Title

Development of Integrated Meso/Microscale Traffic Simulation Software for Testing Fault Detection and Handling Algorithms in AHS: Final Report

Author

Horowitz, Roberto

Publication Date

2003-03-01

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

Development of Integrated Meso/Microscale Traffic Simulation Software for Testing Fault Detection and Handling Algorithms in AHS: Final Report

Roberto Horowitz
University of California, Berkeley

California PATH Research Report
UCB-ITS-PRR-2003-13

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.

Final Report for MOU 383

March 2003

ISSN 1055-1425

MOU 383

Development of Integrated Meso/Microscale Traffic Simulation Software for Testing Fault Detection and Handling Algorithms in AHS

Final Report

P.I. Roberto Horowitz

Department of Electrical Engineering and Computer Sciences
Department of Mechanical Engineering
University of California, Berkeley

December 11, 2002

1 Executive Summary

A large portion of PATH's effort in the area of Advanced Vehicle Control Systems (AVCS) during the past several years has focused on the development of theory and technologies that may significantly increase highway safety and capacity through the use of Automated Highway Systems (AHS). To continue with the analysis of AHS feasibility, it is necessary to perform exhaustive large scale simulations that will provide information on the impact of the proposed technologies upon system safety and capacity. In particular, it is necessary to study and evaluate how faults in a vehicle impact the overall AHS performance and capacity, how the roadside control systems can react to these faults and perform degraded-mode activities at the higher hierarchical levels, and how the roadside control system can detect faults either in the vehicle or in its own infrastructure.

When the response of an AHS is being simulated, different degrees of precision are required. In some sections of the highway the detailed behavior in each vehicle in the section must be simulated to model vehicle or infrastructure faults, or test vehicle level fault detection and handling algorithms or degraded mode maneuvers. This level of AHS simulation, where the dynamics of each vehicle is simulated, is called *microsimulation*. The AHS microsimulation software currently under development at PATH is SmartAHS. This software package is based on the hybrid systems language SHIFT [7].

In most sections, it is convenient to model the highway using fluid-like conservation models of traffic flow, applied to the average traffic characteristics in that section, such as density, flow, and velocity. This level of AHS simulation is called *mesosimulation*. Mesoscale simulators do not provide information about particular vehicles but are instead capable of simulating large highway networks. The mesosimulation software used by PATH for analyzing Automated Highway Systems is SmartCAP [5]. Although it is in principle possible to microsimulate a large scale AHS, the computational cost is prohibitively high and, in many cases yields no advantage over mesosimulation, except for the few sections of the highway where it is needed.

The aim of this project is to bridge the simulation gap between SmartAHS and SmartCAP, by implementing an integrated AHS micro-meso simulation environment for simulating a large scale AHS network, where both SmartCAP and SmartAHS run simultaneously and interact with each other. In this simulation environment, most of the highway sections in the AHS are simulated by the SmartCAP mesosimulation software, except for one or more sections, which are simulated by the SmartAHS microsimulation software. Highway sections that are microsimulated using SmartAHS are referred to as *μ-windows*. The topology of the entire AHS, as well as the location and length of each μ-window is set up by the user, by means of user interface software, named *mminterface*. The benefits of the meso-micro simulation environment are best realized when μ-windows are placed in the regions where microscopic effects are of greatest interest; by allowing the remainder of the AHS to be simulated by the more computationally efficient SmartCAP, the meso-micro software is able to simulate the AHS at a lower computational cost than stand-alone SmartAHS.

This final report documents the theoretical design and software implementation of the integrated micro-meso simulator. It also provides a brief tutorial on its use, including a simple AHS simulation example.

The following tasks were executed in this project:

1. The capabilities of SmartAHS were extended by developing a simplified sensor architecture, a simplified vehicle-roadway environment processor, and a simplified set of regulation-layer components. Under certain conditions, the simplified set produces identical results as the full components, and increases simulation speed 4 to 5 fold.

2. The SmartCAP activity model was extended to include platooning and join/split maneuvers.

3. The interface between SmartCAP and SmartAHS was designed and implemented in SHIFT. A SmartAHS component was created to schedule and monitor all aspects of the interface between SmartCAP and SmartAHS.

4. A batch compiler and a MATLAB-based visual interface were created to allow the user to input simulation parameters, define the highway topology, and view both mesoscale and microscale simulation outputs simultaneously.

5. The developed integrated meso/microscale simulation software was tested for different scenarios. The PATH hierarchical control architecture, specifically the link, coordination, regulation, and physical layers, was tested in the meso-microscale simulator.

The meso-micro simulator is well-suited for testing the response of an AHS to situations that are characterized by localized microscale phenomena, such as: the stalling of an individual vehicle, oversaturation of a particular artery due to external factors (e.g. a sporting event), emergency vehicle maneuvers, and changes in highway capacity due to different on-ramp metering policies. The use of the software is described in Sec. 6 using a simple simulation example. The example shows a traffic density wave propagating forward from the mesoscale region into the microscale region, thus demonstrating the functionality of the software.

2 Introduction

At present, the simulation tools available for analyzing highway traffic behavior, and specifically Intelligent Vehicle Highway Systems (IVHS), generally fall into two categories, according to their level of modeling detail: microscale and mesoscale simulators.

Mesoscale simulators are usually based on fluid-like models of traffic flow, applied to average traffic characteristics such as density, flow, and velocity. They do not provide information about particular vehicles but are instead capable of simulating large highway networks. Microscale simulators are based on models of individual vehicle dynamics, where the behavior of each vehicle depends on its locally perceived environment.

Currently, the mesosimulation software used by the California PATH program for analyzing Automated Highway Systems (AHS) is SmartCAP [5]. This software uses a conservation model of traffic density, coupled with rules of traffic flow behavior given by a finite set of vehicle *activities* (e.g. join, cruise, change lane). SmartCAP allows monitoring of aggregate traffic quantities in an AHS subdivided into sections of approximately 0.5 miles. This level of abstraction has been proven adequate for reproducing large scale traffic phenomena such as upstream moving waves of congestion and disturbances due to emergency vehicles [19, 20]. Additional traffic flow theory used in the development of SmartCAP can be found in [6], and is briefly described in Section 3.1.

The highway microsimulation software currently under development at PATH is SmartAHS. This software package is based on the hybrid systems language SHIFT [7]. SmartAHS was designed to demonstrate several AHS concepts, including the PATH hierarchical control architecture [22, 23]. This architecture divides the task of controlling vehicles in an AHS into a hierarchical structure. The upper levels of the architecture (network and link) are devoted to route planning and flow management across large expanses of highway. The lower levels (coordination, regulation, and physical)

exist onboard each vehicle, and are responsible for vehicle-based communications, maneuver selection, motion control, and actuation. Currently, all AHS control layers, except for network, have been implemented in the SmartAHS environment, and ongoing effort is focused on implementing extended degraded mode structures [15].

In order to simultaneously capture the occurrence of small scale events and their impact on the larger highway, through adjustments to higher level controls and the propagation of congestion, multiple degrees of modeling abstractions are needed. There are many traffic scenarios that are characterized by both mesoscopic and microscopic effects. For example, in an AHS, it is important to study and evaluate how faults in a vehicle impact the overall AHS performance and capacity, how the roadside control systems can react to these faults and perform degraded-mode activities at the higher hierarchical levels, and how the roadside control systems can detect faults either in the vehicle or in their own infrastructure. Strategies have been developed to permit emergency vehicle (EV) transit through an AHS, specifically in the case of stopped traffic [19, 20, 21]. In a realistic situation, EV transit will depend not only on maneuvers of nearby vehicles but also upon the effects of traffic blockage on the entire highway. Another example involves on-ramp metering. Entrance control schemes may be used to determine optimal on-ramp metering rates for a highway system. Such rates are a function of the overall state of the highway, but implementation of entrance control laws takes place at the level of individual vehicles. Although it is in principle possible to microsimulate the entire AHS, the computational cost is prohibitively high (SmartAHS simulation capabilities are in the range of 30 vehicles with a 25/1 real-to-simulation time ratio on a 4 processor Sun Sparc Ultra-II computer). Furthermore, in many cases, such as the previous examples, it may only be necessary to examine selected portions of the AHS at the microscopic scale, while mesoscale simulation is sufficient for the remainder of the highway.

We have noted that (1) it is not possible to simulate large-scale AHS in SmartAHS in a *computationally efficient* manner, and (2) in many situations, we are interested in examining the behavior of individual vehicles only at *specific locations* within the AHS, such as near entry and exit stations. These observations, combined with the need for a tool that is suitable for analyzing extensive AHS designs, have led us to develop a *meso-microscale* simulation environment. The meso-micro simulator

- combines the SmartAHS and SmartCAP simulation models,
- allows the user define an AHS geometry, using the SmartCAP input file format, then select regions of highway, called μ-windows, that will be microsimulated,
- for a given highway configuration, improves simulation speed relative to a purely microsimulated AHS. This is due to the fact that the portions of the AHS outside the μ-windows are simulated by SmartCAP, and unlike SmartAHS, the simulation cost of SmartCAP does not increase with the number of vehicles in simulation,
- facilitates simulation set-up and executable file generation through its batch compiler, and enables visualization of both meso and microscale simulation outputs via its Matlab-based GUI.

The remainder of this report details the development and design of the meso-micro simulator. Section 3 gives background information on both the SmartCAP and SmartAHS models. In Section 4, secondary accomplishments of this project, in particular the extended versions of SmartCAP and SmartAHS, are discussed. The primary contribution of this project is the meso-micro simulator, and the meso-micro software design is described in detail in Section 5. Section 6 contains instructions on how to set up and run a simple simulation example.

3 Background

The main focus of this project was the development of an interface between the mesoscopic AHS simulator SmartCAP and the microscopic simulator SmartAHS. In this section we review the main elements of the SmartCAP and SmartAHS simulation models, in order to familiarize the reader with each simulator, and to introduce notation and terminology that will be used in subsequent sections.

3.1 The SmartCAP model

The SmartCAP model is an example of a *mesoscopic* model of traffic behavior. Its level of abstraction falls between that of a microscopic model, such as SmartAHS, and a macroscopic model, which may consist of a series of interconnected links and nodes. A detailed description of SmartCAP can be found in [5]. Here we give a brief review of the model, and introduce the notation used in Sections 4.1 and 5.4.

SmartCAP is a discrete time / discrete space model. Typically, the highway is divided into sections of about 250 meters in length. The time interval is on the order of 10 seconds. The behavior of automated traffic in SmartCAP is governed by two principles: the conservation of vehicles, and the interaction between the supply of space and the demand given by the *activity plan*. Automated vehicles within the AHS are classified into *types*, according to their origin and destination. Each type has its own activity plan, which may vary with time and space. The set of all activity plans is issued by the *Traffic Management Center* (TMC), which operates according to user defined link layer control laws, and is assumed to have complete information of the state of the AHS. Each activity plan consists of a set of probabilities for engaging in a *maneuver* during the upcoming time interval. For example, an activity plan may specify that vehicles of type j, in section i, have the following probabilities:

- 30% will request a *split* from their current platoon
- 20% will *change lanes to the right*
- 50% will continue to cruise without change.

Each *maneuver* or *activity* has an associated space and time requirement. For example, *cruising as a leader* requires more space than *cruising as a follower*. Changing lanes requires space in both the current and adjacent lanes. The amount of space used by an activity may vary with speed. The principle of space supply versus activity demand dictates that the total space required by the activity plan cannot exceed the available space in each section, for each time interval. If it does, certain activities are rejected according to rules of prioritization, until the space constraint is satisfied.

Notation:

Δt	... Duration of the time interval
$L_{(i)}$	... Length of section i
$n_{(i,j,k)}$	... Number of vehicles in section i, of type j, time interval k
$n_{l(i,j,k)}$	... Number of leaders in section i, of type j, time interval k
$n_{f(i,j,k)}$	... Number of followers in section i, of type j, time interval k
$v_{(i,k)}$	... Average traffic speed in section i during time interval k
$q_{(i,j,k)}$	... Number of vehicles of type j leaving section i, during time interval k

$q_{l(i,j,k)}$ ... Number of leaders of type j leaving section i, during time interval k
$q_{f(i,j,k)}$ ... Number of followers of type j leaving section i, during time interval k
$\texttt{APS}_{(i,k)}$ ... Average platoon size in section i during time interval k

The following relations exist among these variables:

$$n_{(i,j,k)} = n_{f(i,j,k)} + n_{l(i,j,k)}$$
$$q_{(i,j,k)} = q_{f(i,j,k)} + q_{l(i,j,k)}$$
$$q_{(i,j,k)} = v_{(i,k)}\, n_{(i,j,k)}\, \frac{\Delta t}{L_{(i)}}$$
$$\texttt{APS}_{(i,k)} = \frac{\sum_j n_{(i,j,k)}}{\sum_j n_{l(i,j,k)}}$$

The update of the traffic state at every time step is carried out in four steps. First, constraints are applied to the number of maneuvers *commanded* by the TMC. This results in an estimate of the number of *completed* maneuvers. Then an *intermediate state* is calculated by assuming that these maneuvers are completed without forward movement of traffic. The intermediate state, denoted below with superscript bars, accounts for lateral motions (lane changes) and changes in the numbers of leaders and followers through joins and splits. The third step is to compute a speed for the upcoming time interval as the minimum between the command by the TMC and the maximum allowable speed due to the availability of space in the downstream section. Finally, vehicles are moved downstream according to the speed of the third step.

Additional notation:

$\Pi^{m}{}_{(i,j,k)}$ ... Proportion of vehicles of type j completing maneuver m in section i, during time interval k. m can take values `c, l, r, s` or `j`, representing cruise, lane change left, right, split or join.
$\Pi^{\texttt{m}}_{\texttt{TMC}(i,j,k)}$ ... Maneuver proportions requested by the TMC
$\Lambda^{s}_{m}(j)$ ... Average space required by vehicles performing maneuver m, in their current lane
$\Lambda^{r}_{m}(j)$ ... Average space required by vehicles performing maneuver m, in the lane to their right
$\Lambda^{l}_{m}(j)$ ... Average space required by vehicles performing maneuver m, in the lane to their left
$\texttt{TS}_{(i)}$ ... Total space (capacity) of section i
$r(i)$ ... Lane to the right of lane i
$l(i)$ ... Lane to the left of lane i
$u(i)$ ... Lane upstream of lane i
$d(i)$ ... Lane downstream of lane i

Step 1) SmartCAP computes $\Pi^{m}{}_{(i,j,k)}$ for every i,j and m. This is done by first checking whether the following constraint is satisfied in every section i:

$$\sum_j \sum_m \left[n_{(i,j,k)} \Pi^{\texttt{m}}_{\texttt{TMC}(i,j,k)} \Lambda^{s}_{m}(j) + n_{(r(i),j,k)} \Pi^{\texttt{m}}_{\texttt{TMC}(r(i),j,k)} \Lambda^{l}_{m}(j) + n_{(l(i),j,k)} \Pi^{\texttt{m}}_{\texttt{TMC}(l(i),j,k)} \Lambda^{r}_{m}(j) \right] \leq \texttt{TS}_{(i)}$$

If so, then $\Pi^{m}{}_{(i,j,k)}$'s are set to $\Pi^{\mathtt{m}}_{\mathtt{TMC}}(i,j,k)$. If not, the proportions are reduced. The set of rules for estimating the number of completed maneuvers has been extended to include platooning behaviors in MOU 383. These extensions are explained in detail in Section 4.1.

Step 2) The intermediate state is computed as:

$$\bar{n}_{(i,j,k)} = n_{(i,j,k)} + \Pi^{\mathtt{l}}{}_{(r(i),j,k)} n_{(r(i),j,k)} + \Pi^{\mathtt{r}}{}_{(l(i),j,k)} n_{(l(i),j,k)} \tag{1}$$

Step 3) The average speed is computed as: $v_{(i,k)} = \mathtt{min}(v^{\mathtt{TMC}}{}_{(i,k)}\ ,\ v^{\mathtt{space}}{}_{(i,k)})$, where $v^{\mathtt{space}}{}_{(i,k)}$ is the speed at which the downstream section fills to capacity, and $v^{\mathtt{TMC}}{}_{(i,k)}$ is the velocity command from the TMC.

Step 4) Vehicle conservation is applied:

$$n_{(i,j,k+1)} = \left(1 - \frac{v_{(i,k)}\Delta t}{L_{(i)}}\right) \bar{n}_{(i,j,k)} + \left(\frac{v_{(u(i),k)}\Delta t}{L_{(u(i))}}\right) \bar{n}_{(u(i),j,k)} \tag{2}$$

On-ramp and off-ramp flows are treated similarly.

3.2 The SmartAHS model

SmartAHS, a microsimulation framework for Automated Highway Systems, has been developed by the California PATH program using the hybrid systems language SHIFT [7]. The SmartAHS software used in this project is an extended version of a library provided by PATH through [17]. The basic SmartAHS library defines the highway layout, vehicle dynamics and sensor and communication structures [4]. The general structure of SmartAHS is shown in Fig. 1.

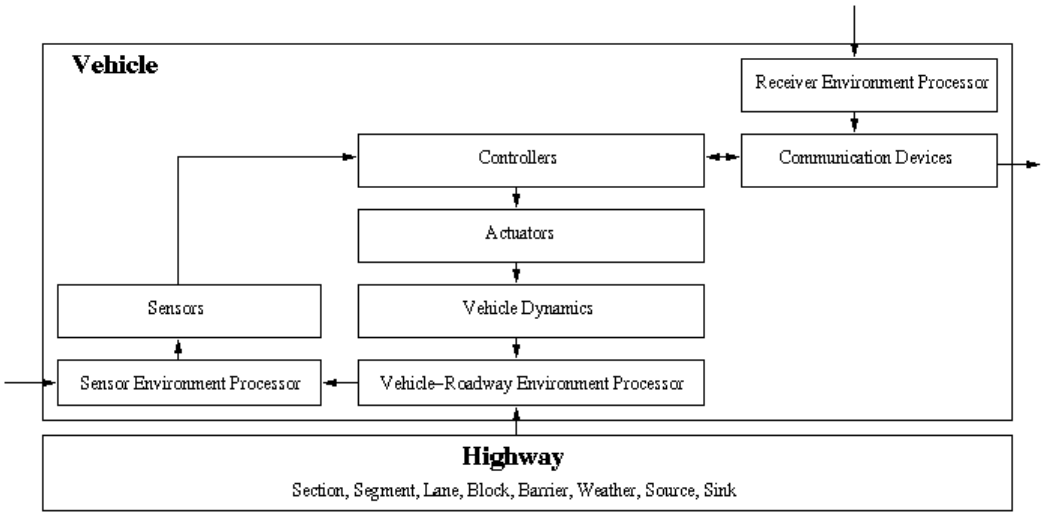

Figure 1: A schematic of SmartAHS architecture

The microsimulation of individual vehicles is based on the PATH multi-layer architecture [22, 23], shown in Fig. 2. This architecture separates the control activities of an AHS into several levels. The network and link layers are concerned with routing and regulating aggregate traffic flows over large-scale highway systems. The coordination, regulation, and physical layers reside on-board each

automated vehicle. These lower layers handle vehicle-based communications, maneuver selection and coordination, motion control, and actuation.

All control layers of the PATH hierarchical control architecture, except for network, have been implemented in SmartAHS. Three of the layers (link, coordination and regulation) were incorporated into SmartAHS under MOUs 238/310 and 287/311; none of these layers are present in the original SmartAHS library provided through [17].

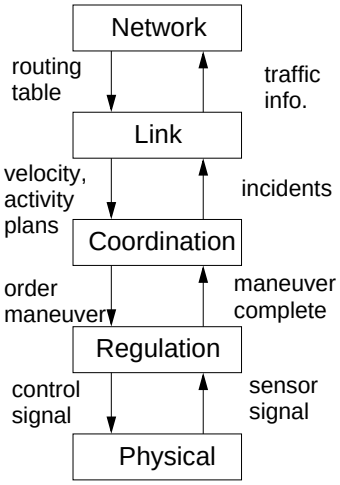

Figure 2: PATH Hierarchical Architecture

One of the primary motivations for developing the meso-micro simulator was our observation of the computational inefficiency inherent in SmartAHS. SmartAHS simulation capabilities are in the range of 30 vehicles with a 25/1 real-to-simulation time ratio on a 4 processor Sun Sparc Ultra-II computer, which indicates that the computational cost of microsimulating an entire AHS, which could include hundreds or thousands of vehicles, is prohibitively high. The low efficiency of the original SmartAHS library, in addition to providing incentive for the integration of SmartCAP and SmartAHS, led us to develop simplified versions of the communication structure, the vehicle-roadway environment processor (VREP), the sensor environment processor (SEP), the vehicle radar, and the regulation-layer control systems, as part of MOU 383. These simplified components are described in Sec. 4.2, while the version of SmartAHS developed under MOUs 238/310 and 287/311 is reviewed in this section. Additional detail on the version of SmartAHS described in this section can be found in the appendix of [24].

3.2.1 Physical Layer

The physical layer consists of vehicle dynamics, sensor and actuator dynamics, and other physical devices. In the SmartAHS library, there are three different types of vehicle dynamics, allowing for simulations of varying complexity: *simple dynamics (kinematics), 2D (flat highway) vehicle dynamics, and 3D (complete) vehicle dynamics* [4]. For this project, we have used the simple kinematic vehicle dynamics exclusively, in order to reduce the computational load. The kinematic vehicle dynamics lumps together the steering, engine, and powertrain dynamics. For the braking actuator, a first order dynamic equation is used to simulate the actuator time delay.

3.2.2 Regulation Layer

The regulation control laws are based on the algorithm given by [13], and most of the implementation in SHIFT can be found in [11]. The structure of the implementation is illustrated in Fig. 3.

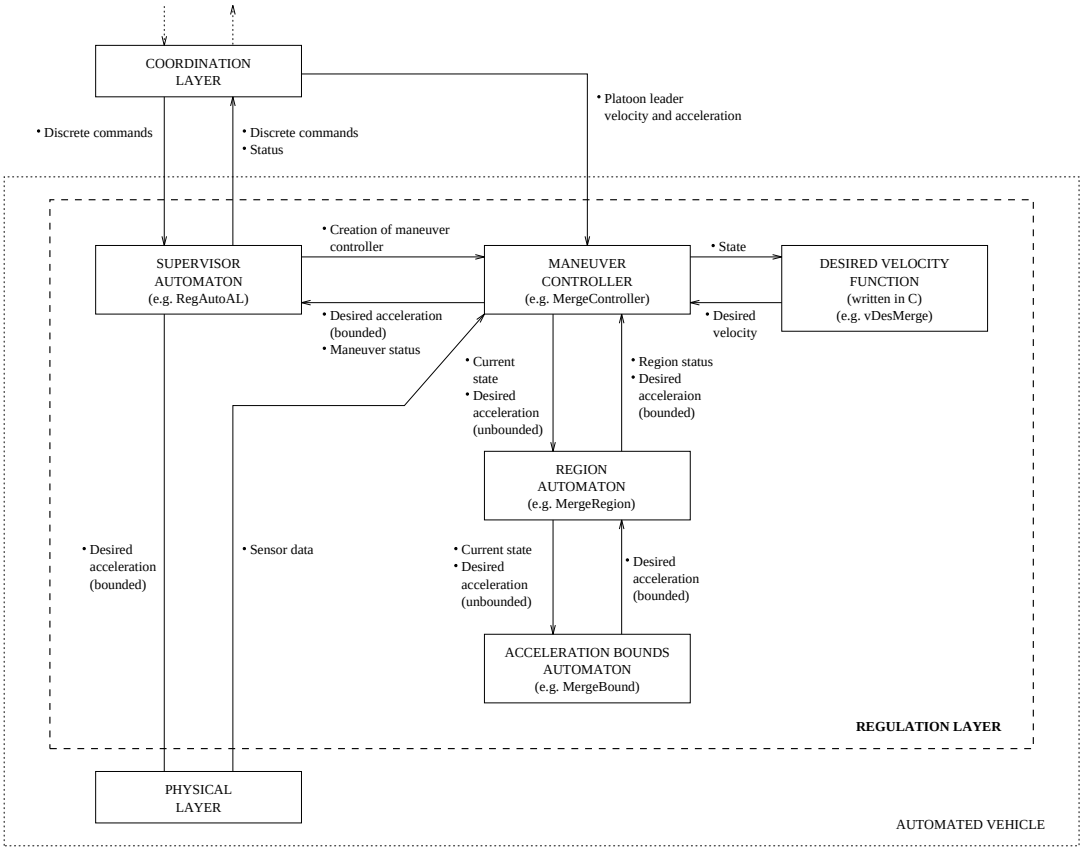

Figure 3: A schematic of regulation layer implementation

The regulation layer implementation consists of two sublayers: the regulation supervisor and the maneuver control laws. The regulation supervisor selects the appropriate regulation control law according to the commands it receives from the coordination layer. It creates the maneuver control law automaton and initiates the maneuver, and then kills this automaton when the maneuver is completed. The maneuver controller consists of four parts: controller, region automaton, acceleration bounds automaton and desired velocity function. More details on each part can be found in [11].

All normal mode regulation control laws were implemented in SmartAHS as part of MOUs 238/310 and 287. *Lead, merge* and *split* maneuvers are based on the control algorithm given in [13], the *follow* controller is given by [18], and the *changelane* controller uses the same safety criterion given by [2] along with a simplified stable lateral motion control. Safety is guaranteed by this controller, which is different from that implemented in SmartPATH [8], for lane change.

3.2.3 Coordination Layer

The coordination layer design is based on that of SmartPATH [8] and the coordination protocol designs of [12] and [22].

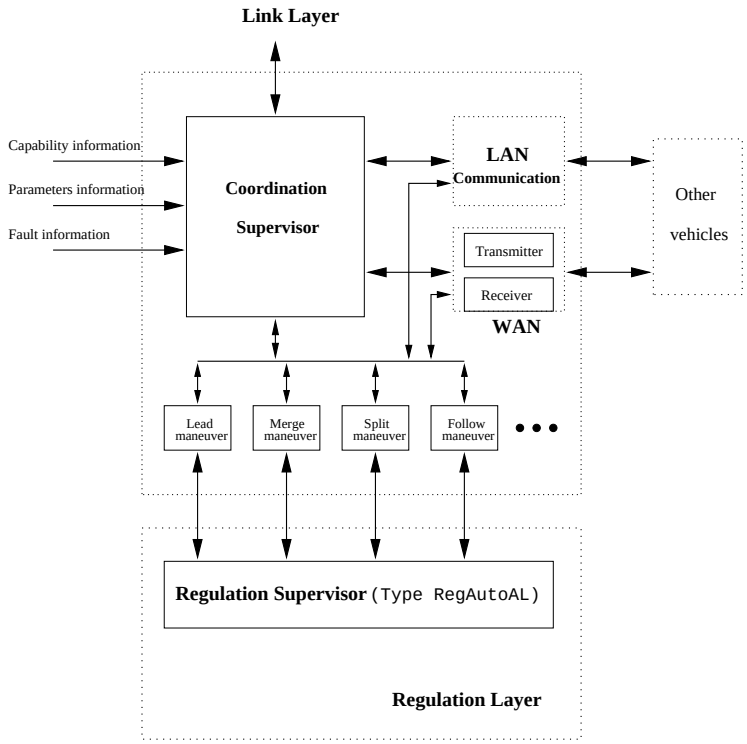

Figure 4: A schematic of coordination layer implementation

The coordination scheme consists of three parts: *coordination supervisor, maneuver protocols* and *communication design*. A schematic of the coordination layer implementation is given in Fig. 4. The coordination supervisor coordinates with different maneuver protocols and manages their initiation and execution by communicating with other vehicles or roadside systems. Basically it is a discrete-event system. The maneuver protocols coordinate with other vehicles to guarantee the correctness and safety of each maneuver. There are two steady state maneuvers: *lead* and *follow*, and some other transient maneuvers such as *split, merge*, and *changelane*. The communication system in the AHS transmits information between vehicles and handles exchange of information between the vehicles and the roadside system. There are two types of communication among vehicles: local area networks (LAN) and wide area networks (WAN). The LAN is used to transmit the velocities and accelerations of both the platoon leader and immediately preceding vehicle to each follower vehicle in the platoon [14], and the WAN is used for passing maneuver messages among different vehicles. The roadside system also broadcasts link layer commands to all the vehicles in the AHS.

The coordination supervisor is similar to the regulation supervisor, except it initiates maneuver protocols instead of regulation control laws. Based on sensor information and messages received through the communication system, the coordination supervisor initiates either a maneuver *initiator* or *responder*. For example, in Figure 5, when the leader of platoon A receives a link layer broadcast with a desired platoon length that favors merging, the leader's coordination supervisor can instantiate a *merge maneuver initiator*, then send out a *merge request* message to the leader of the downstream platoon B. When the coordination supervisor of the leader of platoon B receives this message, it will initiate a *merge maneuver responder* if it is not involved in another maneuver. If it is unable to engage in a merge, for example because it is already performing a split, it will send a rejection message, and the maneuver will be aborted. Otherwise an *acknowledgment* message will be sent and

the appropriate regulation layer controllers will be activated. Once the merge maneuver concludes, the leader of platoon A will become a follower and execute the follower protocol, and the leader of platoon B will become the leader of the new platoon.

The maneuver protocol consists of the coordinated actions among the vehicles involved in a maneuver. For most of the transient maneuvers such as *merge, split*, and *changelane*, two protocols are needed for accomplishment of the maneuver: the maneuver initiator and responder. However, for steady-state maneuvers such as *lead* and *follow*, one protocol is enough.

The implementation of communication in SmartAHS is based on the structure proposed by PATH staff and is different from that of SmartPATH [8]. The communication components must model each layer of the open systems interconnection (OSI) reference model [9]. The OSI structure provided by the library [16] is very complex; in order to increase simulation speed, we used the simplified communication structure developed under MOUs 238/310, 287/311 and 312. In this version of the communication model, vehicle control is integrated with its own communication devices at the message level. For coordination layer communication, the components of the communication structure include: *message, transmitter, receiver*, and a *monitor* with a message queue. These components interface with the coordination supervisor which processes one message from the queue at a time. A message consists of originating vehicle ID, destination ID and message string.

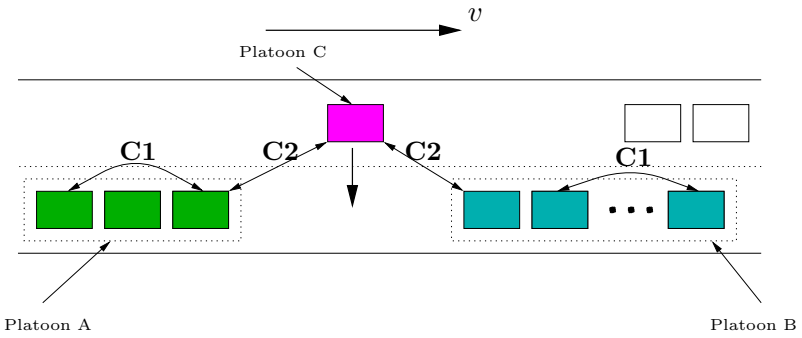

Figure 5: Message level communication schematic

When one coordination control protocol initiates a request for a maneuver, the message will be created and sent to the transmitter. The transmitter broadcasts the messages to the specific vehicles or roadside link layer control systems involved in the maneuver. When the receiver accepts the message it will pass it through to the coordination controller. The monitor works as a centralized component for the physical layer in the communication. The monitor functions as a representation of a set of users adopting the same physical medium; in addition, it models channel properties and keeps track of the transmitters sharing the channel. Also the monitor models the connection type (point to point or broadcast channel). The vehicle ID is passed as part of the vehicle to vehicle communication.

Figure 5 shows the two types of communication that can exist between vehicles in SmartAHS. C1 denotes intraplatoon communication; for example, if the third vehicle in platoon A wants to *split*, it sends the *split* request to the leader of the platoon by communication type C1. This type of message can be broadcast using the point-to-point connection since each vehicle in a platoon knows the ID of the leader. Type C2 communication models communication among different platoons. This type is required to maintain string stability of the platoon. The implementation of communication between the vehicles and the roadside system is similar to that of the coordination layer.

3.2.4 Link Layer

The link layer of the AHS hierarchy is dedicated to the management of traffic over large regions, or *links*, of an automated highway. A link is assumed to be a stretch of highway, in between entrances and exits, which may be subdivided into one or more ∼500m sections. The link layer control system, as implemented in SmartAHS, consists of four main structures:

- a *traffic optimizer*, which determines desired velocities and lane-change profiles for a single link of the highway. The optimization algorithm is based on the method described in [10]. The optimization repeats at a fixed time interval (e.g. every 5 min).
- a *stabilizing controller* is based on the design in [3], and implemented as the SmartAHS component `FlowController`. This controller determines the commanded velocity and lane-change proportions for vehicles in the link. These commands are chosen to stabilize the actual link velocities and lane-change rates about the desired values given by the link optimizer.
- a *communications module*, `LinkTransmitter`. The link transmitter component transmits speed and activity proportion commands (e.g. the proportion of vehicles that should change lane) to the vehicles in a section. For link layer message processing, each car is equipped with a `LinkMonitor`. The link monitor receives the periodic broadcast and processes it. In the case of activity proportions, the link monitor "tosses a coin" to set an activity flag. The coordination supervisor interprets this flag together with the vehicle's state to determine whether to execute the command, which originated from the proportion.
- a *traffic monitor*, `DensityChecker`, that records the vehicle densities and lane change proportions in each section of the link. These measurements are required for the stabilizing control calculations.

Fig. 6 shows a link, divided into several sections, along with the roadside components associated with each section. Using the measured section densities, K_{i-1}, K_i, K_{i+1}, and the lane-change rate matrix, N_i, provided by the traffic monitors, along with the desired density, velocity, and lane change rates, $K_{d,i}$, $V_{d,i}$, $N_{d,i}$, computed by the traffic optimizer, the stabilizing controller for section i periodically calculates commanded velocities and lane-change rates, $V_{c,i}$, $N_{c,i}$, which are broadcast to the vehicles in that section via the `LinkTransmitter`. More detail on the link layer stabilizing controller can be found in [3].

4 Supplementary Accomplishments of MOU 383

In addition to the primary task of developing the dual-scale simulator, MOU383 included a number of secondary tasks, which are described in this section. These tasks were intended to further the development of the SmartCAP and SmartAHS models.

4.1 Extension of SmartCAP Capabilities

The equations underlying the SmartCAP model, and the notation employed in this section are described in Section 3.1.

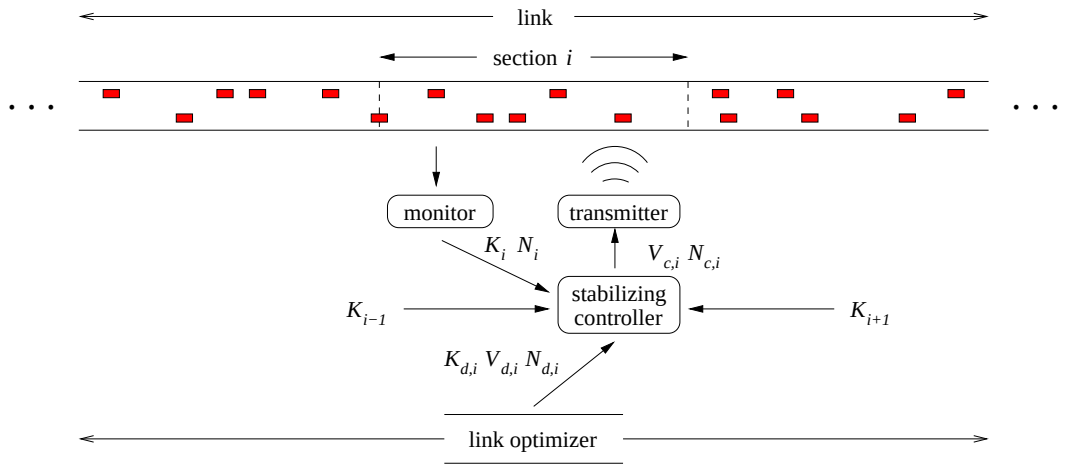

Figure 6: Schematic of SmartAHS link layer components

SmartCAP was originally written for scenarios involving only *independent vehicles*. That is, it only allowed 1-vehicle platoons, or *free agents*. Later developments extended the model to include basic platooning maneuvers such as *join* and *split*, but did not account for several important constraints on the number of maneuvers that can be completed. An improved version of SmartCAP was developed under MOU 383 to more accurately model platooning in the AHS by incorporating constraints on maximum platoon size and the maximum number of completed maneuvers. The update required the definition of an extended state (segregating leaders and followers), and various checks performed on the *activity plan* issued by the TMC (Traffic Management Center). Equations 1 and 2 of Section 3.1 were replaced with Eq. 3 and 4 below:

$$\begin{aligned}
\bar{n}_{(i,j,k)} &= n_{(i,j,k)} + \Pi^{\mathtt{l}}{}_{(r(i),j,k)} n_{(r(i),j,k)} + \Pi^{\mathtt{r}}{}_{(l(i),j,k)} n_{(l(i),j,k)} \\
\bar{n}_{l(i,j,k)} &= n_{l(i,j,k)} + (\Pi^{\mathtt{s}}{}_{(i,j,k)} - \Pi^{\mathtt{j}}{}_{(i,j,k)}) n_{(i,j,k)} + \Pi^{\mathtt{l}}{}_{(r(i),j,k)} n_{l(r(i),j,k)} + \Pi^{\mathtt{r}}{}_{(l(i),j,k)} n_{l(l(i),j,k)} \\
\bar{n}_{f(i,j,k)} &= n_{f(i,j,k)} - (\Pi^{\mathtt{s}}{}_{(i,j,k)} - \Pi^{\mathtt{j}}{}_{(i,j,k)}) n_{(i,j,k)} + \Pi^{\mathtt{l}}{}_{(r(i),j,k)} n_{f(r(i),j,k)} + \Pi^{\mathtt{r}}{}_{(l(i),j,k)} n_{f(l(i),j,k)}
\end{aligned} \tag{3}$$

$$\begin{aligned}
n_{(i,j,k+1)} &= \left(1 - \frac{v_{(i,k)}\Delta t}{L_{(i)}}\right) \bar{n}_{(i,j,k)} + \left(\frac{v_{(u(i),k)}\Delta t}{L_{(u(i))}}\right) \bar{n}_{(u(i),j,k)} \\
n_{l(i,j,k+1)} &= \left(1 - \frac{v_{(i,k)}\Delta t}{L_{(i)}}\right) \bar{n}_{l(i,j,k)} + \left(\frac{v_{(u(i),k)}\Delta t}{L_{(u(i))}}\right) \bar{n}_{l(u(i),j,k)} \\
n_{f(i,j,k+1)} &= \left(1 - \frac{v_{(i,k)}\Delta t}{L_{(i)}}\right) \bar{n}_{f(i,j,k)} + \left(\frac{v_{(u(i),k)}\Delta t}{L_{(u(i))}}\right) \bar{n}_{f(u(i),j,k)}
\end{aligned} \tag{4}$$

The computation of the number of completed maneuvers in the single-lane platooning and multiple-lane platooning cases is explained below.

4.1.1 Single lane platooning

Join and split maneuvers are treated similarly to the original lane change maneuver in SmartCAP. As described in Section 3.1, they are used to compute an *intermediate state* (Eq. 1), which assumes

completion of a portion of the total number of maneuvers requested by the TMC. The intermediate state is then propagated downstream according to the speed determined by the *velocity plan* and the constraint on available space (Eq. 2). The modifications implemented under MOU 383 are related to the estimation of the number of completed joins and splits. Previously it was assumed that all joins and splits requested by the TMC were completed. This led to some unrealistic situations; for example, the number of splits in a section was allowed to exceed the number of followers. The restrictions of the improved version are based on the following assumptions:

1. Each split involves at least one leader and one follower.
2. Each join involves at least two leaders.
3. Every leader can be involved in at most one join or split.
4. The number of joins and splits may not cause the average platoon size to exceed the maximum platoon size.
5. Splits and joins are in *fair competition* for completion as long as the maximum platoon size is not exceeded. If maximum platoon size is exceeded by the TMC command, splits are preferred. Fair competition means that the ratio of completed joins to completed splits equals that of requested joins to requested splits (ρ in Eq. 5).

The problem of estimating the number of completed joins and splits in each section was posed as a constrained optimization. The objective is to maximize the number of completed maneuvers, without exceeding the number of maneuvers requested by the TMC, and conforming to a series of constraints derived from the assumptions listed above. The following notation is used:

For every highway section i, and every time interval k:

$$n_s(i,k) = \sum_j n_s(i,j,k) \quad \ldots \text{Number of completed splits}$$

$$n_j(i,k) = \sum_j n_j(i,j,k) \quad \ldots \text{Number of completed joins}$$

$$t_f(i,k) = \sum_j t_f(i,j,k) \quad \ldots \text{Total number of followers}$$

$$t_l(i,k) = \sum_j t_l(i,j,k) \quad \ldots \text{Total number of leaders}$$

$$d_j(i,k) = \sum_j n(i,j,k)\Pi^{\mathtt{j}}_{\mathtt{TMC}}(i,j,k) \quad \ldots \text{Number of joins requested by the TMC}$$

$$d_s(i,k) = \sum_j n(i,j,k)\Pi^{\mathtt{s}}_{\mathtt{TMC}}(i,j,k) \quad \ldots \text{Number of splits requested by the TMC}$$

In the above definitions, the index j represents vehicle *types*, or different OD pairs sharing the same section. Eq.(5) then represents the problem to be solved for every section i and time interval k (indices i and k have been omitted).

$$
\begin{aligned}
& & & \texttt{max}(n_s + n_j) \\
& \text{subject to :} & & 2n_j + n_s \leq t_l \\
& & & n_s \leq t_f \\
& & & \frac{n_j}{n_s} = \frac{d_j}{d_s} = \rho \\
& & & n_s \geq 0 \\
& & & n_j \geq 0 \\
& & & n_s \leq d_s \\
& & & n_j \leq d_j
\end{aligned}
\tag{5}
$$

Additionally, the solution must be such that the resulting average platoon size (APS) does not exceed the maximum average platoon size (MPS). In SmartCAP, the problem is solved in two steps:

Step 1) Find a solution which ignores the maximum platoon size constraint (i.e. solve Eq.(5)). If this solution yields a feasible average platoon size then it is kept. A solution to problem (5) is given by:

$$
\bar{n}_s = \begin{cases} \texttt{MIN}\left[\frac{t_l}{1+2\rho}\ ,\ t_f\ ,\ d_s\right] & \text{if } d_s \neq 0 \\ 0 & \text{if } d_s = 0 \end{cases}
$$

$$
\bar{n}_j = \begin{cases} \rho \times \bar{n}_s & \text{if } d_s \neq 0 \\ \texttt{MIN}\left[\frac{1}{2}t_l\ ,\ d_j\right] & \text{if } d_s = 0 \end{cases}
$$

The resulting average platoon size is:

$$
\texttt{APS} = \frac{t_l + t_f}{t_l + \bar{n}_s - \bar{n}_j} \overset{?}{\leq} \texttt{MPS}
$$

If $\texttt{APS} \leq \texttt{MPS}$ then the solution is kept. If not, we proceed to Step 2.

Step 2) Because the TMC command results in platoons sizes exceeding MPS, we can assume that some of the joins would have been rejected, and replaced with previously rejected splits. Δ is defined as the number of such replacements, and is found as the minimum of three quantities. $\tilde{n}_s, \tilde{n}_j$ are the corrected numbers of completed splits and joins:

$$
\begin{aligned}
\tilde{n}_s &= \bar{n}_s + \Delta \leq \texttt{MIN}(t_f, d_s) \\
\tilde{n}_j &= \bar{n}_j - \Delta
\end{aligned}
$$

$$
\Delta = \texttt{min}\left[\quad \frac{1}{2}(t_l + \bar{n}_s - \bar{n}_j)\left[\frac{\texttt{APS}}{\texttt{MPS}} - 1\right] \quad ; \quad \texttt{MIN}(t_f, d_s) - \bar{n}_s \quad ; \quad \bar{n}_j \quad \right]
$$

Step 3) The total number of completed maneuvers is now used to compute the proportions of completed maneuvers for each vehicle type. Returning to the fully indexed notation, we have:

$$\Pi^{\mathtt{s}}{}_{(i,j,k)} = \frac{n_s(i,k)}{d_s(i,k)}\, \Pi^{\mathtt{s}}_{\mathtt{TMC}}(i,j,k)$$
$$\Pi^{\mathtt{j}}{}_{(i,j,k)} = \frac{n_j(i,k)}{d_j(i,k)}\, \Pi^{\mathtt{j}}_{\mathtt{TMC}}(i,j,k)$$

4.1.2 Multiple lane platooning

The previous development was extended to multiple lanes by considering two additional maneuvers: lane change left and lane change right. It is assumed that lane changes can be performed by leaders or followers, so there is no need to limit the number of lane changes to the number of free agents (which are currently not explicitly calculated in SmartCAP). Also, it is assumed that these maneuvers can be completed within a single section of the AHS. This requires that followers wishing to change lanes can separate quickly from their platoon with a *short split* maneuver. Thirdly, it is assumed that there is no platoon lane change maneuver.

Additional assumptions are:

1. Each lane change maneuver occupies a platoon (leader) in the origin lane.
2. Whenever the maximum platoon size in the origin lane is not exceeded, all maneuvers are completed in fair competition.
3. If the maximum platoon size is exceeded, a number of joins will be rejected and replaced with other maneuvers, in proportion to their desired quantities.
4. Leader and follower lane changes both cause a loss of a follower in the origin lane and a gain of a leader in the destination lane (i.e. no platoon lane change).
5. The average platoon size used to estimate whether the maximum platoon size will be exceeded as a result of maneuvers is based only on the number of completed joins.

Additional notation:

$n_{l(i,k)} = \sum_j n_{l(i,j,k)}$ ... number of completed splits

$n_{r(i,k)} = \sum_j n_{r(i,j,k)}$ ... number of completed joins

$d_{l(i,k)} = \sum_j n_{(i,j,k)} \Pi_{l(i,j,k)}$ ... number of left lane changes requested by the TMC

$d_{r(i,k)} = \sum_j n_{(i,j,k)} \Pi_{r(i,j,k)}$ ... number of right lane changes requested by the TMC

The optimization problem for the multi-lane case is::

$$\begin{aligned} & \texttt{max}(n_l + n_r + n_s + n_j) \\ \text{subject to :} \quad & n_r + n_l + 2n_j + n_s \leq t_l \\ & n_s \leq t_f \\ & \frac{n_j}{n_s} = \frac{d_j}{d_s} = \rho_1 \\ & \frac{n_r}{n_s} = \frac{d_r}{d_s} = \rho_2 \\ & \frac{n_l}{n_s} = \frac{d_l}{d_s} = \rho_3 \\ & n_s \geq 0 \\ & n_s \leq d_s \end{aligned} \tag{6}$$

Step 1) Again, the MPS constraint has been ignored in (6). Also, the number of lane changes may be further reduced due to insufficient space in the adjacent lane. The maximizing solution to equation (6) is:

$$\bar{n}_s = \begin{cases} \texttt{MIN}\left[\frac{t_l}{1+2\rho_1+\rho 2+\rho 3}\ ,\ t_f\ ,\ d_s\right] & \text{if } d_s \neq 0 \\ 0 & \text{if } d_s = 0 \end{cases}$$

$$\bar{n}_j = \begin{cases} \rho_1 \times \bar{n}_s & \text{if } d_s \neq 0 \\ \beta d_j & \text{if } d_s = 0 \end{cases}$$

$$\bar{n}_r = \begin{cases} \rho_2 \times \bar{n}_s & \text{if } d_s \neq 0 \\ \beta d_r & \text{if } d_s = 0 \end{cases}$$

$$\bar{n}_l = \begin{cases} \rho_3 \times \bar{n}_s & \text{if } d_s \neq 0 \\ \beta d_l & \text{if } d_s = 0 \end{cases}$$

$$\text{with} \quad \beta = \texttt{MIN}\left[\frac{t_l}{d_l + d_r + 2d_j}\ ,\ 1.0\right]$$

Step 2) Again we check the APS condition:

$$\texttt{APS} = \frac{t_l + t_f}{t_l - \bar{n}_j} \overset{?}{\leq} \texttt{MPS}$$

If so, the solution is kept.

Step 3)

If it is violated, it is again assumed that a number Δ of joins will be aborted, and replaced with splits and lane changes. The proportions in which Δ is distributed among other maneuvers are denoted

γ_s, γ_l and γ_r.

$$\begin{aligned}
\tilde{n}_j &= \bar{n}_j - \Delta \geq 0 \\
\tilde{n}_s &= \bar{n}_s + \gamma_s \Delta \leq \texttt{MIN}(t_f, d_s) \\
\tilde{n}_r &= \bar{n}_r + \gamma_l \Delta \leq d_r \\
\tilde{n}_l &= \bar{n}_l + \gamma_r \Delta \leq d_l
\end{aligned}$$

with

$$\gamma_s = \frac{d_s}{d_s + d_l + d_r} \qquad \gamma_r = \frac{d_r}{d_s + d_l + d_r} \qquad \gamma_l = \frac{d_l}{d_s + d_l + d_r}$$

Δ is computed as the minimum of five quantities, with similar interpretations as in the single-lane case:

$$\Delta = \texttt{min}\left[\bar{n}_j - t_l + \frac{1}{\texttt{MPS}}(t_l + t_f) \quad ; \quad \bar{n}_j \quad ; \quad \frac{1}{\gamma_s}\left(\texttt{MIN}(t_f, d_s) - \bar{n}_s\right) \quad ; \quad \frac{1}{\gamma_l}(d_l - \bar{n}_l) \quad ; \quad \frac{1}{\gamma_r}(d_r - \bar{n}_r)\right]$$

Step 3 in the multi-lane case is the same as in the single-lane case.

The implementation of these restrictions on the allowable number of joins and splits required explicit computation of the numbers of leaders and followers in the simulation. This was not available in previous versions of SmartCAP. The SmartCAP state was therefore expanded, and now includes separate variables for leaders and followers, which can be tracked by the user with the following user-functions:

```
double GetNLead(  char *secName, int laneOffset, char *thetaName );
double GetNFollow(char *secName, int laneOffset, char *thetaName );
double GetNFreeA( char *sec‘Name, int laneOffset, char *thetaName );
```

4.2 Extension of SmartAHS Capabilities

As part of the SmartAHS development task, an alternative set of simplified AHS components was developed in order to increase run-time speed, and thus allow more vehicles in the SmartAHS window. Under certain conditions, the simplified set produces identical results as the full components, and increases simulation speed 4 to 5 fold. Full components can still be used on particular vehicles when detailed analysis of a particular function is required. The new components include:

- **2D vehicle/roadside environment processor (VREP):** Each vehicle has associated with it a Vehicle Roadway Environment Processor (VREP) that calculates, at each time step, the 3D position and orientation of the vehicle relative to the road [4]. The simplified VREP identically matches its 3D counterpart in the case of flat highways (i.e. no inclined ramps or banking curves).
- **Simple radars and sensor environment processor (SEP):** The original SmartAHS library contains elements that enable the modeling of vehicle radar in addition to other sensors. The

`RadarSensor` component detects the nearest vehicle, using a radar sensor model which depends on parameters *fov, minRange*, and *maxRange*, i.e. the field of view angle and minimum and maximum detection ranges, and computes the range and range rate of the detected vehicle. In order to prevent the sensors from having to search over the set of all AHS vehicles to determine which ones are in range, the highways are divided into subsections called cells, and a vehicle's sensors restrict their searches to the cells immediately surrounding the vehicle. The Sensor Environment Processor (SEP) is responsible for keeping track of the vehicles that are currently in each cell [4].

The simplified front radar uses a lookup table to detect the front and back vehicles, and an algorithm for updating the table when lane changes occur. The side radar remains the same, but is only activated when needed for changing lanes. Use of these simple radars obviates the need for an SEP, which is the most computationally intensive component of the software. Use of the simplified sensors is highly recommended whenever more detailed analysis, such as sensor fault detection, is not needed.

- **Regulation layer controllers:** As described in Sec. 3.2, each regulation layer controller has an associated acceleration bounds automaton, e.g. the acceleration bound component for the merge controller. This automaton carries out computations to ensure that the velocity, acceleration, and jerk of a vehicle remain within specified limits [11].

  Alternative versions of the regulation layer controllers, such as the merge and split maneuver controllers, have been developed for this project to reduce computations by removing the comfort constraint on the applied jerk. This represents a significant reduction in computations, and yields nearly identical results in most cases.

The components corresponding to each of these extensions to SmartAHS are listed in Table 1.

Type Name	SHIFT Filename	Description
VREP	vrep2D.hs	2D vehicle-roadway environment processor
SimpleRadar	simpleradar.hs	simplified radar
LeadController	lead_controller.hs	lead maneuver controller
LeadRegion	lead_region.hs	state-space region tracker for lead maneuver
MergeController	merge_controller.hs	merge maneuver controller
MergeRegion	merge_region.hs	state-space region tracker for merge maneuver
SplitController	split_controller.hs	split maneuver controller
SplitRegion	split_region.hs	state-space region tracker for split maneuver

Table 1: Modified SmartAHS components

5 Design of the meso-micro simulator

This section describes the methodology that was followed for combining SmartCAP and SmartAHS. The approach is based on the concept of the μ-window, which is the user-defined portion of the AHS network to be simulated in microscopic detail (see Figure 7). The domain of SmartAHS comprises the μ-window along with some auxiliary sections called *transition zones*. The remainder of the network is controlled by SmartCAP. Many aspects of the design are related to the generation of consistent

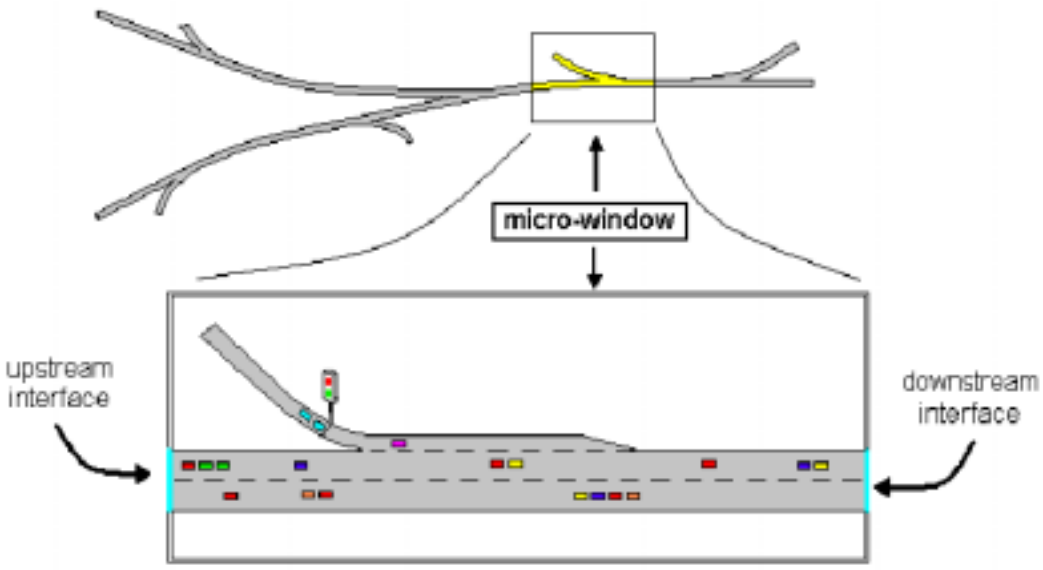

Figure 7: μ-window

boundary conditions for the μ-window. The computation of these boundary conditions takes place at the beginning of every meso-time step (approximately every 10 seconds), and is based on a prediction of the mesoscopic traffic state for the upcoming time interval. A correction to the mesoscopic state, designed to compensate for the differences between the predicted and actual boundary flows, is outlined in Section 5.4. The various tasks of the meso-micro simulator are coordinated by a SHIFT scheduling component shown in Figure 8. These tasks are listed below with labels that are used for referencing throughout this section.

T0: Run SmartAHS / Pause SmartCAP for 1 meso-time interval.
T1: Consistency checks: Use recorded boundary flows to correct the meso-state.
T2: Update the link layer velocity and activity plans for the upcoming meso-time interval using the corrected state.
T3: Compute predicted intermediate meso-state.
T4: Transfer intermediate states to SmartAHS.
T5: Record position of the last vehicle (real or ghost).
T6: Kill all remaining ghost platoons in the upstream transition zone (UTZ).
T7: Run the platoon placement algorithm using the predicted intermediate meso-state.
T8: Create ghost platoons with initial conditions given by the previous step.
T9: Complete meso-state update for the upcoming time interval.

5.1 Boundary requirements for the μ-window

One of the main design issues for the meso-micro simulator was the selection of boundary conditions for the μ-window. These boundary conditions represent the link between the mesoscopic and microscopic traffic states. They must therefore be chosen in a way that preserves compatibility between the two models. The following requirements were established as criteria for choosing the boundary conditions:

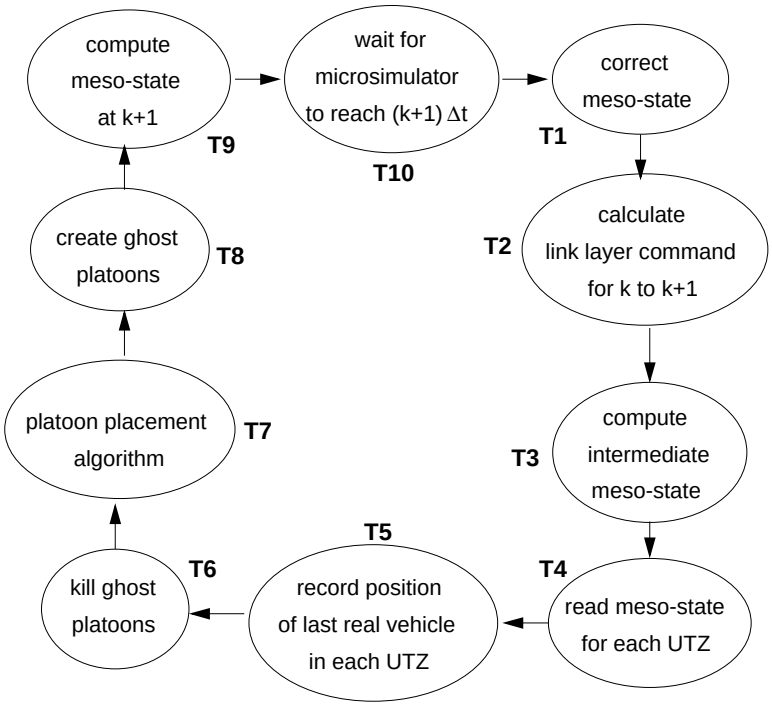

Figure 8: Scheduler

1. They should be such that all density states are conserved. In particular, total numbers of vehicles for every vehicle type, and proportions of leaders and followers (i.e. average platoon sizes) should be preserved in the transition through a μ-window boundary. This is equivalent to requiring that (over every meso-time interval) average flows of leaders and followers, and of each vehicle type, should agree in the meso- and micro- environments.

2. Microscopic vehicles generated at the upstream boundary should be *safe* (as defined in [1]) with respect to other vehicles inside the μ-window.

3. Backward moving waves of congestion should travel uninterrupted over upstream and downstream boundaries of the μ-window.

An example of a possible but inadequate boundary condition would be to equate meso- and micro-upstream flows by generating microscopic vehicles at the upstream boundary at regular time intervals dictated by the predicted mesoscopic flow. For example, if the density in the section immediately preceding the μ-window were 18 veh/km, traveling at a predicted speed of 100 km/hr, the predicted flow into the μ-window over the upcoming time interval would be 0.5 veh/sec. This could be generated by placing one vehicle at the upstream boundary of the μ-window every 2 seconds. The type and maneuver (leader or follower) of each vehicle would depend on the proportions of the preceding section. However, this solution may in some cases (e.g. if there is congestion inside the μ-window) lead to a violation of the *safety* requirement for the upstream boundary condition. This is because vehicles are being placed in the μ-window under the assumption that densities and speeds inside it are evenly distributed.

The boundary conditions implemented in the meso-micro simulator satisfy the requirements outlined above. They are:

- At the beginning of the mesoscopic time interval, set the microscopic densities (leaders, followers, and all types) in the section immediately upstream of the μ-window to the values given by mesoscopic state. Set the target velocity for these vehicles equal to the predicted mesoscopic velocity.
- Set the target speed of microscopic vehicles in the section immediately downstream of the μ-window equal to the predicted average mesoscopic speed.

These boundary conditions do not directly dictate the values of the flows at the boundaries, but instead specify densities and velocities in the sections upstream and downstream of the μ-window. This avoids the possibility of violating the safety requirement. Two auxiliary sections are needed to accommodate these boundary conditions. These are the *upstream transition zone* (UTZ) and the *downstream transition zone* (DTZ). They are described in detail in Sections 5.2.2 and 5.3, and are represented schematically in Fig. 9. Fig. 9(a) shows a μ-window bounded by upstream and downstream SmartCAP sections, while Fig. 9(b) shows the topological elements that comprise the mesoscale and microscale regions of Fig. 9(a).

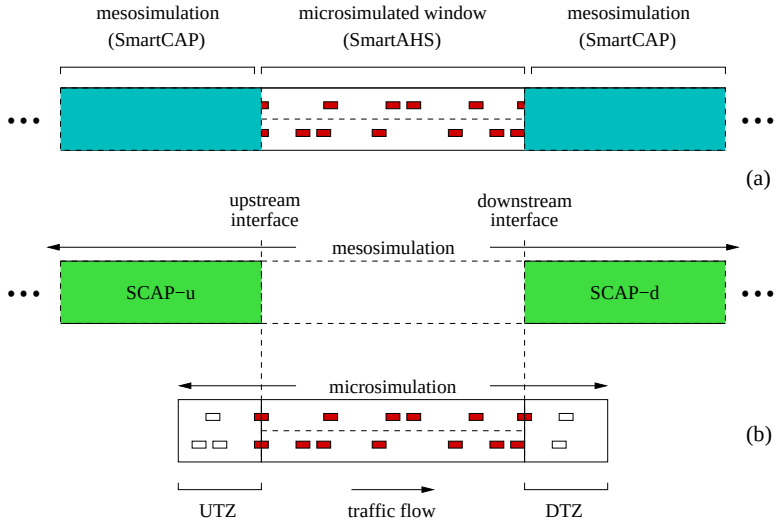

Figure 9: (a) Conceptual drawing and (b) main topological elements of meso-micro simulator

5.2 Elements of the Upstream interface

Vehicles crossing the upstream boundary of the μ-window are influenced by traffic both inside and outside of the μ-window. Their nominal speed is dictated by the upstream section, but may be diminished if traffic is congested inside the μ-window. Furthermore, the backward propagation speed of congestion is determined by the particular regulation layer control laws being used. The *upstream transition zone* (UTZ) is designed to provide space for arranging vehicles with initial speeds and velocities that are within the safety bounds of the regulation layer control laws. The total densities placed in the UTZ are given by the *intermediate meso-state* (Eq. 1, task **T3**). This is the state assuming that all maneuvers for the coming time interval have been completed, but no vehicles have moved downstream. The particular arrangement of vehicles in the UTZ (initial positions and platoon sizes) is calculated by the Platoon Placement Algorithm (PPA, task **T7**), which is explained in Section 5.2.2. While in the UTZ, vehicles are not allowed to perform any maneuvers, since these

are assumed to be completed in the intermediate meso-state. If entry into the μ-window is not impeded by congestion, the flow measured across the upstream boundary in both meso and micro environments should be approximately equal, with differences due to rounding and randomness in the initial vehicle distribution. These differences are cancelled by the meso-state correction task (**T1**). Vehicles in the UTZ are of a simple type called the *ghost platoon*, which lacks many of the AHS vehicle components. A description of this SHIFT type is given below.

5.2.1 Ghost Platoons

The primary reason for using a specialized vehicle type in the UTZ, instead of the fully-equipped SmartAHS vehicle type, is computational efficiency. The total number of vehicles simulated is reduced by replacing each platoon with a single rigid ghost platoon. Another reason is that there is no easy way in the SHIFT language to delete the subset of vehicles remaining in the UTZ at the end of the meso-time interval (task **T6**).

A single ghost-platoon component contains all the information needed to create a platoon of real vehicles (i.e. number, type and order of vehicles in the platoon), but unlike a standard SmartAHS vehicle, a ghost platoon lacks detailed regulation and coordination layers. It travels along the UTZ, controlled by the standard regulation layer *leader* law, and materializes when it enters the μ-window. Because it is important to avoid maneuvers taking place inside the UTZ, leaders are not allowed to request maneuvers until their entire platoon has crossed the boundary.

The behavior of a typical ghost platoon is summarized in Figure 10.

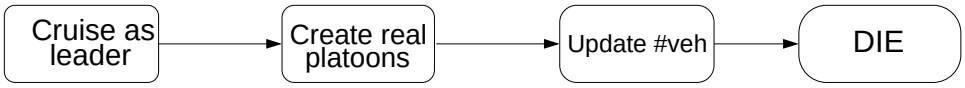

Figure 10: Schematic of *ghost platoon* evolution

A ghost platoon has the following state logic and rules:

1. A ghost platoon is created in the UTZ to represent one platoon, and is modeled with a simple vehicle dynamics and a front sensor, a leader control law and basic information for that platoon, such as the ordering and types of vehicles. The length of the ghost platoon is equal to the length of the platoon it represents;

2. Once a ghost platoon enters the first SmartAHS section in the μ-window, it is converted into a real platoon according to its stored platoon information;

3. Ghost platoons have enough communication structure to deny any requests for maneuvers. For example, a real vehicle in the μ-window could request a changelane. The ghost platoon will ignore this request;

4. Once the real platoon has been created, its corresponding ghost platoon destroys itself;

5. Once the SmartCAP simulation time interval has elapsed, **all** ghost platoons have to be killed wherever they are located in the UTZ or μ-window, and at the same time, the position and velocity of the real vehicles farthest downstream in the UTZ are recorded to be used in the next ghost platoon placement cycle;

6. Ghost platoons are not capable of carrying out any coordination-level maneuvers.

5.2.2 Platoon Placement Algorithm (T7)

This algorithm translates the upstream meso-simulated densities into a list of discrete SmartAHS vehicles, in a way that preserves the number of leaders and followers and the average platoon sizes. It also produces initial positions for each platoon within the UTZ, and a randomized arrangement of vehicles within each platoon.

The main steps of the algorithm are:

1. Check for available space using the intermediate number of leaders and followers in the upstream section $(\bar{n}_{l(u,j,k)}, \bar{n}_{f(u,j,k)})$ and the velocity $(v_{(u,k)})$. These determine the integer numbers of leaders and followers $(n^a_{l\,(u,j,k)}, n^a_{f\,(u,j,k)})$ to be placed in the UTZ.

2. Platoon sizes: The average platoon size for the intermediate state is:

$$\mathtt{APS}_{(u,k)} = \frac{\sum_j \left(\bar{n}_{f(u,j,k)} + \bar{n}_{l(u,j,k)}\right)}{\sum_j \bar{n}_{l(u,j,k)}}$$

   A total of $P = \sum_j n^a_{l\,(u,j,k)} + n^a_{f\,(u,j,k)}$ vehicles must be arranged in $\sum_j n^a_{l\,(u,j,k)}$ platoons, with an average size approximating $\mathtt{APS}_{(u,k)}$. Defining $\bar{\mathtt{APS}}_{(u,k)} = \mathrm{trunc}(\mathtt{APS}_{(u,k)})$, the following rule is used to choose a platoon size PS_n for the n^{th} platoon in the set:

$$PS_n = \begin{cases} \bar{\mathtt{APS}}_{(u,k)} & P - \sum_{j=1}^{n-1} PS_j \geq \bar{\mathtt{APS}}_{(u,k)} \\ P - \sum_{j=1}^{n-1} PS_j & 0 \leq P - \sum_{j=1}^{n-1} PS_j < \bar{\mathtt{APS}}_{(u,k)} \\ 0 & P - \sum_{j=1}^{n-1} PS_j = 0 \end{cases}$$

3. Vehicle arrangement in platoons: Once a size has been determined for each platoon, vehicle types are selected to create the platoons. The first vehicle in the platoon is randomly chosen from the leader types, and the rest from the pool of followers.

4. Ghost platoon spacing: once platoon size and vehicle arrangements have been determined, initial positions for each platoon within the UTZ are calculated. The position of the trailing real vehicle inside the UTZ determines the amount of space available for placing ghost platoons. The space required by ghost platoons is the sum of the physical length of vehicles and the headways required by leaders and followers to ensure safety. This space is subtracted from the total available space to calculate the free space. This amount is then distributed randomly among platoons.

5.3 Elements of the Downstream interface

At the downstream interface, vehicles are phased out of the microsimulation and transformed into an equivalent mesoscopic flow. The transformation is carried out in a smooth, computationally efficient manner by replacing microscale vehicles with *ghost vehicles*. Ghost vehicles are used in order to:

1. increase simulation speed, by reducing the number of SmartAHS components in simulation, as soon as vehicles exit the μ-window;

2. provide continuity between micro and mesosimulated regions. The ghost vehicles track the average velocity of the downstream mesosimulated flow. In addition, the positions and velocities of ghost vehicles are detected by the radar of vehicles in the μ-window. Since the ghost vehicles travel at the same speed as the downstream SmartCAP flow, and the vehicles in the μ-window maintain safe spacing behind the ghost vehicles, downstream flow conditions (e.g. traffic jams) can be propagated backward into the μ-window.

The process of removing microsimulated vehicles from the simulation environment is illustrated by the example of Fig. 11. In the top drawing, vehicle $v1$ has exited the μ-window and has just crossed into the DTZ. The stages that follow are:

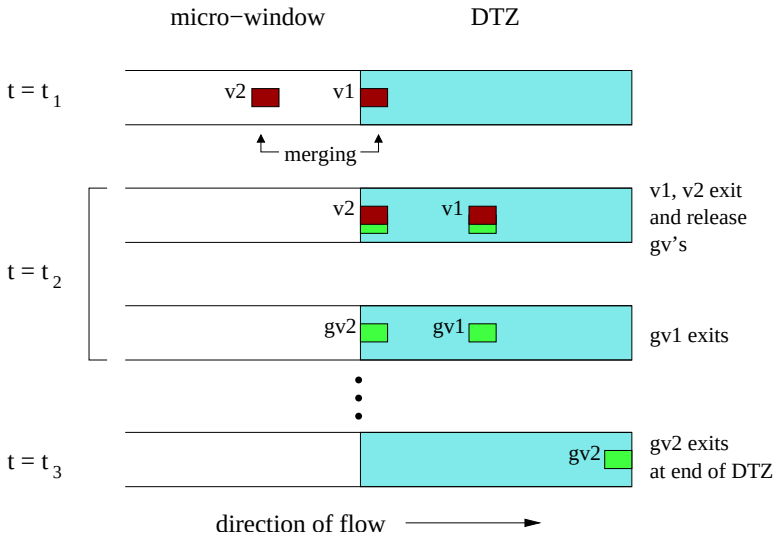

Figure 11: Example: downstream interface

1. $v1$ travels downstream through the DTZ until all its platoon members and/or upstream maneuver partners are inside the DTZ. An upstream maneuver partner of $v1$ is any vehicle upstream of $v1$ that is currently engaged in a pairwise maneuver with $v1$. In Fig. 11, $v2$ is merging with $v1$, thus $v2$ is the upstream maneuver partner of $v1$. If $v1$ were to disappear, with $v2$ still inside the μ-window, this would cause referencing errors for $v2$. Thus, the maneuver pair is kept intact until it crosses into the DTZ, where both partners can be removed safely and simultaneously from the microsimulation.

2. When both $v1$ and $v2$ are inside the DTZ, they spawn *ghost vehicles* ($gv1$ and $gv2$) before deleting themselves from the simulation environment. $gv2$ remains, but $gv1$ disappears since only one ghost vehicle per lane (in this case $gv2$) is needed to preserve the car-following behavior of vehicles in the μ-window. These events take place within time step t_2.

3. $gv2$ travels along the DTZ until (a) it reaches the downstream end, as in Fig. 11, or (b) another ghost appears upstream in the DTZ. In either case, $gv2$ is removed from the simulation.

Flow characteristics are preserved across the downstream micro-meso boundary. The average velocity and flow rate of vehicles exiting the μ-window are used as inlet parameters for the downstream mesosimulated region.

5.4 Consistency enforcing adjustments (T1)

As mentioned in Section 5.1, one of the requirements on the μ-window is that the variable representing the flow across either of its boundaries have the same value in both the meso- and micro-environments. These flows are depicted in Figure 12 with the following notation:

$q^{\texttt{CAP}}(u,j,k\text{–}1)$ ... # of vehicles in SmartCAP that crossed upstream boundary during time interval $k-1$
$q^{\texttt{AHS}}(u,j,k\text{–}1)$ ... # of vehicles in SmartAHS that crossed upstream boundary during time interval $k-1$
$q^{\texttt{CAP}}(d,j,k\text{–}1)$ ... # of vehicles in SmartCAP that crossed downstream boundary during time interval $k-1$
$q^{\texttt{AHS}}(d,j,k\text{–}1)$ ... # of vehicles in SmartAHS that crossed downstream boundary during time interval $k-1$

However, as the task scheduler reaches the end of state **T0**, it is not necessarily the case that $q^{\texttt{CAP}}(u,j,k\text{–}1) = q^{\texttt{AHS}}(u,j,k\text{–}1)$. There are two sources of discrepancy:

- The conversion of real densities to integer numbers of vehicles in the PPA.
- The SmartCAP assumption that density is distributed uniformly within each section.

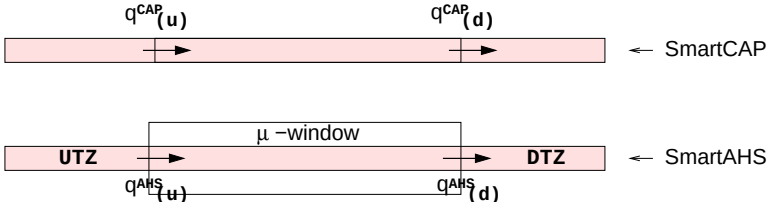

Figure 12: Traffic flow across μ-window boundaries

The second source might become significant if, for example, a vehicle stops inside the μ-window, causing a temporary queue to form near the upstream boundary. If the μ-window is relatively empty, the SmartCAP speed prediction for the previous section is high. This translates (through the PPA) into large initial speeds in the UTZ, and a large predicted flow between the UTZ and the μ-window by SmartCAP. However, the flow measured by SmartAHS after the time interval has concluded is small, due to the blockage. This discrepancy also affects the densities in the section upstream of the window, and in this case results in an underestimation of the true speed of growth of the queue.

The *consistency enforcing adjustments* to the meso-state described in this section are based on the assumption that the flows measured in SmartAHS are more precise than the prediction made by SmartCAP. This is the case in the example described previously, where the assumption of uniformly distributed densities in SmartCAP caused an overestimation of the upstream boundary flow. The SmartAHS version of the boundary flows is preferred over the SmartCAP prediction because the representation of traffic within the μ-window is more precise in SmartAHS than in SmartCAP.

Replacing the SmartCAP flow prediction with its SmartAHS counterpart implies additional adjustments to the densities and velocities in the sections upstream and downstream of the μ-window. These are given in Eq.(7) through (10). The $\leftarrow$ symbol is used to denote that the variable on the left-hand-side *takes the vale* given by the right-hand-side. Also, the variables defined at the beginning of this section are further divided into leaders and followers by adding subscript l's and f's:.

Upstream density adjustments :

$$\begin{aligned} n^{\texttt{CAP}}(u,j,k) &\leftarrow n^{\texttt{CAP}}(u,j,k) + q^{\texttt{CAP}}(u,j,k\text{–}1) - q^{\texttt{AHS}}(u,j,k\text{–}1) \\ n_l^{\texttt{CAP}}(u,j,k) &\leftarrow n_l^{\texttt{CAP}}(u,j,k) + q_l^{\texttt{CAP}}(u,j,k\text{–}1) - q_l^{\texttt{AHS}}(u,j,k\text{–}1) \\ n_f^{\texttt{CAP}}(u,j,k) &\leftarrow n_f^{\texttt{CAP}}(u,j,k) + q_f^{\texttt{CAP}}(u,j,k\text{–}1) - q_f^{\texttt{AHS}}(u,j,k\text{–}1) \end{aligned} \tag{7}$$

Upstream speed adjustments :

$$v^c(u,k\text{–}1) \leftarrow \frac{\sum_j q^{\texttt{AHS}}(u,j,k\text{–}1)}{\sum_j \bar{n}(u,j,k)} \left(\frac{L(u)}{\Delta t} \right) \tag{8}$$

Downstream density adjustment :

$$\begin{aligned} n^{\texttt{CAP}}(d,j,k) &\leftarrow n^{\texttt{CAP}}(d,j,k) - q^{\texttt{CAP}}(d,j,k\text{–}1) + q^{\texttt{AHS}}(d,j,k\text{–}1) \\ n_l^{\texttt{CAP}}(d,j,k) &\leftarrow n_l^{\texttt{CAP}}(d,j,k) - q_l^{\texttt{CAP}}(d,j,k\text{–}1) + q_l^{\texttt{AHS}}(d,j,k\text{–}1) \\ n_f^{\texttt{CAP}}(d,j,k) &\leftarrow n_f^{\texttt{CAP}}(d,j,k) - q_f^{\texttt{CAP}}(d,j,k\text{–}1) + q_f^{\texttt{AHS}}(d,j,k\text{–}1) \end{aligned} \tag{9}$$

Boundary flow adjustments :

$$\begin{aligned} q^{\texttt{CAP}}(u,j,k\text{–}1) &\leftarrow q^{\texttt{AHS}}(u,j,k\text{–}1) \\ q_l^{\texttt{CAP}}(u,j,k\text{–}1) &\leftarrow q_l^{\texttt{AHS}}(u,j,k\text{–}1) \\ q_f^{\texttt{CAP}}(u,j,k\text{–}1) &\leftarrow q_f^{\texttt{AHS}}(u,j,k\text{–}1) \\ q^{\texttt{CAP}}(d,j,k\text{–}1) &\leftarrow q^{\texttt{AHS}}(d,j,k\text{–}1) \\ q_l^{\texttt{CAP}}(d,j,k\text{–}1) &\leftarrow q_l^{\texttt{AHS}}(d,j,k\text{–}1) \\ q_f^{\texttt{CAP}}(d,j,k\text{–}1) &\leftarrow q_f^{\texttt{AHS}}(d,j,k\text{–}1) \end{aligned} \tag{10}$$

The adjustments to the boundary flows should take place after the other three in order for the values of the SmartCAP flows to correspond to those used at the previous time step. To implement these equations, the μ-window scheduler component keeps track of the numbers of leaders and followers that enter and exit its boundaries. This information is passed via *foreign functions* to a C-based code that implements the meso-state adjustments in SmartCAP. The three C functions involved are `AHStoSC_upflow()`, `AHStoSC_corrupvel()`, and `AHStoSC_dnflow()`. They can be found in `sc.c` in the SmartCAP library.

6 An example

A simple test scenario is presented here to illustrate the use of the input and output user interfaces. The test network is shown in Figure 13. It is a straight one-lane highway with a bifurcation. The upstream boundary is supplied with a flow of 1800 vph, divided equally among two vehicle types; f1 takes the left branch and f2 the right. At the bifurcation, vehicles of type f2 must perform a lane change maneuver. A μ-window was placed at the bifurcation to focus on these lane change maneuvers.

Three input files are needed to set up the meso-micro simulation:

1) The SmartCAP input file (`SCexample`) containing the details of the highway geometry and traffic demands. A μ-window spanning sections `hw_16` to `hw_20` is created as follows:

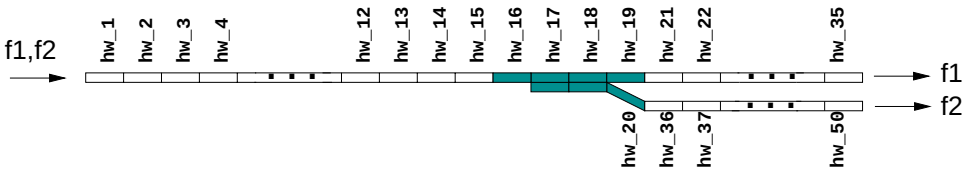

Figure 13: Test network

```
//
HIGHWAY hw_16
LANE 1 AUTOMATED NONE_BARRIER MMUP
GEOMETRY LINE 500
//
HIGHWAY hw_17
LANE 1 AUTOMATED NONE_BARRIER MM
LANE 2 AUTOMATED NONE_BARRIER MM
GEOMETRY LINE 500
//
HIGHWAY hw_18
LANE 1 AUTOMATED NONE_BARRIER MM
LANE 2 AUTOMATED NONE_BARRIER MM
GEOMETRY LINE 500
//
HIGHWAY hw_19
LANE 1 AUTOMATED NONE_BARRIER MMDN
GEOMETRY LINE 500
//
HIGHWAY hw_20
LANE 1 AUTOMATED NONE_BARRIER MMDN
GEOMETRY LINE 500
//
```

The two flow types (f1 and f2) are also defined in the SmartCAP input file. The following lines determine their origin, destination and demand flows (4 sec/veh = 900 veh/hr):

```
SECTION FLOW
//   FlowTypeName   CarType     DestHwName DestLane
FLOW      f1        passenger   hw_35        1
FLOW      f2        passenger   hw_50        1
//
SECTION        INFLOW
//          EntryHwName  EntryLane
INFLOW      hw_1         1
//      flowType   startTime   endTime  interval(sec/veh)   weight  activity
FLOW    f1         0           3600      4.00                1.0    automated
FLOW    f2         0           3600      4.00                1.0    automated
//
```

2) The TMC velocity plan is coded in `VEPexample.c` by modifying `VelocityPlan()`. In this example, the link layer velocity command is maximum speed (`GetVmax`) throughout the highway:

```
void VelocityPlan()
{
  char secName[28];
  int secId, lane;
  for ( secId=1; secId<=50; secId++ )
  {
     sprintf(secName,"hw_%d",secId);
     lane = 1;
     SetVdesired(secName,lane, GetVmax());

     if( secId==17 | secId==18 )
     {
        lane = 2;
        SetVdesired(secName,lane, GetVmax());
     }
  }
}
```

3) The activity plan in this example uses only *cruise* and *lane change right* maneuvers. Vehicles of type f2 were assigned a 100% likelihood of changing lanes in sections `hw_17` and `hw_18`. This was done by modifying `ActivityPlan()` in `TMCexample.c`:

```
void ActivityPlan()
{
  int    secId, lane;
  char   secName[20];

  % -- Activity plan for type "f1" (all cruise) -----
  for ( secId=1; secId<=50; secId++ )
  {
      sprintf(secName,"hw_%d",secId);
      lane = 1;
      SetPi(secName, lane, "f1", "join"   , 0.0);
      SetPi(secName, lane, "f1", "split"  , 0.0);
      SetPi(secName, lane, "f1", "lcleft" , 0.0);
      SetPi(secName, lane, "f1", "lcright", 0.0);
      SetPi(secName, lane, "f1", "cruise" , 1.0);
      if ( secId ==17 | secId==18 )
      {
        lane = 2;
        SetPi(secName, lane, "f1", "join"   , 0.0);
        SetPi(secName, lane, "f1", "split"  , 0.0);
        SetPi(secName, lane, "f1", "lcleft" , 0.0);
        SetPi(secName, lane, "f1", "lcright", 0.0);
```

```
      SetPi(secName, lane, "f1", "cruise" , 1.0);
    }
  }

  % Activity plan for type "f2" ------------------
  for ( secId=1; secId<=50; secId++ )
  {
    sprintf(secName,"hw_%d",secId);
    if ( secId ==17 | secId==18 )
    {
      lane = 1;
      SetPi(secName, lane, "f2", "join"   , 0.0);
      SetPi(secName, lane, "f2", "split"  , 0.0);
      SetPi(secName, lane, "f2", "lcleft" , 0.0);
      SetPi(secName, lane, "f2", "lcright", 1.0);
      SetPi(secName, lane, "f2", "cruise" , 0.0);
      lane = 2
      SetPi(secName, lane, "f2", "join"   , 0.0);
      SetPi(secName, lane, "f2", "split"  , 0.0);
      SetPi(secName, lane, "f2", "lcleft" , 0.0);
      SetPi(secName, lane, "f2", "lcright", 0.0);
      SetPi(secName, lane, "f2", "cruise" , 1.0);
    }
    else
    {
      lane = 2;
      SetPi(secName, lane, "f2", "join"   , 0.0);
      SetPi(secName, lane, "f2", "split"  , 0.0);
      SetPi(secName, lane, "f2", "lcleft" , 0.0);
      SetPi(secName, lane, "f2", "lcright", 0.0);
      SetPi(secName, lane, "f2", "cruise" , 1.0);
    }
  }
}
```

This completes the definition of input. The next step is to compile the program using `mmface`. The configuration file for mmface (`mmexample`) should contain the appropriate directories and filenames:

```
#
MESOMICROINTERFACE tmc_c_File
/mminterface/src/TMCexample.c
#
MESOMICROINTERFACE vep_c_File
/mminterface/src/VEPexample.c
.
.
.
MESOMICROINTERFACE SCinput_Input
```

```
/mminterface/SCexample
#
```

The meso-micro simulator is compiled by typing:

```
> mmface mmexample
```

The actual simulation can be executed with either the command line debugger or with TkShift. The simulation output consisted of 48 *vehicle* files (e.g. vehicle12.m), 2 SmartCAP output files (`SCexample_d.m` and `SCexample_sv.m`), and `hinfo.m`. These were placed, along with `SCexample.inp`, generated by `mmface`, in a data directory called `\mmplot\ExampleData\`. This information was provided to `mmplot.m`:

```
basedir = 'c:\mmplot\';
datadir = [basedir 'ExampleData'];
ahstopol = [datadir '\SCexample.inp'];
captopol = [datadir '\SCexample'];
numveh=48;
```

Figure 14 shows the graphic output of `mmplot`. The mesoscopic state is illustrated in the top half of the figure, which clearly shows a wave of uncongested traffic flow moving through the μ-window. The bottom half of the figure shows a microscopic view of the μ-window, which can be used to observe lane changes in more detail.

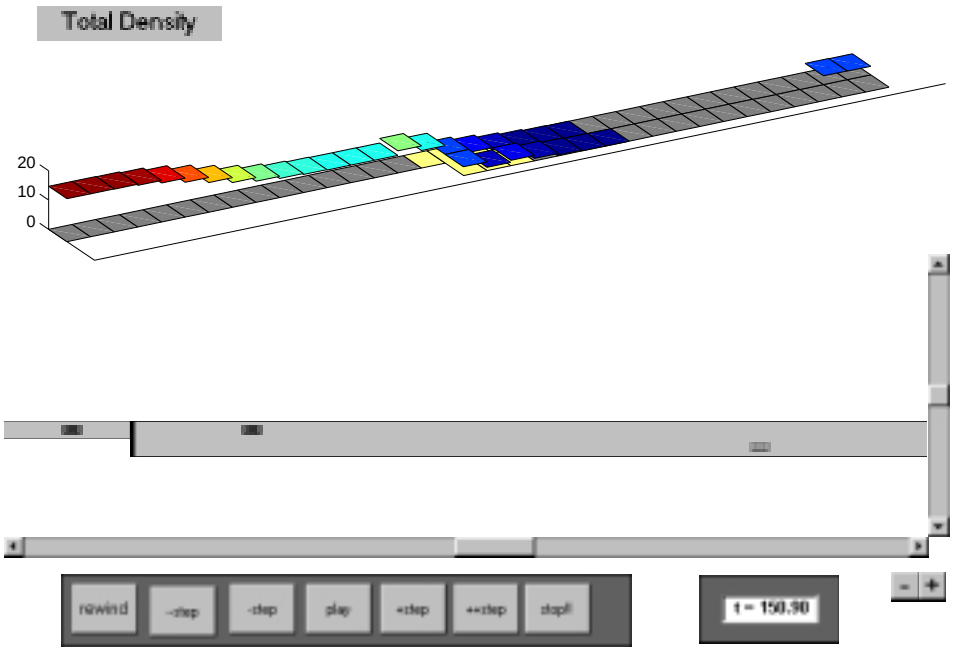

Figure 14: Graphical user interface

References

[1] L. Alvarez and Horowitz. Safe Platooning in Automated Highway Systems. PATH Technical Reports UCB-ITS-PRR-97-46, Institute of Transportation Studies, University of California, Berkeley, 1997.

[2] Luis Alvarez. *Automated Highway Systems: Safe Platooning and Traffic Flow Control*. PhD thesis, Department of Mechanical Engineering, University of California at Berkeley, 1996.

[3] Luis Alvarez and Roberto Horowitz. Traffic flow control in automated highway systems. Technical Report UCB-ITS-PRR-97-47, 1997.

[4] Marco Antoniotti *et al. SmartAHS User's Manual*. California PATH, Berkeley, California, 1998. available at website, *http://www.path.berkeley.edu/smart-ahs/sahs-manual/manual.html*.

[5] M. Broucke, P. Varaiya, M. Kourjankski, and D. Khorramabadi. Smartcap User's Guide. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, May 1996.

[6] Mireille Broucke and Pravin Varaiya. A theory of traffic flow in automated highway system. *Transportation Research, Part C: Emerging Technologies*, 4(4):181–210, 1996.

[7] Akash R. Deshpande, Aleks Göllü, and Luigi Semenzato. SHIFT *Reference Manual*. California PATH, Berkeley, CA, 1997.

[8] Farokh Eskafi. *Modeling and Simulation of the Automated Highway Systems*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1996.

[9] Farokh Eskafi. Communication Structure for Automated Highway System. California PATH presentation slides, 1998.

[10] Gabriel Gomes, Luis Alvarez, and Roberto Horowitz. Traffic flow patterns in AHS: system user optimals. In *American Controls Conference*, 2000.

[11] R. Horowitz, S. Sastry, P. Varaiya, L. Alvarez, K. Lueng, C. Toy, and D. Gulick. Emergency Vehicle Maneuvers and Control Laws for Automated Highway Systems. PATH Technical Reports to Caltrans 98-C2, Institute of Transportation Studies, University of California, Berkeley, 1998.

[12] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya. Protocol Design for an Automated Highway System. *Discrete Event Dynamic Systems*, 2(1):4–16, 1994.

[13] Perry Li, Luis Alvarez, and Roberto Horowitz. AVHS safe control laws for platoon leaders. To appear in IEEE Transactions on Control Systems Technology, 1997.

[14] Antonia E. Lindsey. *Design, Verification and Simulation of Communication Protocols for a Fault Tolerant Automated Highway System*. PhD thesis, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, 1997.

[15] John Lygeros, Datta Godbole, and Mireille Broucke. A fault tolerant control archiecture for automated highway systems. *IEEE Transactions on Control Systems Technology*, 8(2):205–219, March 2000.

[16] Valerie Murgier. *SmartAHS Components for Communication Simulation*. California PATH, Berkeley, California, 1998. available at website, http://www.path.berkeley.edu/smart-ahs/sahs-manual/communication_devices.html.

[17] California PATH. Smartahs. http://www.path.berkeley.edu/smart-ahs.

[18] D. V. A. H. G. Swaroop. *String Stability of Interconnected Systems: An Application to Platooning in Automated Highway Systems*. PhD thesis, Department of Mechanical Engineering, University of California at Berkeley, 1994.

[19] Charmaine Toy, Luis Alvarez, and Roberto Horowitz. Non-stationary velocity profiles for emergency vehicles on automated highways, part I. To be published. ASME JDSMC.

[20] Charmaine Toy, Luis Alvarez, and Roberto Horowitz. Non-stationary velocity profiles for emergency vehicles on automated highways, part II. To be published. ASME JDSMC.

[21] Charmaine Toy, Kevin Leung, Luis Alvarez, and Roberto Horowitz. Emergency vehicle maneuvers and control laws for automated highway systems. In *Proceedings of the 1998 ASME International Mechanical Engineering Congress and Exposition. Anaheim, CA*, November 1998.

[22] P. Varaiya. Smart Cars on Smart Roads: Problems of Control. *IEEE Transactions on Automatic Control*, 38(2):195–207, 1993.

[23] P. Varaiya and Steven E. Shladover. Sketch of an IVHS systems architecture. Technical Report UCB-ITS-PRR-91-3, Institute of Transportation Studies, University of California, Berkeley, 1991.

[24] J. Yi, A. Howell, R. Horowitz, K. Hedrick, and L. Alvarez. Fault detection and handling for longitudinal control. Technical Report UCB-ITS-PRR-2001-21, PATH MOU 312 Final Report, 2001.