Traffic Interaction in the Urban Challenge: Putting Boss on its Best Behavior

Christopher R. Baker and John M. Dolan

***Abstract*— We describe 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 software system that controls "Boss", Tartan Racing's winning entry in the competition. Design criteria are presented, followed by the application of software design principles to derive an architecture well suited to the rigors of developing complex robotic systems. Combined with a discussion of robust behavioral algorithms, the design's effectiveness is highlighted in its ability to manage complex autonomous driving behaviors while remaining adaptable to the system's evolving capabilities.**

I. Introduction

The Urban Challenge[4] was an autonomous vehicle competition sponsored by the US Defense Advanced Research Projects Agency (DARPA). Contestant robots were required to autonomously execute a series of navigation missions in 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 the previous challenges[6], [7], which focused on rough-terrain navigation, this competition required the development of 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 Fig. 1. The overall software system that controls Boss is discussed in brief in Sec. II to provide context for discussion of the Behavioral Executive, which begins with a presentation of design goals and resultant architectural decisions in Sec. III. A detailed discussion of the core functionality follows in Secs. IV, V and VI, highlighting the combination of strong decoupling with robust, tunable algorithms to allow the Behavioral Executive to remain adaptable to the evolving needs and capabilities of the rest of the system. This adaptability is further highlighted in Sec. VII, which discusses implementation of several secondary functions based entirely on intermediate data products from the core elements. Sec. VIII discusses the strengths and weaknesses of the subsystem's design and summarizes the lessons learned over the course of development.

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 winning entry to the Urban Challenge.

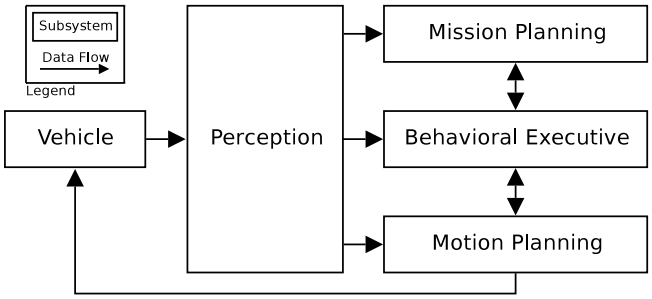

Fig. 2. System Architecture, showing primary subsystems and data paths.

II. System Architecture

The software system that controls Boss is divided into four primary subsystems: *Perception*, *Mission Planning*, *Behavioral Execution*, and *Motion Planning*. Their dominant communications paths are shown in Fig. 2, and they communicate via message-passing according to the anonymous publish-subscribe[9] pattern. Their individual responsibilities are described here in brief, and a more thorough presentation of the system's functionality may be found in [3].

The *Perception* subsystem processes sensor data from the vehicle and produces a collection of semantically-rich data elements such as the current pose of the robot, the geometry of the road network, and the location and nature of various obstacles such as road blockages and other vehicles. These data products are called the *Vehicle State*, *Road Model*, *Road Blockage Set*, and *Moving Obstacle Set*, respectively.

The *Mission Planning* subsystem computes the fastest route to reach the next checkpoint from any point in the road network, using 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 a *Value Function* that maps each waypoint in the road network to an estimated time to reach the next checkpoint[1].

The *Behavioral Executive* follows the Value Function from the robot's current position, generating 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. In the case of 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 *Motion Planning* subsystem is responsible for the safe, timely execution of the incremental goals issued by the Behavioral Executive. These goals fall into two broad contexts: structured, on-road driving and unstructured zone navigation. A separate path-planning algorithm is used for each context, and the nature and capabilities of each planner have a strong influence on the overall capabilities of the system[1]. The Motion Planner periodically publishes its progress on the current goal, used by the Behavioral Executive to cue the selection and publication of subsequent or alternate goals as appropriate.

III. Design Motivations and Subsystem Architecture

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. To incorporate an evolving understanding of the problem domain, the team followed the *Spiral*[2] development process, wherein incrementally more complex abilities are added to a growing core of functionality. This incremental process was a significant influence on the design of the Behavioral Executive, as it depended heavily on the capabilities of all other subsystems 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 several of the criteria that drove the design of the Behavioral Executive, most importantly that it should:

- Support the incremental development of a diverse set of capabilities;
- Minimize the impact of changes in other subsystems through abstraction and strong decoupling;
- Support instrumentation for on-line examination and off-line analysis of the subsystem's internal state;
- Configurably support alternate behavioral algorithms for comparative evaluation and diversity in simulation.

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

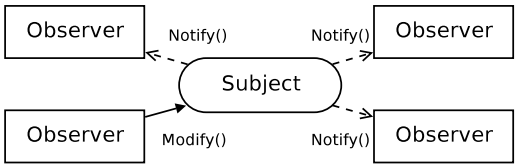

Fig. 3. The Observer Pattern: Observers expose a *notify()* method that is called whenever a registered Subject is modified.

These design goals were satisfied by the selection and application of several design patterns, the most important of which is the *Observer Pattern*[5], illustrated in Fig. 3. The Observer Pattern is a class collaboration pattern wherein a set of active elements, called *Observers*, communicate 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 them 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. 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 intermediate data products as Subjects allows for the straightforward addition of secondary functionality, such as instrumentation for on-line analysis, discussed in Sec. VII.

The Observer Pattern is complemented in the Behavioral Executive by the use of an *Abstract Factory*[5] to instantiate Observers and Subjects. 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, as opposed to compile-time. This supports the rapid reconfiguration of the Behavioral Executive, allowing the incremental inclusion or substitution of functionality by simply adding or replacing Observers. For example, various algorithms for behaviors such as tactical lane selection can be developed as separate Observers with the active lane selection algorithm determined by a configuration file. This allows field testers to switch between variations more readily, and allows for competitive evaluation in simulation by running multiple variations on different simulated vehicles within the same simulated environment.

The core functionality of the Behavioral Executive is implemented in nine Observers, which are grouped into three functional contexts as shown in Fig. 4. The Behavioral Executive acts in direct response to novel message inputs, iteratively modifying internal Subjects and propagating changes according to the rules in each Observer to generate the outputs to the rest of the system as appropriate. There are, in fact, many more functional elements and more convoluted data paths than are illustrated, but they generally belong to auxiliary functionality such as diagnostic state reporting or

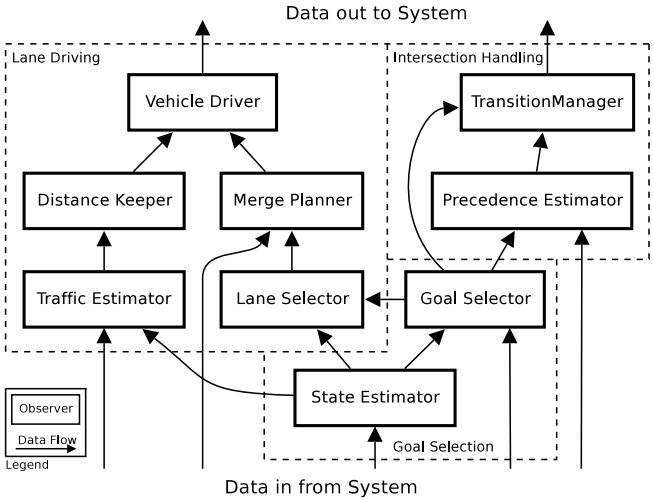

Fig. 4. Abstract data flow view of the Behavioral Executive, showing dominant elements and data paths, grouped by functional context.

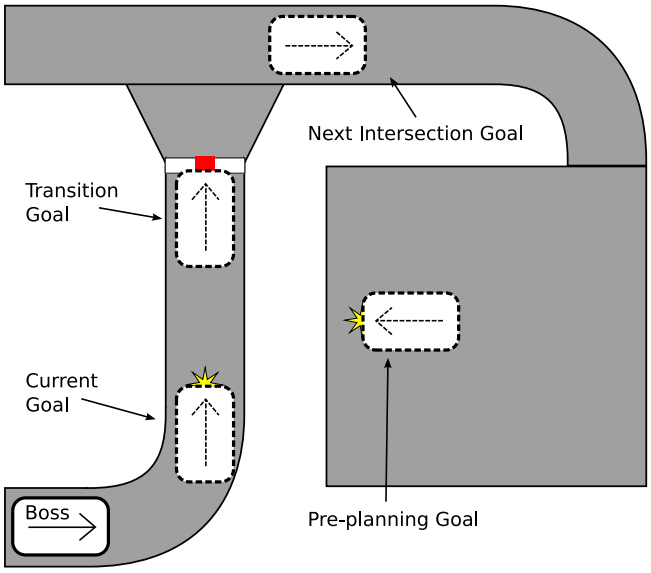

Fig. 5. The set of Motion Goals generated by the Goal Selector.

else are an artifact of the system's rapid development cycle. Some of these elements are discussed in Sec. VII, but many are omitted to allow for a more clear, concise description of the subsystem's functionality in the following sections.

IV. Goal Selection

The core responsibility of the Behavioral Executive is to select, issue and monitor the execution of incremental *Motion Goals* along the route to the next checkpoint. Two Observers, the State Estimator and the Goal Selector, work together to fulfill this role. The nature of their interactions is strongly influenced by the structure of the Road Model, which is specified as a directed graph of latitude/longitude waypoints that are hierarchically grouped into larger elements such as roads and parking zones. Each element in the hierarchy is assigned a unique identifier, called a *World ID*, which may be used to retrieve extended information, such as the location of a waypoint or the geometry of a road lane, from the Road Model.

The State Estimator uses the vehicle's geometric position, provided by the Vehicle State message, to determine the robot's logical position within the road network, which is maintained in a Subject called the *Current World ID*. In the nominal case, this is the ID of the most recently visited waypoint, but during recovery maneuvers or on system initialization, this can be the ID of the lane, zone or intersection that the system failed in or was initialized near. If the vehicle's autonomy is interrupted by manual override or emergency stop, the Current World ID is invalidated, causing the system to enter an idle state until the override is cleared.

The Goal Selector combines the Current World ID with the Value Function the Mission Planner to generate a set of Motion Goals, illustrated in Fig. 5, that describe the current action to be taken, called the *Current Goal*, and up to three specific actions that will be taken at critical junctures in the future. It is important to note that the three future goals have specific semantics and do not generally specify a complete incremental path to the current checkpoint. First, the *Transition Goal* is the action the system intends to take immediately after the Current Goal. It is used internally to guide forward searches for relevant obstacles along the road network and externally, by the Motion Planner, to ensure smooth transitions between actions. Second, the *Next Intersection Goal* represents the next action the vehicle intends to take through an intersection. The Next Intersection Goal is generally unique, but can be a duplicate of the Transition or Current Goal, depending on whether the robot is approaching or moving through an intersection. If the system will not encounter an intersection in the immediate future, e.g. if it is performing a long series of parking zone maneuvers, the Next Intersection goal will be invalid. Lastly, a *Pre-planning Goal* is computed and provided to the Motion Planner to describe the next action the system will take through a parking zone. It is used to reduce planning delays by initializing the zone planner in advance of reaching the parking zone.

In the nominal case, these goals represent an edge in the waypoint graph, specifying the initial and final waypoints, minimum, maximum and terminal speed constraints, and a target pose for the robot to achieve upon completion To handle anomalous situations, such as initialization near, but not on, the road network, or else to compensate for failure of a previously-specified goal, the Goal Selector includes extensive mechanisms for computing recovery goals. These goals treat the area around the robot as an obstacle zone and specify a pose on the road network that would allow the system to return to normal operation. The details of the recovery selection algorithms are beyond the scope of this paper; a more thorough analysis may be found in [1].

Rather than publishing these four goals directly to the Motion Planner, the Goal Selector are stores them in intermediate Subjects for use by other elements in the system, most notably by the Observers in the Intersection Handling group. This group, discussed in the next section, determines the appropriate time to transmit these goals, decoupling the policies for goal selection from the policies for goal propagation.

V. Intersection Handling

The Intersection Handling group consists of two Observers, the Precedence Estimator and the Transition Manager. To illustrate their combined responsibility, imagine that the robot is sitting at a stop line, waiting for precedence. While it waits, the Current Goal from the Goal Selector specifies the intended maneuver through the intersection. That goal is withheld until the Intersection Handling group determines it is time to proceed, whereupon the goal is issued and the robot travels through the intersection.

The Precedence Estimator encapsulates all of the algorithms for determining when it is safe to proceed through an intersection and controls two boolean Subjects:

- **Precedence**: whether it is the robot's turn among the vehicles waiting at stop lines; and
- **Clearance**: whether the intersection is clear of vehicles and there is sufficient opportunity to safely cross or merge into any relevant lanes of moving traffic.

The estimation of Precedence and Clearance are implemented according to the Urban Challenge Technical Evaluation Criteria[4], which specify that the system shall:

- Respect the arrival-based precedence order at intersections and not proceed out of turn;
- Wait until the intersection is free of other vehicles before proceeding.
- Yield precedence to the right in cases of simultaneous or near-simultaneous arrival;
- Determine when the precedence order has been broken by waiting for at least 10 seconds of inactivity, then may proceed into the intersection regardless of the perceived precedence order;
- Perform safe merges into or across moving traffic, allowing for sufficient separation that other vehicles do not have to slow down;

The Precedence Estimator uses the Next Intersection Goal, which determines the intersection to be monitored, and the Moving Obstacle Set, which represents the estimated vehicles around the robot, to calculate the precedence ordering at the upcoming intersection. To accommodate an evolving ability to detect and track other vehicles, the Precedence Estimator must be able to cope with several possible sources of error. Foremost, it is impossible to guarantee that vehicles will be tracked persistently over long periods of time, requiring the retention of the intersection's state through temporary tracking failures. Also, sensing and modelling uncertainties can affect the estimated shape, position and velocity of a tracked vehicle, and any one real vehicle may be represented by a number of separate Moving Obstacles, each flickering in and out of existence around the vehicle's true location.

To compensate for these possible errors, the algorithm for computing precedence does not make use of any information beyond the geometric properties of the intersection and the instantaneous position of each Moving Obstacle. Instead of assigning precedence to specific vehicles, it is assigned to individual stop lines using a geometric notion of "occupancy" to determine arrival times. Polygons are computed around each stop line, extending backward along the incoming lane by a configurable distance and exceeding the lane geometry by a configurable margin. If the front bumper of a Moving Obstacle is inside this polygon, it is considered an *occupant* of the associated stop line. Boss's front bumper is added to the pool of front bumpers, and is treated in the same manner to ensure equitable estimation of arrival times.

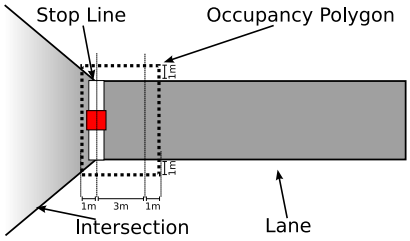

Fig. 6. A Typical Exit Occupancy Polygon

Fig. 6 shows a typical occupancy polygon extending three meters back along the lane from the stop line, with one meter of padding on all sides. Once the polygon becomes "occupied", it retains both the time of initial occupancy and the time of most recent occupancy. The former is treated as the arrival time for the purposes of precedence ordering, and the latter is used to implement a timeout on the stop line's "occupied" state, whereby it remains "occupied" for a configurable time delay once it is otherwise free of vehicles. This allows Moving Obstacles to flicker out of existence for short intervals without disturbing the precedence ordering.

In support of incremental development, this time delay, along with the polygon's size parameters, may be tuned to reflect the system's competence at tracking other vehicles. Larger polygons and longer delays are generally more robust to the sources of error mentioned above, but at the cost of the ability to discriminate between two vehicles that are queued very close together. During the Urban Challenge, the capabilities of the Perception subsystem allowed these values to be quite small, with the polygon sizes from Fig. 6 and a time delay of one second yielding highly reliable results.

To handle the case of simultaneous arrival, the arrival times are biased as a function of their position relative to Boss. Stop lines that are "to the right" receive a negative bias, effectively giving them an earlier arrival time and encoding an implicit yield-to-right rule. Similarly, stop lines that are "to the left" receive a positive bias, seeming to have arrived later and causing the system to take precedence from the left. This bias may also be tuned to the robot's perceptive capabilities, and a value of $0.5s$ proved experimentally sufficient to encode this rule with an update rate of roughly $15Hz$ and a positional accuracy well under $0.25m$ for Moving Obstacles within the scope of an intersection.

Sorting the occupied polygons in ascending order by their biased arrival time yields a direct representation of the intersection's precedence ordering. When the front of that list matches Boss's target stop line, Boss is considered to have precedence at the intersection. Deadlocks are broken by a ten-second timer that is reset every time the precedence ordering changes. If that timer expires, the Precedence Sub-

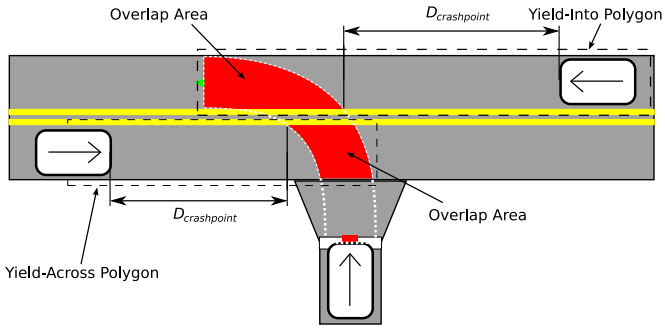

Fig. 7. Yield Determination for a Tee Intersection

ject is forced to *true*, and that state is held until the robot successfully moves through the intersection. The expiration of the timer causes the Vehicle Driver element, discussed in Section VI, to suppress the robot's maximum speed to $5mph$, causing it to proceed cautiously through the intersection. Once the robot has begun moving through intersection, there is no facility for aborting the maneuver, and the system instead relies on the Motion Planner's collision-avoidance algorithms, and ultimately the intersection recovery algorithms in the Goal Selector, to guarantee safe, forward progress should another robot enter the intersection at the same time.

The computation of Clearance begins with the construction of an occupancy polygon over the interior of the intersection in question. The same occupancy logic as used for stop lines is applied, and its "occupied" state is a direct representation of whether there is a vehicle inside the intersection, whereupon Clearance is set to *false*. Otherwise, Clearance is determined by whether there is sufficient time to merge into or across any lanes of moving traffic that intersect the robot's path through the intersection. Bounding polygons, shown in Fig. 7, are computed backwards along each of these *yield lanes*, with their lengths determined by the equation:

$$L_{polygon} = V_{max}(T_{action}+T_{delay}+T_{spacing})+D_{safety} \quad (1)$$

Where:

- V_{max} is the speed limit of the intersecting lane;
- T_{action} is the time necessary to complete the maneuver through the intersection, plus any additional time required to accelerate up to the speed limit if the robot is merging into a lane of moving traffic;
- T_{delay} compensates for system delays between the issuance of a goal and the initial motion of the robot;
- $T_{spacing}$ is the desired temporal spacing between vehicles, where one second approximates the vehicle-length per $10mph$ rule from [4];
- D_{safety} is a configurable safety margin.

On each notification from the Moving Obstacle set, a *yield window* is computed for each yield lane as the minimum estimated time of arrival (ETA) among all vehicles approaching the crash point for that lane. The ETA of any moving obstacle that is inside or overlaps the yield polygon is considered in this computation, which is simply:

$$ETA = D_{crashpoint}/V_{obstacle} \quad (2)$$

for some conservative (i.e., faster than reported) estimate of the obstacle's velocity.

The overall yield window is considered to be *instantaneously* open when $ETA_{min} > (T_{action}+T_{delay}+T_{spacing})$ for all yield lanes. In order to account for transient tracking loss, the yield window must be continuously open for a configurable period of time before Clearance is passed to the rest of the system. As with the precedence estimation algorithms, this delay, along with the parameters T_{delay} and D_{safety} are also tuned to the robot's perceptive abilities, supporting incremental development in the Perception subsystem.

The Transition Manager observes the outputs of the Precedence Estimator, waiting for both Precedence and Clearance to be *true* before propagating the goals produced by the Goal Selector onto the appropriate message outputs. Once propagated, the Transition Manager periodically retransmits these goals until the Motion Planner acknowledges their receipt, adding robustness to temporary failures and out-of-order initialization relative to the Motion Planning subsystem.

VI. Lane Driving

The third group within the Behavioral Executive is dedicated to interacting with other traffic while driving along a lane or group of lanes, performing functions analogous to the "tactical level" of reasoning in prior work on autonomous highway driving [10], [8]. Such functions include speed government in reaction to leading traffic and the management of lane-change maneuvers to pass slow-moving vehicles or to hit certain goals, such as a mid-lane checkpoint.

These functions are carried out by five Observers in the Behavioral Executive: the Traffic Estimator, Distance Keeper, Lane Selector, Merge Planner and Vehicle Driver. The ultimate output of these modules is the Motion Parameters message, which is published periodically to adjust the maximum travel speed and the desired travel lane. The Motion Planner reacts immediately to these adjustments, and the resultant maneuvers can be highly dynamic, so their computation must be as safe and reliable as possible.

Government of the maximum travel speed begins with the Traffic Estimator, which observes both the Moving Obstacle and Road Blockage message inputs and generates an abstract notion of the distance to and the speed of the closest obstacle along the lane in front of Boss. These are provided as the Lead Vehicle Distance and Lead Vehicle Speed Subjects, respectively, and their values are guarded against transient tracking loss as in the computation of Precedence and Clearance, retaining the smallest value of the Lead Vehicle Distance for a configurable time before allowing it to grow again. As the Urban Challenge was expected to consist largely of simple roads with one lane in either direction, this abstraction played an important role in decoupling the details of the obstacle inputs from the implementation of the most commonly-encountered lane behaviors: maintaining a safe distance behind a slow-moving vehicle and queueing in stop-and-go traffic.

These behaviors are implemented by the Distance Keeper, which governs the maximum speed of the robot in order to

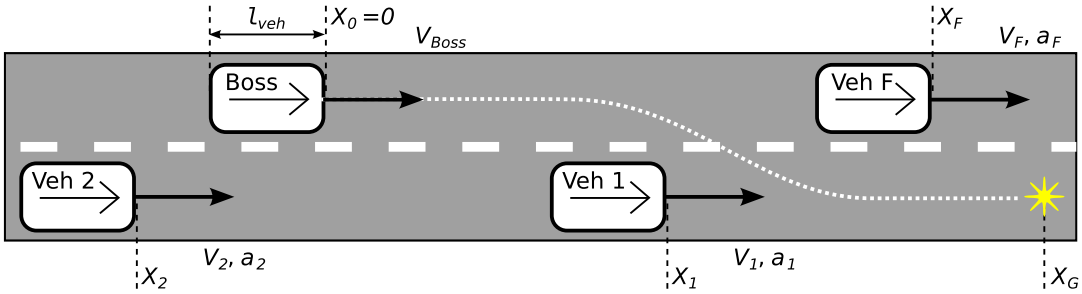

Fig. 8. An example lane driving scenario.

maintain a minimum inter-vehicle spacing of one vehicle-length per $10mph$, with a minimum of one vehicle-length in travel lanes and tapering down to $2m$ on approach to an intersection. In the steady-state, the Distance Keeper equalizes Boss's speed with the Lead Vehicle Speed, while trying to simultaneously drive the Lead Vehicle Distance to match a desired gap. The commanded velocity is computed using a simple feedback-control law:

$$v_{cmd} = K_{gap}(gap_{actual} - gap_{desired}) \tag{3}$$

where $gap_{desired}$ is computed according to the rule listed above, gap_{actual} is the Lead Vehicle Distance and K_{gap} is a configurable gain. As a simple proportional controller, this exhibits a steady-state lag in the tracking of $gap_{desired}$, but experiments showed that this had no appreciable impact on the mission execution time. Moreover, the simplicity of the control law, when combined with the abstracted Lead Vehicle Distance, afforded smooth, repeatable performance in the face of noisy obstacle data. Note that this control law will drive the commanded speed to zero on approach to a stopped car or roadblock. If the obstacle subsequently starts moving along the road, the commanded speed will increase again, resulting in a stable queuing behavior.

If the obstacle does not start moving, a passing maneuver may be warranted, which is the combined responsibility of the Lane Selector and the Merge Planner. The Lane Selector is responsible for the determination of tactical intent, or which lane Boss "wants" to be in at any given moment. The Merge Planner manages the process of seeking to, or waiting for, an appropriate opportunity to merge into the intended lane. These responsibilities are analogous to the "meta-tactical" and "tactical" levels in the simulated highway lane-planning work described in [8], with the important differences that Boss operates in an urban environment, with more structure and restrictions than on a highway, and that the Lane Selector and Merge Planner must cope with the uncertainties of real sensor data.

The Lane Selector determines both the *Current Lane* that Boss is driving along, and the *Intended Lane* that Boss should merge into. In the nominal case, these two Subjects represent the same lane, indicating that no lane-change maneuver is necessary. The Lane Selector will choose a different lane if the Current Goal does not terminate in the Current Lane, or else when the robot's progress is being hindered or blocked by traffic in the Current Lane. If the robot's progress is being hindered, the Lane Selector will choose the Intended Lane according to the available passing lanes and the nature of the blockage. When a passing lane is available in the same direction of travel, it will be selected once the robot has been slowed to a configurable fraction of the current speed limit, allowing passing maneuvers at-speed. On roads with one lane in each direction, the oncoming traffic lane will be selected only after the robot has come to a complete stop and enough time has passed to determine that the obstruction must be circumvented. In both cases, several parameters, such as the time delay before initiating the passing maneuver, may be tuned to accommodate changes in the robot's ability to perceive other vehicles, further promoting incremental development.

When the Intended Lane differs from the Current Lane, the Merge Planner is responsible for monitoring the feasibility of, and eventually executing, the lane-change maneuver. Feasibility is determined as the ability to maintain proper spacing with surrounding vehicles while ensuring the system reaches the Intended Lane within constraints such as the length of the remaining road and the distance to the Current Goal. To illustrate the Merge Planner's functionality, consider the two-lane unidirectional case is depicted in Fig. 8. In this scenario, the Merge Planner has been commanded to merge into the right-hand lane in order to hit a checkpoint.

First, the merge is checked for kinematic feasibility, which is determined by comparing the distance required for Boss to complete a lane-change maneuver to the "merge-by" distance, which is the minimum of:

- The distance to the current goal, i.e. X_G;
- The remaining distance in the current lane;
- The distance to the closest blockage in the current lane;
- The projected future position of the closest vehicle, i.e. Veh_F, in the current lane.

Assuming that the merge is kinematically feasible, each Moving Obstacle in the merge-to lane is evaluated both for whether it can be overtaken for a front-merge within the merge-by distance and for whether it is possible to slow down and wait for a back-merge opportunity. Subsequently, the set of available merge "slots" are evaluated for feasibility, where n obstacles describe $n+1$ slots to consider for merging. For example, there are three slots in Fig. 8: in front of Veh_1, between Veh_1 and Veh_2, and behind Veh_2.

For the front and rear slots, feasibility is associated with a single obstacle and is thus directly determined by the foregoing. A "between" slot is deemed feasible if the front-obstacle back-merge and rear-obstacle front-merge are both

possible, and the gap between the obstacles is, and will remain, large enough to allow proper spacing.

The target slot is selected from the set of feasible slots according to the context of the maneuver. If both the Current and Intended Lanes are in the same direction, the foremost feasible slot is selected in order to make the best time. If the lanes are in opposing directions, such as when passing a disabled vehicle, the closest feasible slot is selected in order to remain in the wrong-direction lane for the shortest time.

The target slot is then evaluated for whether proper spacing has been met. For a front-merge in Fig. 8, this requires:

$$X_0 - l_{veh} - X_1 \geq max((V_1 * (l_{veh}/10)), gap_{min}) \quad (4)$$

If proper spacing has been met, one last check is made to make sure the rear vehicle's velocity can be matched after the merge without violating proper spacing. If so, the merge is commanded by propagating the Intended Lane as the *Commanded Lane*. Otherwise, if proper spacing has not been met or cannot be maintained, the Commanded Lane is held as the Current Lane, and another output Subject, the *Merge Speed* is set to indicate whether the robot should speed up or slow down to synchronize with the target slot. This process repeats on each update to the Moving Obstacle Set until the merge is completed or there are no more feasible merge slots, whereupon the Merge Speed is set to the road's minimum speed and it is left to the Goal Selector to choose an alternate path forward.

The outputs of the Distance Keeper and the Merge Planner are monitored by the last Observer in the Lane Selection group, the Vehicle Driver. This Observer applies a set of rules that arbitrates between the two sets of outputs, imposes safety constraints, and forwards the appropriate values to the Motion Parameters message output. This separation of concerns allows each behavior to be developed in isolation, further promoting incremental development within the Behavioral Executive.

VII. Secondary Functionality

Beyond the core functionality, the flexibility of the Behavioral Executive's design allowed for the incorporation of several secondary functions, leveraging the power of the Observer Pattern to introduce them without interfering with core functionality. Two of the most illustrative examples are the Diagnostic Interface and the Panhead Planner.

First, the maintenance of all intermediate data products as Subjects allows the Diagnostic Interface to trivially observe and export the Behavioral Executive's internal state as a Diagnostic Status message, fulfilling the requirement for instrumentability as a side effect of the design. Conversely, the Diagnostic Interface also supports the run-time modification of Subjects, via a Diagnostic Command message input, to interactively enable or disable new or experimental functionality while running live tests, allowing the real-time comparison of alternate approaches to a given problem.

Second, and late in the development process, the Behavioral Executive acquired the responsibility of managing two sets of panning long-range laser and radar sensors. Although sensor coverage optimization was far outside the scope of the original requirements, the design accommodated the functionality easily in a separate Observer, the Panhead Planner. It uses the Occupancy Polygons from the Precedence Estimator to point the panheads in directions most likely to include moving traffic that would not be detected by the vehicle's statically-configured sensors, such as at a T-intersection, providing more reliable long-range detection of fast-moving vehicles and allowing the system to make safer yield decisions.

VIII. Analysis and Conclusions

Despite the Behavioral Executive's ability to accommodate these extra responsibilities, the design still exhibited one of the primary difficulties of the Observer Pattern[5]. As the competition approached and various shortcuts were taken to try to fix problems, the inability to fully understand the complex notification patterns among the Observers led to many unexpected and erroneous results. For instance, the accidental introduction of a circular dependency among the Observers in the system inadvertently caused an "idle" goal to be spuriously emitted by the Behavioral Executive, which in turn caused the robot to slam on the brakes for no apparent reason. Debugging these problems required consideration of the Behavioral Executive as a whole, defying the decoupling and encapsulation that the design strove to achieve.

Nevertheless, the design of the Behavioral Executive proved well suited to its purpose, providing a solid framework for growing incremental capabilities and fertile ground for the testing of new and different ideas. The isolation of functionality in individual Observers allowed the Behavioral Executive to remain highly adaptive to Boss's evolving capabilities. When combined with behavioral algorithms that can be tuned to exploit strengths and accommodate shortfalls in the rest of the system, this yielded a reliable, competent subsystem for managing the traffic-interactive behaviors that characterized the Urban Challenge.

References

[1] Christopher R Baker, David I. Ferguson, and John M. Dolan. Robust mission execution for autonomous urban driving. In *International Conference on Intelligent Autonomous Systems*, 2008.

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

[3] Chris Urmson, et al. Autonomous Driving in Urban Environments: Boss and the DARPA Urban Challenge. *Accepted to Journal of Field Robotics*, 2008.

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

[5] 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.

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

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

[8] Jun Miura, Motokuni Ito, and Yoshiaki Shira. A three-level control architecture for autonomous vehicle driving in a dynamic and uncertain traffic environment. In *ITS*, pages 706–711, Boston, MA, 1997.

[9] Mary Shaw and David Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[10] Rahul Sukthankar, Shumeet Baluja, and John Hancock. Multiple adaptive agents for tactical driving, 2002.