical passing maneuvers on multi-lane roads, either at speed into another lane in the same direction, or from a stop and into an oncoming travel lane if the system encounters a disabled vehicle.

Second, the Behavioral Executive had to guarantee the successful execution of the system's mission, even if it meant temporarily violating the normal conventions of traffic flow and interaction. The Urban Challenge rules provided for several likely situations such as deadlock resolution at intersections and turning around and re-routing at a roadblock, but there were many other possible circumstances with no strictly legal path forward. In these cases, the only guidance was that the vehicle should perform qualitatively “safe” actions, and that judges would penalize “unsafe” actions at their discretion.

This open-ended functional specification meant that the Behavioral Executive's design had to maximize its ability to incrementally accommodate as much functionality as could be developed in the time available, all without introducing unnecessary delays into the system. In terms of non-functional requirements, the design had to:

- Minimize the latency of behavioral decisions by reacting quickly and efficiently to novel input data;
- Support multiple and alternative behavioral configurations at load-time, especially allowing for fast prototyping of new functionality and the testing of a variety of alternate or aberrant behaviors;
- Support isolated runtime configurability, such as interactively enabling/disabling new features for in-situ performance comparison;

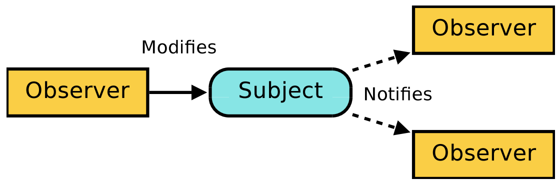


Fig. 3: The Observer Pattern: Observers are notified whenever a Subject is modified, usually by another Observer.

- Support extensive instrumentation for on-line examination and off-line analysis of the status and performance of the subsystem, especially providing details of the robot's situational awareness and intent;
- Easily accommodate additional responsibilities that may be identified later in the development process without interfering with the functionality of existing elements.
- Exploit, to the greatest extent possible, the availability of an extensive simulation environment that allows separate instances of the complete software system to control separate vehicles operating in the same simulated world.

Beyond explicit requirements, the design was also constrained by architectural decisions that were intended to mitigate various team-level risks of such a rapid development effort. Most notably, all system tasks were required to use a custom C++ application framework that provided common configuration, scheduling, and interface management with the following properties:

- Message delivery is best-effort, potentially requiring loss detection and retransmission of critical messages;
- Message interfaces must be periodically polled for new data, rather than providing asynchronous notification of message arrival;
- Polling an interface consumes a message if one is waiting, requiring the user to cache incoming message data as necessary.

These properties encouraged a certain amount of uniformity among tasks in the system and made the infrastructure accessible to team members of varying programming skill, both of which were highly valuable to the overall development effort. However, it assumed a fixed duty cycle and a fairly straightforward mapping of inputs to outputs that was not entirely consistent with the diverse requirements of the Behavioral Executive. As such, the subsystem's design had to provide a more fine-grained unit of functional decomposition and interaction that would provide the modularity necessary to fulfill the subsystem's requirements.

IV. Behavioral Executive Architecture

A. Design Patterns

These requirements were satisfied by the selection and application of several design patterns, the most important of which, the *Observer Pattern*[6], is illustrated in Figure 3. The Observer Pattern is a class collaboration pattern wherein a set of active elements, called *Observers*, communicate indirectly via intermediate data elements, called *Subjects*. On initialization, an Observer may register an interest in any number of Subjects, after which the Observer's *notify()* method is called whenever one of its Subjects is modified, allowing the Observer to update its internal state and perhaps modify other Subjects in the system.

Multiple Observers can register for notifications from the same Subject, and the resultant pattern of communication is very similar to the anonymous publish-subscribe pattern employed at the inter-process level. The key difference is that the Subjects are single, persistent elements, whereas the messages passed according to anonymous publish-subscribe are multiple and transient. Nevertheless, the Observer Pattern conveys similar benefits, especially in strong decoupling of functionality. Algorithms for specific behaviors can be encapsulated in individual Observers, effectively isolating disparate functionality and promoting the design's ability to accommodate incremental growth. In addition, maintaining important intermediate data products as Subjects allows for the straightforward addition of secondary functionality such as instrumentation for on-line analysis or the transmission of various status messages, as described in Section VI.

The largest risk associated with the Observer Pattern is the potential for complex interdependencies between Observers and Subjects to lead to an infinite cycle of change notifications. As suggested in [6], this was mitigated by decoupling the modification of a Subject from the associated change notification through an external entity, called the *Observatory*. This element is responsible for periodically iterating over all Subjects and triggering notifications for the ones that have changed, repeating the process until the system settles (i.e. there are no more "changed" Subjects), or until some maximum number of cycles is reached. In the latter case, the Observatory logs a prominent warning, waits for the next round of execution, and starts again to ensure that the system remains responsive to novel message inputs. This decoupling has the additional benefit of keeping the execution context within exactly one Observer at a time, rather than recursively moving through multiple Observers, eliminating the need to deal with reentrancy in an Observer's *notify()* method.

The Observer Pattern is complemented in the Behavioral Executive by the use of an *Abstract Factory*[6] to instantiate Observers and Subjects. The most important benefit is that this allows multiple Observers to be developed to fulfill the same role in different ways, with the active set of Observers specified at load-time by a configuration file. This allows for rapid load-time reconfiguration of the Behavioral Executive, directly supporting the incremental development process and the testing of alternate or aberrant behaviors without affecting the functionality of the primary behavioral modules. For example, various algorithms for behaviors such as tactical lane selection can be developed as separate Observers and all compiled into the same program. The active lane selection algorithm is determined by a configuration file, allowing field testers to switch between them more readily, and allowing for competitive evaluation in simulation by running multiple variations of the Behavioral Executive on different simulated vehicles operating in the same simulation environment.

Another complementary pattern, the *Adaptor*[6] Pattern, leverages the similarity between the Observer and Publish-Subscribe patterns to transparently convert the messaging interfaces into Subjects. The only element that is aware of the difference is the controlling Observatory, which must explicitly update the message-input Subjects at the start of a computational cycle and flush "changed" message-output Subjects at the end. Beyond that, message inputs and outputs blend seamlessly with the rest of the architecture, with message arrival triggering notification of registered Observers, and the modification of message-output Subjects having the side effect of message transmission.

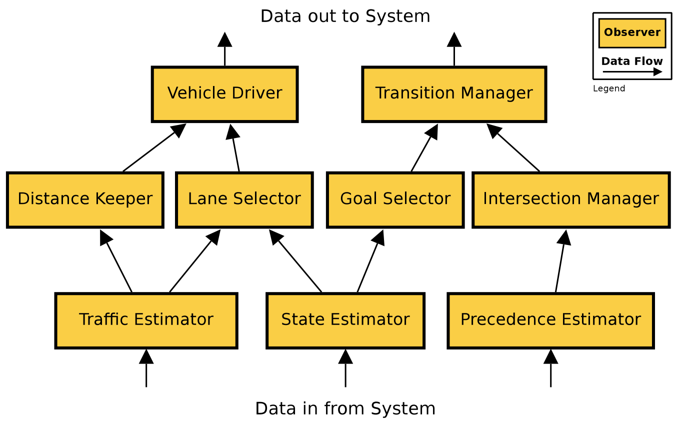


Fig. 4: Original Behavioral Executive Architecture, showing the primary functional elements and their approximate dependencies.

The fourth major pattern applied to the design, the *Strategy Pattern*[6], encapsulates the common responsibilities of an architectural role, such as gathering inputs and propagating the results to the rest of the system, in an abstract class and uses polymorphism to promote the rapid development of strategically-different approaches to the same role. In support of incremental development, a simple, typically naive, variant can fulfill a given strategic role in early stages, deferring the development of advanced tactical algorithms until the necessary functionality is available in other subsystems. This pattern also supports varietal simulation and competitive evaluation, as it is straightforward to introduce subclasses that exhibit aberrant behaviors for testing in simulation or to expose some specific case for field testing. A good example of this would be a Lane Selector class that oscillates or otherwise randomly selects among the available lanes on a road. This is useful in simulation for testing the system's ability to cope with this kind of behavior in other vehicles, and in the field for testing the Motion Planner's lane-changing performance on an otherwise open road.

B. Functional Decomposition

A rough, abstract layout for the pattern of change-propagation was generated to guide the development process, and it is reproduced in Figure 4, showing the core Observers in the system and the data dependencies between them.

The overall intent was for the Behavioral Executive to act in direct response to novel message inputs, iteratively modifying internal Subjects and propagating changes according to the rules in each Observer to ultimately generate the outputs to the rest of the system. The original role specification for these Observers was as follows:

- **Traffic Estimator** shall combine the list of known other vehicles with the world model to produce an abstract representation of local traffic conditions;
- **State Estimator** shall combine the vehicle's position with the world model to produce a representation of the robot's logical position within the road network;
- **Precedence Estimator** shall use the list of known other cars to determine the precedence ordering at the upcoming intersection;
- **Distance Keeper** shall make use of the surrounding traffic conditions to determine the necessary safety gaps and govern the robot's speed accordingly;
- **Lane Selector** shall make use of the surrounding traffic conditions to determine the optimal lane to travel in, executing a merge into that lane if necessary;
- **Vehicle Driver** shall combine the outputs of the Distance Keeper and Lane Selector with its own internal rules to generate the *Motion Parameters* message, governing details such as the vehicle's speed and tracking lane;
- **Goal Selector** shall make use of the current logical location as reported by the State Estimator to generate a series of incremental goals for execution by the Motion Planner;
- **Intersection Manager** shall make use of the current location in the road network to determine the next intersection of consequence and combine that with the precedence ordering from the Precedence Estimator to produce a boolean indication of whether the robot should proceed through the intersection;
- **Transition Manager** shall manage the discrete goal interface between the Behavioral Executive and the Motion Planner, using the goals from the Goal Selector and the output from the Intersection Manager to determine when to transmit the next sequence of goals.

Given the time constraint on the development effort, the Subjects that actually form the data flow between these elements were only loosely specified at design time. However, strong emphasis was placed on using the minimal effective representation of each Observer's outputs, which would collectively become the critical internal state of the Behavioral Executive. This had the dual benefit of encouraging the careful selection of intermediate data products and suppressing an early explosion of complex interdependencies, which would have led to a highly brittle system in the long run. In addition, cyclic and multi-path notifications were avoided when possible, and carefully documented when necessary.

All of these elements were expected to evolve with the rest of the system, and the nature of the Observer pattern allowed Observers to be easily combined when they became too closely interdependent, or else subdivided when they grew too large or acquired too many responsibilities. Some Observers underwent these transformations, and others were added along the way, but much of the original design remains visible in the final architecture, shown in Figure 5.

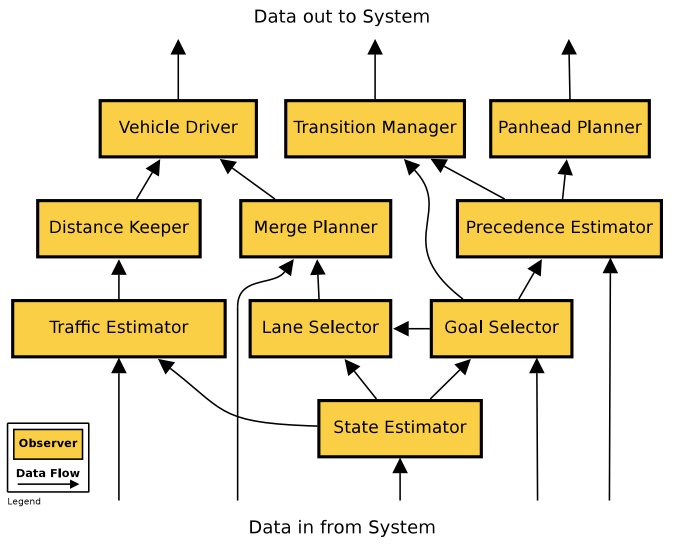


Fig. 5: Final Behavioral Executive Architecture, showing primary functional elements and their dependencies.

Beyond slightly expanded and rearranged data dependencies, there are three primary differences between the initial and final designs that are worthy of mention. First, the Intersection Manager and the Precedence Estimator proved too closely related to maintain separately, so they were combined into a single entity, retaining the name "Precedence Estimator", early in the development process. Second, the Lane Selector was split into two Observers, the first, retaining the name "Lane Selector", responsible for tactical decision-making, and the second, called the "Merge Planner", responsible for synchronizing with and merging into an adjacent lane of traffic. This separation of concerns proved very effective and is further discussed in section V-C. Third, and late in the development process, the Behavioral Executive acquired the responsibility of managing two sets of long-range sensors which could be pointed in various directions to gain more information about moving traffic around the robot. While this was a significant departure from the subsystem's original requirements, the design easily accommodated this functionality, encapsulated in the Panhead Planner, as discussed in Section VI.

V. Detailed Module Discussion

It is important to note that the final implementation includes many more functional elements and more convoluted data paths than listed above, but these generally belong to auxiliary functionality, such as diagnostic state reporting, or else are an artifact of the system's rapid development cycle, as in the case of the Panhead Planner. On race day, the Behavioral Executive was configured with 15 Observers and 93 Subjects, and a complete, detailed discussion of these elements and their interactions, covering nearly 25,000 lines of code, is beyond the scope of this article. Instead, the core elements depicted in Figure 5 are discussed in more detail in the following subsections, grouped by functional context. Thereafter, Section VI presents several secondary responsibilities that are fulfilled entirely through notifications from core Subjects, highlighting one of the key strengths of the design. Additional discussion of the joint necessities of careful design and flexible algorithms in the Behavioral Executive may be found in [1].

A. Goal Selection

The goal selection group encapsulates mission execution requirements of the Behavioral Executive, determining the best action to take toward the next checkpoint, monitoring the progress of that action, and selecting alternate actions in case of failure. This is achieved in the collaboration of two Observers, the State Estimator and the Goal Selector, as illustrated in Figure 6.

The State Estimator converts the data provided by the *Vehicle State* message, which describes the pose and velocity of the robot's inertial measurement unit (IMU) into forms that are more directly relevant to the Behavioral Executive. The *Bumper Pose* and *Bumper Speed* Subjects are transformations of the IMU pose and speed into the pose and speed of the robot's front bumper, which is the primary reference point for the Urban Challenge rules regarding discrete traffic interaction. In addition, the State Estimator throttles the update rate of these Subjects to between 1 and 13 Hz, depending on the speed of the robot, whereas Vehicle State updates occur in excess of 200 Hz. This reduces extraneous computation in Observers that depend on the physical position of the robot and decouples them from the details, both in terms of content and arrival rate, of the Vehicle State message.

The *Current World ID* represents the robot's logical position in the road network, nominally describing the waypoint in the road network that is closest to the robot's current position or that the robot has most recently passed over. However, it can also be set to the World ID of a larger entity, such as a lane or parking zone, if the robot is initialized or otherwise encounters an erroneous situation at an arbitrary position in the road network. The Current World ID may also be set to a special "invalid" value, or *invalidated*, indicating that there is not enough information available to proceed on-mission, such as during initialization when the Vehicle State and Road Model are not yet available.

Updates to the Current World ID are sparse and represent critical transition points in the execution of the robot's mission. Primarily, the State Estimator monitors the *Planner Status* messages and updates the Current World ID either on the successful completion or failed termination of the current goal, indicating the need to select a new goal. Otherwise, the Current World ID is updated only under certain special circumstances, such as the engagement of the robot's emergency-stop or manual-override switches, whereupon it is explicitly invalidated with the intent of placing the Behavioral Executive, and by extension the rest of the system, into a quiescent state. The Current World ID is held invalid until the associated circumstance has passed, whereupon the State Estimator re-resolves the vehicle's position in the road network as on initialization. This representation promotes strong decoupling in the rest of the Behavioral Executive, as both the instantaneous value of and the occurrence of updates to the Current World ID provide clear meaning to the rest of the subsystem.

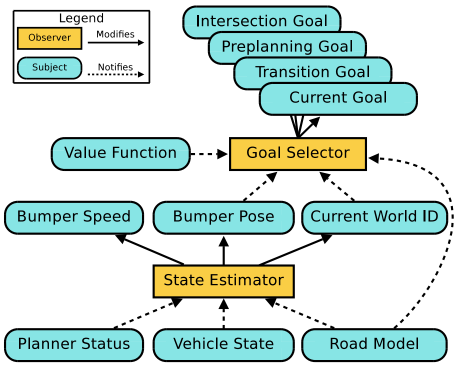


Fig. 6: Goal Selection Observer/Subject collaboration diagram.

On updates to the Current World ID, the Goal Selector searches along the road network, starting at the node represented by the Current World ID and following the *Value Function* towards the current checkpoint. A series of incremental goals are generated to describing the current action to take, called the *Current Goal*, and three critical points in the future: the *Transition*, *Pre-Planning*, and *Intersection* goals. The Transition Goal represents the action to take immediately after the Current Goal and allows smooth transitions between goals at-speed. The Pre-Planning goal describes the next action the robot will take in a parking lot, providing the Motion Planner with an opportunity to pre-initialize the zone planner in advance of reaching the parking lot. The Intersection Goal describes the next intersection that the robot will interact with and is invalid if no intersections will be encountered in the near future. Also, the Intersection Goal will be identical to the Current Goal when the robot is about to proceed through an intersection.

If the Current World ID is invalid, the Road Model is not available, or there is no Value Function for the current checkpoint, the three meta-goals are invalidated and the Current Goal is set to a special "Idle" goal, which causes the Motion Planner to stop the vehicle and wait for further instructions. Notification from any of these inputs will trigger the recomputation of all four goals, guaranteeing that they instantly reflect the system's current location and intent. This promotes additional decoupling in the subsystem, as other Observers may make direct use of these goals to perform their individual functions, as in the case of managing intersections.

B. Intersection Handling

The Intersection Handling Group consists of two Observers, the Precedence Estimator and the Transition Manager, as shown in Figure 7. The Precedence Estimator uses the Intersection Goal to determine which intersection to monitor, for which it computes the set of all possible areas around the intersection where a relevant vehicle might be found. This set of *Occupancy Polygons* is critical to the algorithms in the Precedence Estimator, which are described in more detail in [9]. They were initially exported as a Subject for diagnostic purposes, but also proved useful for other secondary responsibilities, as discussed in Section VI.

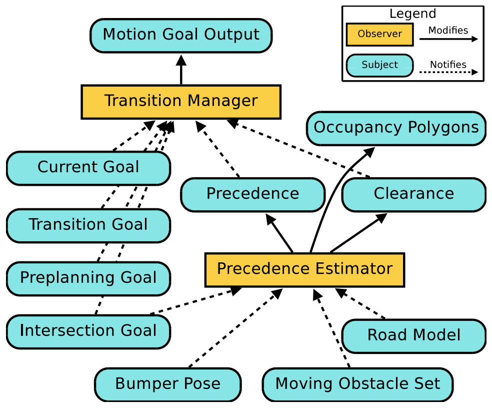


Fig. 7: Intersection Handling Observer/Subject collaboration diagram.

After setting up the Occupancy Polygons, notifications from the *Moving Obstacle Set* cause the Precedence Estimator to compute and update two boolean Subjects, *Precedence* and *Clearance*. Precedence indicates whether it is Boss's turn to proceed through the intersection and Clearance indicates whether there is a sufficient window of opportunity to merge into or across any relevant lanes of moving traffic. This representation promotes incremental development by allowing the Precedence Estimator's role to be initially filled with a skeleton that simply sets Precedence and Clearance to "true", proceeding with incremental enhancements to the estimation of each value as the robot's perceptive capabilities evolve over time. Similarly, this supports varietal testing in simulation, as simplistic or erroneous behaviors, such as waiting until there are no cars visible, or waiting for exactly five seconds, can be easily implemented and introduced as alternate Precedence Estimators.

The two boolean outputs of the Precedence Estimator are observed by the Transition Manager, which incorporates their values into a set of rules that determine when the goals from the Goal Selector should be propagated to the *Motion Goal Output*. If the Current Goal is "Idle", it is propagated immediately, causing the system to halt abruptly if it is already moving and guaranteeing the immediate propagation of the effects of the emergency-stop or manual-override states described above. Otherwise, if the Current Goal describes an action through an intersection (i.e., it is identical to the Intersection Goal), then it will wait on Precedence and Clearance from the Precedence Estimator, causing the robot to wait on its turn at the intersection. The Transition Manager also encapsulates the details of the interface to the Motion Planner, periodically retransmitting the goals until the Planner acknowledges their receipt and performing various checks on the outgoing goals to ensure that they are sensible and safe.

C. Lane Driving

The Lane Driving Group consists of five Observers which are active whenever the robot is driving along a traffic lane. The Traffic Estimator abstracts the data in the *Road Blockage Set* and *Moving Obstacle Sets* into a representation of the distance to the closest leading obstacle in the current lane, called the *Lead Vehicle Distance*. It also propagates the speed of the associated obstacle, zero if it is a Road Blockage, as the *Lead Vehicle Speed*, and compensates for transient data loss or tracking errors by extrapolating these values over a short time. The Distance Keeper uses this abstracted representation to implement the speed government policy that guarantees minimum spacing between vehicles moving along a road, through its *Distance Keeping (DK) Speed* output. The distance-keeping behavior was the most frequently encountered behavior that was also safety-critical, so this separation of concerns provided important support for incremental development in other subsystems, as the robot's ability to detect traffic and discrete obstacles on the road ahead was continuously in flux.

In parallel, the Lane Selector uses the current pose of the robot along the road to keep track of the *Current Lane* of travel, and to determine whether that differs from the desired, or *Intended Lane* of travel. The Intended Lane is generally the result of a tactical decision based on the distance to Current Goal, the lane required at the termination of the Current Goal, and whether surrounding traffic conditions necessitate a passing maneuver.

When the Current and Intended Lanes differ, the Merge Planner monitors the set of available merge slots between vehicles in the Intended Lane for an opportunity to change lanes without violating vehicle separation rules. When active, the Merge Planner governs the speed of the robot via the *Merge Speed* output, slowing to wait for a merge slot and to synchronize with traffic in the intended lane before issuing the merge command by propagating the Intended Lane onto its *Commanded Lane* output. Otherwise, the Merge Planner propagates the Current Lane as the Commanded Lane.

Both the Lane Selector and Merge Planner roles can be fulfilled in a variety of ways, so they are implemented according to the Strategy Pattern discussed in Section III. This promotes incremental development by allowing their roles to be initially filled by simple heuristics such as "stay in the goal lane" and "merge after three seconds". Later in the development process, this separation allowed for rapid prototyping of alternate or enhanced strategies as more and better data became available from the Perception subsystem.

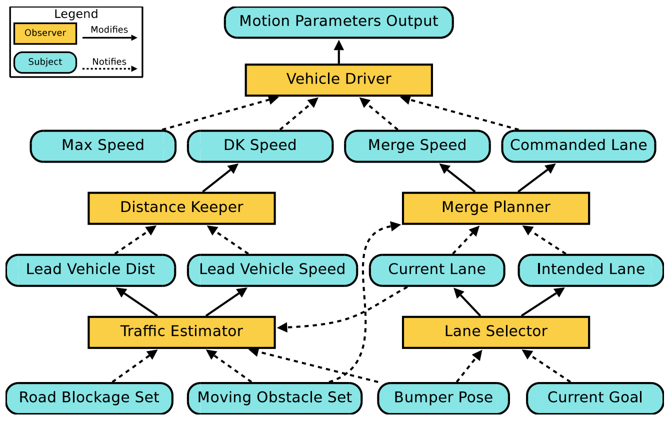
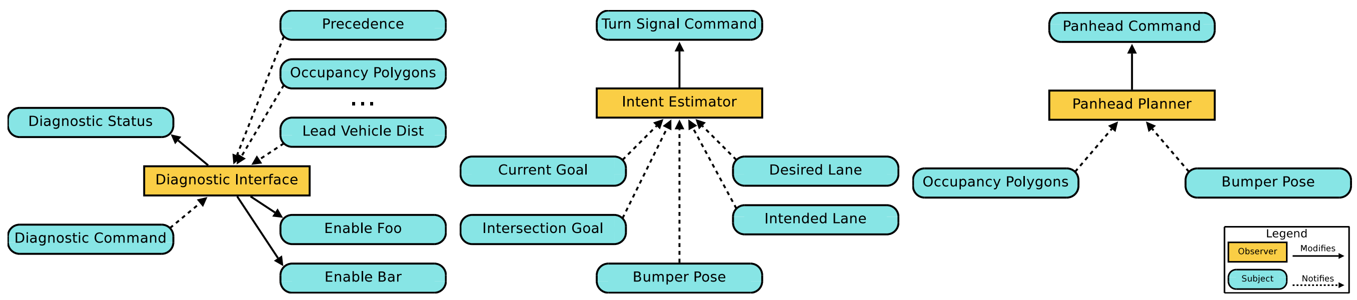


Fig. 8: Lane Driving Observer/Subject collaboration diagram.



(a) Diagnostic Status Reporting (b) Intent Estimation (c) Panhead Planning

Fig. 9: Secondary responsibilities that were fulfilled entirely using core data products.

Finally, the Vehicle Driver combines the outputs of the other Observers in this group according to its internal rules to produce the *Motion Parameters Output*. In general, the commanded speed of the robot is the minimum of the distance-keeping and merge speeds, along with a globally imposed *Maximum Speed*, which provides a single point to throttle the overall speed of the robot. In addition, the Vehicle Driver encapsulates the details of the parametric interface to the Motion Planner, imposing safety checks and guaranteeing the periodic transmission of the Motion Parameters message, which the Motion Planner treats as a liveness indicator from the Behavioral Executive.

VI. Secondary Responsibilities

The design of the Behavioral Executive made it straightforward to incorporate elements that use the core data products discussed above to fulfill several secondary responsibilities without interfering with core functionality. Three of these roles, shown in Figure 9, are discussed in this section to highlight some of the key benefits of the subsystem's design.

First, it was necessary to periodically export the subsystem's internal state in a diagnostic message to provide the robot's operator with real-time insight into its situational awareness and intent. This role was filled by the *Diagnostic Interface*, shown in Figure 9a, which observes essentially all of the Subjects in the Behavioral Executive and transcribes their contents to a *Diagnostic Status* output. Notifications from certain critical Subjects, such as Precedence or Clearance at an intersection, cause the status message to be transmitted immediately. Otherwise, it is transmitted at a maximum of 10 Hz to avoid flooding the system with redundant status messages. When combined with comprehensive message logging and the availability of a powerful simulation environment, this also allowed for very effective off-line debugging of the Behavioral Executive. Problems witnessed in the field could be marked in the logs and subsequently scrutinized using a playback utility to determine the inciting circumstances, which could then be carefully re-created in simulation, using the Diagnostic Status messages as a guide. The problem could then be analyzed, resolved and verified without consuming on-vehicle test time or otherwise endangering the robot or its operators. In addition, the Diagnostic Interface monitors the *Diagnostic Command* message input and can manipulate various, typically boolean "Enable Foo"-style Subjects as a function of incoming requests, leveraging the nature of the Observer Pattern to fulfill the requirement for minor runtime configurability.

Second, the Behavioral Executive acquired the responsibility of managing the robot's turn signals on approach to an intersection, and also when changing lanes. The Intent Estimator, shown in Figure 9b, applies a set of fairly simple heuristics to the data from several core Subjects to generate the *Turn Signal Command* that is transmitted to the vehicle controller. The Intersection Goal is used to determine which direction the robot will turn, if any, at the upcoming intersection, and the Bumper Pose is used to activate the appropriate signal when the robot is within 30 meters of the turn. The Current and Intended Lanes are used to determine lane-change intent, and when they differ, a simple geometric check is used to determine whether the intended lane-change is to the left or to the right. In addition, the Intent Estimator activates the hazard flashers when the Current Goal is "Idle", providing a clear indication that the robot is inactive.

Lastly, and late in the development process, the Panhead Planner, shown in Figure 9c, was added to allow the Behavioral Executive to perform situation-dependent optimization of the coverage of two sets of panning long-range laser and radar sensors. The Panhead Planner uses the Occupancy Polygons and the Bumper Pose to perform a set of geometric calculations that optimize the coverage of these sensors for oncoming and cross-traffic lanes that are not covered by the robot's statically-configured sensors, updating the *Panhead Command* angles as necessary. For example, if the robot is waiting at a typical T-intersection, the panning sensors are pointed along the crossing road to provide more reliable long-range detection of fast-moving vehicles, improving the system's ability to make safe merge decisions.

VII. Analysis and Conclusions

The ability to absorb these additional responsibilities without interfering with any of its core functionality is a strong indication that the architecture of the Behavioral Executive was well suited to its purpose. Even still, the design began to deteriorate as the competition approached and various shortcuts were taken to try to quickly fix problems. Under these conditions, the inability to fully understand the complex notification patterns among the Observers led to some unexpected and erroneous results. For example, the accidental introduction of a circular dependency among the Observers in the system inadvertently caused an "Idle" goal to be spuriously emitted, which in turn caused the robot to momentarily slam on the brakes, in certain circumstances when traveling through an intersection. Debugging this and similar problems required consideration of the Behavioral Executive as a whole, defying the decoupling and encapsulation that guided the subsystem's design. Often, these dependency cycles were broken by converting some notification-based task into a periodic task, relieving the issue at the cost of some small amount of latency. For most other projects, these problems would have been mitigated by adjusting the delivery schedule instead of compromising performance, but that option was unavailable given the fixed date of the final competition.

These difficulties were, however, the exceptions, rather than the rule. Overall, the design allowed the Behavioral Executive to remain highly adaptive to Boss's evolving needs and capabilities. The effective abstraction of messaging details insulated the subsystem from ongoing changes in external functionality, mitigating one of the most significant risks of complex robotic system development according to the Spiral process. The many internal avenues of support for the incremental development process and the clear separation of responsibilities among Observers were critical to the successful development of the Behavioral Executive, providing a solid framework for growing core functionalities and fertile ground for testing new and different ideas. This ultimately allowed for the development of a reliable, competent system for managing the traffic-interactive behaviors that characterized the Urban Challenge.

References

[1] Christopher R. Baker and John M. Dolan. Traffic Interaction in the Urban Challenge: Putting Boss on its Best Behavior. In *IEEE/RSJ International Conference on Intelligent RObots and Systems*, pages 1752–1758, 2008.

[2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley, 2003.

[3] B Boehm. A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes*, 11(4):14–24, 1986.

[4] Defense Advanced Research Projects Agency (DARPA). Urban challenge website, July 2007. http://www.darpa.mil/grandchallenge.

[5] David I. Ferguson, Christopher R. Baker, Maxim Likhachev, and John M. Dolan. A Reasoning Framework for Autonomous Urban Driving. In *IEEE Intelligent Vehicles Symposium*, pages 775–780, 2008.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[7] Karl Iagnemma and Martin Buehler. Special Issue on the DARPA Grand Challenge, Part 1. *Journal of Field Robotics*, 23(8), 2006.

[8] Karl Iagnemma and Martin Buehler. Special Issue on the DARPA Grand Challenge, Part 2. *Journal of Field Robotics*, 23(9), 2006.

[9] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, and Jason Ziglar. Autonomous Driving in Urban Environments: Boss and the DARPA Urban Challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

Christopher R. Baker Christopher Baker earned a B.S. degree in Computer Science from Carnegie Mellon University in 2002 and an M.S. degree in Robotics from Carnegie Mellon University in 2004. He is currently a Ph.D candidate in the Robotics Institute at Carnegie Mellon University, advised by Dr. John Dolan and researching the challenges of the design and integration of complex robotic systems. He has participated in the development of robotic systems for autonomous subterranean exploration and mapping, and was most recently a subsystem architect on Carnegie Mellon's Tartan Racing team, winners of the 2007 DARPA Urban Challenge.

John M. Dolan John Dolan is a senior systems scientist at Carnegie Mellon University's (CMU) Robotics Institute, where he has been a member of the research faculty since receiving his Ph.D. in mechanical engineering from CMU in 1991. His research interests include multi-robot cooperation, human-robot interaction, and sensor-based control. He was the behaviors lead for Carnegie Mellon's Tartan Racing team, and is currently PI for NASA projects involving telesupervision of multiple autonomous oceangoing platforms for harmful algal bloom characterization and reliability analysis of planetary rover teams. His previous work at CMU includes distributed multirobot reconnaissance and surveillance (DARPA "Cyberscout") and formulation of the kinematics and control of filament winding for non-axisymmetric transformers. Dr. Dolan studied for a year at the Technical University of Munich on a Fulbright Scholarship, working simultaneously at the German Space Agency (DLR) on the stability and ride quality of magnetically levitated trains. He is currently a colonel and military intelligence officer in the U.S. Army Reserves.