Title
An Interface Between Continuous And Discrete-event Controllers For Vehicle Automation

Authors
Lygeros, John
Godbole, Datta N.

Publication Date
1994

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

An Interface Between Continuous and Discrete-Event Controllers for Vehicle Automation

John Lygeros
Datta Godbole

California PATH Research Report
UCB-ITS-PRR-94-12

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department of Transportation, Federal Highway Administration.

April 1994

ISSN 1055-1425

An Interface between Continuous & Discrete-Event Controllers for Vehicle Automation *

John Lygeros and Datta N. Godbole

Intelligent Machines and Robotics Laboratory
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720
lygeros@robotics.eecs.berkeley.edu
godbole@robotics.eecs.berkeley.edu

Abstract

The work presented here is part of a bigger effort to design an automated highway system to improve the capacity and safety of the current highways. Automation of highways and in particular platooning of vehicles raises a number of control issues. In the design proposed in [1] these issues are addressed by a hierarchical structure consisting of both discrete event and continuous time controllers. The work presented here is an attempt to construct a consistent interface between these two types of controllers. The design proposed is in the form of a set of finite state machines that interact with the discrete controllers through discrete commands and flags and with the continuous controllers by issuing commands that get translated to inputs for the vehicle actuators. The operation of the proposed design is verified using COSPAN and tested in simulation. By virtue of the fact that the interface touches on both the discrete and continuous worlds, the design might provide insight to interesting problems related to the hybrid nature of the system.

1 Introduction

The work presented here was carried out with the particular Automated Highway System (AHS) structure of [1, 2, 3] in mind. In this context, it is assumed that traffic on the highway is organized in platoons of tightly spaced vehicles. This is done in an attempt to maximize the capacity and the throughput of the highway, while avoiding exposing the passengers to additional risk. Theoretical studies indicate that the capacity increase if such a scheme is

*Research supported by the PATH program, Institute of Transportation Studies, University of California, Berkeley, under MOU 100

implemented successfully will be substantial. Moreover, this will be done without a negative impact on passenger safety as, by having all the vehicles of a platoon follow each other with a small intra-platoon separation (about 1 meter), then, if there is a failure and an impact is unavoidable, the relative speed of the vehicles involved will be small (hence the damage to the vehicles and the injuries to the passengers will be minimized). The inter-platoon separation, on the other hand, is large (of the order of 30 meters) to physically isolate the platoons from each other, so that the probability of a collision is minimized and decelerations are attenuated as they propagate up the highway.

Clearly the realization of such a scheme requires automatic control of the vehicles. The control of such a large scale system poses a formidable problem. There needs to be some form of Autonomous Intelligent Cruise Control (AICC) law to maintain a safe separation between platoons, a different control law to keep the platoon tightly spaced and specialized control laws to carry out various maneuvers (such as forming and breaking up platoons and moving vehicles from one lane to the next). Moreover, there needs to be some coordination between these laws to ensure that the operation of the system is safe and efficient. Finally, the scheme requires a controller that monitors the conditions of the entire highway and decides on a long term strategy aimed at maximizing capacity.

The control structure suggested in [1] consists of four layers (Figure 1). The top layer, called the **network layer**, is responsible for the flow of traffic on the entire highway system. Its task is to prevent congestion, maximize throughput and minimize travel times by dynamic routing of traffic.

The second layer, called the **link layer**, coordinates the operation of whole sections (links) of the highway. Its primary concern is to maximize throughput while maintaining safe conditions of operation. With these criteria in mind, it calculates an optimum platoon size and an optimum velocity for each highway section. It also decides which lanes the vehicles should follow to get to their destinations as fast as possible. Finally, it monitors incidents on the highway and diverts traffic in order to minimize the impact of the incident on traffic flow and safety. Because the link layer bases its control actions on large numbers of vehicles, it inevitably has to use some form of aggregate information. Therefore it treats the vehicles in a section statistically rather than considering the state of individual vehicles or platoons. Likewise, the commands it issues are not addressed to individual vehicles but rather to all the vehicles in the section as a whole; a typical command would be "30% of the vehicles who wish to get off the highway at the next exit change lane now" or "all platoons in this section should try to be 10 vehicles long". The design proposed in [4] makes use of flow equations to model the traffic in the given section.

The next level of hierarchy is the **coordination layer**. Its task is to coordinate the operation of platoons with their neighbors. It receives the link layer commands and translates them to specific maneuvers that the platoons need to carry out. For example, the coordination layer will ask two platoons to merge to a single platoon whose size is closer to the optimum or, given a command like "30% of the vehicles going to the next exit change lane now", it will decide which vehicles will be in this 30% and split the platoons accordingly. The design proposed in [5] uses protocols, in the form of finite state machines, to organize the maneuvers in a systematic way. They receive the commands of the link layer and aggregated sensor information from the individual vehicles (of the form "there is a vehicle in the adjacent

lane"). They then use this information to decide on a control policy and issue commands to the lower layer. The commands are typically of the form "accelerate to merge to the preceding platoon" or "decelerate to let another vehicle move into your lane ahead of you".

Below the coordination layer in the control hierarchy lies the **regulation layer**. Its task is to receive the coordination layer commands and translate them to throttle, steering and braking input for the actuators on the vehicle. For this purpose it utilizes a number of continuous time feedback control laws ([6, 7, 8, 9]) that use the readings provided by the sensors to calculate the actuator inputs required for a particular maneuver. The regulation layer occasionally needs to communicate with the coordination layer to inform it of the outcome of a maneuver.

There is one more layer in the system which is not part of the control architecture. It is the **physical layer**, i.e., the actual vehicle. For the purposes of the control design it is modeled as a set of differential equations and transfer functions that translate the actuator inputs (provided in this case by the regulation layer) to the state of the vehicle (position, velocity and acceleration). The physical layer also includes the sensors that provide sampled information about the state to be used by the control algorithms.

The work presented here focuses on the interface between the regulation and coordination layers. [5] describes how the coordination layer protocols were designed and tested. A great deal of work has also been done on continuous time control algorithms that the regulation layer uses to carry out the various maneuvers ([6, 7, 8, 9]). Between these two areas of development, there is still, however, a gap. As discussed above, the commands of the coordination layer typically are of the form "accelerate to merge with the preceding platoon". The continuous time control laws are unable to directly interpret these commands and introduce them in their actuator input calculations. Similarly, the coordination layer needs some way of interpreting the sensor readings and the state of the continuous time controllers in a form that it can understand. In other words, there is a need for an interpreter because the coordination layer (discrete event system) and the regulation layer (continuous time system) speak in different languages. We seek to fill this gap by providing an interface that allows this communication to take place.

The interface proposed here is in the form of a discrete event system (DES). It has a finite number of states (finite state machine) representing commands directed towards either the regulation layer (e.g., invoke a specific controller) or towards the coordination layer (e.g., notify the appropriate protocol that the requested maneuver was completed). The DES will transition from one state to another depending on the commands from the coordination layer, the readings of the sensors and the state of the continuous time controllers. The design will be arranged so that there is never a conflict or a deadlock, and the coordination layer commands are followed whenever this is possible.

The paper is arranged in three main parts. In the first part, the framework into which our design fits is outlined. We briefly describe existing work on the regulation and coordination layers and provide references that contain more details. This outline will motivate our work, which is presented in Sections 3 and 4. In Section 3, the assumptions we make about the interaction of the interface with the rest of the system are presented and the tasks that the design will be expected to perform are specified in detail. In Section 4, the formal specification of the proposed design is given and it is verified that the required tasks are indeed performed.

The verification is done automatically by means of COSPAN, a program that verifies whether all event sequences (runs) that can be generated by a collection of interacting finite state machines satisfy the specified properties. In the closing section, directions for extending this work are outlined.

2 Coordination & Regulation Layer Design

2.1 Coordination Layer

As discussed in the introduction, the task of the coordination layer is to systematically organize the traffic in platoons. It is assumed that the vehicles are equipped with communication devices that allow them to exchange messages and coordinate maneuvers in order to form and break up platoons and move vehicles between the lanes. The coordination layer design proposed in [5] uses only three such maneuvers: merge, split and lane change. The reason behind the small number of maneuvers is to keep the design as simple as possible. To further simplify the problem it is assumed that only leaders or free agents can initiate maneuvers; the followers can request their leader to initiate a maneuver for them[1]. Finally the current design requires that each platoon will be involved in at most one maneuver at a time.

We will now briefly describe the actions involved in each one of the coordination layer maneuvers of [5]. The **merge** maneuver is used to join two platoons in the same lane and form a single platoon. The following platoon requests the leading platoon permission to merge. The permission is granted, if the leading platoon is not engaged in another maneuver and if the size of the resulting platoon will not exceed the upper limit set by link layer. After an agreement is reached, the coordination layer of the following platoon orders its regulation layer to accelerate to catch up with the leading platoon. At the end of the maneuver, the following platoon becomes part of the leading platoon. The **split** maneuver does exactly the opposite. It is used to break the platoon into two smaller platoons. The trailing platoon decelerates after break up to create safe inter-platoon separation from the parent platoon. Finally the **change lane** maneuver is used to move vehicles from one lane to another. For simplicity, it is required that only free agents may change lanes. Apart from the obvious lateral movement, lane change may also involve longitudinal movement. The possible longitudinal actions that may be required for changing lanes are summarized in Figure 2. Free agent A wants to change to the lane where platoon B is moving. It is assumed, for purposes of safety, that A can move over only if its speed is close to that of B and their spacing is close to some safety distance (*dsafe*). There are three scenarios that allow A to move over in safety: A has to decelerate and move in behind B, B has to decelerate and let A in ahead of it, or B has to split and let A enter in the middle. One of the three alternatives is chosen, depending on the size of B and the position of A relative to B. Overall, the lane change maneuver consists of two steps. In the first step, labeled *Decel_to_Change*, the vehicles adjust their longitudinal positions so as to align the vehicle changing lane with a gap. In the second step, called *Move*, the lateral action of the lane change is carried out.

[1] In the context of platooning each vehicle will be either a leader (first vehicle of the platoon), a follower or a free agent (single vehicle platoon).

The coordination required in order to carry out these maneuvers in safety was specified in [5] by a structured set of communication messages, in the form of protocols. Modeling these protocols by interacting finite state machines the logical correctness of the design was verified using COSPAN.

2.2 Regulation Layer

The regulation layer consists of a number of feedback control laws that make use of sensory information to produce throttle, brake and steering inputs for the vehicle actuators. The current design is based on a continuous time, ordinary differential equation model of the vehicle dynamics. The control laws are designed to take into account vehicle capabilities and passenger comfort standards. The different control laws needed for normal operation of the regulation layer are summarized below:

Lead Control: The primary goal of the lead control law is to maintain safe spacing between platoons. In the design of [9] the safe spacing is calculated according to the formula:

$$D = \lambda_a \ddot{x} + \lambda_v \dot{x} + \lambda_p$$

where $\dot{x}$ and $\ddot{x}$ denote the velocity and acceleration of the platoon. For normal operation, the values $\lambda_a = 0$, $\lambda_v = 1\text{sec}$, $\lambda_p = 10\text{m}$ are currently used. Provided that the primary task is carried out without a problem, the controller also tries to perform a secondary task, namely to track the optimum velocity calculated by the link layer as closely as possible.

Follower: The follower control law has a single objective: it tries to match the velocity and acceleration of the preceding vehicle in the platoon, while staying close (1 meter) behind it. It also has the advantage that the vehicles within a platoon are connected via an infrared communication link, so they have access to information about their neighbors other than that provided by their sensors. [6, 8] provide details of possible designs of the follower control law.

Merge: The merge control law is expected to take two vehicles (or platoons) with an initial spacing d_o (typically 30 meters) and a initial velocity mismatch δv_o (typically a few meters per second) to a final spacing equal to the intra-platoon spacing (typically 1 meter) and zero velocity mismatch. The whole maneuver should be carried out as fast as possible but without pushing the engine or the brakes to their limits (thus compromising safety) and without affecting passenger comfort. A continuous time feedback controller that fulfills the above requirements was designed and is presented in [9]. It is based on the calculation of a desired trajectory at the beginning of the maneuver. Feedback from the sensors is then used to keep the actual vehicle trajectory as close as possible to the desired one.

Split: The split controller is expected to take a pair of vehicles, initially at intra-platoon spacing and zero velocity mismatch (assuming that the follower law operation is close to perfect), to inter-platoon safe spacing and zero velocity mismatch. The design is very similar to the one for the merge maneuver: a trajectory that carries out the desired task

and does not violate any limits is calculated and then feedback is added to guarantee tracking [9].

Decel_to_Change: A controller capable of carrying out the longitudinal actions expected by Decel_to_Change (refer to Figure 2) is presented in [9]. The general principle is again very similar to that of the merge maneuver.

Move: The move control law involves lateral motion. Again the design should be such that the required input does not force the actuators close to their limits or makes the passengers uncomfortable. A design satisfying these requirements is presented in [10].

Lane Follow: The lateral control law that maintains lane position is required to keep the vehicle at the center of the lane. The current design uses magnets placed at regular intervals along the center of the lane and magnetometers mounted on the vehicle to obtain deviations from the center of the lane. A frequency shaped LQ optimal controller is designed in [7] to achieve the lane keeping objective.

3 Interface Design Assumptions & Requirements

The interface structure and its interactions with the surrounding controllers are outlined in more detail in Figure 3. In this figure the entry marked "Regulation Layer" in Figure 1 is expanded (between the dotted lines) to reveal the details of the internal structure. In the center of the regulation layer lies the "interface", which is the main subject of this paper. It communicates with the coordination layer through two channels, one for receiving requests and one for sending out responses. The interaction is facilitated by the presence of two buffers that can be used to store the commands and responses. The interface also has to interact with the continuous time controllers that are used to calculate the actuator inputs (in this case assumed to be steering, throttle and brake). Note that the regulation layer contains a number of different control algorithms (in the figure they are indicated by the slots under the collective name *control input calculation*), each designed to carry out a specific maneuver. Therefore it is important that the interface keeps track of the controller it has to invoke in a systematic manner. Finally, the interface has to make certain assumptions about the physical layer. It also has to interact with it directly through the sensor information that is needed to make decisions. In this section we will lay out the assumptions we make and the specifications we set for all these interactions of the interface.

3.1 Interaction with Coordination Layer

Our primary goal is a design that will allow the coordination and regulation layers to operate asynchronously and at different time scales. This is achieved in this case by the use of the *reg_response* and *reg_request* communication lines and the command and flag buffers. The coordination layer decides what maneuver needs to be carried out and stores the appropriate command in the command buffer. It then notifies the regulation layer through the *reg_request* line. Whenever the regulation layer is ready, the interface reads the command from the command buffer and invokes the appropriate control law to carry out the maneuver. When

the maneuver is completed, the interface stores a flag that signifies success in the flag buffer and notifies the coordination layer through the *reg_response* line. If the maneuver was aborted (because it was unsafe to proceed with it), the flag signifying failure is stored in the buffer and *reg_response* is used to notify the coordination layer. Whenever the coordination layer is ready, it reads the flag from the buffer, updates its state accordingly and decides on the next action.

This arrangement gives a lot of flexibility to the interface. While the coordination layer is waiting for the *reg_response*, it can carry out other tasks (e.g., plan its next move). Likewise, the regulation layer can operate autonomously without having to synchronize with the coordination layer; in fact, communication is necessary only when a new maneuver is requested. In between requests the regulation layer can go about its business as if the coordination layer is not there. It is assumed that in an actual implementation of the control scheme, *reg_request* and *reg_response* will use interrupt lines. In the current implementation within the framework of the SmartPath simulator [11], the communication channels are modeled by "events" in the C-Sim programming language, which can be thought of as a form of software interrupts. To simplify the figures the abbreviation *nr* will be used to indicate that the coordination layer has no request or the regulation layer has no response (the interpretation will be clear from the context).

3.1.1 Commands

The commands stored in the command buffer reflect the maneuvers that a vehicle may be requested to carry out under the platooning scenario.

Accel_to_Merge: Asks the vehicle to join the preceding platoon.

Decel_to_Change: Asks for a deceleration so that vehicles in adjacent lanes end up in a relative position from which a lane change can take place safely. Figure 2 shows three possible scenarios for lane change. Which of the three alternatives is chosen is decided by the coordination layer; the regulation layer simply carries out the chosen maneuver. If scenarios 1 or 2 are chosen the coordination issues a Decel_to_Change command to the appropriate regulation layer (A or B respectively). If scenario 3 is chosen the command Split_Change (see below) is issued to the regulation layer of the appropriate vehicle in platoon B.

Move: Asks the regulation layer to move the vehicle to the adjacent lane.

Split_Free: Splits the platoon so that a car can become a free agent.

Split_Change: Creates a split so that a vehicle from an adjacent lane can change lane in the middle of the platoon, as in the scenario 3 of Figure 2. The maneuver is almost identical to the one of Split_Free, the only difference being that the final separation of the vehicles is twice as much for the case of Split_Change.

The first two commands can be issued only when the vehicle is either the leader of a platoon or a free agent. The third can be issued only when it is a free agent. Finally, the last two can be issued only when the vehicle is a follower in a platoon. The interface expects the coordination layer to keep track of these facts. In the figures the command names

will be abbreviated to keep the notation simple: Accel_to_Merge will be denoted by *mrg*, Decel_to_Change by *ch*, Split_Free and Split_Change by *sp* and Move by *mv*. The abbreviation *nc* will be used when the coordination layer does not command any special maneuver. In this case the regulation layer will execute the default control law (either leader or follower).

3.1.2 Flags

For the communication from the interface to the coordination layer, two flags are used.

Succ: is issued if the requested maneuver was completed successfully.

Not_Succ: is issued if the maneuver had to be aborted to avoid some hazardous situation.

3.2 Interaction with Continuous Time Controllers

As discussed in Section 2, the interface has a number of continuous time control laws at its disposal, which it can invoke to perform the various maneuvers requested by the coordination layer. From the interface point of view, the details of these control laws are irrelevant; we only need to consider them from an input-output point of view. These laws use the sensory information to calculate the engine input over short time intervals. For the SmartPath simulator implementation, this interval is taken as 0.1 seconds, a value dictated by the sampling frequency of the sensors. At the end of the interval, the continuous time law returns the control to the interface which checks whether a new *reg_request* has occurred, whether the current maneuver has completed or not and, if not, whether it is still safe to go ahead with it. Depending on the outcome of these checks the interface then selects another continuous time control law and the process is repeated. It should be noted here that the lead control law is the most natural as it is similar to the control that human drivers carry out most of the time. It is also more robust, in the sense that it can tolerate larger spacing and velocity errors and does not depend on communication between vehicles (as the follower law does). Therefore, it is invoked by the interface as a default, i.e. whenever a maneuver is aborted, in the case of a communication breakdown, etc.

3.2.1 Initialization

Every time a maneuver is requested by the coordination layer, the interface must make sure that the appropriate control law is ready to respond before invoking it. For this reason, the interface should first carry out some form of initialization. For the control algorithms presented in [9], the initialization involves:

- Setting the state of the controller to the right initial condition (for controllers which are dynamic, such as the lead controller).
- Updating parameter values that might have changed since the controller was last invoked (e.g., the optimum velocity).
- Calculating the desired trajectories used by the merge, split, decelerate to allow lane change and move to adjacent lane maneuvers.

3.2.2 Safety Checks

Before turning over the control to the continuous time laws, the interface must make sure that the requested action can be carried out safely. For this reason, it performs certain safety checks. The checks should be repeated whenever new sensor data comes in. The details of the safety checks depend on the maneuver in question and the control law implementation. They are grouped in five classes:

- General safety checks that will alert the system if a malfunction occurs (e.g., communication device failure, engine breakdown or tire burst). Formalizing such safety checks can be difficult, as the number of possible malfunctions is large and the way they affect the system is diverse. In [12] and extensive list of malfunctions is presented. A predicate hierarchy is introduced to model the system capability[2]. The levels of the predicate hierarchy reflect the levels of the control hierarchy and the values of the higher level predicates depend on those of the lower level ones. System malfunctions cause certain physical layer predicates to return 0. This may cause some regulation layer predicates to return zero, which in turn may cause some coordination layer predicates to return zero and so on. The malfunction safety check at the interface involves checking the predicate of a control law (such as the merge law) before invoking it. If the predicate returns zero the control law is incapacitated because of some malfunction and the interface has to abort the maneuver and select a different law. In [13] an extension of the control architecture is proposed to guarantee that at least one control law is operational in any situation.

- Safety checks that deal with the constraints imposed upon the state of the vehicles by road conditions, engine capabilities and the need for passenger comfort. These factors impose bounds on the acceleration and the jerk produced by the engine and the brakes; typically the acceleration has to lie in the range $[-5, 3]ms^{-2}$ while the jerk in the range $[-5, 5]ms^{-3}$, but the bounds may be even tighter in adverse conditions (e.g., rain). As discussed above, the trajectories designed for the various maneuvers are chosen so that they lie well within these bounds. However, in certain cases it may be possible for the actual trajectories to come close to the bounds. For example, the leading platoon in a merge maneuver may start accelerating half way through the maneuver. This extra acceleration will be reflected on the merging vehicle by the action of the feedback law and, when added to the acceleration normally required by the merge trajectory, may cause the state of the trailing vehicle to come dangerously close to the bounds. A safety check is therefore introduced to abort a maneuver when situations like this are encountered, so that the trajectory can be redesigned and the maneuver reinitialized. Clearly such a safety check is only applicable to maneuvers requested by the coordination layer. There is no way of aborting lead control, for example, even if it causes the states to go close to the bounds.

- Safety checks involving the detection of new vehicles in the vicinity. If, while a merge or a split is taking place, a vehicle moves into the lane, between the two vehicles involved in the maneuver, the maneuver must be aborted and the lead control invoked to bring

[2]Predicates are functions that return 1 if the system possesses a certain capability and 0 if it does not.

the vehicles to a safe position. Similarly, if, while a move to an adjacent lane is taking place, another vehicle moves into the target position, the move must be aborted. The presence of these intruding vehicles is detected by comparing the current sensor readings to the values that are expected from the previous readings. If the difference is found to be too large (more than the length of an average vehicle for example) the check fails. It should be noted that situations like these are unlikely in a fully automated highway, provided that the coordination layer is well designed. We introduce these checks, however, to deal with the cases of semi-automated highways and malfunctions that may cause unpredictable changes.

- Safety check for the move maneuver. The move maneuver results in different longitudinal neighbors (i.e., the front and rear vehicles) after the vehicle changes lane. The safety checks are designed to ensure collision free operation during the move maneuver and after its completion. The interface initiates the move maneuver only if a safe inter-platoon spacing exists on either side of the vehicle's desired position in its target lane. The execution of the move maneuver takes a finite amount of time (of the order of 5-10 sec). During this time, the vehicles in the target lane may not be able to maintain the required spacing due to the traffic condition downstream. The safety checks make sure that the gap exists throughout the maneuver. Given the bounds on the capabilities of the vehicles, we calculate the region of the state space from which the lead controller [9] can *safely* take the state of the vehicle to the desired inter-platoon spacing. The move maneuver is aborted if the state of the vehicle changing lane goes outside the safe region for the lead controller (with respect to the preceding vehicle in the target lane) or the state of the trailing vehicle in the target lane goes into the unsafe region of its lead controller (with respect to the vehicle changing lane). If the move maneuver is aborted, the vehicle returns to its lane of origin.

- Safety check for the Decel_to_Change maneuver. This check is carried out only if the vehicle that is decelerating detects another vehicle ahead of it, in its own lane. In this case the interface calculates the time that will elapse before the preceding vehicle comes dangerously close (inside the safety region discussed above), assuming that the velocity of both vehicles will remain constant. If this time is less than the time required to carry out the maneuver plus the time required to move from one lane to the next the maneuver is aborted. In the following section this abort will be referred to as *abort_safe*.

It should be noted that, with the exception of the malfunction check, all safety checks are essentially hybrid, as they involve extracting discrete information (safe vs. unsafe) from continuous data (like the positions of adjacent vehicles or the acceleration of the vehicle in question).

3.3 Interaction with Physical Layer

The "Physical Layer" represents the vehicle itself. For the AHS scenario considered here, it is assumed that all vehicles will be equipped with communication devices, sensors (that monitor the state of the vehicle and its position relative to neighboring vehicles) and actuators (to apply throttle, steering and brake inputs).

The communication capabilities are only used at the coordination layer or higher. The sensors are assumed to operate perfectly (there is no fault detection in our design so far) and provide samples of the states at fixed intervals. Finally the engine and brake inputs act on the third derivative of position ("jerk") [9] and are applied to the engine directly from the controllers (without the intervention of the interface). The steering input affects the second derivative of the lateral position and orientation of the vehicle [7]. It is also applied directly to the actuators by the relevant control laws.

For the purpose of simulations, the vehicle dynamics were approximated by a 7^{th} order continuous time model. Three of the states (position, velocity and acceleration) are related to the longitudinal dynamics and are affected by throttle and brake inputs while the remaining four (lateral position, lateral velocity, orientation and angular velocity) are related to the lateral dynamics and are affected by the steering input. The equations were integrated using a 4^{th} order, variable step, Runge-Kutta algorithm. The sampling time for the sensors is taken to be 0.1 seconds while that of the actuators is 5 milliseconds. A zero order hold is used to interpolate between actuator samples.

4 Formal Specification & Verification

An interface that meets all the above specifications was designed in the form of a number of interacting finite state machines (FSM). The advantages of this format are many: it is easy to translate to code (in C or other programming languages), it is possible to verify automatically and it provides a direct way of communicating with the coordination layer which is in FSM form in the current design.

In the subsequent discussion five such machines will be presented. *INTERFACE* will be the central machine; it will carry out all the tasks specified above. It will cooperate with *FLAG*, a machine that keeps track of the flag that will be passed to the coordination layer, *COMMAND*, which keeps track of the maneuver requested by the coordination layer, *RES*, which keeps track of the *reg_response* communication channel and *REQ*, which keeps track of the *reg_request* channel. A sixth machine, *COORD* will be introduced for the purpose of automatic verification. Its role is to mimic the operation of the coordination layer, from the regulation layer point of view. For all these machines we use the following convention: to each state we associate one or more "outputs". Each time the machine lands in a given state it has to select one of the outputs associated with that state. The outputs are the only things that the other machines have access to and they are denoted by lower case letters (whereas the states are denoted by upper case). The transitions between the states of a machine depend only on the outputs - either those of the machine itself or those of the other machines. Our design is deterministic in the sense that a single transition is enabled for every possible set of outputs. To avoid confusion, the name of the machine is added before the name of the output when labeling transitions. For example *FLAG: Not_Succ* means that the output of the machine *FLAG* is *Not_Succ*. This convention helps keep figures tractable and simplifies the task of coding the machines in the *Selection/Resolution* format that COSPAN, the verification language, accepts as input.

4.1 Finite State Machine for the Interface

A rough outline of the FSM structure for the interface is given in Figure 4. The two modes of operation, leader and follower are centered about the two *Read Command* states. In these states the interface checks the command buffer and selects the appropriate maneuver. Transition from leader mode (*Read Command 1*) to follower mode (*Read Command 2*) is effected by a successful merge maneuver. If the maneuver is interrupted (by a new command or by an abort) the leader mode is reestablished. Transitions in the other direction (from the follower mode to the leader mode) are effected by some form of split maneuver (Split_Free or Split_Change). The difference here is that even if the maneuver is interrupted half way through, the leader mode of operation is established. Clearly, the maneuvers that involve deceleration to allow a lane change and the moving of the vehicle to the adjacent lane do not affect the mode of operation; the vehicle is in leader mode both before and after the maneuver, whatever the outcome. We now present the detailed structure of the part of the interface machines used for each maneuver.

Leader: Has the simplest structure (see Figure 5). If no request comes in, the interface resorts to the default AICC law. Some initialization takes place and then the control inputs to be applied to the vehicle actuators over the next 0.1 seconds are calculated. Then the interface checks if a maneuver request came in. If yes, it returns to *Read Command 1* to initiate the requested maneuver. If no, the control input calculation is resumed (without initialization). *reg_response* is never issued by this part of the protocol.

Follower: The overall structure (Figure 6) is very similar to that of the leader. Again *reg_response* is never used.[3]

Merge: The protocol is shown in Figure 7. Whenever the command merge (*mrg*) is read from the buffer the maneuver is initialized and safety checks are carried out. If there is some problem, the maneuver is aborted and the lead control takes over to bring the vehicle to safety. If it is safe to proceed, the continuous time merge control law is invoked to calculate the engine input. After 0.1 seconds, the interface checks for a new *reg_request*. If there is one, it goes back to read the new command. If not it checks if the maneuver is complete and either returns to the safety check to continue or goes into follower mode accordingly. A *reg_response* is issued during the transitions labeled *abort* and *complete*. The flag passed is *Not_Succ* and *Succ* respectively.

Move: The sequence of events is exactly the same as for the merge maneuver (Figure 8). The only difference is that the maneuver both starts and finishes in the lead (*Read Command 1*) mode.

Split: The protocol used for the Split_Free and Split_Change maneuvers is shown in Figure 9. The same sequence of events is used in both cases. The only difference is that Split_Change completes when the vehicles have reached twice the distance required by Split_Free. This difference is taken care of in the initialization step, where the trajectory that will be tracked

[3] In a future version, safety checks may be added to the lead and follow parts of the interface. Their role will be to notify the coordination layer about malfunctions (tire bursts, communication breakdowns, etc.) and to invoke emergency control laws.

is calculated. The sequence of events involved in splitting is very similar to the sequence for merge, with the obvious difference that the vehicle starts in the follower mode and ends up in the leader mode. A more subtle difference is that the flag issued is always *Succ*. It is assumed that the coordination layer will not initiate a split unless it is safe to do so. If a hazard emerges half way through, the maneuver is aborted and lead control is invoked to take the vehicle to safety. Completion is signaled to the coordination layer, however, as the vehicle is no longer part of the original platoon.

Decelerate for Lane Change: This is by far the most complicated maneuver. The reason is mainly that it involves vehicles in two lanes. The event sequence involved is shown in Figure 10. We will refrain from detailed discussion of the basic steps (initialization etc). There are two main loops for this maneuver. The top loop (safety check - control calculation - check for interrupts - done - initialize lead) is very similar to the ones encountered in the previous maneuvers. It is used during normal deceleration. It can be exited by a new *reg_request* (in the state *check for requests*), by an *abort* (reflecting a major safety hazard such as a breakdown) or by an *abort_safe*. This last option reflects the fact that the vehicle preceding the one that is decelerating can end up in a position that may be dangerous if the maneuver continues. It leads to the lower loop where the vehicle decelerates to safety under the lead control. This loop is exited if the safety problem is resolved (in which case the upper loop is reinitiated), if the vehicle finds itself in a position where the lane change can be carried out safely (in which case the maneuver is declared complete) or if the interface decides that it is impossible to complete the maneuver (in which case an abort is issued). The *reg_response* is set by the transitions labeled *complete* and *abort*. The flag passed is *Succ* and *Not_Succ* respectively.

4.2 Supporting Finite State Machines

The finite state machine formalism was also used to implement the remaining parts of the design: the flag and command buffers and the two communication channels. These machines can be viewed as monitors that observe the transitions of the two major machines (the coordination layer and the interface) and change their own state accordingly.

The flag buffer is a simple two state machine (Figure 11). It indicates *Succ* and *Not_Succ* by being in state S and NS respectively. It transitions to NS whenever a maneuver is aborted, unless this maneuver is a split. It transitions to S whenever a maneuver is completed, or if a split is aborted. Clearly the state of this machine is only of importance when a *reg_response* message "awakens" the coordination layer.

The command buffer is a five state machine (Figure 12). Similarly to the flag buffer its state is of importance only when a *reg_request* message is passed to the regulation layer. Its states reflect the maneuver that should be carried out. Its transitions are governed by the output of the coordination layer.

The *reg_request* machine (Figure 13) has two states: R and NR indicating whether there is an incoming request or not. The machine transitions from NR to R whenever the coordination layer output indicates that a new maneuver is needed. It transitions from R to NR whenever the request is read by the interface and the command starts being serviced (*INTERFACE: read*).

Finally the *reg_response* machine (Figure 14) also has two states, R and NR, indi-

cating whether the regulation layer has something to tell the coordination layer or not. It transitions from *NR* to *R* whenever there is a need to notify the coordination layer and back when the coordination layer has taken note of the message (*COORD: read*).

4.3 Automatic Verification

The design described above was verified automatically using COSPAN [14]. COSPAN is a verification tool that works by symbolically analyzing a given set of FSM to make sure that their performance satisfies certain requirements specified by the user. It should be noted that symbolic testing is different from simulation or execution of the system; it is an automated mathematical proof that the system fulfills the requirements.

The machines for the interface, the buffers and the communication channels described in the previous section were translated to code in the Selection/Resolution (*S/R*) FSM model used by COSPAN. An additional state machine that plays the role of the coordination layer (Figure 15) was used to create the inputs. The transitions that are not uniquely determined by the current state of the machines (that is the commands of the coordination layer and the sensor inputs that determine whether a maneuver is complete or has to be aborted) are "selected" by the verification algorithm to take on all possible values, thus recreating all the runs that the FSM may produce. Monitors were used to test if our design satisfies the following properties:

1. The interface looks for a new request exactly once in each 0.1 second interval. This is to make sure that there can be no loop where the regulation layer is stuck to initiating maneuvers without carrying out any control. The same monitor also makes sure that regulation layer does not ignore the coordination layer commands.

2. The coordination and regulation layer are in the same mode, that is, follower commands (e.g., split) are not issued while the vehicle is a leader and vice versa.

3. The flag returned by the interface whenever a split maneuver is requested is always *Succ*. This guarantees that the requirement that the split maneuver is never aborted is indeed satisfied.

4. The interface carries out exactly one safety check in each 0.1 second interval, except when the vehicle is a leader or a follower in which case it carries out no safety checks at all. This guarantees that the latest sensor data is always used for the safety checks.

COSPAN verified that our design indeed performs all the above tasks.

5 Concluding Remarks

The automatic verification described above suggests that the design proposed here will perform well under the assumed conditions. As a further test the interface was implemented in C together with the continuous time control laws described in the references. It was then introduced in the SmartPath simulation platform [11]. For the purpose of SmartPath, the

reg_request and *reg_response* interrupts were modeled by software "events" (in the C-Sim programming language) and parameters (commands and flags) were passed via global variables. The complete design is currently being tested in this framework by simulating various scenarios that reflect actual highway conditions. The results indicate good performance in most traffic situations. Moreover, they highlight the problems that may be encountered when dealing with multilayered, hybrid control systems like this. These issues are discussed further in [15].

It should be noted that the automatic verification described above depends on the underlying assumption that the coordination layer behaves like the abstraction of Figure 15, at least as far as the regulation layer is concerned. Therefore, given a design for the coordination layer, one needs to do some more verification to ensure that, when coupled, the two layers will perform as required. Alternatively one can try to prove, either by automatic verification or by theoretical analysis, that the proposed coordination layer design is equivalent to the abstraction of Figure 15 from the regulation layer point of view. Work in both these directions is currently underway for the coordination layer design proposed in [5].

The interaction with the continuous time control laws is more challenging. Unlike the interaction with the coordination layer, which can be easily investigated using standard FSM tools, the interaction of the interface state machine with the continuous domain is much more complicated. There are no tools yet to perform automatic verification on hybrid systems like this. Some tools exist for verification of timed FSM, that is FSM that have "clocks" associated with each state, based on the work of Alur, Courcoubetis and Dill [16]. However the dynamics of our system are a lot more complicated than the simple $\dot{x} = 1$ dynamics of clocks. Therefore attempts to directly use such verification techniques on our system soon run into trouble. One possible solution to this problem is to construct a conservative abstraction of our system that falls into the realm of timed automata, that is an abstraction that contains only "clocks" whose behavior includes all possible behaviors of our system. Then we could verify our system by verifying the conservative abstraction. Work in this direction is also underway.

The interface presented here was introduced as a way of coupling discrete event and continuous time systems. Even though the details are specific to the problem at hand we believe that our work illustrates a more general approach to obtaining such a coupling. It should be noted however that, despite the fact that the immediate task of achieving communication between the layers was performed there is still no guarantee that the coupled system will operate as required under all possible operating environments. Unfortunately there is no formal theory at the moment to support the analysis of our system. Moreover the automatic verification techniques also fall short, as described above. We hope that further work on this problem will provide useful insight for hybrid systems in general and help us induce a formalism capable of dealing with systems like this.

Acknowledgment: The authors would like to thank Farokh Eskafi, Bobby Rao, Shankar Sastry and Pravin Varaiya for helpful discussions providing insight into the problem.

References

[1] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. AC-38, no. 2, pp. 195–207, 1993.

[2] P. Varaiya and S. E. Shladover, "Sketch of an IVHS systems architecture," Tech. Rep. UCB-ITS-PRR-91-3, Institute of Transportation Studies, University of California, Berkeley, 1991.

[3] S. Shladover, C. Desoer, J. Hedrick, M. Tomizuka, J. Walrand, W. Zhang, D. McMahon, H. Peng, S. Sheikholeslam, and N. McKeown, "Automatic vehicle control developments in the PATH program," *IEEE Transactions on Vehicular Technology*, vol. 40, no. 1, pp. 114–130, 1991.

[4] B. S. Y. Rao and P. Varaiya, "Roadside intelligence for flow control in an IVHS," *Transportation Research - C*, vol. 2, no. 1, pp. 49–72, 1994.

[5] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, "Protocol design for an automated highway system," *Discrete Event Dynamic Systems*, vol. 2, no. 1, pp. 183–206, 1994.

[6] J. K. Hedrick, D.McMahon, V. Narendran, and D. Swaroop, "Longitudinal vehicle controller design for IVHS system," in *American Control Conference*, pp. 3107–3112, 1991.

[7] H. Peng and M. Tomizuka, "Vehicle lateral control for highway automation," in *American Control Conference*, pp. 788–794, 1990.

[8] S. Sheikholeslam and C. A. Desoer, "Longitudinal control of a platoon of vehicles," in *American Control Conference*, pp. 291–297, 1990.

[9] D. N. Godbole and J. Lygeros, "Longitudinal control of the lead car of a platoon," *IEEE Transactions on Vehicular Technology*, vol. 43, no. 4, pp. 1125–1135, 1994.

[10] W. Chee and M. Tomizuka, "Lane change maneuver of automobiles for the intelligent vehicle and highway systems (IVHS)," in *American Control Conference*, pp. 3586–3587, 1994.

[11] F. Eskafi, D. Khorramabadi, and P. Varaiya, "SmartPath: An automated highway system simulator." PATH Technical Report UCB-ITS-94-4. Institute of Transportation Studies, University of California, Berkeley, 1994.

[12] J. Lygeros, D. N. Godbole, and M. E. Broucke, "Design of an extended architecture for degraded modes of operation of AHS." PATH Working Paper, UCB-ITS-PWP-95-3, Institute of Transportation Studies, University of California, Berkeley, 1995.

[13] J. Lygeros, D. N. Godbole, and M. E. Broucke, "Design of an extended architecture for degraded modes of operation of IVHS," in *American Control Conference*, 1995. To Appear.

[14] Z. Har'El and R. Kurshan, *Cospan User's Guide*. AT&T Bell Laboratories, 1987.

[15] D. N. Godbole, J. Lygeros, and S. Sastry, "Hierarchical hybrid control: An IVHS case study," in *IEEE Control and Decision Conference*, pp. 1592–1597, 1994.

[16] R. Alur, C. Courcoubetis, and D. Dill, "Model checking for real-time systems," *Logic in Computer Science*, pp. 414–425, 1990.

A Figures

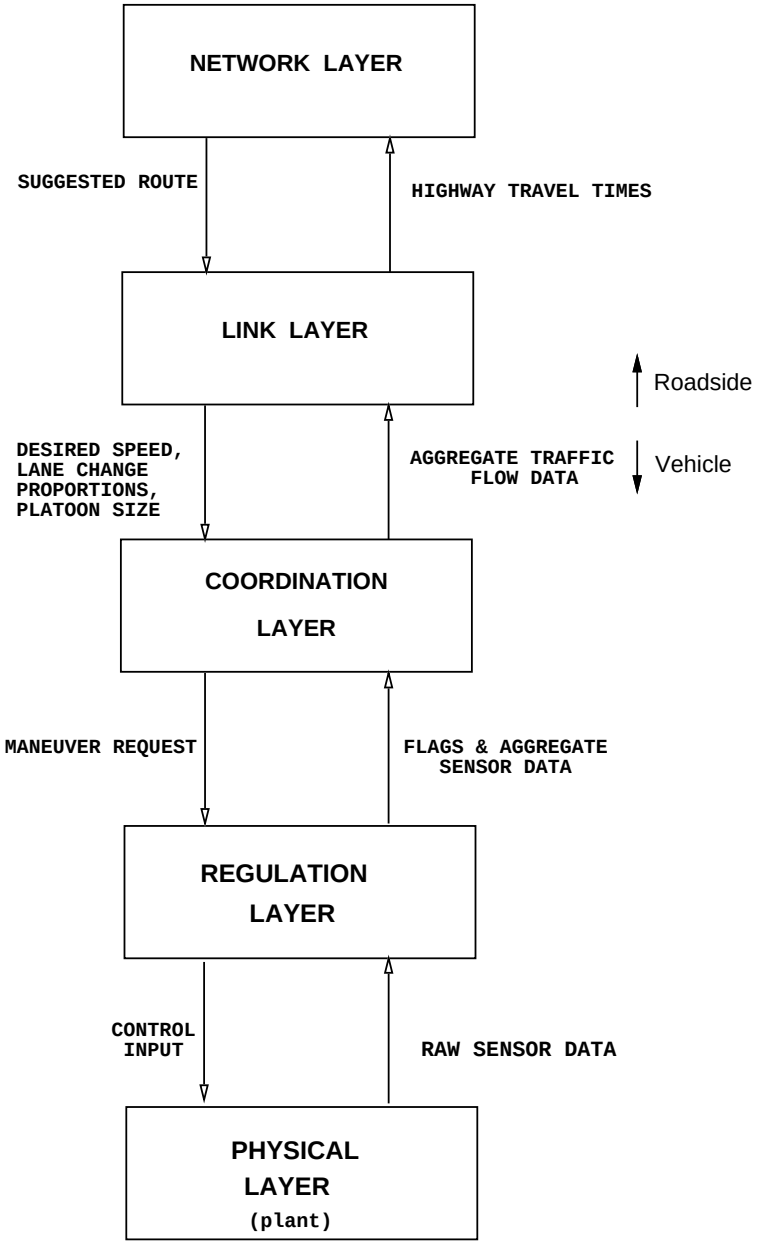

Figure 1: Hierarchical structure of the control system

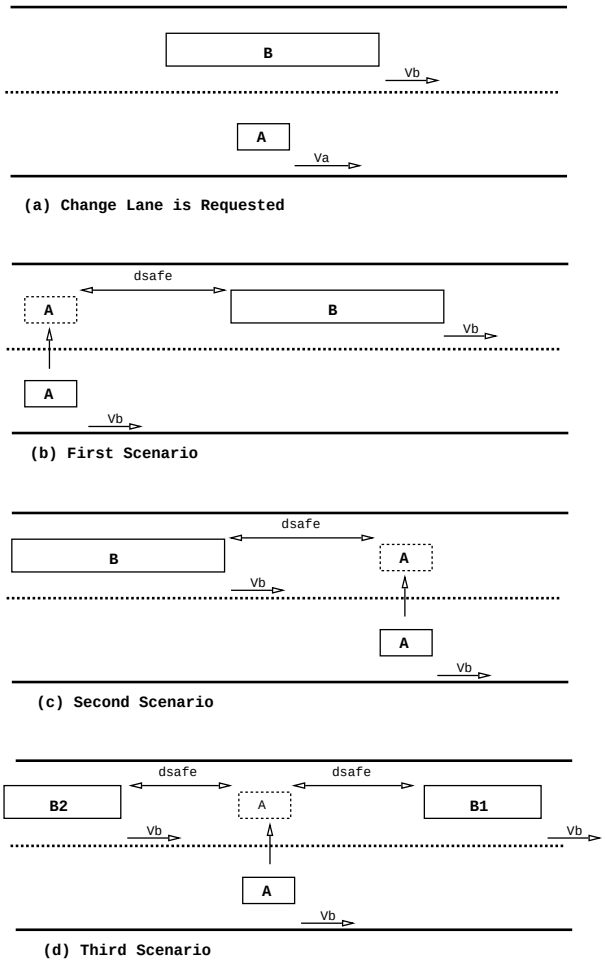

Figure 2: Three scenarios for changing lane

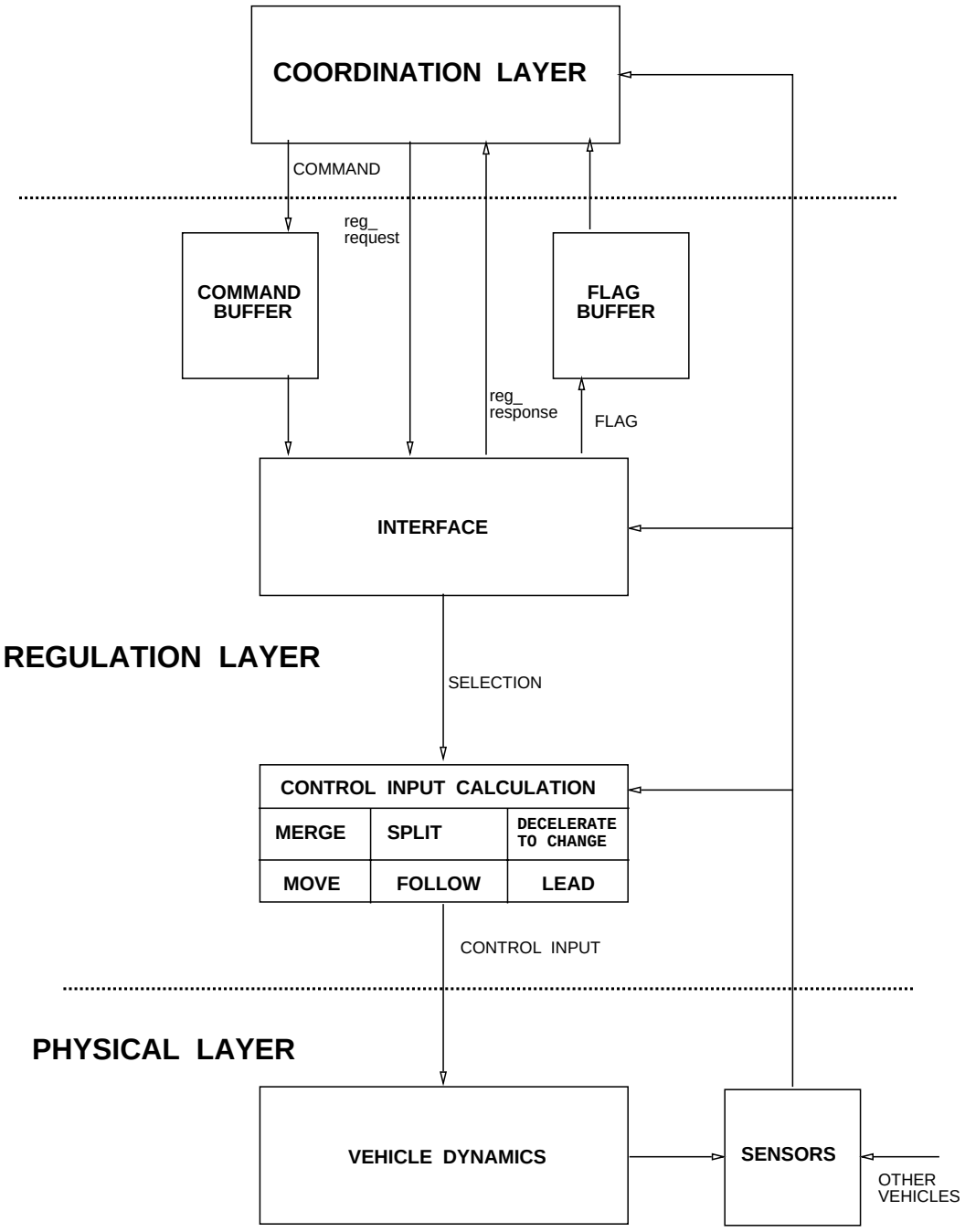

Figure 3: Interactions between controller layers

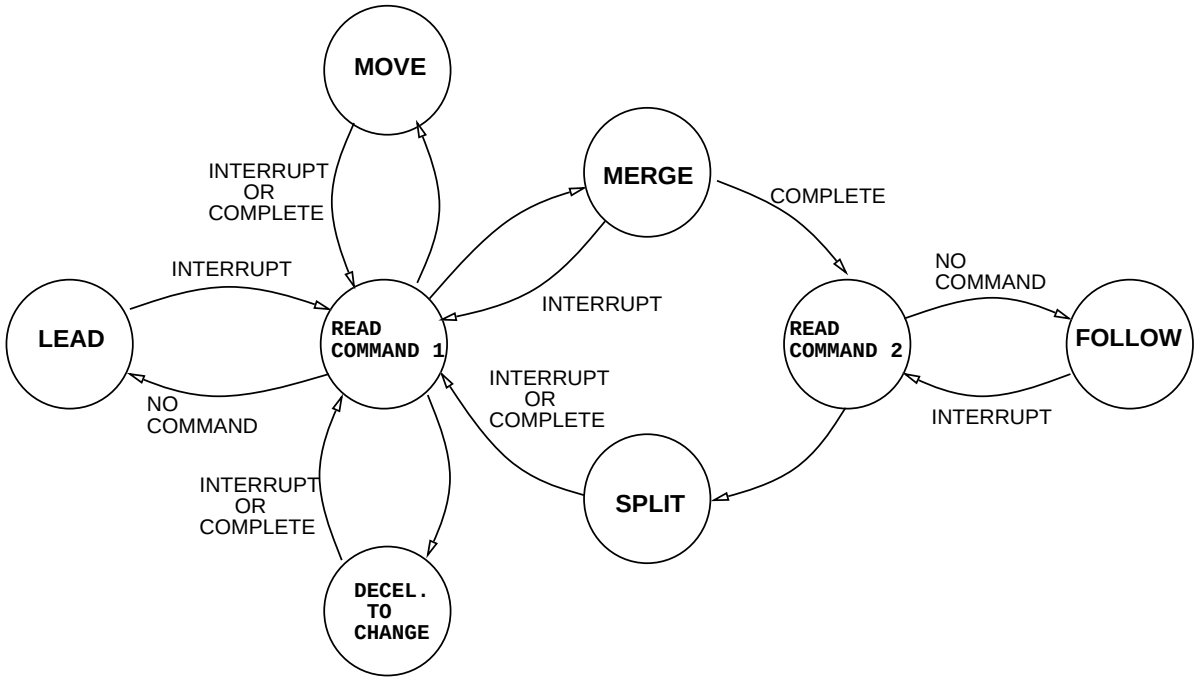

Figure 4: Outline of proposed Interface FSM

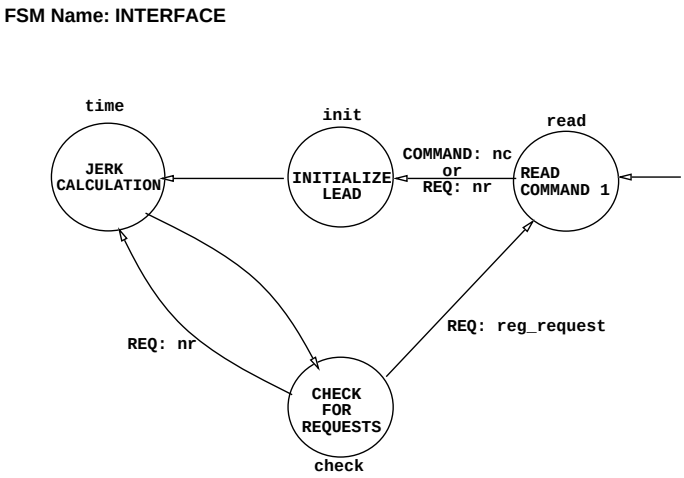

Figure 5: Leader actions

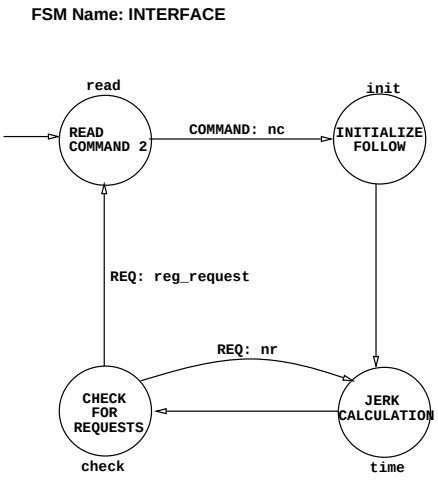

Figure 6: Follower actions

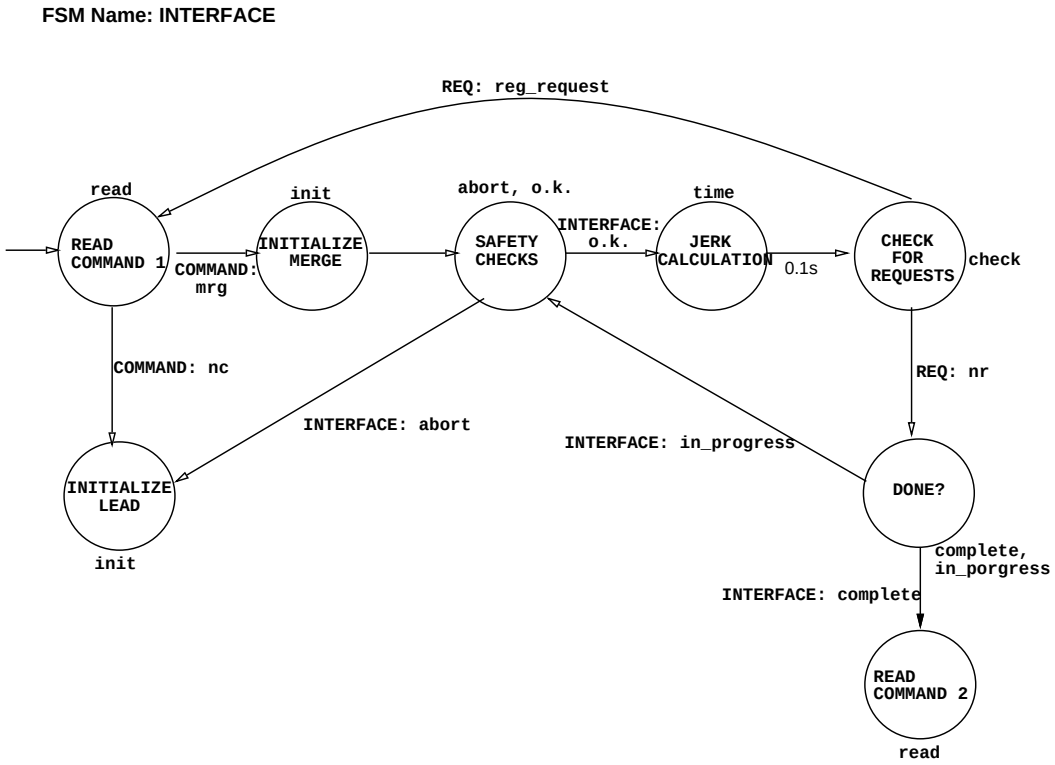

Figure 7: Merge maneuver protocol

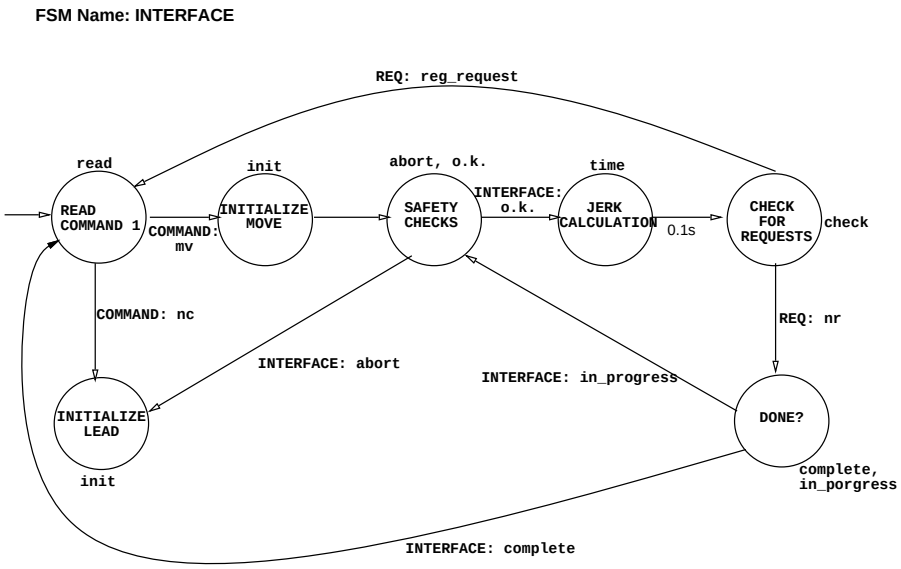

Figure 8: Move maneuver protocol

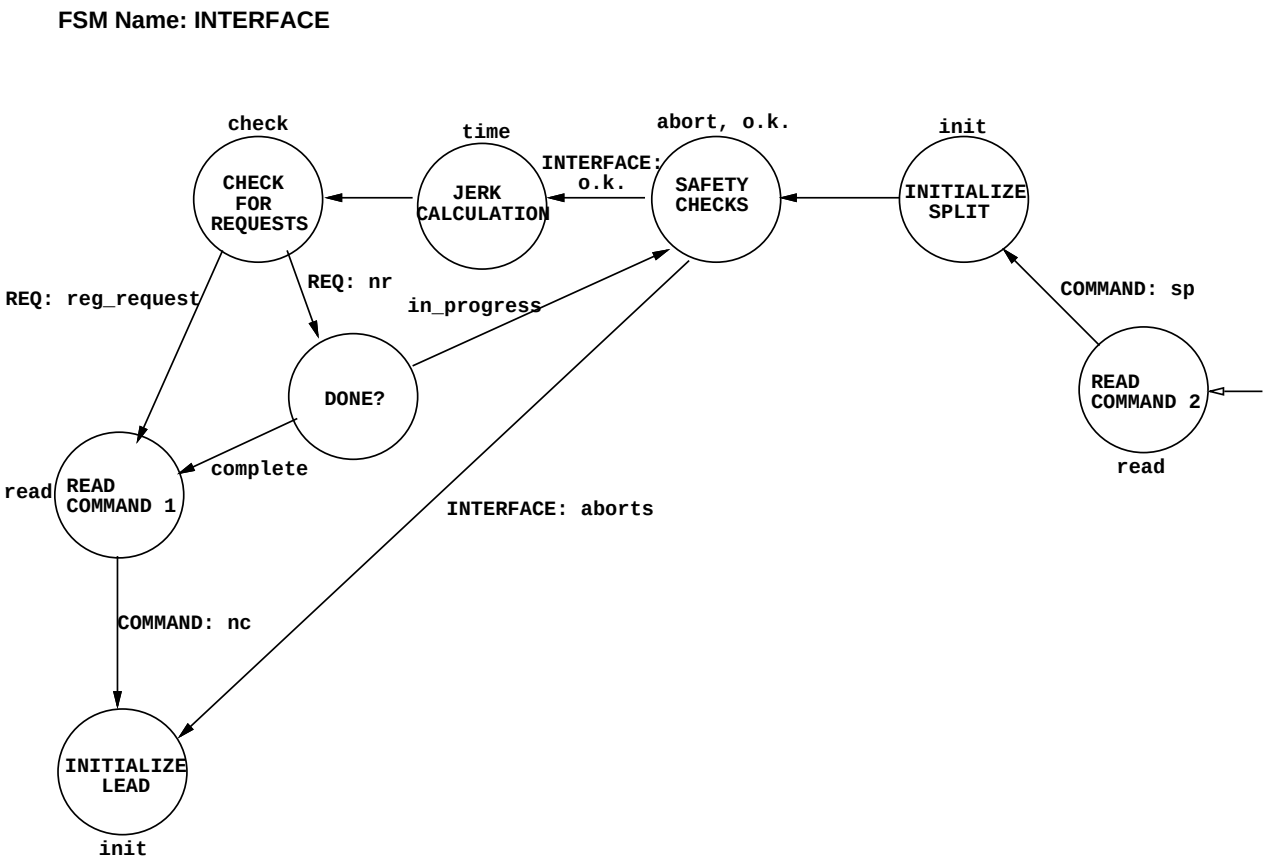

Figure 9: Split maneuver protocol

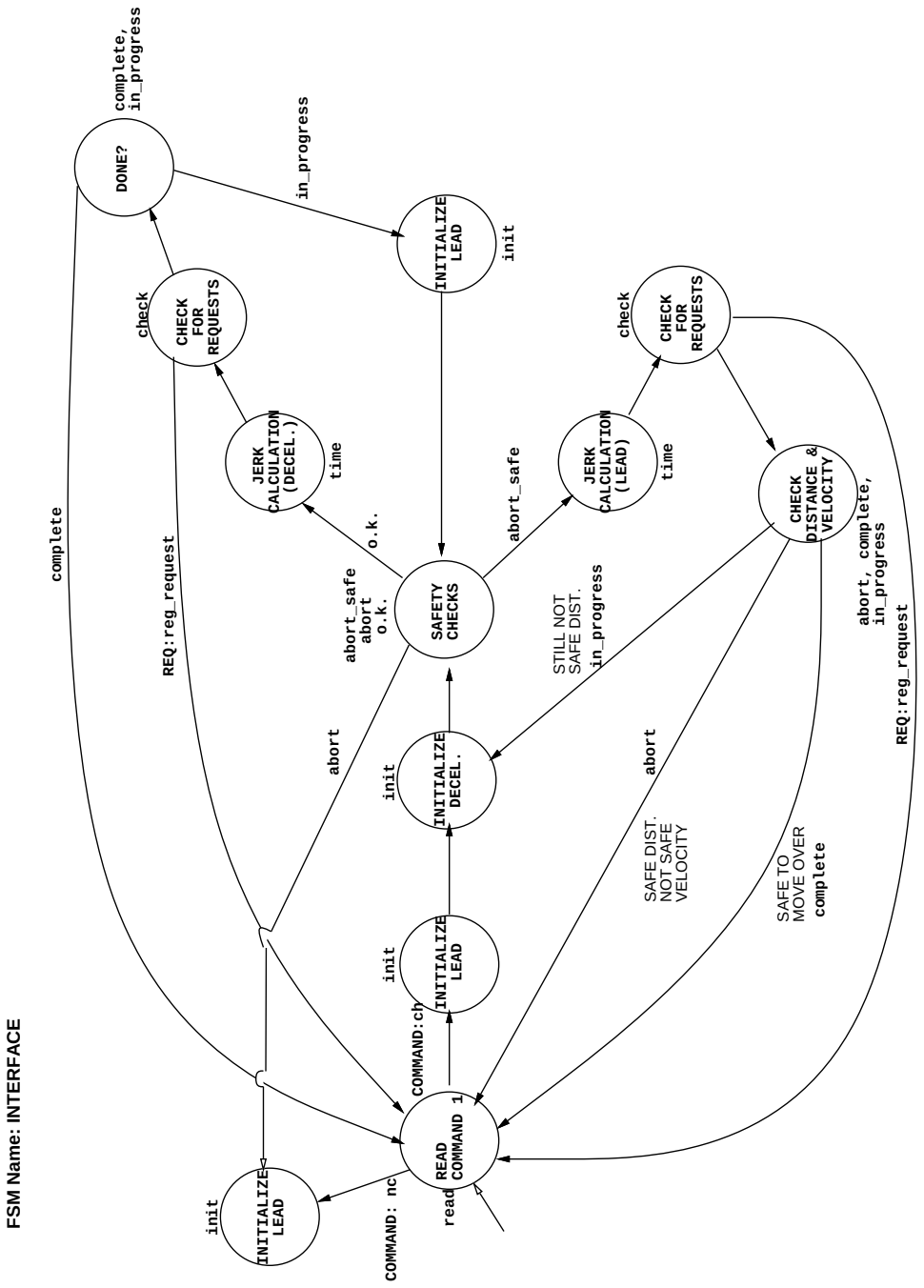

Figure 10: Change maneuver protocol

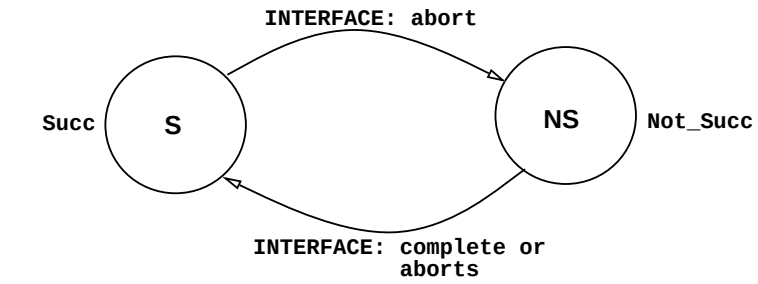

Figure 11: Flag Buffer

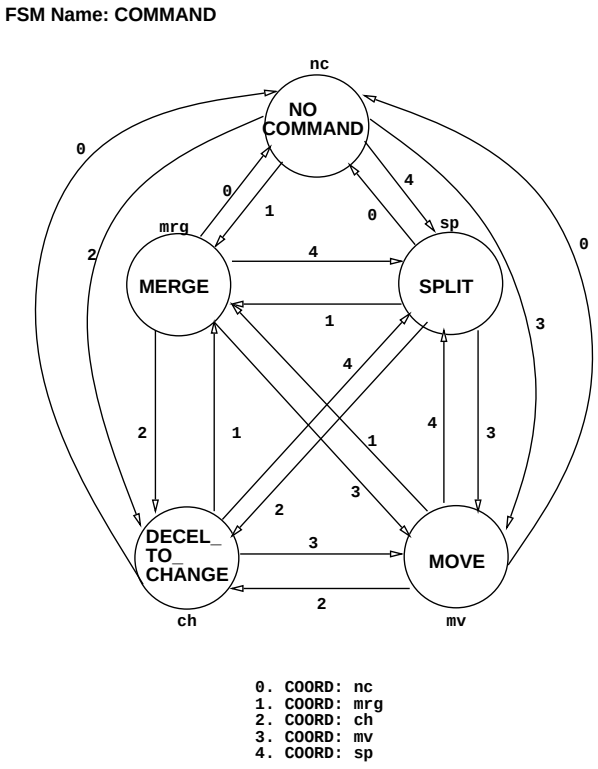

Figure 12: Command Buffer

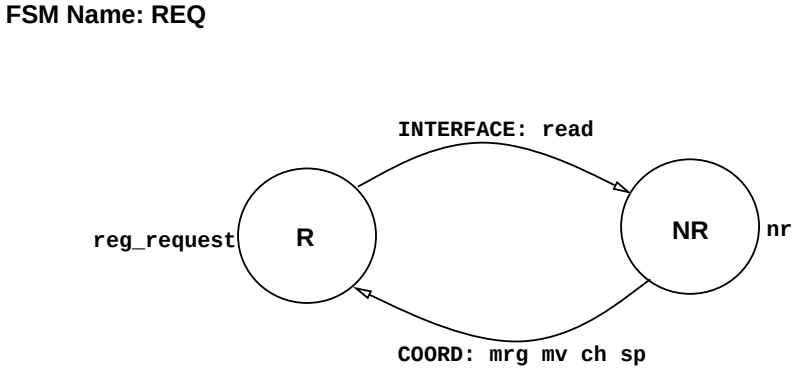

Figure 13: reg_request Channel

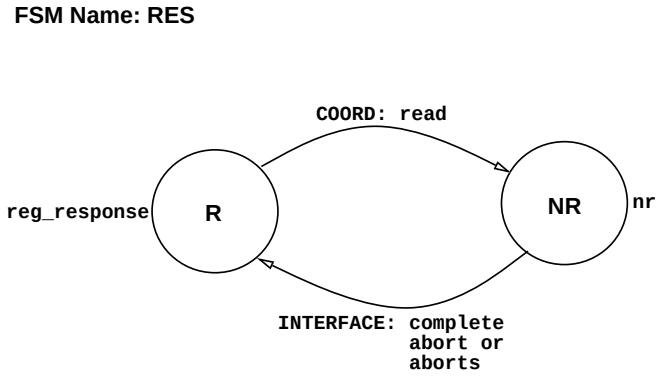

Figure 14: reg_response Channel

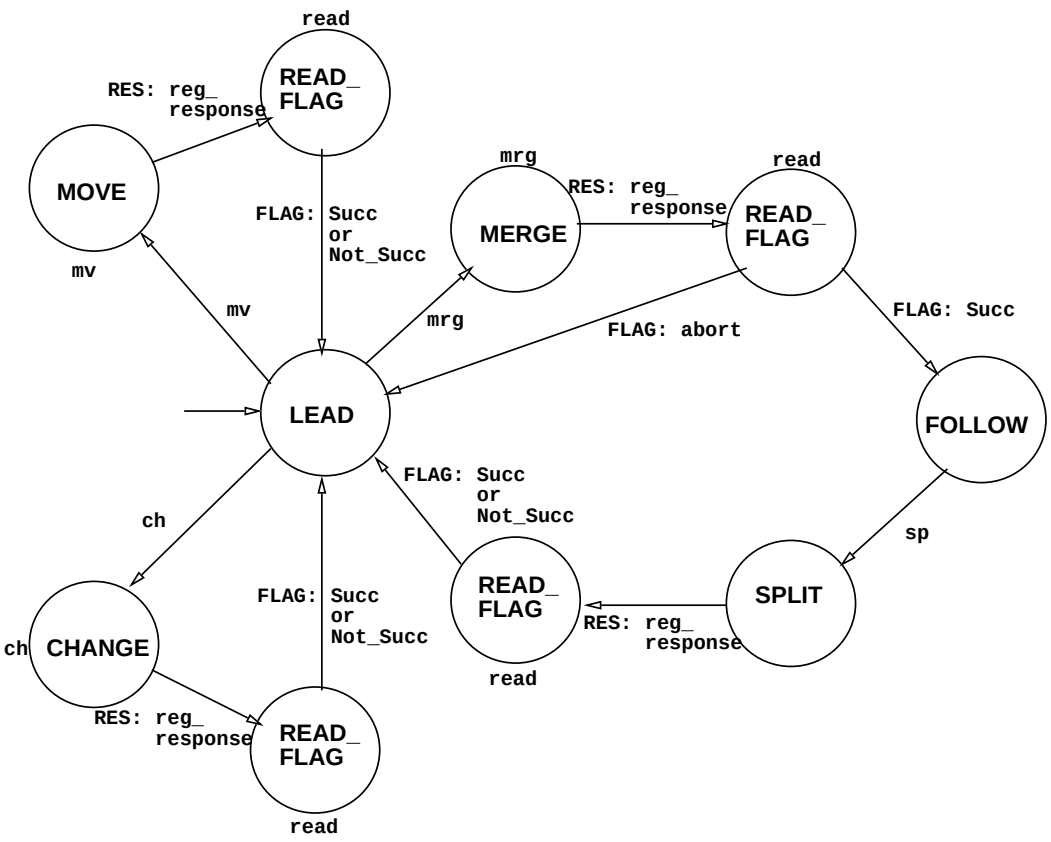

Figure 15: Coordination Layer Abstraction