

UC Berkeley

Research Reports

Title

Modeling And Simulation Of The Automated Highway System

Permalink

<https://escholarship.org/uc/item/11m6t11p>

Author

Eskafi, Farokh H.

Publication Date

1996

This paper has been mechanically scanned. Some errors may have been inadvertently introduced.

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

Modeling and Simulation of the Automated Highway System

Farokh H. Eskafi

**California PATH Research Report
UCB-ITS-PRR-96-19**

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.

The contents of this report reflect the views of the authors who are responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official views or policies of the State of California. This report does not constitute a standard, specification, or regulation.

July 1996

ISSN 1055-1425

Modeling and Simulation of the Automated Highway System

by

Farokh Hassanzadeh Eskafi

B.S. (University of California, Berkeley) 1991

M.S. (University of California, Berkeley) 1992

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Engineering-Electrical Engineering
and Computer Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY of CALIFORNIA at BERKELEY

Committee in charge:

Professor Pravin Varaiya, Chair

Professor Shankar Sastry

Professor Roberto Horowitz

1996

Abstract
Modeling and Simulation of the Automated Highway System
by

Farokh Hassanzadeh Eskafi

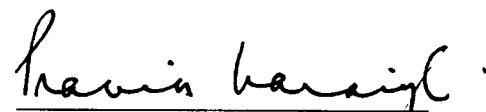
Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences
University of California at Berkeley
Professor Pravin Varaiya, Chair

We present the hierarchical structure for the control design of the Automated Highway System (AHS). This control hierarchy has four layers: network, link, coordination, and regulation. The network layer routes vehicles from their origin to their destination. The link layer provides the immediate path a vehicle should follow. The network and link layer controllers are on the roadside. The coordination layer supervises a vehicle's activities and engages the vehicle in different maneuvers as needed. The regulation layer executes the maneuvers by providing the control inputs to the vehicle actuators. We show that this control hierarchy can be used to model different AHS proposals.

The next step is to provide the controllers for the control layers. We follow the PATH-AHS proposal of platooning as the means to reduce congestion and increase highway capacity. A platoon is a group of vehicles traveling in close proximity to each other with large inter-platoon distance. In the PATH-AHS proposal vehicles are under automatic control. We describe the internal structure of each control layer and the interfaces between them. There are three basic maneuvers: join, split, and change lane. In the join maneuver two platoons join together to form one platoon. In the split maneuver one platoon is divided into two platoons. In the change lane maneuver a single vehicle changes lane. Since the vehicles are automated, each maneuver use a communication protocol to acquire the permission from the surrounding vehicles to perform the maneuver. The communication protocols are designed to ensure maneuver safety and efficiency.

In order to observe the behavior of any specific proposal and to test, evaluate, and compare different proposals, we simulate the design. We have developed the SmartPath simulation tools for AHS scenarios. SmartPath demonstrates that our modeling approach is sound. SmartPath is a visual simulation package. It provides a graphical interface to view the simulated data (vehicles and highway) in a natural way. SmartPath is a micro-simulation: the behavior of each functional element of the vehicle and highway is individually

modeled and simulated. SmartPath is also a distributed simulation, so that different sections of the highway network can be simulated in different processors.


Professor Pravin Varaiya, Chair

Contents

| | |
|--|-----------|
| List of Figures | v |
| 1 Introduction | 1 |
| 1.1 Automated Highway System | 2 |
| 2 AHS Control Architecture | 5 |
| 2.1 Introduction to AHS Controllers | 5 |
| 2.2 AHS Control Hierarchy | 6 |
| 2.2.1 Highway Model | 6 |
| 2.2.2 Network Layer | 7 |
| 2.2.3 Link Layer | 13 |
| 2.2.4 Coordination Layer | 16 |
| 2.2.5 Regulation Layer | 21 |
| 2.3 PATH-AHS Control Architecture | 22 |
| 2.4 Design of the Coordination and Regulation Layer Controllers for PATH-AHS Proposal | 23 |
| 2.4.1 PATH-AHS Maneuver Sublayer | 24 |
| 2.4.2 PATH-AHS Supervisor Sublayer | 31 |
| 2.4.3 PATH-AHS Regulation Layer Controllers | 33 |
| 2.4.4 Interfaces Among Control Layers | 37 |
| 2.5 PATH-AHS Extension for Mixed Traffic, Mixed Lane Assignment | 38 |
| 2.5.1 Extended Highway Infrastructure | 38 |
| 2.5.2 Extended Network Layer | 39 |
| 2.5.3 Extended Coordination Layer | 39 |
| 2.5.4 Extended regulation Layer | 44 |
| 2.6 Support Modules for PATH-AHS Design | 46 |
| 2.6.1 Vehicle Sensors | 46 |
| 2.6.2 Highway Sensors | 48 |
| 2.6.3 Highway-Vehicle and Vehicle-Vehicle Communication | 48 |
| 2.7 Concluding Remarks | 49 |
| 3 AHS Simulation | 50 |
| 3.1 Introduction: Simulation of Hybrid Systems | 50 |
| 3.2 SmartPath Simulation Environment | 51 |
| 3.2.1 SmartPath Primary Constructs | 52 |

| | |
|---|------------|
| 3.2.2 SmartPath Secondary Constructs | 55 |
| 3.2.3 SmartPath Tertiary Constructs | 57 |
| 3.3 Performance of the Simulation | 58 |
| 4 Parallelization and Distribution | 62 |
| 4.1 Distributed Simulation of AHS | 63 |
| 4.1.1 Simulation Entities and Their Interactions | 65 |
| 4.1.2 Partitioning the AHS Simulation | 67 |
| 4.1.3 Partitioning the Highway for Minimum Communication Cost | 70 |
| 4.2 Distributed Simulation of Hybrid System | 73 |
| 4.2.1 The Processor Model | 75 |
| 4.2.2 Simulation Supervisor (SS) | 76 |
| 4.2.3 Distribution Supervisor (DS) | 80 |
| 4.2.4 Assumptions and Correctness of the DS Algorithm | 83 |
| 4.2.5 Extension to the DS Algorithm | 87 |
| 4.3 Implementation of the Distributed SmartPath | 88 |
| 4.4 Performance of the Distributed SmartPath | 91 |
| 5 Dynamic Load Balancing | 94 |
| 5.1 Introduction | 94 |
| 5.2 Problem Formulation and Structure of Optimal Policy | 95 |
| 5.2.1 Dynamic Programming | 96 |
| 5.2.2 Dynamic Load Balancing: Problem Formulation | 98 |
| 5.2.3 Dynamic Load Balancing: Structure of the Optimal Policy | 99 |
| 5.3 Implementation of the Load Balancing Scheme | 101 |
| 5.4 Performance of the Load Balanced SmartPath | 102 |
| 6 Conclusion and Future Extensions | 105 |
| Bibliography | 107 |

List of Figures

| | |
|---|----|
| 1.1 Four Views from the Graphical User Interface of SmartPath | 4 |
| 2.1 Modeling the Highway and the Data Structure | 7 |
| 2.2 Modeling the Network Patches | 8 |
| 2.3 Modeling the Network graph | 9 |
| 2.4 Modeling the Network graph | 11 |
| 2.5 Modeling the routing inconsistency | 12 |
| 2.6 Link layers highway model | 14 |
| 2.7 Network-Link Interface model | 15 |
| 2.8 An Example of the Mealy FSM | 18 |
| 2.9 A Supervisor Sublayer FSM model | 19 |
| 2.10 General Model for Regulation Layer Controller | 21 |
| 2.11 Sequence of Events in <i>Join</i> Maneuver | 25 |
| 2.12 The FSM for <i>Join</i> Maneuver | 25 |
| 2.13 Sequence of Events in Split Maneuver | 27 |
| 2.14 The FSM for the Split Maneuver: Follower Splitting | 28 |
| 2.15 Sequence of Events in Change Lane Maneuver. <i>A</i> is the vehicle that wants to change lane, <i>B</i> is the platoon in the adjacent lane (lane 2), and <i>C</i> is the vehicle in the next to the adjacent lane (lane 3). | 30 |
| 2.16 A Supervisor Sublayer FSM Model | 32 |
| 2.17 Regulation Layer Controller Layout for PATH-AHS | 36 |
| 2.18 Entry Maneuver State Diagram | 40 |
| 2.19 Detail of Stop Light State Diagram | 41 |
| 2.20 Exit Maneuver State Diagram- Leader | 42 |
| 2.21 Exit Maneuver State Diagram- Follower | 43 |
| 2.22 The Supervisor Sublayer for the Extended PATH-AHS Proposal | 45 |
| 2.23 Required Longitudinal and Lateral Sensors on the Vehicle | 47 |
| 3.1 Grid structure on the highway for detection of vehicles | 53 |
| 3.2 Real Time Spent per Simulation sample time as a Function of Number of Vehicles | 60 |
| 3.3 Real Processing Time as a Function of Simulation Time in Sun SPARC 5 Workstation | 61 |
| 3.4 Memory Usage of SmartPath as a Function of Number of Vehicles in Simulation | 61 |
| 4.1 Highway and its Associated Graph | 70 |

| | |
|--|----|
| 4.2 Modeling of a Processor | 75 |
| 4.3 Modeling of Multiprocessor System | 77 |
| 4.4 Modeling of the Simulation Supervisor | 78 |
| 4.5 The Discrete-Time Simulation | 79 |
| 4.6 Formal State Diagram for the simulation supervisor | 80 |
| 4.7 Formal State Diagram for the distribution supervisor- 1st and 2nd branches | 82 |
| 4.8 Formal State Diagram for the distribution supervisor- 3rd branch | 83 |
| 4.9 Formal State Diagram for the distribution supervisor- 4th branch | 84 |
| 4.10 The Test Highway for Load Balancing | 92 |
| 4.11 Speed up of the Distributed Simulation | 92 |
| 4.12 Scaled Speed up of the Distributed Simulation | 93 |

Acknowledgements ¹

I was fortunate enough to work with Prof. Varaiya and benefit from his knowledge, insight, and guidance during the last 5 years. For these and his friendship I am grateful.

I would like to thank Professors Shankar Sastry, Roberto Horowitz, and James Demmel for being in the qualifying exam and my reading committee. It was always encouraging to hear Roberto's comments about SmartPath and his support for my work.

If it wasn't for Delnaz' insight, experience, work, and support, SmartPath wouldn't have the same organization, robustness and reputation that it has now, and my graduate work may have extended for another couple of years. Aleks, David, Datta, John, Karim, Karl, Lokesh, and Mireille, have contributed enormously to this thesis with their works, discussions, encouragements, and friendships.

Nargues by just being there when needed, bearing with me during the deadlines and exams, and letting me stay long, long hours in the Cory Hall made this work possible. The presence of little Eidin accelerated the last steps of the research and the writing process for which I am truly thankful.

This research was supported 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 Army Research Office under contract DAAH04-94-9-0026.

Chapter 1

Introduction

Traffic congestion is an everyday problem for many people commuting to and from work in metropolitan areas of the United States. In 1995 the average speed of vehicles during peak hours was 35 mph and is expected to drop further to 11 mph by the year 2005. This will increase fuel consumption and air pollution[1]. The traditional solution of constructing more highways to meet growing demand is no longer possible in many urban areas. Telecommuting opens a new approach to the congestion problem by keeping people out of the urban centers, but for the telecommuting to be implementable across the workforce, affordable high speed communication links should be available, and more importantly the business control structure which relies on direct supervision has to change. The expansion of public transportation cannot provide a cost-effective solution in areas facing dispersion of the workforce, and the vast majority of commuters continue to use private automobiles. There is a growing research effort worldwide in the use of technology to improve the throughput of the highway system. This research has two main threads:

- Increase the throughput by providing reliable traffic information which helps drivers to make better decisions regarding their route selections [2,3]. This approach is limited to the “behavioral law” that with human drivers, there is a limit to the maximum achievable traffic flow (about 2000 vehicles/hour/lane).
- Increase the throughput by automating the decision-making for route selection and control of the vehicle which effectively removes the human driver from the driver seat. This approach claims dramatic improvements in capacity, safety, and energy efficiency and leads to the concept of the Automated Highway System (AHS).

My focus in this dissertation is on the AHS approach to increasing throughput.

1.1 Automated Highway System

Planning an Automated Highway System is a huge task. It involves the design and integration of the intelligent vehicles and intelligent highways to increase throughput without compromising safety.

There are several proposals about the structure of an AHS and its elements. At one extreme lie proposals in which a centralized controller determines the position of every vehicle similar to the way trains are controlled. These designs were studied by TRW, GM, Rohr Industries and some other groups, and are reviewed in [4]. At the other extreme are proposals that are inspired by robotics and AI-based approaches to the control of an autonomous vehicle navigating in an unstructured and even hostile environment [5,6]. These approaches emphasize recognition, learning, and trajectory planning in the face of diverse threats and obstacles. Proposals in between these extremes include the PATH (Partners for Advanced Transit and Highways) proposal for AHS (PATH-AHS), which I describe in the next Chapter.

We would like to be able to test and evaluate every AHS proposal. For example, we would like to answer the the following questions for each proposal: Is it feasible? Is it safe? Does it perform as claimed? How would the public react to such a system? What is the difference between this proposal and the others with respect to some measure of performance?

In order to test, evaluate, and compare these proposals, we must build a model of the proposed AHS design and simulate the design. Simulation provides a cost effective tool that can be used to model different strategies and configurations; therefore, it is suitable for AHS planning, modeling, and testing. But one has to be careful in interpreting the simulation results. A simulation environment simulates what is modeled; if the model is far from reality, the simulation will not represent the reality. Therefore, it is essential to validate the simulated model. Also, due to the discrete nature of any simulation, one can only approximate the continuous elements of the system, and when the approximation is too coarse, it may diminish the reliability of the simulation results. Note that the discrete approximation may lead to unreasonably pessimistic or optimistic estimates, depending on the system being simulated and on what can be expected. We will see in Chapter 3, how we use discretization to simulate AHS scenarios.

This dissertation present an approach to the modeling of the Automated Highway System and describes the SmartPath simulation tools for AHS scenarios. SmartPath is

used to demonstrate that our modeling approach is sound. The SmartPath project started in Spring of 1991, and its first and second releases were used by AHS researchers within universities and industry. We are now in the process of releasing SmartPath3.0.

SmartPath provides an environment for testing various controller designs, evaluating their performance, and observing the interaction between the vehicle controllers and the highway. It effectively responds to the questions of feasibility, safety, and performance. SmartPath is a micro-simulation, i.e., the functional elements and the behavior of each vehicle with regard to normal operations or the anomalies that may occur in the highway are individually modeled. The output of SmartPath is a comprehensive state description of each vehicle on the highway throughout the simulation.

SmartPath is also a visual simulation package. It provides a graphical interface to view the simulated data (vehicles and highway) in a natural way. Figure 1.1 shows four views of the highway from different angles which are available through the graphical interface to SmartPath users. This interface can ultimately be used to respond to the question of public reactions toward a given scenario or AHS architecture.

In Chapter 2, I will give the detail design of an AHS control architecture which is general enough to cover a wide array of scenarios. I will also give the description of a set of specific controllers needed to model the PATH-AHS design.

In Chapter 3, I will discuss the detailed design of SmartPath, the structures and functions it provides, and the performance of the simulator when a specific set of controllers are employed.

Chapter 4 presents the extension of SmartPath to a distributed environment.

Chapter 5 discusses load balancing and a simple heuristic for dynamic load balancing of the distributed SmartPath simulation.

In Chapter 6, I will discuss briefly the on-going projects that use SmartPath as their primary simulator and some concluding remarks.

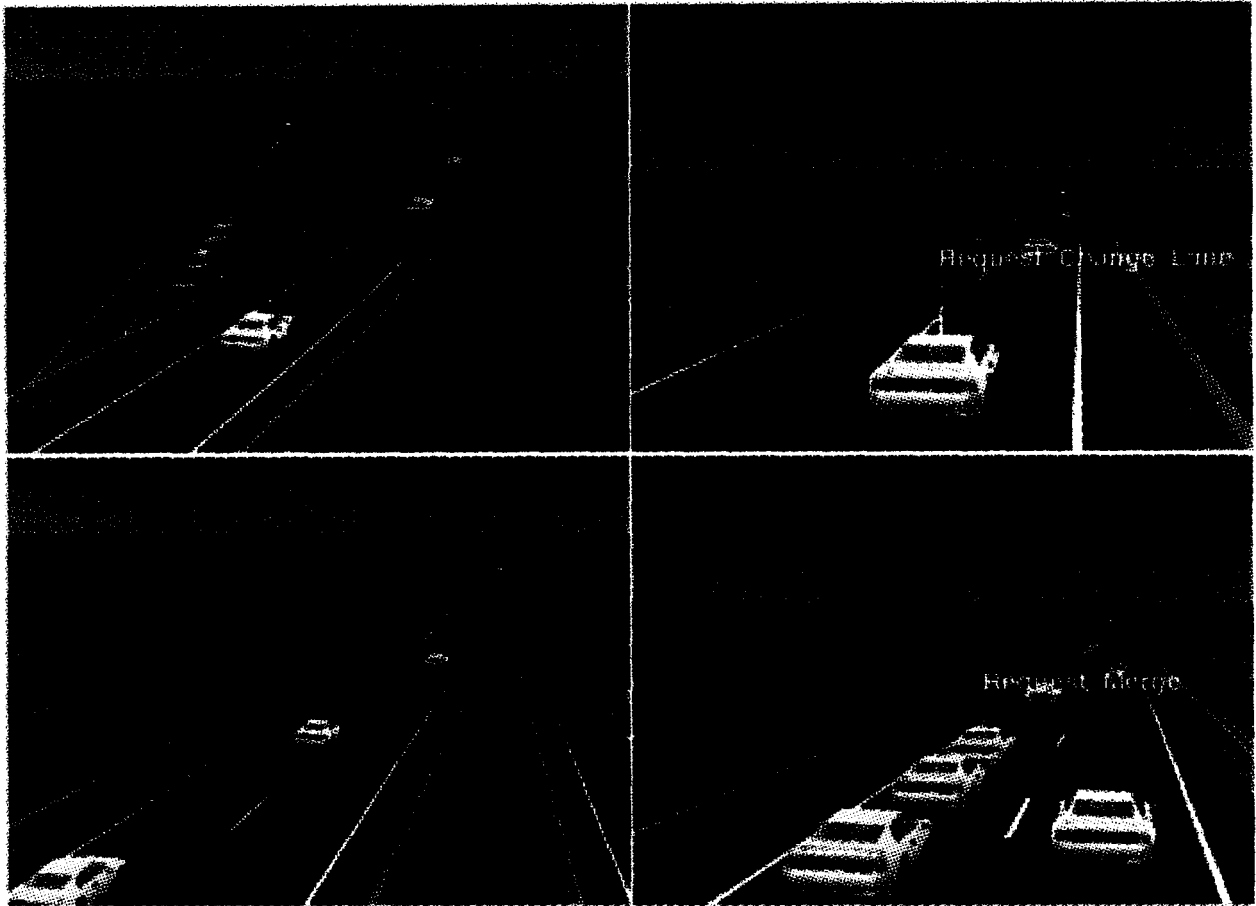


Figure 1.1: Four Views from the Graphical User Interface of SmartPath

Chapter 2

AHS Control Architecture

2.1 Introduction to AHS Controllers

The Automated Highway System consists of two elements: the automated highway and the automated vehicle. The technical challenge facing any AHS proposal is the design, development, and testing of a set of controllers in both highway and vehicle that enable the AHS to produce performance superior to today's. In general, the controllers on the highway should help the driver choose the route with the shortest travel time, and the controllers on the vehicle should be able to drive the vehicle safely and efficiently. We can, therefore, group the controllers in loosely coupled control layers according to their function and domain of operation. This allows for independent design and development of controllers by different groups of control engineers, so long as we can define a robust interface among the layers.

A hierarchical control design proposed originally in a 1991 PATH report [7] and later extended in [8] has four layers: network, link, coordination, and regulation layers. The first two layers are on the highway and the last two on the vehicle. This proposal, though originally designed for the PATH-AHS proposal, is the most elaborate proposal in terms of generality and completeness. It only specifies the control layers and not any specific controller, and it encompasses both the highway and the vehicle.

In what follows, I will describe each control layer in detail, the interactions among them, and show how we can accommodate in this control hierarchy other AHS strategies and scenarios. I will also present the specific controllers that can implement the PATH-AHS proposed structures and maneuvers.

2.2 AHS Control Hierarchy

I start this section with the highway model, since it is the connecting structure between the highway and the vehicle controllers; therefore, all controllers have to agree on it. As we will see later, the controllers use the highway model to create their own internal structure. This is particularly true for the network and link layer controllers, which comprise the “intelligence” of the highway.

2.2.1 Highway Model

A *highway topology* is divided into sections. A *section* has a specific length defined as the length of the inner most lane within that section (lane 1 or the high speed lane). The only requirement for a section is that it should have the same number of lanes throughout its length, and all the lanes within a section have the same geometry. However, the number of lanes from one section to another can change. Every section comprises a number of segments. A *segment* can be either a line or an arc; in the former case its only attribute is its length, but in the latter case it has length, the arc radius, and the direction of turn which can be left or right.

The connections between the sections needs to be defined only at the level of the lanes, i.e., the sections are connected together via their lanes. For every lane of a section we need to know whether the lane has a *barrier* to its right or left, or whether the lane is open. Barriers are one-foot long structures through which a vehicle cannot pass. There are also other structures beside the barriers that may be present on the highway like stop lights, check-in and check-out stations, and gates. Their positions along the lane should be specified in the lane structures. The lane also can be a *source* (an entrance to the highway) and/or a *sink* (an exit). These data structures are shown in the bottom of figure 2.1. The top diagram in figure 2.1 is an example of a highway topology and the corresponding data model shown graphically.

In this example the highway splits and merges back again. There are barriers between two of the lanes and at some point there are gates through which the vehicles can move to the other side of the barrier. Also note that section 4 consists of three segments; all of them are arcs but with different radius and turn direction. For example, the first segment is an arc 400 meter long, radius of 200 meters, and turns to the right.

With this highway model in mind, I will describe each layer beginning from the top of the hierarchy.

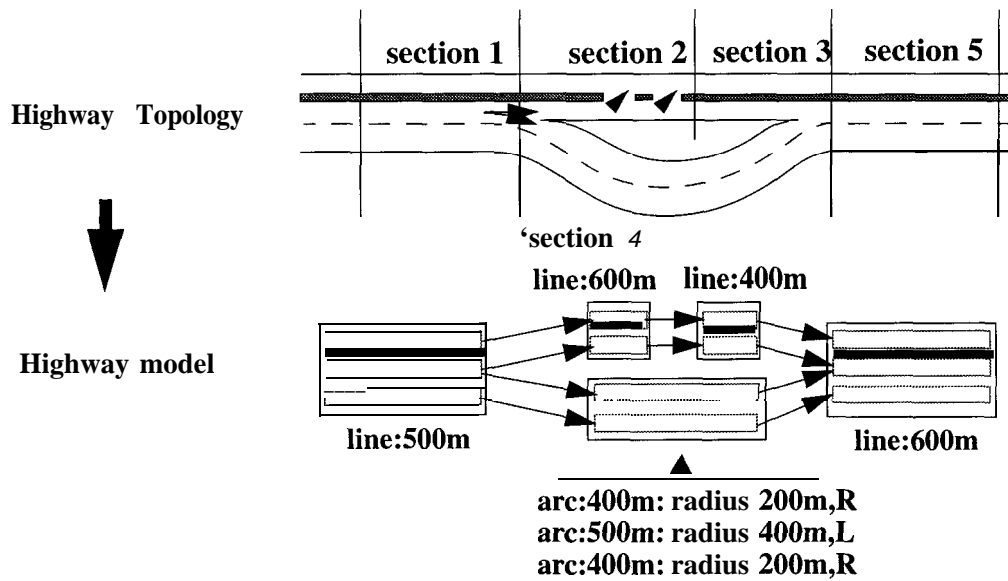


Figure 2.1: Modeling the Highway and the Data Structure

2.2.2 Network Layer

The network layer controller provides routing information from any point on the AHS to any exit of the AHS that may be the destination of the vehicles. The routing information must be logical and unambiguous, in the sense that the vehicles should be able to comply with the suggestions and should be able to choose a route when they reach a junction or a point in the highway where there is a possibility of traffic splits. For example in figure 2.1, a vehicle in section 2 should not be routed to section 4, and when a vehicle is in section 1, the network layer's information should be sufficient to enable the vehicle to choose the correct direction (section 2 or 4). The same should be true for a vehicle in section 2, where it can enter through the gate or otherwise continue straight. As we see from figure 2.1, the choices that a vehicle has can depend on the lanes that it happens to occupy. For example the vehicle in lane 1 of section 1 only can go to the section 2, but the vehicle in lane 2 can go to either section 2 or section 4.

To describe the routing choices, the network layer decompose the highway into *patches*. A patch is a collection of lanes that have open access to each other, i.e., a vehicle traveling in one lane can freely move to the adjacent lanes within the same patch. Patches are derived from sections of the highway as follows. Every section is inspected; if there is no barrier between the lanes, a patch will be created with the same length and number of lanes as the section. Therefore, when there is no barrier between the lanes of a section, the patch

and the section are identical. Note that by definition of a section, we cannot encounter a highway split within a patch.

The identity between the section and the patch is violated for sections that contain barriers. For these sections we create a set of adjacent patches, such that each patch contains the lanes with open access to each other.

As an example let us look at the highway of figure 2.1. We can derive the network's patch model as follows: (figure 2.2) Section 1 has three lanes with a barrier between lanes 1 and 2. Therefore, we create two patches. Patch 1 has only one lane (lane 1) and patch 2 has two lanes (lanes 2 and 3). In this way we continue until all sections are inspected and all patches are created.

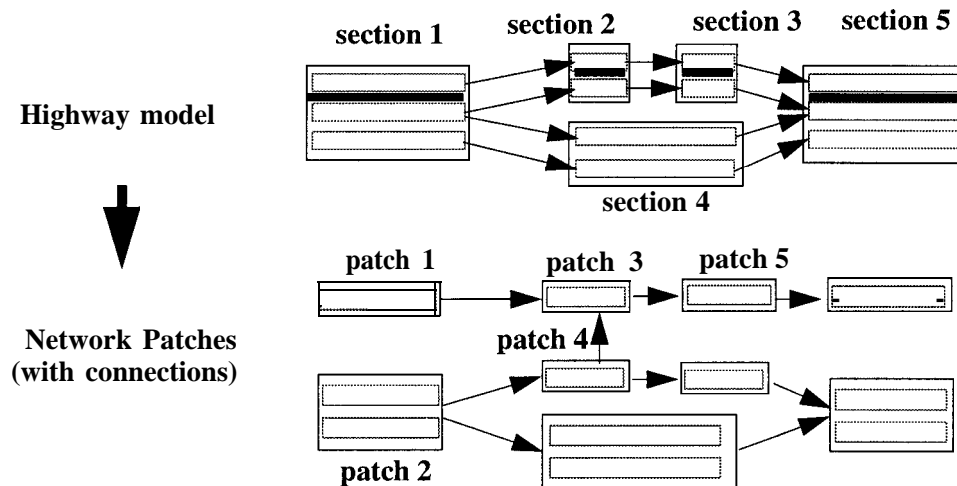


Figure 2.2: Modeling the Network Patches

The next step is to connect the patches. Two patches are connected if either the lanes that they contain are connected together, or there is a gate between the lanes. Therefore, into connect the patches together, we inspect the connections of every lane within the patch and if the lane is connected to another lane, we connect the patch to the patch which contains the next lane. To continue with our example (figure 2.2), we see that lane 1 of section 1 is connected to lane 1 of section 2, which translates to lane 1 of patch 1 to lane 1 of patch 3, so patches 1 and 3 are connected. Also we see a gate between lane 1 and 2 of section 2 (figure 2.1), which translates to a connection between patch 3 and 4, as shown in the bottom diagram of figure 2.2.

Now we can formulate more precisely the function of the network layer controller: it is to provide a sequence of patches that in shortest time leads a vehicle to the patch

in which its destination is located. Let R_{ij} be the routing from patch i to patch j , the destination of the vehicle. Then $R_{ij} = \{p_k\}_{k=0}^m$ where p_k is the index of the k th patch through which the vehicle should travel. Note that $p_0 = i$ and $p_m = j$.

In this manner, we have effectively changed the highway topology to a directed graph $G = (N, E)$ where $N \in 1, \dots, n$ is the set of patch indices, and $E \in N \times N$ is the set of connections between the patches: each member of E is a pair (i, j) where i and j are the indices of the beginning and end of the edge, respectively. The problem of routing is then transformed to finding the shortest path between the nodes of the graph, where the length of a path is the sum of the length of the edges in the path, and the length of edge (i, j) is the average travel time of the vehicles in patch i .

We can reduce the size of the graph by observing that some of the patches can be excluded from the routing path, since they have only one next patch and one previous patch. For example, we have the patch configuration shown in the top diagram of the figure 2.3, we immediately can derive the graph G (middle diagram) where for every patch a node is assigned and there is an edge between two nodes if the corresponding patches are connected. However, as we see from the graph a vehicle in patch 5 can only go to the next patch, and the same is true for a vehicle in patch 6. Therefore, we can eliminate all the nodes that are not necessary for routing and we get the Network graph G' , shown in the bottom diagram of the figure 2.3.

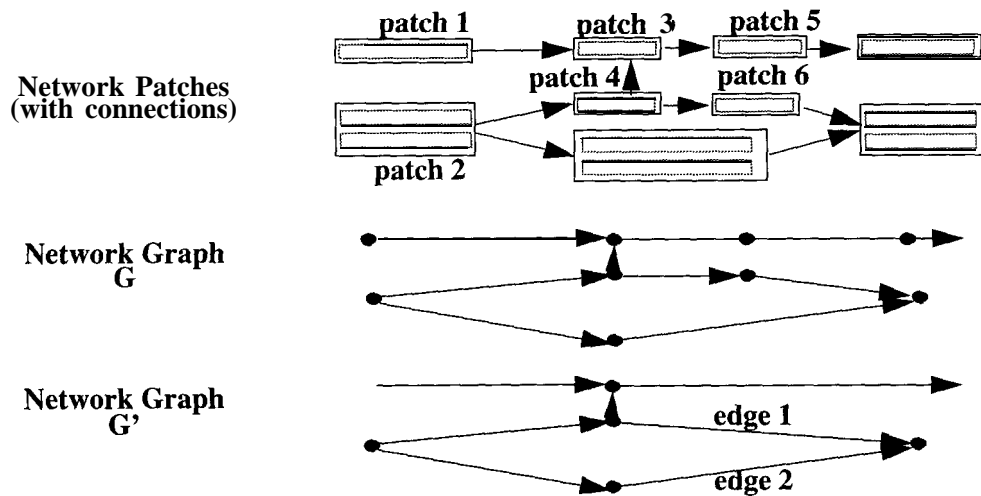


Figure 2.3: Modeling the Network graph

We can construct the graph $G' = (N', E')$ directly from the network patches, by categorizing the patches as follows:

1. patches with sink
2. patches with source
3. patches with more than one next patch
4. patches with more than one previous patch
5. patches with gates
6. patches which do not have any of the above characteristics.

Then we can assign the nodes to the categories 1-5 as follows:

- for sinks, splits (category 3), and gates, we assign a node to the patch and its next patch(es). Note that in the case of gates, one of the next patches is adjacent to the current patch;
- for sources, and merges (category 4) we assign a node only to the current patch. The node assignment for these categories is not necessary; however, since it terminates the two arriving edges, e.g., edges 1 and 2 in figure 2.3, every patch has one and only one edge through it; therefore, the computation and bookkeeping of the weights on the edges are reduced;
- for other patches we do not assign any nodes.

This categorization and the resulting node structures is shown in figure 2.4.

In this manner we can derive the graph $G' = (N', E')$ where $N' \subset N$ is the set of patches p_j $j = 1, \dots, m$, $m \leq n$, such that p_j is the index of the patch to which a node is assigned, and $E' \in N' \times N'$. The length of any $e' \in E'$ is the sum of the average travel time in all the patches through which e' passes.

We now show that for all patches, we can find a shortest route.

Proposition 2.1 *Let us assume that we have constructed the graph G' as described above, and we can find the shortest route R'_{jk} from any node j to any node k on the graph. Then for every node i such that $(i, j) \in E'$ and any patch $p_l \neq p_i$ such that the edge (i, j) passes through it, the route $\hat{R}_{lk} = \{p_l R'_{jk}\}$ is the shortest path from patch p_l to node k in patch p_k .*

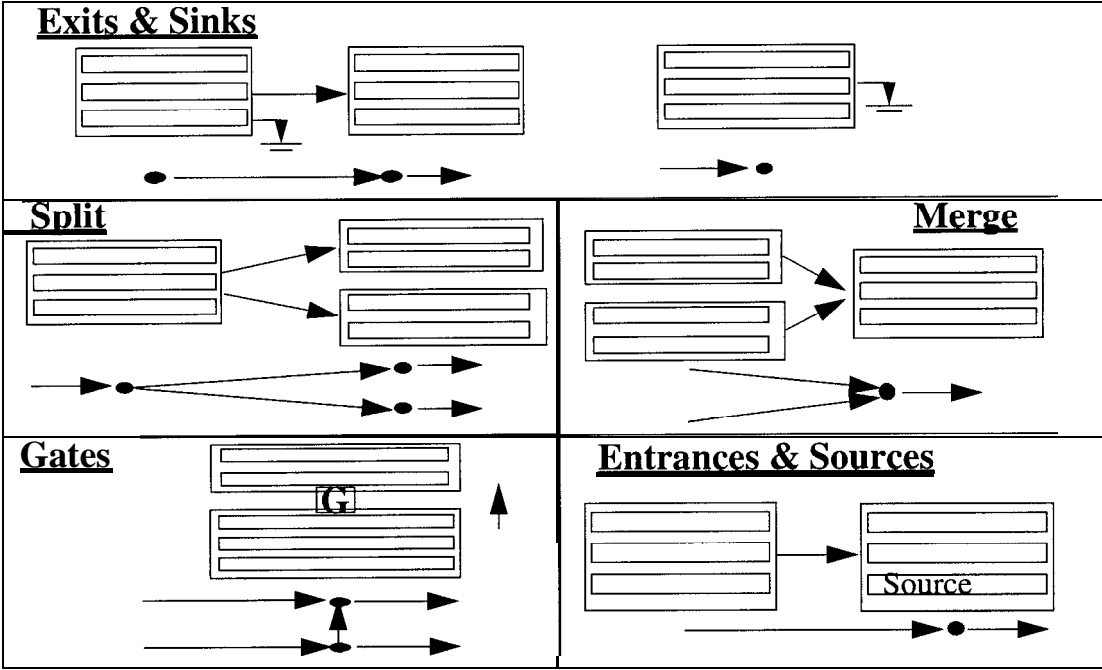


Figure 2.4: Modeling the Network graph

Proof: By construction of the graph G' if the patch is not assigned a node, only one edge passes through a patch, i.e., for the patch p_l where edge (i, j) passes through it, the next node where there is a possibility of route selection is j . Therefore, all vehicles in patch p_l have to travel through patch p_j . Since R'_{jk} is the shortest path from node j to node k , it will be the shortest path from patch p_l if we append p_l to the R'_{jk} . \square

This approach of organizing the highway in patches, provides several advantages:

- The vehicle will receive a list of patches that it should follow, and since the list is broadcasted periodically (the period depends on the controller design), the network layer can provide the best expected routing at any time during the travel time of the vehicle.
- Since the information is based on the patches (not vehicles), another control layer can receive this routing information and provide micro-routing information (e.g., to travel in a particular lane), to the vehicle. This point is discussed in the link control layer.
- The approach is scalable for large metropolitan areas like the Los Angeles basin or nationwide networks, where the number of nodes in the network is so large that real time computation of the edge weights and shortest path is not possible. In this case, we create small subnetworks, each with the routing described earlier for

destinations within their domain, and for other destinations they provide the routing to pre-specified entry points of the appropriate next subnetworks. In this way, by providing a mapping from the destinations in other domains to the patches that are the entry points to the neighboring networks, we can treat those patches as the destination of the vehicles.

In the present software implementation of the network layer, the shortest path is obtained by the distributed Bellman-Ford shortest path algorithm (see [9]). Note that since the weight on the edges are average travel times, they should be adjusted both periodically and on demand (for emergency or unpredictable cases like an accident), Whenever there is a significant change in the weights, the shortest path should be re-calculated.

To obtain the above formulation we made three implicit assumptions:

1. The highway topology is such that given a sequence of patches the vehicle can indeed follow the route. This means that if the next patch is a downstream patch, the vehicle is indeed able to carry out the required maneuvers to follow the path. Let us look at an example.

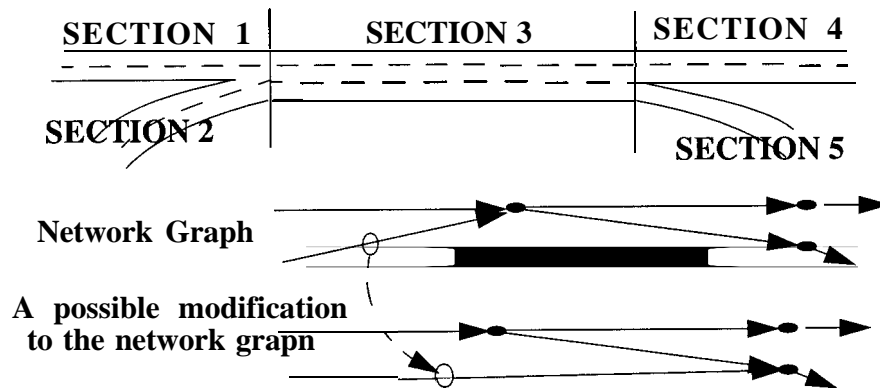


Figure 2.5: Modeling the routing inconsistency

Figure 2.5 shows the highway and its network graph. A vehicle in section 3 can go to either section 4 or 5. However, a vehicle in section 2 may not be able to reach section 4, perhaps due to the short length of section 3 or the number of lanes that connect 2 to 3 and 3 to 4. This problem can easily be solved if we know what the capability of the vehicles are, i.e., how many lane changes per kilometer are possible, and what the traffic condition is in the sections. One possible solution when the traffic is heavy, is to connect the edge that passes through section 2 to the node in section 5 directly, as shown in the bottom diagram of figure 2.5. This will result in a path which does

not go through section 3 for vehicles in the sections upstream of section 2 with a destination downstream of section 3.

Another possible solution which does not require a change in the graph topology is to introduce *switching costs* at nodes. Switching cost is defined as the cost (in terms of travel times) associated solely with the reduction of the traffic flow as a result of change lane. Every node has a table that assigns a cost to every connection from an incoming edge to an outgoing edge. The switching cost is set to zero if the traffic condition is normal, and the required change lane associated with the connection does not reduce the traffic flow. The shortest path algorithm includes the costs of the switching in calculation of the path cost. In the example of figure 2.5 when the traffic is heavy, the node assigned to section 3 increases the switching cost in its table for connection between the edge in section 2 and the edge in section 4.

2. The network layer assumes equal average travel time for lanes within a patch. This allows us to treat the patch as a homogeneous block. This assumption may not hold for short durations, for example, an accident blocks one lane.
3. The last assumption is that when a vehicle reaches its destination patch, it can reach its destination lane. In other words, as the vehicle nears its destination, it moves toward the outer lanes.

These assumptions make it clear that the network layer is not concerned with the routing within a patch. The responsibility of choosing a *lane* within a patch for a given vehicle is relegated to the link layer, which I consider next.

2.2.3 Link Layer

The link layer controllers are responsible for the smooth flow of traffic within the AHS. Its tasks are to balance the traffic among the lanes by determining the immediate path every vehicle should follow while traveling in the AHS. It provides micro-level routing commands to a vehicle traveling in a given section. The routing commands are a function of the actual traffic flow within the section the vehicle is traveling on and its neighbors, the destination of the vehicle, and the route provided by the network layer.

The link layer controller uses the highway model described before. However, it organizes the highway in interconnected lanes, each lane being connected to the front, right, and left lanes (figure 2.6). Also each lane may have two next lanes as is the case for

the second lane of section 1 in figure 2.6. The interconnection among the lanes also limits the behavior of vehicle in a lane. For example, a vehicle in section 1 lane 1 (figure 2.6) can only stay in the same lane.

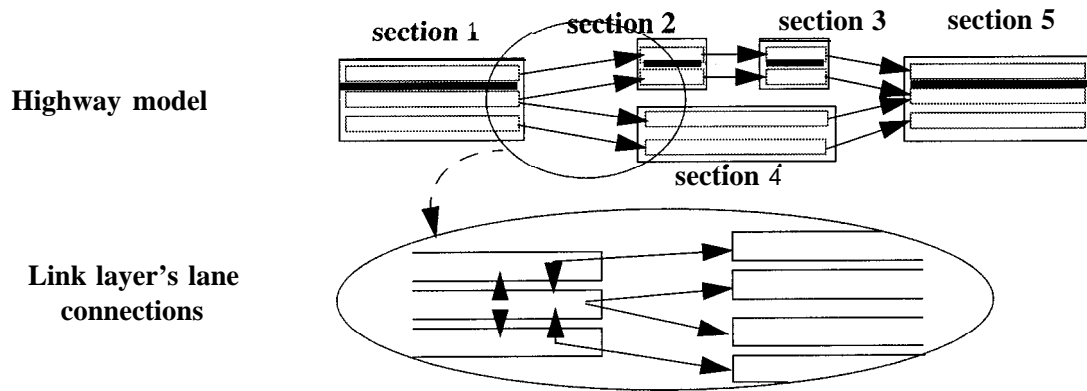


Figure 2.6: Link layers highway model

Since a lane can at most connect to two next lanes and two adjacent lanes, it is clear that the most basic commands to route a vehicle are: change right, change left, straight, stay right, and stay left. The last two commands are for cases when the present lane splits into two lanes. These commands are the most basic, since they require only simple maneuvers, if any, from the vehicles.

The highway model allows the link layer controllers to be distributed across the highway, such that each controller serves a number of sections and the vehicles on them. However, for the proper routing of vehicles, it is necessary that an overlapping region exists between two neighboring link layer controllers. Also, the amount of information that has to be passed between the controllers depends on the design of the controllers, but in general every controller should know about the downstream expected travel time, and the upstream expected incoming flow. The distribution of the controllers allows the link layer to respond to the highway condition in real time, which is crucial, especially when there is an incident. A controller design for link layer is developed in [10, 11, 12].

Now let us examine how the network and link layers can be used in different AHS settings.

- For a proposal which is based on in-vehicle routing advice and decision making, the network layer can be used as the routing guide; however, depending on the available information, we can use either a static router, in which all the weights are fixed (perhaps proportional to the distances along the highway), or a dynamic router of

the network layer, whenever there is a possibility of updating the routing information within the vehicle. The link layer can now be ignored, since no micro-router is needed, and the driver is the decision maker. In this case, the design of the vehicle and its controllers are more complex, as we will see later in this chapter.

- For an AHS proposal which is based on centralized planning (like the PATH proposal), the network and link layer controllers are of paramount importance. The information that the two layers send to each other is shown in figure 2.7.

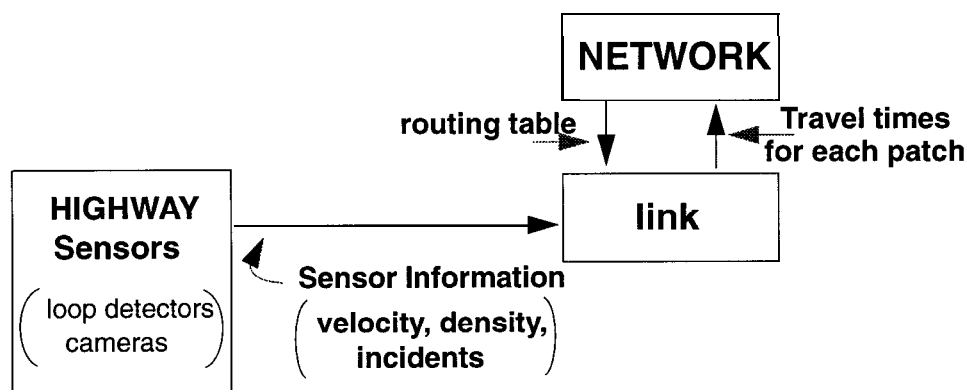


Figure 2.7: Network-Link Interface model

The network layer uses the travel time estimates from the link layer to calculate the weights on the edges of its graph. In turn, the link layer will receive for each lane the routing table from the network layer. This routing table, as explained earlier, consists of a sequence of patches which connects that lane to any exit which is reachable (reachability & destination means that a vehicle can travel to that exit, or the exit lane is in a downstream patch). The link layer controllers also receive from the highway sensors the traffic flow information in each lane and from vehicles the information regarding the destination of vehicles (i.e., highway number and exit number) that are traveling on that lane. The latter information can be transmitted to the link layer by the vehicle via transmitters installed on vehicles, or it can be acquired using special sensors on the highway. The information may be aggregate and statistical in nature. Using this information, the link layer issues near-term routing commands to the vehicles that should be executed by the vehicles. As the vehicle travels along the highway, it receives a sequence of link layer commands, guiding the vehicle toward its destination in the shortest possible time.

A point of caution: if the amount of information needed for decision-making is high, one should thoroughly analyze the feasibility of acquiring that information. For example, if the controllers require the exact position of each vehicle at every second, then the highway should be either instrumented with new high-precision sensors, or a very high bandwidth communication channel should be provided, so that the vehicles can transmit their positions to the roadside controllers.

We now turn to the vehicle and describe the lower two control layers, namely the coordination and regulation layer controllers.

2.2.4 Coordination Layer

The coordination layer determines which maneuver to undertake and coordinates the movement of the vehicle with neighboring vehicles. When the vehicle has to do a maneuver (e.g., change lane), this layer acquires permission from neighboring vehicles. This activity can be done explicitly, by sending radio messages asking for permission, or implicitly, using the vehicle sensors. When permission is obtained, this layer instructs the regulation layer to execute the maneuver (e.g., move the vehicle from one lane to the adjacent lane). The decision about which maneuver to undertake and when to start it is based on safety, the shortest path to the destination of the vehicle, and the local traffic condition. Aside from safety, these considerations can be made by the coordination layer itself or by the link layer, if there exist a path planning controller on the highway. In the latter case, the coordination layer receives the commands left, right, straight, and so on, from the link layer and proceeds to perform the appropriate maneuvers.

We can further split the coordination layer into two distinct sublayers: supervisor and maneuver sublayers. The supervisor sublayer ensures that the vehicle performs the appropriate maneuver. If a maneuver becomes inappropriate at any time, the supervisor sublayer aborts the maneuver and initiates the appropriate one. The appropriateness of a maneuver is determined by the AHS strategy being modeled. When a vehicle is engaged in a maneuver, it activates a protocol machine, which is a structured exchange of messages between the vehicle and the neighboring vehicles. As stated before, this protocols can be as simple as turning on the vehicle's signal, or as complex as coordinating all neighboring vehicles. The maneuver sublayer contains the protocol machines for all the maneuvers a vehicle can perform.

We would like to be able to model the supervisor and maneuver sublayers and to

answer such questions as: Does the maneuver as modeled result in the desired behavior? How can we guarantee that the desired behavior is the only behavior the maneuver can produce? Is there a possibility of deadlock between the initiator of the maneuver and its respondent? There are many methods of modeling; we found the model and the syntax of Finite State Machines (FSM), which are finite automata with output, the simplest and the most convenient. I will explain briefly the general FSM model and syntax, and then show how it is used to model the coordination layer controllers under different AHS strategies.

Finite State Machine (FSM) The finite state machine is a generalization of the finite automaton, which is a mathematical model of a system with discrete inputs and outputs. The system has many internal configurations or “states”; each state is a summary of all the past history needed to determine the next state upon arrival of an input. We can therefore characterize a finite state machine by:

- Q : finite set of states
- q_0 : an initial state
- Σ : set of input symbols
- $\lambda : Q \times \Sigma \rightarrow Q$: the transition function
- A : set of output symbols
- $\gamma : Q \rightarrow A$ or $\gamma : Q \times \Sigma \rightarrow A$: the output generating function

We define a finite state machine **M** as the six-tuple $(Q, q_0, \Sigma, \lambda, A, \gamma)$. The function λ takes as input the pair $(q, \sigma) \in Q \times \Sigma$ and outputs the next state $q' \in Q$. There are two distinct approaches to specify the function γ . If the argument of γ is the state alone, the FSM is called a “Moore” machine; if the argument is the pair $(q, \sigma) \in Q \times \Sigma$, the FSM is called a “Mealy” machine. Moore and Mealy machines are equivalent and have the same expressive power. We use the Mealy model to model the coordination layer, since it is more convenient to describe the communication protocols using the Mealy machines.

We can associate a directed graph with a finite state machine by assigning the states of the system to the nodes of the graph, and whenever for any two nodes i and j , there exist a σ such that $\lambda(i, \sigma) = j$, there is an edge from i to j , denoted by the pair (i, j) , and the label on the edge (i, j) is, in the Mealy machine representation, a/S where $\delta \in A$, and for the Moore machine representation, the label is σ .

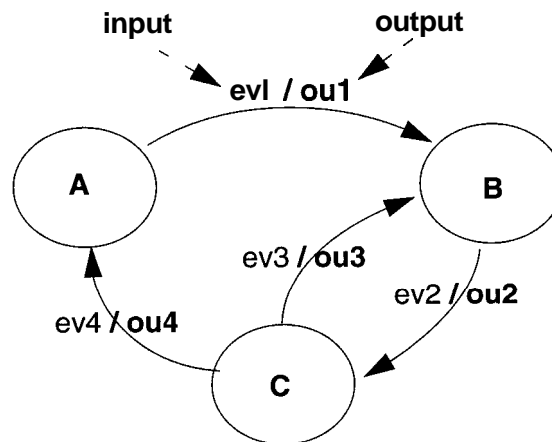


Figure 2.8: An Example of the Mealy FSM

An example of a three-state Mealy FSM is shown in figure 2.8. As we see from the figure, the input (sometimes called the event) $ev1$ causes the system to move from state A to state B and generates the output $ou1$. To create a large system, it is easier to create system modules first and later connect them, This is down by coupling (sometimes called synchronizing) different FSMs by using the output of one machine as the input to another, as we will see in the description of protocols we employ in the coordination layer.

The behavior of a Mealy-FSM is defined by the states that it can traverse and the outputs it can generate. Therefore, we can answer the question of the desirability of the system behavior by analyzing the FSM and tracing its states and outputs. This can be done by manually analyzing the FSM when the number of states is small, which I have done for a relatively small system in section 4.2.3; however, this becomes difficult as the state space increases. For large state spaces there are some formal methods and programs that can be used to analyze the system. Assume that the behavior of the system is \mathcal{F} and the desired behavior of the system is \mathcal{A} . Then one way to show that the modeled system will not generate any undesired behavior is by checking the intersection of \mathcal{F} and \mathcal{A}' which is the complement of the desired behavior, $\mathcal{F} \cap \mathcal{A}'$, and if the intersection is empty, we conclude that $\mathcal{F} \subset \mathcal{A}$, so the modeled system can not generate any undesirable behavior. Among the formal verification software packages are COSPAN [13] and HSIS [14], which follow the above procedure. For example in COSPAN, \mathcal{A} is called a monitor, which is an automaton specified to define those sequences of (state, event) pairs produced by the FSM algorithm which constitute the performance of the stated task. Both packages allow modular design of the FSMs, and have been extensively used to verify VLSI designs.

Now, let us return the coordination layer. I will describe next the supervisor FSM and the maneuver FSM.

- **Supervisor Sublayer-** The supervisor sublayer can be abstractly regarded as a two-state machine. In the first state it waits for the occurrence of an event. When the event occurs, the supervisor initiates the appropriate action. In the the second state, it waits for the completion of the action it started. The events that the supervisor awaits are of three types:

1. Events related to changes in the position of the vehicle on the highway. For example, if the vehicle is at the end of an exit ramp, at a place in the highway where changing lane is illegal or unsafe (like the places where solid lines are drawn between lanes), or if another vehicle is in the proximity of the vehicle, the supervisor should be notified. The events of this type are detected by the vehicle's sensors, which in turn notify the supervisor. Accordingly as it is a manual or an automated vehicle, the sensors are the human driver's eyes and ears or a set of radars and detectors.
2. Arrival of a message or signal from another vehicle requesting a maneuver or from the link layer relaying a specific instruction. The communications module receives the message and forwards it to the supervisor for a reply.
3. Arrival of a message from the maneuver module indicating successful or unsuccessful completion of a maneuver (maneuver-done).

An example is shown in figure 2.9, where $ev\text{-}type\textit{i-j}$ is the $j\textit{th}$ event of type i and $initiate\textit{-i}$ is the initiation of the $i\textit{th}$ maneuver (a request or a response).

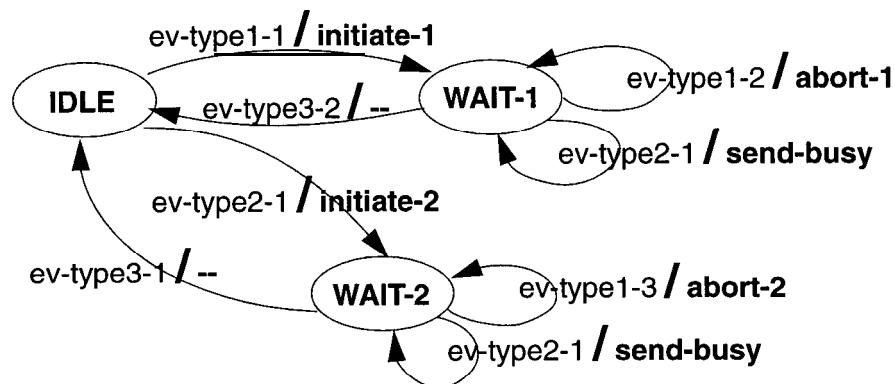


Figure 2.9: A Supervisor Sublayer FSM model

The system is initially in the *IDLE* state. The next state is any of several *WAIT* states. In figure 2.9, there are two *WAIT* states, *WAIT-1* and *WAIT-2*, that can be reached by the two events, *ev-type1-1* and *ev-type2-1*, and the system will then generate the outputs, *initiate-i* and *initiate-2*, respectively.

We can reduce the complexity of the supervisor by adding the requirement that the supervisor sublayer can only participate in one maneuver at a time, either as initiator or as respondent. With this simplifying assumption we can see that when the system is in any of the *WAIT* states, and another vehicle requests a maneuver, *ev-type2-1*, the supervisor generates the output *send-busy*, interpreted as the rejection of the request. A specific design for the supervisor sublayer of the automated vehicles will be discussed later.

- **Maneuver Sublayer-** The maneuver sublayer contains the protocol machines for maneuvers. The two basic maneuvers every vehicle should be able to perform regardless of the underlying AHS strategy, are lane keeping and lane changing. The former does not require the coordination layer, since there is no need to reach an agreement with neighboring vehicles; the vehicle only uses its sensors and checks the relative velocity and the distance between vehicles to avoid collision. However, completing a change-lane maneuver requires an empty space in the adjacent lane, and if the space is occupied by another vehicle then the lane-changing vehicle should either wait, decelerate, or ask the neighbor to decelerate, by using some form of communication (signaling, direct radio communication, or just turning in front of the other vehicle and hoping that there is no collision!).

Although maneuvers can be very different from each other, their structure should follow the following four-step general algorithm:

1. When initiated by the supervisor sublayer, activate the communication protocol specific to the maneuver and secure the agreement of all vehicles that are involved. If successful, go to step 2; otherwise, go to step 4.
2. Ask the regulation layer to implement the maneuver.
3. Wait until the regulation layer returns the outcome of the maneuver. If the regulation layer was successful, do the bookkeeping and notify other vehicles involved, if necessary.
4. Notify the supervisor layer of the outcome of the maneuver.

When the initiated maneuver has notified the supervisor layer, it can terminate.

2.2.5 Regulation Layer

The regulation layer implements and executes the maneuver. The controllers at this layer should provide the appropriate inputs to the vehicle's actuators in order to perform the maneuver started by a coordination layer controller. An example of such inputs is the jerk or acceleration value that the vehicle should implement. Regulation layer controllers are in general time-driven systems, since they are heavily dependent on the sensors. The sensors on the vehicle have their own sampling time, i.e., the sensors provide the surrounding information every T seconds, where T can be as short as 20 milliseconds; therefore, the regulation layer can at most observe the outside world and the effect of its commands every T seconds. For example, when the vehicle has to change lane the information regarding where the vehicle with respect to the lane marker is can be obtained every T seconds, and when the regulation layer issues a tire angle or lateral acceleration to the engine, the amount of the vehicle movement can be assessed at every T seconds. We still can model the regulation layer as a FSM, if we assume that whenever the new sensor values are ready, the sensors module within the vehicle will issue an event to the regulation layer. So let us look at the general model of the regulation layer, see figure 2.10.

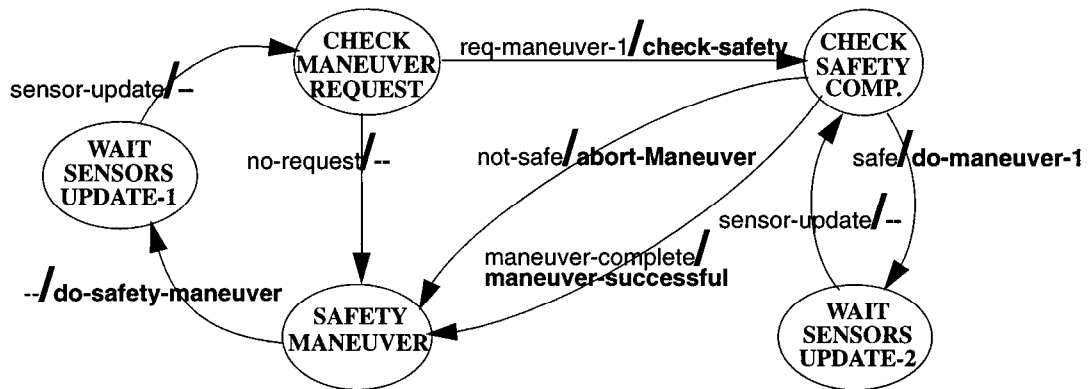


Figure 2.10: General Model for Regulation Layer Controller

The initial state for the regulation layer is *WAIT-SENSORS-UPDATE-1*. When the new set of updated values from the sensors is available, the regulation layer checks the requests from the coordination layer. If there is no request from the coordination layer, the regulation layer should only do the safety maneuver, the controller which makes sure the vehicle will not run into a vehicle in front of it. This is shown as the *SAFETY-MANEUVER*

state from which the regulation layer takes the edge back to *WAIT-SENSORS-UPDATE-1*, the initial state, with the output *do-safety-maneuver*. However, when there is a request from the coordination layer, shown in the figure as the event *req-maneuver-1*, the regulation layer first should check the safety conditions and if the maneuver does not violate the safety constraints, it performs the maneuver. The state *CHECK-SAFETY-COMP* checks both the safety and the completion of the initiated maneuver. When the maneuver is safe and not completed as of the current time, the regulation layer outputs *do-maneuver* and moves to the *WAIT-SENSORS- UPDATE-2* state, until the next update from sensors is available. In the case that the maneuver is not safe to perform, it aborts the maneuver and moves to the *SAFETY-MANEUVER* state. When the maneuver is completed, the regulation layer still moves to *SAFETY-MANEUVER* state but its output is *maneuver-successful*, which also notifies the coordination layer of the outcome of the maneuver. In general, there may be many requests from the coordination layer, which increases the number of states in the regulation layer; however, the overall design of this layer remains the same.

The coordination layer together with the regulation layer comprise the intelligent vehicle.

Now that we have the FSM tool and some general models for the coordination layer and regulation layer, we can be more specific and show the design of the coordination layer and regulation layer controllers within the PATH-AHS control architecture.

2.3 PATH-AHS Control Architecture

The PATH-AHS proposal is built on the premise that if we can decrease the distance between the vehicles moving in the highways, we can accommodate more traffic within the existing highway network. Therefore, it proposes to organize the traffic in platoons: a group of vehicles traveling in close proximity with each other with the intraplatoon distance of one to three meters. There is no requirement on how many vehicles can belong to one platoon; however, through simulation the average platoon size is about 10 vehicles when the flow of traffic is about 8000 vehicles per hour per lane. In order to maintain close spacing within the platoon, the driving is done automatically.

Let us see how you will “drive” on an AHS structured and built according to the PATH proposal.

You drive your vehicle through an entrance to the highway and push a button on the dashboard, enabling the Automatic Vehicle Driving System (AVDS), which in turn registers the vehicle and its pre-entered destination with the highway controllers. The AVDS activates the vehicle's radio transmitters and receivers and listens to the messages that come from roadside. The messages are divided into two parts: destination and command; the command part can be *change-lane-right*, *change-lane-left*, or *keep-lane*. If the destination is the same as the the vehicle's destination, The AVDS tries to comply with the command. When the command is *keep-lane*, the AVDS, using its sensors, looks for other vehicles in front of it, in order to create platoons or responds to other vehicle's requests; when the command is *change-lane*, the AVDS checks the adjacent lane with its lateral sensors, and if there is a vehicle there, it asks for permission to change lane; otherwise, it calculates a trajectory starting from its present position and ending in the adjacent lane, and sets the vehicle's throttle, brakes, and steering actuators to follow that trajectory. When the vehicle reaches its destination, the AVDS sounds an alarm, and asks you to take over the control of the vehicle. During the travel, you may be drinking coffee and reading a newspaper. If at the end of travel, you are not able to take the vehicle's control, the AVDS will park your vehicle in a designated place at the end of the exit lane.

2.4 Design of the Coordination and Regulation Layer Controllers for PATH-AHS Proposal

The first vehicle in a platoon is called the *leader* and the other vehicles are *followers*. A platoon with one vehicle is called a *free agent*. Network and link layers controllers are operating on the highway broadcasting commands and information to the vehicles.

The basic maneuvers required of an automated vehicle in this environment are: join to form platoons, *split* to break up the platoons, and *change lane* to move a vehicle from one lane to another. The coordination layer can combine these basic maneuvers to produce more complex maneuvers like exit or free agent maneuver. In designing these maneuvers, we are assuming certain sensing and communication capabilities. Every vehicle is equipped with a transmitter and receiver that can send and receive messages to and from other vehicles and road side controllers. Within a platoon, there is an established communication link for periodic message transmission. Vehicles have longitudinal and lateral sensors that can calculate the intervehicle distance, and the relative velocity. The lateral sensors can sense vehicles in the adjacent lane and next-to-adjacent lane (two lanes away).

In what follows, I describe the maneuver sublayer, the supervisor, and then the regulation layer controllers.

2.4.1 PATH-AHS Maneuver Sublayer

The maneuver sublayer contains the primary maneuver and the free agent maneuver protocols, which I describe next. In the description of the primary maneuvers, I first identify the involved vehicles, then the protocol machine, and finally the required updating and bookkeeping.

• Join Maneuver

The *join* maneuver creates one platoon from two platoons. Leaders of both platoons are involved in the maneuver; however, only the leader of the rear platoon is responsible to execute the required trajectory to reach the front platoon. The join maneuver is initiated by the leader of a platoon when the following conditions are satisfied:

1. The vehicle is in its assigned lane.
2. The platoon leader is not engaged in any other maneuver.
3. The sensors detect another platoon in the same lane.

Let us assume that platoon A is in front of platoon B , and A_i and B_i are the i th followers of A and B , respectively. Figure 2.11 shows the sequence of events, and figure 2.12 shows the FSM for the *join* maneuver. B_1 is the leader of the rear platoon which initiates *join* and A_1 is leader of the platoon A .

If the above conditions are met and the size of the platoon B is smaller than the platoon size allowed in that section, B_1 transmits a *request-join* message to the platoon A . The size of the B platoon is also encoded within the message. Depending on how the communication module is built, the vehicle that receives the message may be the leader A_1 or a follower of the platoon A , in which case it forwards the message to its leader, A_1 .

A_1 checks the conditions 1 and 2 above, and also adds up the sizes of the two platoons, and if the total size is less than the maximum platoon size specified by the link layer controller, it sends an acknowledgement, *ack-request-join*. Otherwise it sends a negative acknowledgement, *nack-request-join*, which causes B_1 to terminate the *join* maneuver.

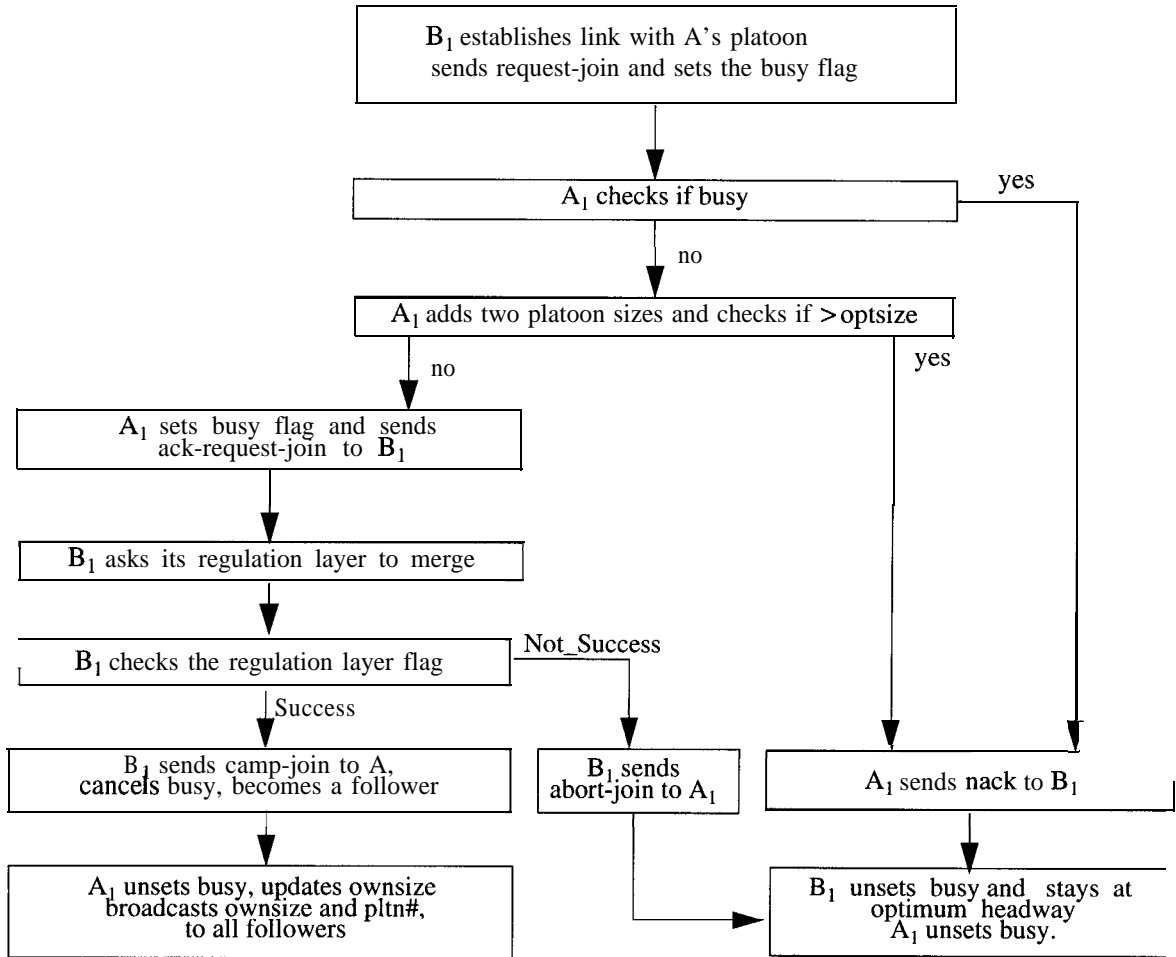


Figure 2.11: Sequence of Events in *Join* Maneuver

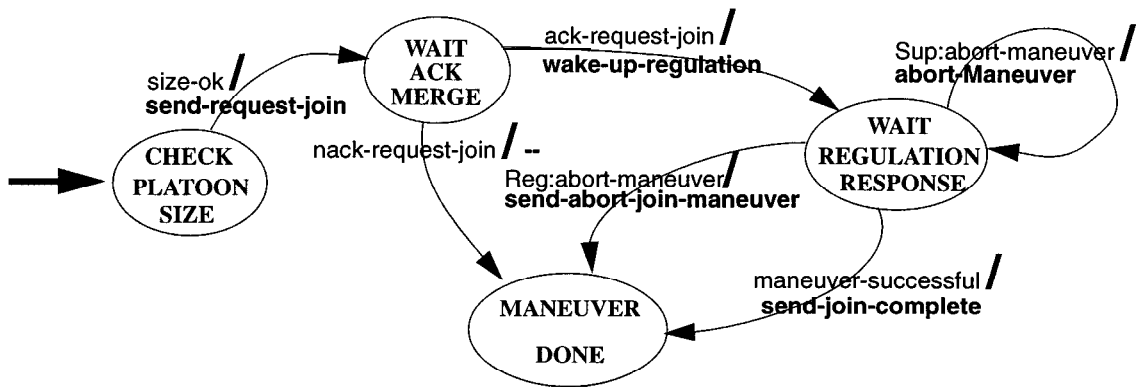


Figure 2.12: The FSM for *Join* Maneuver

Upon receiving *ack-request-join*, B_1 's coordination layer controller instructs its regulation layer to execute the *join* feedback law. The regulation layer controller causes the B platoon to follow the trajectory needed to reach a pre-specified distance (the intraplatoon distance) from the tail car of A 's platoon. During this time, the distance from the tail car of the A 's platoon is continuously sensed, and if another vehicle has moved between the two platoons, the execution of the maneuver is aborted by the regulation layer, the coordination layer of B_1 then sends an *abort-join-maneuver* message to the platoon A . While B_1 is accelerating, the rest of its platoon, under the follower feedback law, maintain platoon formation. When B_1 's regulation layer has completed the maneuver, it notifies the coordination layer, which transmits *join-complete* to A_1 . A_1 then updates its own state information regarding its new platoon, and sends a message to its followers (which now include the vehicles in platoon B) to update their states. At the end of the update, the two platoons are joined, and the maneuver is completed.

From the state diagram of figure 2.12 one can see that the supervisor may issue an *Sup:abort-maneuver* when the vehicle is accelerating (in *WAIT-REGULATION-RESPONSE* state); when this happens, the regulation layer is notified, but the maneuver cannot be aborted, until the regulation layer aborts the maneuver and issues *maneuver-successful* or *Reg:abort-maneuver*.

The members of platoon A must update the size of the platoon in their state information. The tail car of platoon A *also* has to update the back car communication id; the members of platoon B have to update their position in the new platoon and the platoon leader information.

- **Split Maneuver**

The *split* maneuver splits a platoon to the two platoons. There are two types of split: *fast-split* and *slow-split*. In *fast-split*, the maneuver is completed as soon as the platoons are logically separated which occurs when the updatings of the platoons are completed; in *slow-split*, the maneuver is complete when the platoons are at a safe distance from each other. *Fast-split* is used mainly for emergency purposes. Depending on which vehicle in the platoon initiates the maneuver, we can also classify the split maneuvers as follows:

1. The leader of the platoon wants to become a free agent. In this case the vehicles involved in the maneuver are the leader and the second car in the platoon.

2. A follower wants to split from the platoon. The involved vehicles are the splitting vehicle and the leader of the platoon.
3. The leader of the platoon asks a follower to split from the platoon. The involved vehicles are the splitting vehicle and the leader of the platoon.

The third split type is needed to accommodate a change lane maneuver by a vehicle in an adjacent lane or in emergencies; however, except in the initiation phase, it is the same as the second type; thus, I will only describe the first two of the above types. Figure 2.13 summarizes the sequence of events for both cases, and figure 2.14 illustrate the FSMs for the initiator's communication protocol.

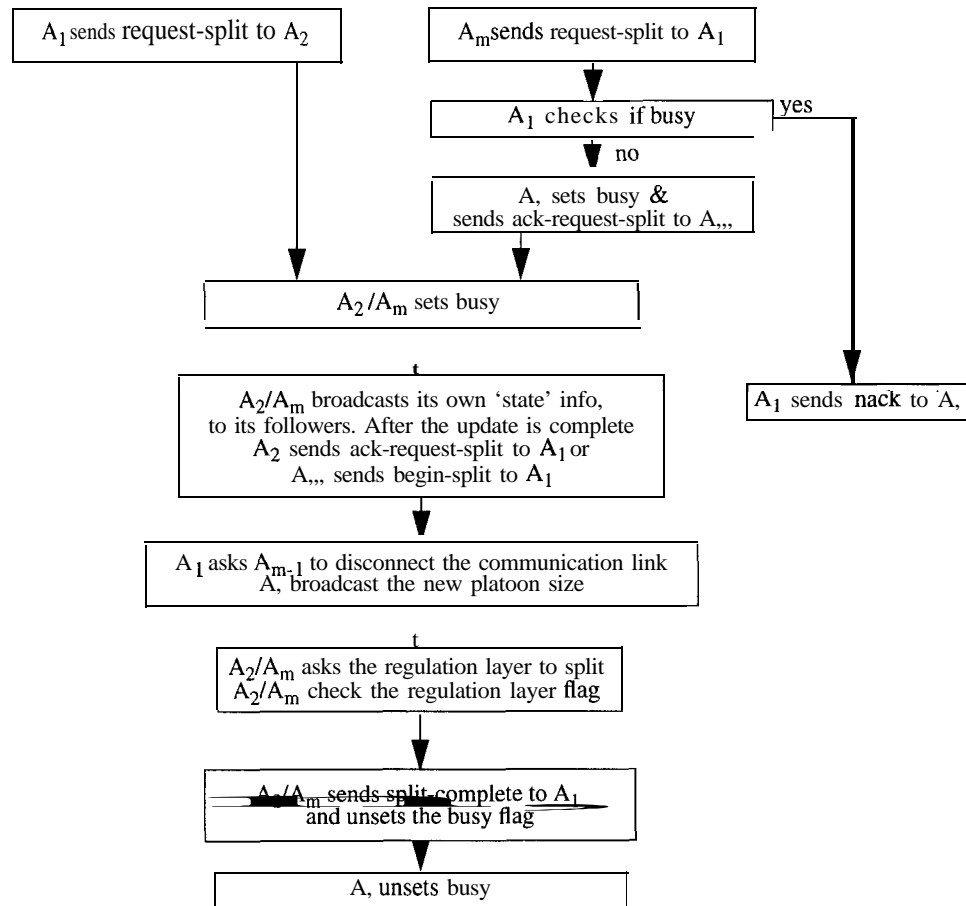


Figure 2.13: Sequence of Events in Split Maneuver

Let us assume that A , is the m th follower in platoon A with A_1 as its leader. If A_1 wants to split, it sends the message *request-split* to A_2 . Follower A_2 then updates its state (replacing the platoon id by its own id), and sends a message to all other vehicles

in the platoon to update their state, and to follow the A_2 's lead from then on. When A_2 receives the *update-complete* message from the tail car of the platoon, it sends the message *ack-request-split* to A_1 and activates the split regulation layer controller. As soon as the regulation layer completes the maneuver, A_2 sends the *split-complete* to A_1 . The vehicle A_1 in turn updates its own state to reflect the fact that it is now a free agent (one-car platoon). The maneuver is now complete. Note that A_2 always accepts the split maneuver initiated by the leader.

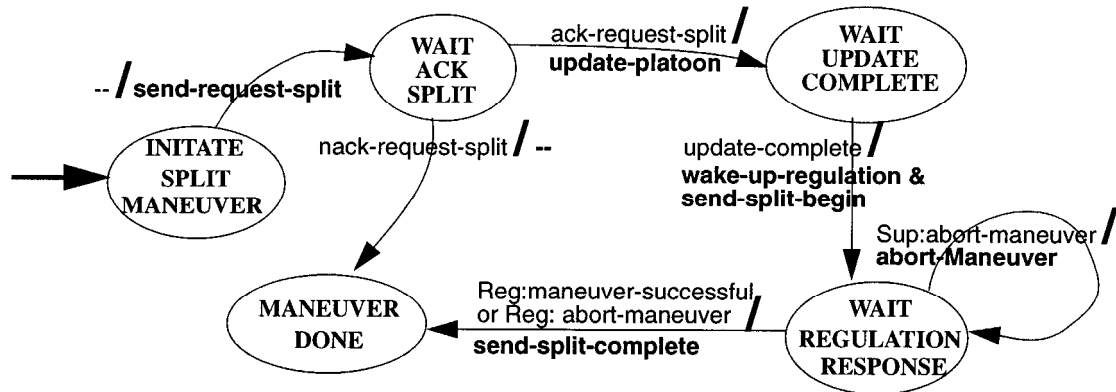


Figure 2.14: The FSM for the Split Maneuver: Follower Splitting

If a follower, say A_i , in the A platoon wants to split, it sends the message *request-split* to A_1 . Upon receiving this message, A_1 checks its busy flag, and if it is not set, it replies with the message *ack-request-split*, and waits until it receives the *begin-split* from A_i , after which it instructs A_{m-1} to change its back car id to 0 and disconnects the communication link of its own platoon with the platoon with A_i as its lead vehicle. A_{m-1} is now the tail car of A 's platoon. When A_i receives the acknowledgement from A_1 , it updates its state (since it is now a leader) and instructs its followers to update their states and follow the vehicle A_i . When the tail car of the platoon has updated its state, it returns *update-complete* to A_i . A_i then transmits the message *split-begin* to A_1 and instructs its regulation layer to execute the split feedback law, which causes A_i to decelerate until it is at a safe distance from platoon A . When that distance is reached, A_i transmits the message *split-complete* to A_1 , and the maneuver is complete.

The members of the new platoon have to update their states regarding the size of the platoon, their position in the new platoon, and the lead vehicle information, if applicable.

If a vehicle within a platoon wants to change lane, it must first become a free agent. The free agent maneuver is built on top of the split maneuvers. The free agent maneuver requires one split, if the vehicle is first or last in the platoon, and two splits, if it is a follower in the middle of the platoon.

- **Change Lane Maneuver**

The change lane maneuver is initiated by a free agent when the lane assigned by the link layer is different from the vehicle's current lane. If the vehicle is not a free agent, the required split(s) should be initiated at the end of which the vehicle is a free agent.

Now suppose that A is a free agent in lane 1 of a three-lane highway and wants to change its lane to lane 2. It can do so only if there is adequate space in lane 2 and no vehicle in lanes 2 or 3 is planning to move into that space. Therefore, the vehicles that are involved in the *change lane* maneuver are vehicle A , the nearest platoon to the vehicle A in lane 2, and if there is no platoon in lane 2 in the lateral sensor range of the vehicle A , the nearest platoon in lane 3, if any. The sensors on the vehicle determine the presence or absence of a vehicle within some detection range in the the adjacent lanes. Vehicle A can change its lane under one of the following three conditions:

1. No vehicle is detected in lanes 2 and 3; the vehicle can move to lane 2.
2. No vehicle is detected in lane 2, but a vehicle is detected in lane 3; vehicle A then sends a message to that vehicle, asking whether the vehicle wants to move into lane 2.
3. A vehicle is detected in lane 2. A then conducts a protocol exchange with the platoon in the manner described below.

Figure 2.15 describes the sequence of events that must occur before A changes lane.

Suppose A has detected vehicle B_m in lane 2. It sends *request-change-lane* to B_m which the latter forwards to its leader, B_1 . If B_1 is busy, it sends a *nack-request-change-lane* to A . If it is not busy, it returns *ack-request-change-lane*. At this point B_1 determines how to create a space in lane 2 in order to accommodate A 's request. There are three options:

1. Vehicle A can decelerate until it reaches the safe distance behind the platoon B . If this option is chosen, B_1 sends the message *decelerate-your-platoon* to the vehicle A . Vehicle A then activates the appropriate regulation layer feedback law to create the trajectory for moving the vehicle to a safe distance behind platoon

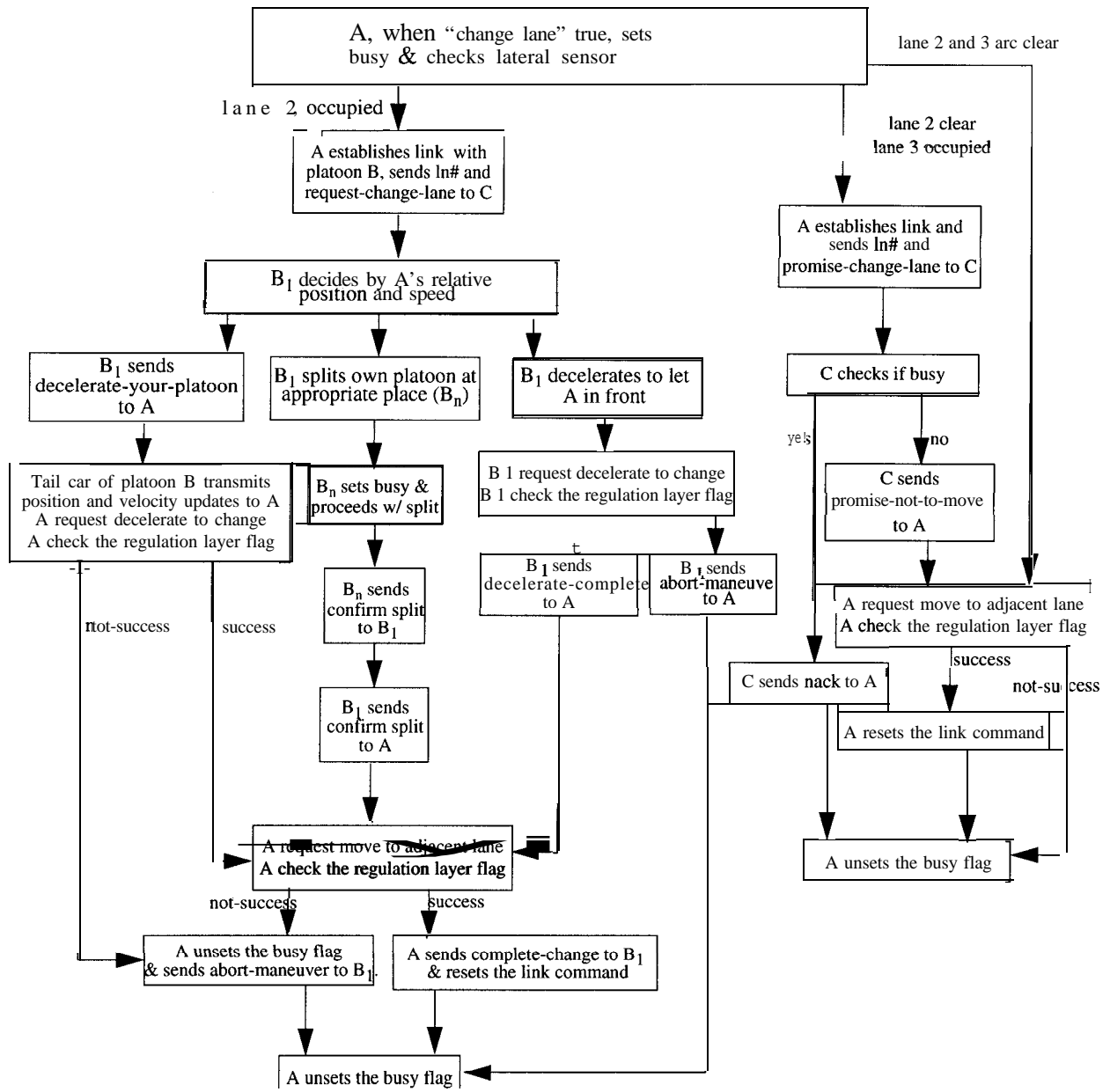


Figure 2.15: Sequence of Events in Change Lane Maneuver. A is the vehicle that wants to change lane, B is the platoon in the adjacent lane (lane 2), and C is the vehicle in the next to the adjacent lane (lane 3).

B. During the deceleration, the tail car of the platoon *B* sends information regarding its velocity and position along the highway to the vehicle *A*, since acquiring this information by direct line-of-sight detection may not be possible. If the deceleration is successfully completed, *A* activates the lateral controller to move the vehicle to the adjacent lane. Depending on the outcome of the maneuver, *A* sends either of the messages *change-lane-complete* or *abort-change-lane* to B_1 .

2. Platoon *B* can decelerate until it reaches a safe distance behind the vehicle *A*. At the end of the deceleration, the leader B_1 sends the message *deceleration-complete* to the vehicle *A*, which causes *A* to activate its lateral controller of the regulation layer. The rest is the same as the previous option.
3. Platoon *B* can split at a suitable position. In this case B_1 initiates the split maneuver and asks the follower B_n to split where n is the appropriate index. Note that the split distance between the two platoon could be as large as twice the safe distance, since presumably vehicle *A* will move in between the two platoons. When split maneuver is completed, B_1 sends the message *split-complete* to the vehicle *A* which causes *A* to activate its lateral controller. The rest is the same as the previous options.

In the first approximation, B_1 's decision is based on the location of *A* relative to the platoon *B* and the safe distance, and one of the above options is chosen depending on the minimum distance of the deceleration, i.e., if vehicle *A* is almost at the head of the platoon *B*, and *B* is a large platoon, platoon *B* decelerates, and if *A* is at the tail of the platoon *B*, vehicle *A* decelerates. Option three is chosen whenever the platoon is a large platoon and vehicle *A* is almost at the middle of the platoon *B*.

In this maneuver, excluding the split maneuver that may occur, there is no state change beside the new lane for vehicle *A*.

2.4.2 PATH-AHS Supervisor Sublayer

A simple and general supervisor FSM for the above maneuver sublayer is shown in figure 2.16. The striped line in the middle divides the supervisor for maneuvers for traveling in the AHS (above the line) from the maneuvers for termination of the trip and the release of the vehicle control by the AVDS (below the line). In the figure the letters A to K correspond to the events to which the supervisor will respond. These events are:

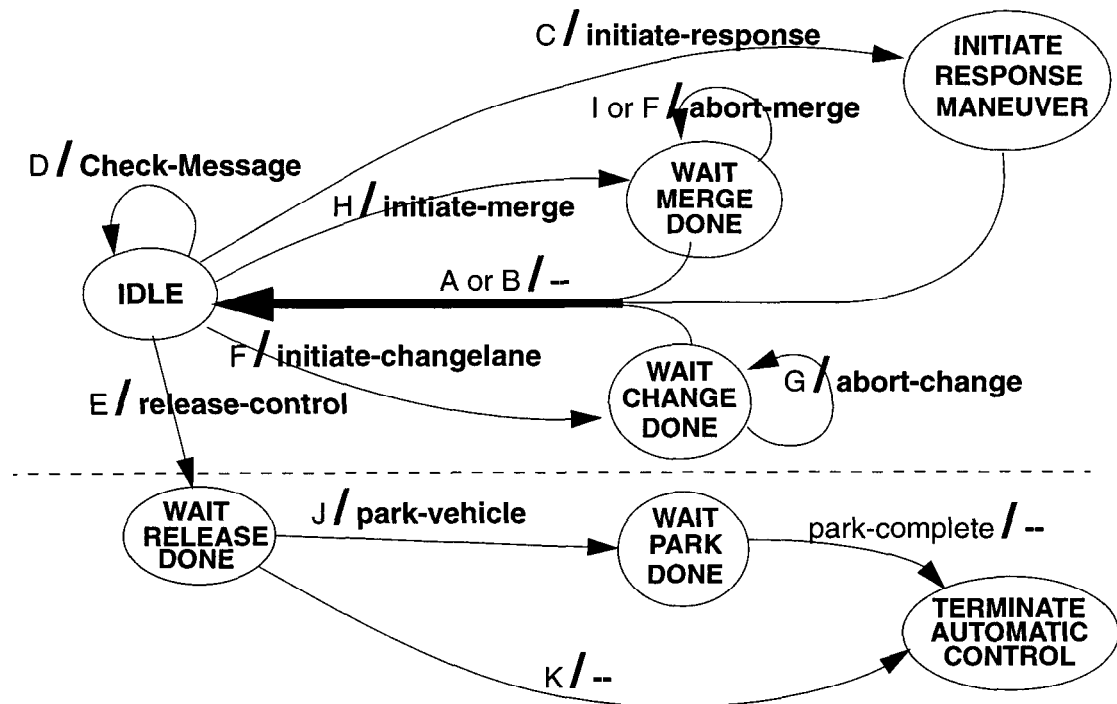


Figure 2.16: A Supervisor Sublayer FSM Model

- *A*: The maneuver sublayer notifies the supervisor that the initiated maneuver was not successful.
- *B*: The maneuver sublayer notifies the supervisor that the initiated maneuver was successful.
- *C*: Vehicle receiver notifies the supervisor of a message from another vehicle.
- *D*: Vehicle receiver notifies the supervisor of a message from roadside controller (link layer controller).
- *E*: Vehicle sensors notify the supervisor that the vehicle is at the end of an exit lane and entering the surface street.
- *F*: Vehicle sensors notify the supervisor that the vehicle is not in the correct lane (the lane the link layer controller has assigned).
- *G*: Vehicle sensors notify the supervisor that at this position on the highway the change lane maneuver is illegal.
- *H*: Vehicle sensors notify the supervisor that there is a vehicle in front.

- *I*: Vehicle sensors notify the supervisor that at this position on the highway the *join* maneuver is illegal.
- *J*: The release of the control maneuver was not successful.
- *K*: The release of the control maneuver was successful.

The events A, B, J, and K are from the maneuver sublayer, C and D are generated from the communication module, and E through I are from sensors module. The *release-control* and *park-vehicle* maneuvers are not yet modeled.

The action that any of these events causes depends on the state of the supervisor, since, as described earlier, the output of a Mealy machine is a function of both the state and the event. For example, if C (arrival of a message from another vehicle) occurs, the supervisor will either initiate a response maneuver (*initiate-response*) or send a busy message back (*send-busy*), depending on whether it is in the *IDLE* state or one of the *WAIT* states, respectively.

The system starts in the *IDLE* state; when the event H or F occurs, it initiates the corresponding maneuver, *join* or *change lane*, and moves to the state *WAIT-CHANGE-DONE* or *WAIT-MERGE-DONE*, respectively. In either state there is a possibility of aborting the maneuver. For example, when event I or F occurs and the supervisor is in state *WAIT-MERGE-DONE*, the output will be *abort-join*, but the system will not change its state, until it gets either of the A or B events, which tells the supervisor that the maneuver sublayer has successfully or unsuccessfully ended the maneuver. This is for the purpose of safety, since the supervisor does not know how the maneuver has progressed, and it is possible that completing the maneuver is safer than aborting it. When the sensors notify the supervisor that the vehicle is at the end of an exit, the supervisor activates the *release-control* maneuver and awaits its result. If the maneuver is successful, the supervisor terminates itself; otherwise, it parks the vehicle in a pre-determined parking lot and, then, terminates itself.

2.4.3 PATH-AHS Regulation Layer Controllers

After the supervisor chooses the maneuver that the vehicle should engage, and the maneuver sublayer acquires permissions from the other vehicles and successfully executes its communication protocol, it is the regulation layer's task to execute and implement the maneuver. For example, when the *join* command is issued from the maneuver sublayer,

the regulation layer should devise a trajectory that accelerates and decelerates the vehicle appropriately, so that **at** the end of the trajectory the vehicle is within the intraplatoon distance of the tail vehicle of the front platoon.

From the description of the maneuver sublayer we see that we need the following controllers within the regulation layer to successfully drive the vehicle in AHS:

- *Accelerate to join* is used by a leader that wants to join the platoon ahead. The controller calculates an acceleration and deceleration trajectory which results in the joining of its own platoon to the front platoon. The primary inputs to the controller are the relative velocity of the front platoon and the distance to the tail vehicle of the platoon. This is the only maneuver that accelerates the vehicle beyond the maximum velocity allowed in the highway, and therefore it is very important that the controller be tested thoroughly in software and hardware to avoid high-speed collisions. The maneuver may be aborted in three cases:
 1. If from one sample time to the next there is a step change in the intervehicle distance, which means that another vehicle has changed lane in between the two platoons. This dangerous condition may cause an accident, depending on the velocity of the joining vehicle and the distance to the lane-changing vehicle.
 2. If the maneuver is not safe to perform, which may happen if the platoon in front is decelerating rapidly.
 3. If the coordination layer requests an abort of the maneuver. In this case, the regulation layer will abort the *join* maneuver, if it is safe to do so, i.e., the maneuver is not almost completed.
- *Decelerate to split*, which is used by a follower that is splitting from its platoon. The controller designs a deceleration and acceleration trajectory, which causes the vehicle to be at the safe distance from the platoon ahead of it. This controller uses only the difference of the speed and distance between itself and the platoon ahead of it.
- *Decelerate to change lane*, used by a free agent that is supposed to move behind the platoon in the adjacent lane. This controller is similar to the *decelerate to split*, but instead of tracking the vehicle in front of it, it tracks the tail vehicle of the adjacent platoon. In order for this controller to be safe, it also has to track the velocity and distance of the vehicle in front of it.

- *Move to adjacent lane*, used by the change lane maneuver. When the vehicle is in the position to move to the desired lane and the sensors do not see any threat from other vehicles, it activates this controller. The inputs to this controller are the sensors value of the relative speed and distance of the vehicles in the vicinity.

We have to add the following two controllers for the default cases, i.e., when the vehicle is not involved in any maneuver.

- *Lead control* is the default controller for the leader of the platoon. *Lead control* is active whenever the vehicle is not involved in a maneuver. This controller operates in one of the following two modes: safety or follow the specified maximum speed. In the first mode the sensors have already detected a vehicle within the safety domain of the vehicle and traveling slower than the specified speed; this causes the *lead control* to calculate a deceleration and acceleration trajectory according to the distance between the vehicles and the velocity difference, at the end of which, the vehicle is at the safe distance from the vehicle in front. In the second mode, the sensors may have detected another vehicle, but it is not within the safety domain. The input to the *lead control* is either the difference of the distance and velocity of the vehicle ahead, if applicable, or the optimum velocity set by the link layer for that lane.
- *Follower control* is the default controller for the followers in the platoon. Since the vehicles are traveling in close proximity to each other (one to three meters), the follower law depends not only on its sensors to track the velocity and the distance to the front vehicle, but also on the acceleration information transmitted to it from the regulation layer of the front vehicle. We will specify the required communication link later in this chapter. The inputs to the controller, therefore, comprise the sensor information and the acceleration of the front vehicle. The communication link is a crucial part of the regulation layer, since, to guarantee safety, as soon as the required information from the front vehicle is not received, the regulation layer informs the supervisor sublayer, which in turn initiates the *split* maneuver.

Figure 2.17 gives the state diagram for the regulation layer.

Since vehicles enter the AHS as free agents, the initial state of the regulation layer is *LEAD-CONTROL*, which implements the *lead-control* controller and waits for the next

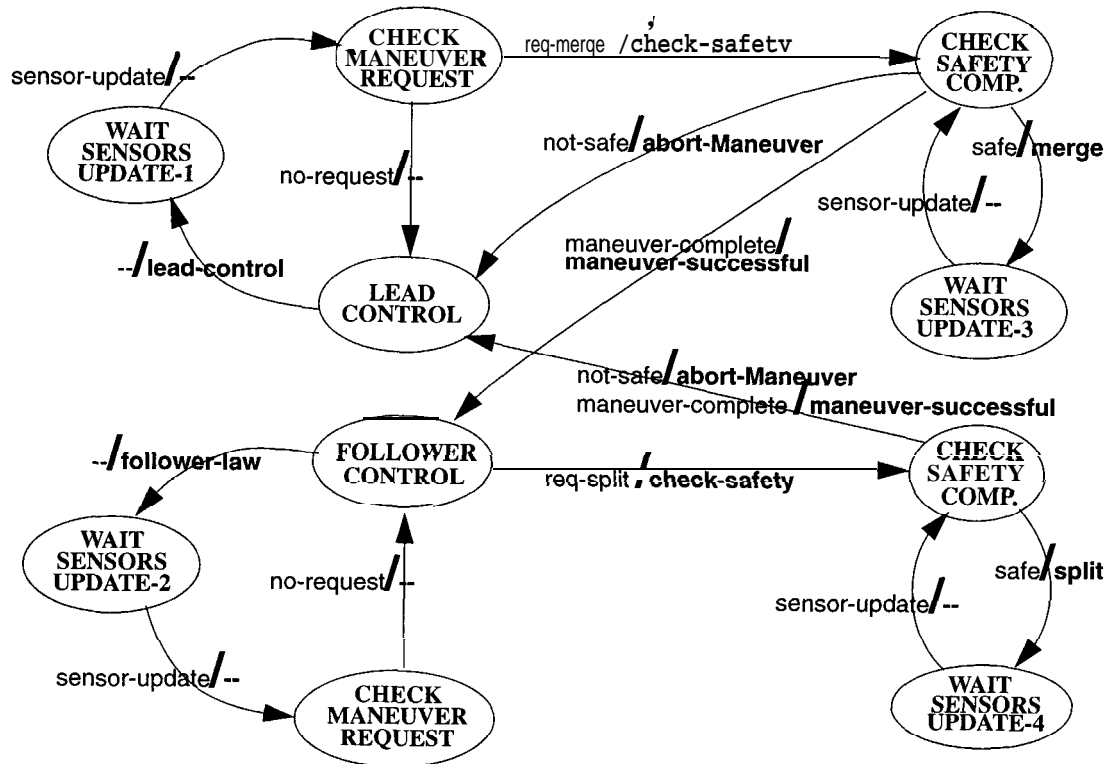


Figure 2.17: Regulation Layer Controller Layout for PATH-AHS

update of the sensors. As soon as the update is available, the regulation layer checks the maneuver requests that may have been issued by the coordination layer. The request can be either *req-join* or *request-move-to-adjacent* (only *req-join* is shown in the figure). In either case the regulation layer checks the safety or completion of the maneuver each time sensors updates are available. When the *join* maneuver is successfully performed the state moves to the *FOLLOWER-CONTROL*, since at this point the vehicle is a follower. In the *FOLLOWER-CONTROL* the only acceptable maneuver is *req-split*, which moves back the state of the regulation layer to the *LEAD-CONTROL* state, whether the *split* maneuver is successful or not. The reason is that after the platoon is split, a join maneuver is needed to join them back.

For a full description of the longitudinal controls (*Accelerate to join*, *Decelerate to split*, and *decelerate to change lane* see [15, 16, 17]; for lateral controls see [18], and for *follower control* see [19, 20]).

2.4.4 Interfaces Among Control Layers

In this section, I summarize the interfaces among the control layers.

Network-Link The network layer provides the routing table to the link layer controllers. The routing table can be accessed by the pair (*patch-i*, *destination-j*) where *patch-i* is the current location of the vehicle and *destination-j* is its destination patch. The link layer provides the average travel time for each patch. The average is taken over the travel times of the lanes in the corresponding patch.

Link-Supervisor The link layer broadcasts a command (change right, change left, straight, stay right, and stay left) periodically to every lane of a section for every reachable destination from that lane. The vehicles receive the command list and using their destination as the key, decode the command specific to their destination. The supervisor submodule also periodically transmits its position (lane and section number) and its destination address to the link layer controller.

Supervisor-Maneuver The supervisor initiates the maneuver that should be performed. It also may request the termination of the maneuver by notifying the maneuver sublayer. However, the maneuver may not be aborted because of safety or other factors involved in the maneuver. The maneuver sublayer notifies the supervisor when the maneuver terminates successfully or unsuccessfully.

Supervisor-Regulation The supervisor sublayer initiates the regulation layer controller and request its termination when necessary. The regulation layer notifies the supervisor

sublayer when the termination is successful.

Maneuver-Regulation The maneuver sublayer request the execution of a maneuver from the regulation layer. The regulation layer notifies the maneuver sublayer of the outcome (successful or unsuccessful) of the maneuver execution.

2.5 PATH-AHS Extension for Mixed Traffic, Mixed Lane Assignment

In order to implement an AHS design incrementally, i.e., to deploy the AHS on part of an existing highway such that some lanes are converted to the AHS use and others are used by the manual traffic as before, we need to provide the necessary controllers for transitions from manual lanes (ML) to automated lanes (AL) and vice versa. One of the proposals for highway configuration which can safely utilize the mixed traffic is the *island* configuration which was shown above in figure 2.1. For safety purposes [21] there are barriers between the manual and automated lanes. Since in the AL the vehicles are driven automatically, a portion of the roadway called *transition lane*, TL, is assigned for transfer of the control from the human driver to the computer on the vehicle. The barriers between the TL and AL are breached by gates at specific points to permit the movement of the vehicles from the TL to AL and vice versa. For other proposed highway configurations and the quantitative analysis in terms of traffic flow improvement, control complexity, and construction costs see [22].

In this section, I will describe the coordination and regulation layer controllers required for the entry and exit maneuvers for an automated vehicle that wants to join the AL or wants to exits the AL, the network layer controller extension, and the extra infrastructure needed to facilitate the entry and exit maneuvers. We do not need an extension of the link layer controllers; however, the link layer controllers have to be designed such that it does not require the periodic message transmission from the manual vehicles, and when some specific vehicle information is needed that cannot be acquired through highway sensors, the link controller has to rely on statistical prediction.

2.5.1 Extended Highway Infrastructure

Check Station At beginning of every TL a check station is installed to test if the vehicles that enter the TL in order to enter the AL are automated and in proper working condition with the capabilities necessary for the AL. The testing can be with the sensors installed on

the check station and a series of message transmissions to verify the state of the controllers on the vehicle. If the vehicle fails any of the test, it is not permitted to enter the AL. The actual enforcement method has yet to be determined.

Stop Light A stop light is installed on the TL at a point where all automated vehicles after passing the check station test stop. The stop light can be used to meter the inflow to the AL, synchronize vehicles waiting to enter and the gaps in the AHS, and initiate the maneuvers between the a platoon in the AL and the vehicle. To perform these tasks, the stop sign is connected to a highway sensor which detects vehicles and gaps in the AL at some distance upstream. The stop sign notifies the vehicle in the TL of the available gap and the communication id of the platoon in the AL in front of the gap, if any.

2.5.2 Extended Network Layer

Recall that the network layer provides the routing table for the AHS. Since the highway is used by both human drivers and automated vehicles (though they are segregated), the network layer provides two routing tables-manual table and an automated table. Routes in the manual table consist solely of the patches which contain the manual lanes, but the routes in the automated table can consist of any patch. In this manner the network layer can also be used as a centralized electronic route guidance for the human drivers driving in the manual lanes. The structure of the network layer as described in 2.2.2 can support both manual and automated vehicles.

2.5.3 Extended Coordination Layer

In the PATH-AHS we defined the behavior of a vehicle to be the following: as leader a vehicle keeps a safe distance from the vehicle in front, and it can change lane, accelerate to join another platoon, and split to create two platoons. A follower keeps a fixed intraplatoon distance and can perform the *split* maneuver. However, when a vehicle is in a transition lane and in transit from manual to the automated mode and from manual lanes to the automated lanes or vice versa, its behavior is different and we need to add a new set of controllers to the maneuver sublayers and a new set of states to the supervisor sublayer.

- **Extended Maneuver sublayer** There are two new maneuvers added to the maneuver sublayer, entry and exit. Each of the maneuvers is a sequence of phases that the controller should go through, as I describe next.

Entry Maneuver When a car enters the transition lane, it should have the ability to become automated and engage in the entry maneuver. The entry maneuver consists of five states, as shown in figure 2.18, which are:

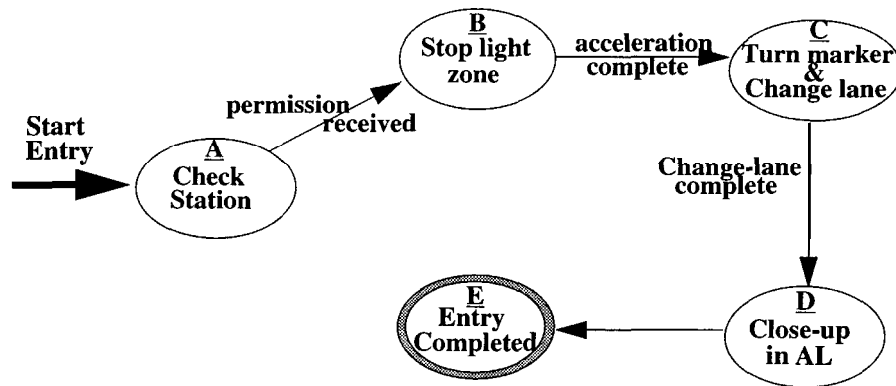


Figure 2.18: Entry Maneuver State Diagram

Check Station Before a vehicle can engage in the entry maneuver, it has to become automated. As was explained above, if the car is automated, it receives the permission to enter the TL and move to its next state, i.e., the *stop-light* zone state. Before reaching the check station the vehicle should be under the automatic control of its coordination and regulation layer controllers.

Stop Light Zone A car in the TL after the check station may halt either behind another car or at the stop light.

In the first case, the car either gets a message from the vehicle in front of it, to become a follower and enter the AL as part of a platoon with the vehicles in front of it, or as the vehicles in front of it depart, its regulation layer, eventually, will notify the it that it is at the stop light.

The state diagram for the second case is shown in figure 2.19. When a car is at the stop light, it transmits a message to the stop light, registering its presence; it then waits for the response from the stop light. The stop light, in turn, activates the roadside sensor on the AL. The roadside sensor, at regular intervals, sends the length of the available gap in the AL to the stop light, and if it exceeds the space needed for a vehicle to change lane to AL, the stop light notifies the waiting vehicle.

The message that the vehicle receives from the stop light has the information about the length of the available gap and the communication id of the platoon

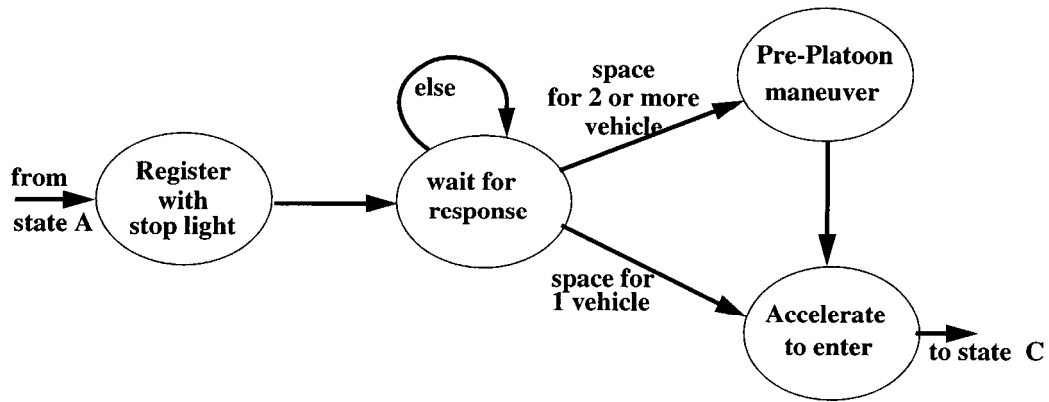


Figure 2.19: Detail of Stop Light State Diagram

nearest to that gap. Depending on the available space, the vehicle decides how many vehicles can enter along with it, and if more than one, it initiates a pre-platooning maneuver which is similar to the join maneuver described in the previous section; if the space is not adequate for more than one vehicle, the vehicle remains a free agent. At this point, the entry maneuver issues the command *accelerate-to-enter* to its regulation layer and waits until the regulation layer controller completes the acceleration and reaches the turn-marker at the gate. While the vehicle accelerates, it requires the velocity and the position of the tail car of the platoon in the AL relative to the entry gate. This information can be acquired through periodic messages that the tail car of the platoon transmits to the entering vehicle (or the leader of the pre-platoon). It may as well be possible that there is no vehicle within the range of the roadside sensor; in this case, the returned communication id is zero.

Turn-Marker As the car reaches the turn-marker, its velocity matches the velocity of the platoon on the AL, if any, or the average speed on the AL. At the turn-marker, the vehicle checks the gate sensors to make sure that the required gap is still available. If the gap is occupied, the entry maneuver is aborted, and the unsuccessful car either proceeds on the TL to retry at the next entry gate or returns to the ML. In either case, the vehicles in the pre-platoon split and become free agents which is the same maneuver as the split maneuver explained before.

When the gap is available, the coordination layer issues a command to its regulation layer to change lane to the AL. The maneuver enters the next state as

soon as the regulation layer completes the change lane.

Close-up In AL After successfully entering the AL, the vehicles perform a close-up maneuver, which causes the pre-platoon members to become followers of the platoon in the AL. After the close-up is completed, i.e., the entering vehicle is at the intraplatoon distance from the tail car of the platoon in the AL, the leader of the platoon in the AL updates the state of the platoon (size of the platoon and new tail car communication id), and requests all followers to update their state, too. At this stage the entry maneuver is completed.

Exit Maneuver When a car wants to exit from AL and enter the TL, its coordination layer controller initiates the exit maneuver protocol machine. The exit maneuver is symmetric to the entry maneuver. If there is more than one vehicle in a platoon that wants to exit, they will eventually form another platoon in the TL. Figure 2.20 shows the informal state diagram of the exit maneuver for leader of a platoon, and figure 2.21 is the state diagram for an exiting follower.

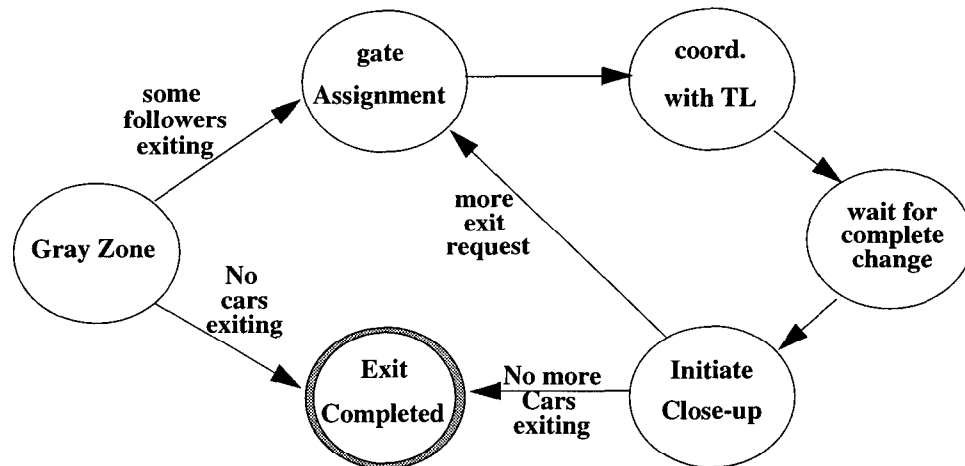


Figure 2.20: Exit Maneuver State Diagram- Leader

Gray Zone At the start of every section that contains exit gates is a zone in which every platoon leader accepts the *exit request* messages from its followers. We call it the gray zone. At the end of the gray zone, the leader assigns to each exiting vehicle in its platoon a gate number, if there are enough gates in that particular section. The leader also compares the number of requests with the number of gates to determine whether there is a possibility of reassignment for an exiting

follower, if it fails to exit at its assigned gate. If the leader itself is among the exiting vehicles, it should notify the second vehicle in the platoon to become a leader. This transfer of leadership is required every time the current leader wants to exit.

Coordination with TL Before each gate, the lead vehicle on the AL communicates with the leader on the TL, if any, to determine if the next exiting vehicle will be a leader or a follower upon entering the TL. The leader on the TL may be a vehicle from an entry section (if the exit section follows an entry section) which had aborted its entry maneuver, or a vehicle previously belonging to the platoon and exited from a previous gate. The platoon leader on TL requires the information regarding the position of the exiting vehicle in the platoon to execute the catch-up maneuver, the maneuver that aligns the head or tail of the platoon in the TL with the exiting vehicle. This maneuver ensures that each exiting car will remain in the “shadow” of the platoon in AL, which is an important factor in the safety criteria [22].

The leader of the platoon notifies each exiting vehicle in the platoon of its assigned gate before the platoon reaches the gate. Upon receiving the assignment, the vehicle will direct its regulation layer to initiate a change lane maneuver; the regulation layer waits until the turn-marker on the gate appears and then initiates the move *to adjacent lane* regulation layer controller.

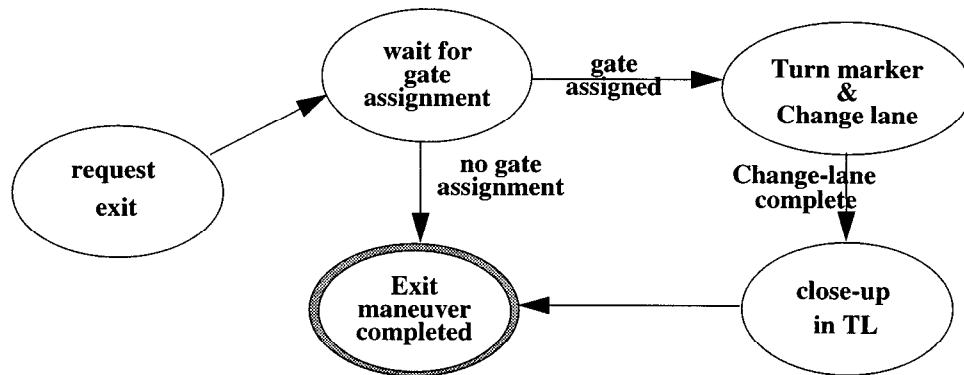


Figure 2.21: Exit Maneuver State Diagram- Follower

In the interim , the platoon on the TL has executed a catch-up maneuver and is ready to accept this newly exiting car. When the vehicle enters the TL successfully, it will send the *exit complete* message to both leaders in the AL and TL.

The AL platoon leader then asks the vehicle behind the exiting vehicle to initiate a *close-up* maneuver which closes the gap created by the exiting vehicle. If the vehicle is unsuccessful in exiting, which is possible if the vehicle has not started the *move to adjacent* lane controller, the leader on AL may reassign a new gate, if one is available, and this vehicle may retry the exit maneuver later. At the turn-marker, the exiting vehicle checks the gate sensors to find out if the space it wants to move into is vacant. If the space is not free, the exit maneuver is aborted.

The above procedure is repeated until either no gate or no exiting car remains. At the end of the exit section, the platoon on the TL decelerates to match the average velocity on the ML. By the end of the TL, the members of the platoon have already executed the required split maneuvers, and all are free agents. A final change lane and switching of modes from automated to manual brings the vehicles back on the ML and successfully concludes the exit maneuver.

- **Extended supervisor sublayer** The extension to the supervisor sublayer is shown in figure 2.22 where the letters *A-E* are events already described in section 2.4.2.

When the vehicle is in the TL, it initiates the supervisor sublayer in the *IDLE-TL* state. The supervisor then checks the position of the vehicle, and if it is in an entry section, it initiates the entry maneuver and awaits its outcome. If the maneuver is successfully completed which means that the vehicle is in the AL, the supervisor sublayer moves to the *IDLE-AL* state. If the entry maneuver is not successful, the vehicle should be prepared to reenter the manual lanes.

The exit maneuver is initiated when the vehicle has to change lane to an adjacent transition lane. If the exit maneuver is successful, the next state of the supervisor is to prepare the vehicle to reenter the manual lane; otherwise, the supervisor moves back to the *IDLE-AL* state.

2.5.4 Extended regulation Layer

A vehicle in the transition lane, in addition to the *lead controller*, *follower controller*, and *move to adjacent lane* which were described above, have the following controllers in its regulation layer:

- *Stop-light* controller is activated when the vehicle is in the stop light zone. This controller designs the trajectory so that the vehicle stops when it reaches the stop

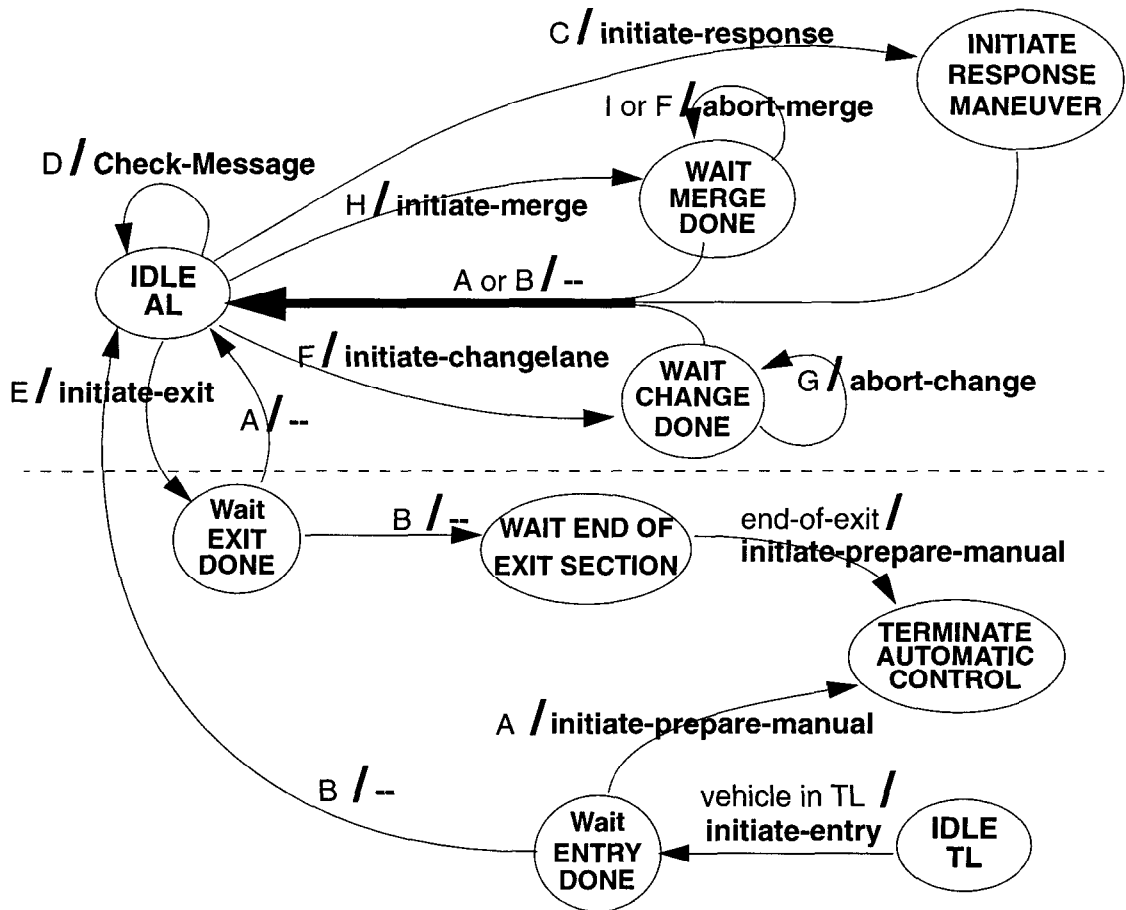


Figure 2.22: The Supervisor Sublayer for the Extended PATH-AHS Proposal

light. This feedback controller is similar to the *lead controller*. If there is a queue at the stop light, the vehicle stops at the intraplatoon distance behind the last car in the queue. This maneuver requires the relative distance between the vehicle and the stop light.

- *Accelerate-to-enter* controller is initiated after the stop light notifies the vehicle of an acceptable gap in the AL. The controller calculates a trajectory for the stopped vehicle to accelerate so that by the time the entering vehicle reaches the gate it has the same velocity as the platoon in the AL (or the average speed of the AL) and is behind the platoon in the AL at the intraplatoon distance. It requires the velocity and the relative distance of the AL platoon and its own distance from the gate during the execution of the controller. The controller aborts, if the alignment of its vehicle to the target platoon in the AL is not within the specified safety tolerances.

- *Accelerate-to-close-up* controller executes a trajectory at the end of which the vehicle is at the intraplatoon distance of the platoon ahead of it. It is activated by the vehicle entering the AL and by the vehicles behind an exiting vehicle when the exit is completed. This controller requires the velocity of the platoon ahead. The controller may abort, if the platoon ahead is dangerously close and has smaller velocity.
- *Accelerate-to-catchup* controller aligns the platoon in the TL to the platoon in the AL so that the exiting vehicle from the AL upon finishing the change lane maneuver becomes either the leader or the tail car of the platoon in the TL. The controller requires the relative position and the velocity of the exiting vehicle, and it is aborted whenever the exiting vehicle is dangerously close to the platoon.
- *Prepare-manual* controller of each vehicle is activated when the vehicle is in the transition lane and eventually will enter the manual lanes of the highway. When executed, the controller decelerates the vehicle until it is at the manual safe distance from the vehicle ahead of it.

2.6 Support Modules for PATH-AHS Design

At the beginning of this chapter we made a set of assumptions regarding the availability of appropriate sensors and communication facilities for vehicles and highways. I will now quantify the minimum sensor and communications requirements for deployment of the PATH-AHS proposal.

2.6.1 Vehicle Sensors

The sensors on the vehicle are mainly used for detection; however, in some cases they may be used to decode the information (usually the vehicle's distance from some feature on the highway) that is provided from the roadside; in the latter case, the type of encoding determines what types of sensors are needed.

Detection The vehicle should be able to detect the following:

- Sensors should provide the relative velocity and the distance of the nearest vehicle in front and in adjacent lanes. There are two sets of sensors on the vehicle, longitudinal and lateral, shown in figure 2.23.

The longitudinal sensors are used for safety purposes and join maneuver; therefore,

they should be able to detect any object which is in the path of the vehicle. The range of detection depends on the maximum allowable speed, the maximum deceleration, and the rate at which the vehicle can achieve such deceleration. For example, if the speed is 25 m/sec , maximum deceleration is 5 m/sec^2 , and the vehicle can achieve the maximum deceleration in one second, then the distance needed to stop a vehicle is about 75 meters¹ Thus, the detection range for the sensors should be at least 75 meters, which allows a vehicle to stop before it hits a disabled vehicle or any other non-moving object on the highway.

The lateral sensors are mainly used in the change lane maneuver. As we said earlier the change lane checks both the adjacent lane and next-to-adjacent lane, so as not to move in between two joining platoons, and not to have a collision with another vehicle that is changing into the same lane close by. Therefore, the lateral sensors should be able to detect vehicles in adjacent lanes in front and in the back of the vehicle with at least the same range as in the longitudinal sensors; otherwise, there is the possibility of high speed crash between a joining vehicle and the changing-lane vehicle.

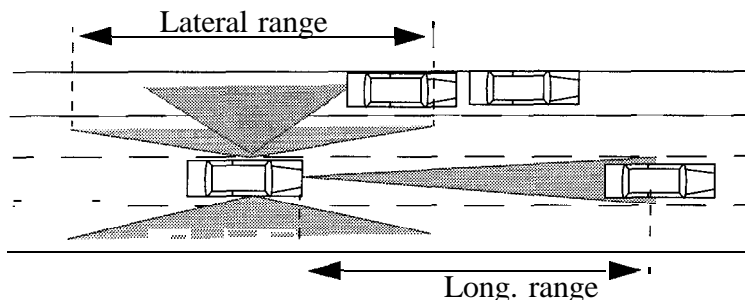


Figure 2.23: Required Longitudinal and Lateral Sensors on the Vehicle

- Sensors should provide the following information regarding the highway and the lane the vehicle is traveling on: section number, lane type (automated, transition, or manual) and the lane number which are used by the coordination layer to access the routing provided by the link layer, existence of barriers and the beginning and end of the gates used in the entry and exit maneuvers by the extended version of PATH-AHS plan and end-of-exit notification to the coordination layer in order to disable the

¹During the one second it takes for the vehicle to achieve the maximum deceleration of 5 m/s^2 , the vehicle's acceleration follows $a = -5t$, the vehicle's velocity is $v = 25 - \frac{5}{2}t^2 = 22.5 \text{ m/s}$ after the one second transition time, and the distance the vehicle has traveled is $x = 25t - \frac{5}{6}t^3 = 24.2 \text{ meters}$. After the first transient, the vehicle deceleration is a constant (5 m/s^2), and it takes $\frac{v^2}{2a} = \frac{(22.5)^2}{10} = 50.63 \text{ meters}$ for the vehicle to stop. Therefore, the total stopping distance is 74.8 meters.

automatic vehicle controller and hand over the vehicle to the human driver.

Information Decoder In the entry and exit maneuvers, the regulation layer of the vehicle on different occasions needs to know the distance of the vehicle from the stop light, entry gates to the automated lanes, and exit gates to the transition lanes. This information may not be acquired through direct line-of-sight detection. Therefore, they are encoded in the highway, so that sensors can read and decode them as the vehicle travels along the highway. As we saw earlier in the description of the maneuvers, since this information is only needed when the vehicle wants to enter or exit the automated lanes, the decoding sensors are turned on by the coordination layer, as the need arises.

2.6.2 Highway Sensors

Two types of highway sensors are needed for the PATH-AHS proposal:

1. Sensors that provide information regarding the state of the traffic (traffic flow and average velocity) to the link layer controllers.
2. Sensors that provide occupancy information to assist the entry and exit maneuvers. The occupancy information is needed when no direct line-of-sight detection by the vehicle is possible. For example to design the metering scheme for the entry maneuver, the controller on the stop light requires the traffic gap information at a specific point on the highway.

2.6.3 Highway-Vehicle and Vehicle-Vehicle Communication

Highway-vehicle and vehicle-vehicle communication is an active area of PATH research. Previous work and proposals can be seen in [23,24].

The highway-vehicle communication is in the form of broadcast. Since the information is grouped according to the section and the lane the vehicles are traveling on, different channels should be allocated to each section and lane, with the possibility of channel reuse when the sections are far apart and interference is limited, Every vehicle has to tune its receiver to the new channel as soon as it enters a new section. The reverse channel (vehicle to the highway) can work in the same manner, but since the number of vehicles can be large (up to the capacity of a lane in a section), there should be a robust scheme to minimize the interference among the transmitters.

Communication among vehicles can be divided into intraplatoon and interplatoon communication. Currently in the field tests intraplatoon communication is handled with radio communication under the token bus protocol ([24]). Since the follower controller in the regulation layer requires a periodic update of the acceleration and velocity of the front vehicle and the platoon leader, and the token bus has a finite turn-around time for the token, only a few vehicles can be part of a platoon. Also, the bandwidth that is used for the token bus may be used for the interplatoon communication. A better communication scheme may be based on the infra-red communication, but it has not yet been tested [25, 26]. To perform a maneuver, the vehicles must have point-to-point communication among themselves. Therefore, every vehicle listens to a general shared frequency and a specific frequency (*a communication address*). To establish the point-to-point link the vehicles may use the shared frequency to transmit their request with their communication address encoded in the message. The appropriate vehicle should respond directly to the requesting vehicle using the encoded communication address.

2.7 Concluding Remarks

The protocols of section 2.3 were verified at various stages of the protocol development by Ann Hsu and myself. The maneuvers have been extensively simulated using SmartPath. Subsequently, Ekta Singh and Sonia Sachs have verified the communication protocols of section 2.5. For the verification, we have used the COSPAN software package.

The question that may arise at this stage is whether these protocols are implementable on a real vehicle. At first glance, we might be tempted to answer this question positively. Though the answer may be correct, we shall test the complexity of the algorithms in a real-time operating system to evaluate the performance of the system and to see whether it is feasible to acquire the needed information from sensors and communication devices in real time. This has not been done yet.

At this point we have finished the modeling of the Automated Highway System. The next section describes the simulation environment and how the SmartPath simulation is organized.

Chapter 3

AHS Simulation

3.1 Introduction: Simulation of Hybrid Systems

The Automated Highway System is a hybrid system in which both continuous and discrete elements are present. The coordination layer and the communication protocols employed in the maneuver sublayer are discrete controllers, and the vehicle's actuators, the sensors, and the feedback control laws employed in the regulation layer are continuous elements of the system. The continuous element is sometimes called the plant and the discrete element is called the controller. The plant is usually described by means of differential or difference equations, and the discrete controllers by discrete event description languages like FSMs and Petri nets. The output of the plant will generate a set of input events for the FSMs and the discrete controllers. For example, the detection of a vehicle in the safety zone of another will cause an event for the safety controllers of the latter vehicle. The controller after making its transitions will output a set of symbols that encode trajectories which should be followed by the continuous plants; however, since there may be more than one discrete controller, it is possible that more than one transition takes place. Let us look at an example. In the join maneuver, as we saw before, the coordination layer has to be notified of the detection of a vehicle in front of it; the detection comes from the vehicle's sensors which generate an event for the supervisor sublayer of the coordination layer. The coordination layer then performs the required protocol and if successful, asks the regulation layer for the execution of the maneuver which in turn designs a trajectory for the maneuver. Until this point we are in the realm of the discrete controllers. the desired trajectory is followed by the accelerate *to* join feedback controller which produces the control parameters (desired jerk or acceleration) for the actuators of the vehicle.

The state of the hybrid system is the state of the discrete controller together with the state of the continuous plant; the former defines the trajectory and the latter the actual position of the system on that trajectory. But how does the plant behave during the time it takes for the controllers to decide on the next trajectory? If we assume that the transitions all occur in zero time, i.e., the communication protocols are executed instantaneous by then time stops during discrete state transition and the plant will not evolve. The assumption can be justified, if the discrete controllers can reach the decision much faster than the sampling period of the observation. For example, if the sampling period T is 0.1 seconds, then a communication protocol between two entities, each entity modeled as an FSM, can be assumed to be instantaneous, since the speed of communication is much faster than 0.1 seconds, and virtually infinite number of transactions can happen within the 0.1 seconds.

In order to simulate a hybrid system, we build a scheduler for the execution of each layer of the system, continuous and discrete. In its most general form, such scheduler has two phases. In the first phase, it schedules the continuous plants and increments the simulation time by T seconds; in the second phase, after the continuous plants are simulated, the discrete controllers are scheduled for simulation, and during this step the simulation time is stopped.

In this chapter we will describe SmartPath simulation system which provides the environment to simulate the control layers and the continuous elements of the Automated Highway System.

3.2 SmartPath Simulation Environment

SmartPath is a simulation environment for the AHS. It has a simulation program and an animation program; the animation can be used to show the simulation results as the simulation progresses (concurrent visualization), or it can read the simulation results from a simulation-prepared data file. The latter case is more efficient and better suited for long simulation runs; however, with concurrent visualization, one can use the animation to choose a vehicle and change the control parameters interactively, see [27,28] for information on how to use SmartPath. SmartPath's scheduler is written on top of CSIM, which provides a light-weight, quasi-parallel, process-oriented simulation environment. For further information regarding CSIM, see [29]. The network layer and the link layer are each modeled as one process within the simulation. However, every vehicle is modeled as a collection of processes, since each vehicle has a coordination and a regulation layer controller and other supporting

modules like communication and sensors.

Although we have optimized SmartPath for the PATH-AHS proposal, any proposal that can be mapped to the four layer control hierarchy, described in chapter 2, can be modeled and simulated in SmartPath. To show how an AHS proposal can be simulated within SmartPath, I will describe three sets of constructs-primary, secondary, and tertiary-that are provided within this framework.

3.2.1 SmartPath Primary Constructs

The primary constructs in SmartPath provide the basic structures for the simulation of any AHS scenario that complies with the four layer control hierarchy, described in section 2. The highway structure is the basis for the vehicle movement, the detection schemes, and the interactions among the network, link, and coordination layers of the vehicles. The highway is modeled (as described in detail in section 2) by sections and lanes. Every section contains a number of segments, which in turn contain the geometrical information about that section; a segment can be either a straight line or an arc of a circle; in the former case its only attribute is its length, but in the latter case it has length, the arc radius, and the direction of turn which can be left or right (see figure 2.1).

On top of the implementation of this highway model, we provide the following functions:

- A set of routines to create the four different control layers-network, link, coordination and planning, and regulation layers.
- A set of routines to populate the highway to some initial condition, and generate vehicles from the entrances (sources) at some fixed time intervals. The created vehicles are initialized by activating their coordination layer, and each is given a unique id. This id is used to identify the vehicle for communication and sensing purposes, as we will see later. Since the coordination layer of the vehicle is the only layer initialized within the vehicle, it should in turn initialize the regulation layer, sensors, and communications, if desired.
- Every control layer has a wake-up routine, used by the scheduler to activate the time driven part of the control layer, if any. The simulation calls the *wake-up* routine at every simulation time step.
- A set of support routines to ease the burden of AHS implementation. Among these

are a simple fixed step *Kutta-Mersa 4-5* integration routine for non-stiff differential equations and a simple vehicle model.

- Simple detection scheme (for implementing a sensor process) for vehicle-vehicle and highway-vehicle detection. There are two sets of sensors: the vehicle sensors and the highway sensors. The vehicle sensors provide the vehicle the relative distance and the velocity of the front vehicle, the nearest right-adjacent vehicle, and the nearest left-adjacent vehicle. The highway sensors provide the highway controllers aggregate information, like the average flow of vehicles and the average velocity in a specific lane of a section. The sensor requirement is, as explained in Chapter 2, the detection of the vehicles in front, in the right and left adjacent lanes, and in the right and left next-to-adjacent-lanes, if applicable. These regions are shown in figure 2.23. To simulate a such sensor scheme we place a grid on every section. The length of each cell in the grid depends on the geometry of the section. The length of the largest possible cell on all sections is equal to the length of the smallest vehicle present in the simulation. The grids for the two cases of line section and arc section are shown in figure 3.1.

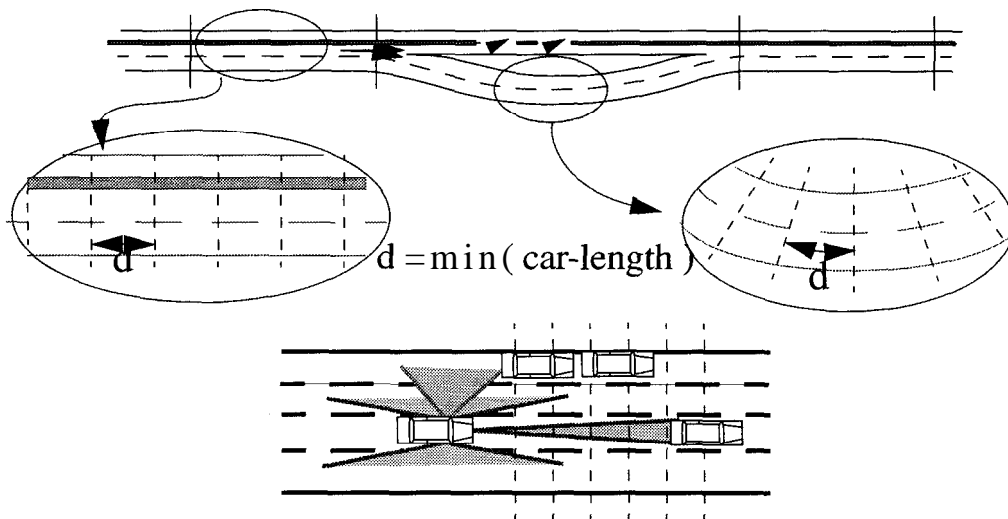


Figure 3.1: Grid structure on the highway for detection of vehicles

As the vehicles travel along the highway, they are placed in the appropriate cells by the simulation, and because the length of the cells is the minimum length of the vehicles in the simulation, a cell can be occupied at most by one vehicle; otherwise, there is a collision. The sensors now only have to traverse the cells around the vehicle

for detection, and if they are occupied a detection has been made. For example, the front sensor will traverse the cells in front of the vehicle that are in the same lane. When a vehicle is detected, its velocity and relative distance is returned.

- The simulation and animation can run simultaneously. The connection is provided by the Unix interprocess communication, so it is possible to run the simulation on one workstation and the animation on another. This feature will be used in the secondary constructs to provide the interactive change of the control parameters while the simulation is running.

The following interfaces exist between the control layers:

- An event delivery system between coordination and regulation layer controllers, which can be extended to other layers. Each layer defines a set of events to which it will respond. The events function in the same way as the input symbols in the FSM definition, which may cause a transition in the FSM. Event names should be provided by the layer developer.
- A simple communication scheme between the coordination layers, such that the coordination layer of a vehicle can activate the transmitter and the receiver modules in order to send or receive messages. The transmitter and the receiver form an error-free point-to-point communication system.

At the heart of the simulation is the scheduler, which activates the control layers and synchronizes vehicle movements. The `SmartPath` scheduler calls each layer according to the following cycle:

1. Update the position of all vehicles on the highway. The synchronous updating of the positions of the vehicles allows the vehicle sensors to return consistent values, i.e., the returned value will not depend on the fact that the regulation layers of some vehicles are activated after the sensors detected the vehicle.
2. Call the regulation layer wake-up routine.
3. Wait until the regulation layer of all vehicles are activated and the new acceleration, velocity, and coordinate (X, Y) increments are calculated.
4. Move all vehicles in the highway by the coordinate (X, Y) increments provided by the regulation layers in the previous step and calculate their new positions and cell locations on the highway for detection purposes.

5. Call the coordination layer *wake-up* routine.
6. Call the link layer *wake-up* routine.
7. Call the network layer *wake-up* routine.
8. Wait until all the control layers are executed.
9. Increment the simulation clock by the simulation time step, and go to 1.

Note that the coordination, link, and network layer controllers are called at the same time; therefore, there is no guarantee on the order of their execution.

These primary constructs provide the basic means to create different layers and to initiate the interactions between them. The animation program can then be used to observe the behavior of the traffic or individual vehicles. However, since the constructs are primitive, the AHS developer has to create all aspects of the AHS, e.g., the controllers to move a vehicle, the communication protocols for sending and receiving maneuver requests, and so on, which is a burden if the purpose of the AHS developer is to experiment with one of the control layers. The burden is even greater, if the developer only wants to see whether the performance of a maneuver is satisfactory. For these reasons, we implement the secondary constructs that are more specific but provide greater functionality.

3.2.2 SmartPath Secondary Constructs

The secondary constructs are created to ease the implementation of a specific layer or a controller within the layer without worrying about how other layers and controllers are implemented. This requires a set of robust interfaces between modules as I explain after I introduce the new constructs.

- The complete structure of the network layer is incorporated in SmartPath. The network layer structure closely follows the description in Chapter 2. The shortest path routing is obtained using the Bellman-Ford shortest path algorithm ([9]).
- In the coordination layer, we have created the two sublayers: supervisor and maneuver sublayers. The supervisor layer is modeled according to the discussion in section 2 as Finite State Machines. In the maneuver sublayer we have added the verified protocol machines for *join*, *split*, and *change lane*. For the *join* and *split* maneuvers to make sense, we need the concept of platooning and point-to-point communication between the vehicles. There are two *change lane* protocols in the maneuver sublayer: manual

change lane which uses sensors as the only means of coordination between the involved vehicles (no communication between vehicles) and *automated change lane* which uses radio communication to coordinate the adjacent vehicles.

- In the regulation layer a set of controllers to accelerate, decelerate, and move the vehicle to an adjacent lane are included. The regulation layer of the vehicle can either be in the lead-mode (when the vehicle is a leader or free agent as defined in Chapter 2), or in the follow-mode. When the regulation layer is in the follow mode, it expects to receive a control message (speed, acceleration) at every simulation time sample. A brief description of these controllers is in Chapter 2 and [30].
- We extend the simultaneity of the animation and simulation and provide for interactivity between them so that control parameters can be changed through the animation as the simulation runs.
- The simulation can be stored at a specific time and restored at a later time. This feature can help to re-run a maneuver with different parameters and observe the differences.

The primary interfaces are too general to be of very good use for these extensions. Therefore, we specialize them as follows:

- The communication facilities for vehicle-to-vehicle and highway-to-vehicle message transmission is included in the secondary constructs. There are three means of communication: point-to-point radio, broadcast radio, and infra-red. The distinction between the radio communication and infra-red is that radio communication has a larger area of propagation and can be transmitted over relatively large distance, whereas infra-red has a limit of 50 meters. Therefore, infra-red communication is used only for intra-platoon communication. We specialize each type even further. The point-to-point radio communication provides message-passing from the supervisor or maneuver sublayer of one vehicle to the supervisor or maneuver sublayer of another vehicle. The broadcast radio is used to transmit messages to all the vehicle in a lane of a section. The messages are used by the link layer controllers to send route information to the vehicles. The infra-red communication is used to transmit control messages between the regulation layers of the vehicles within the same platoon. The infra-red communication is also point-to-point, and the current controllers use it to send control information (velocity, acceleration) to the vehicle behind in the same platoon.

- The event delivery system is extended to include the supervisor, maneuver, and regulation layer. In the present implementation of coordination and regulation layer (SmartPath3.0), the only event issued by the supervisor sublayer to the regulation layer is the *abort-regulation-process* event, which causes the regulation layer to terminate itself. The request for a maneuver execution is only issued by the maneuver sublayer to the regulation layer.

The secondary constructs allow us to model and simulate the PATH proposal for a fully Automated Highway System, see [31,32] for more details on simulation and modeling and [33,11] for some performance results. But in order to simulate the extended PATH-AHS scenario, described in Chapter 2 which includes mixed traffic and mixed lane assignments in the highway, we need to add even more structure and functionality. These tertiary constructs are explained next.

3.2.3 SmartPath Tertiary Constructs

The tertiary constructs are created for simulation of the extended PATH-AHS scenario. The following are added to the secondary constructs:

- In the tertiary constructs, the structure of a lane also contains the type of the lane (which can be automated, transition, and manual) and existence of the barriers, gates, checking station, and stop-sign. A barrier has a beginning and an end and can be placed on all lane types. Gates are assumed to be at the places where there are no barriers. A checking station has a position on the highway, and a stop-sign has a position on the lane and is associated with a highway sensor in the automated lane upstream. Both the checking station and the stop-sign can only be placed on the transition lanes.
- Two operational modes are defined for the vehicles in SmartPath, automated and manual. To ease and decouple the implementation of different maneuvers for the vehicles in different lane-types, we provide multiple behaviors for each operational modes in the coordination layer. The number of behaviors is equal to the number of lane-types available. The manual mode is designed solely for the interactive simulation and visualization, so that when a vehicle is set to operate in the manual mode, its regulation layer takes the acceleration and steering command directly from the SmartPath graphic user interface. In this fashion, the user can “drive” a vehicle in the highway. Within the automated mode, we have three behaviors: automated,

transition, and manual. The automated behavior is the same as the secondary constructs with the addition of exit maneuvers to the maneuver sublayers. The transition behavior encapsulates the extended PATH-AHS proposal with checking-station, stop-sign, and entry maneuvers, and the required preparation for an automated vehicle to be re-introduced to the manual lanes. The manual behavior only has the lane change maneuver with the *lead controller* in its regulation layer. The supervisor of every behavior knows about the required support modules such as vehicle-to-vehicle communication or vehicle-to-highway communication. In order to switch from one behavior to another, the previous supervisor sublayer should prepare the vehicle for the new supervisor sublayer by terminating all on-going maneuvers and deactivating its support modules. The switching from automated mode to the manual mode follows the same guidelines with the difference that if a vehicle that is chosen to become manual (by the user) is a follower, the supervisor of that vehicle first initiates a *fast split* maneuver, so that it becomes a free agent in the shortest possible time, and it then switches the control of the vehicle to the manual mode supervisor sublayer.

- A simple behavior model of the check station and stop light is added to the simulation. The check station uses a highway sensor to detect vehicles in its vicinity. As soon as a vehicle is detected, a message is sent to it. If the message is answered by the vehicle, the check station issues a permission to that vehicle; otherwise, no permission is granted. The stop light is activated by a message from the vehicle, indicating its presence at the stop light. The stop light then activates the highway sensor located upstream in the automated lane adjacent to the transition lane. The sensors periodically returns the available gap to the stop light, and when the gap is large enough to accept one vehicle, the stop light notifies the waiting vehicle. The stop light then waits for another message.

In the third release of SmartPath, SmartPath3.0, all three constructs are included. In the next section I will describe the performance of the SmartPath3.0 simulation package.

3.3 Performance of the Simulation

The SmartPath simulation program consists of about 50,000 lines of code, excluding the CSIM library. Its execution time and memory usage is a linear function of the number of vehicles in the simulation: as the number of vehicles (processes) in the simulation

increases, the time it takes to simulate one simulation time step increases proportionally. Figure 3.2 shows the CPU time for simulation of different number of vehicles for one minute of simulation time, and figure 3.4 shows the memory usage of the simulation. Both figures are for a 80 km. long highway and 0.1 seconds simulation time step. To calculate the time and memory usage for N vehicles, we ran the simulation program ten times and averaged the results. Each of the ten simulation runs was initialized with N vehicles using a different intervehicle spacing and platoon size. Our studies have shown that on the average 80% of the simulation time is spent for the calculation of vehicle trajectories. Therefore, we conclude that the configuration of the vehicles in the highway and the extent of their involvement in maneuvers have had little effect on the memory and timing results.

As we see from figure 3.2, SmartPath simulates 70 vehicles in real time (i.e., it takes 1 cpu minutes to simulate one simulation minute), 120 vehicles in 1.60 times real time, and 300 vehicles in 4.6 times real time. The graph remains linear for larger numbers of vehicles; for 1800 vehicle it took about 27 times real time with a memory requirements of about 20 megabytes.

These experiments with SmartPath were performed on a SGI Indigo II workstation with MIPS R4400 CPU (200 MHz), 32 MB memory, and IRIX 5.3 operating system.

The simulation performance under Sun SPARC-5 workstation is shown in figure 3.3.

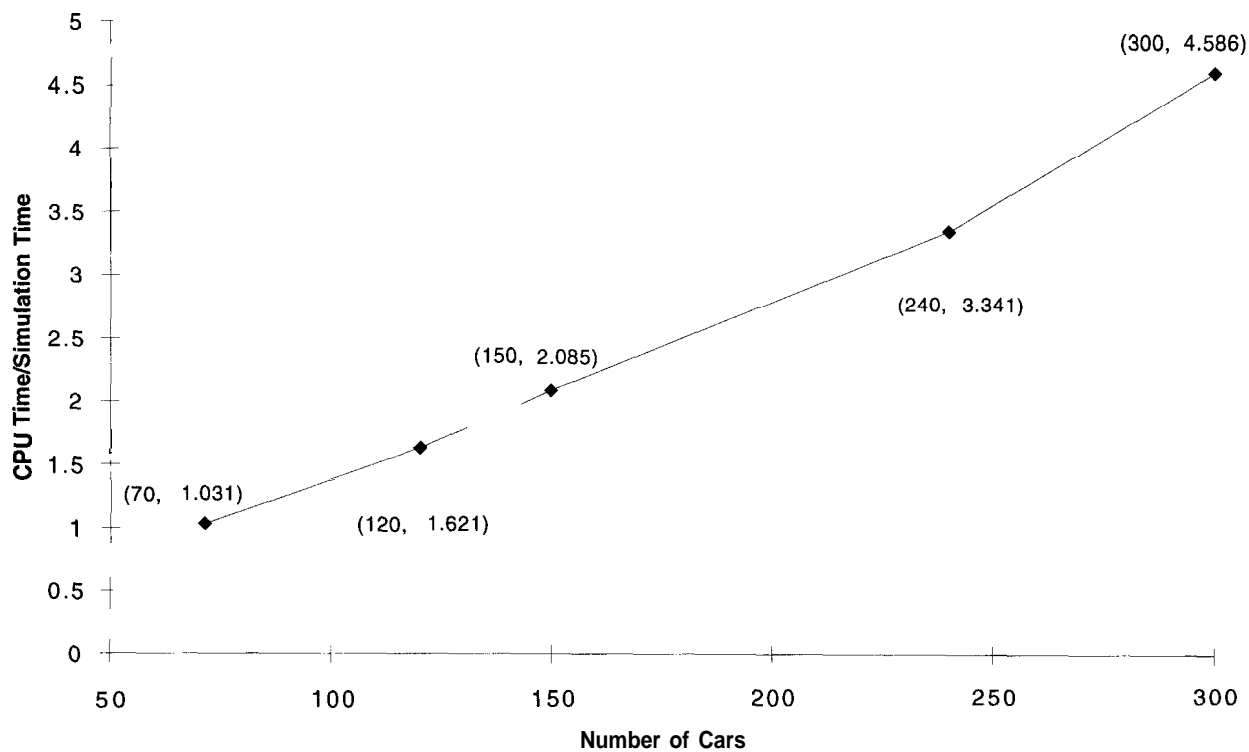


Figure 3.2: Real Time Spent per Simulation sample time as a Function of Number of Vehicles

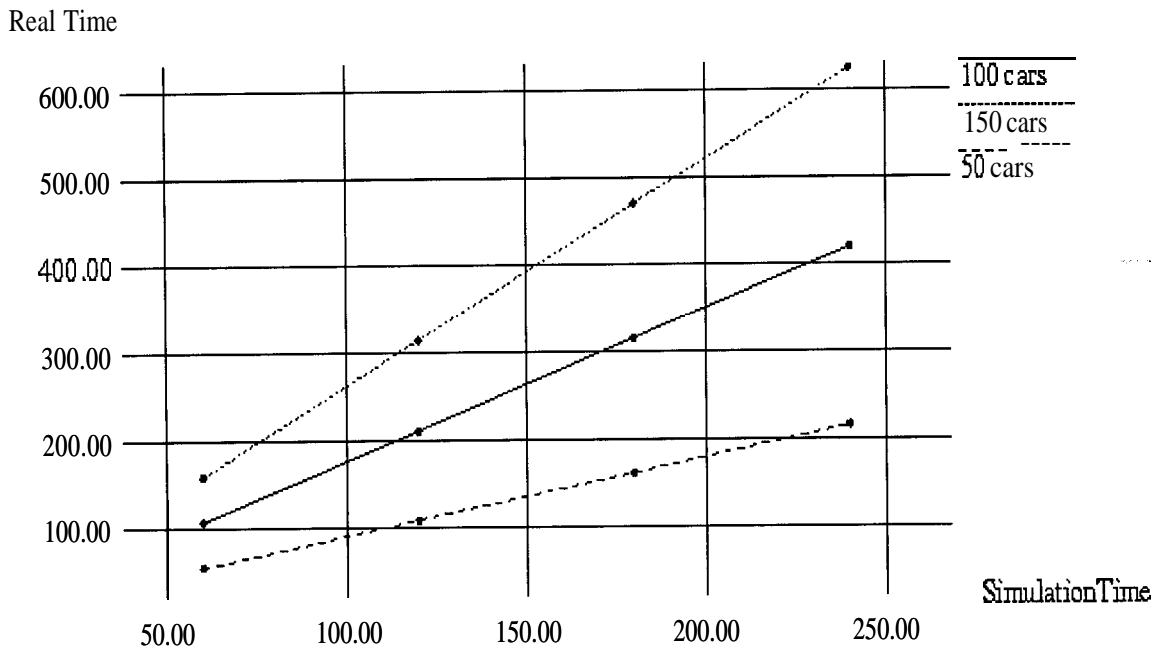


Figure 3.3: Real Processing Time as a Function of Simulation Time in Sun SPARC 5 Workstation

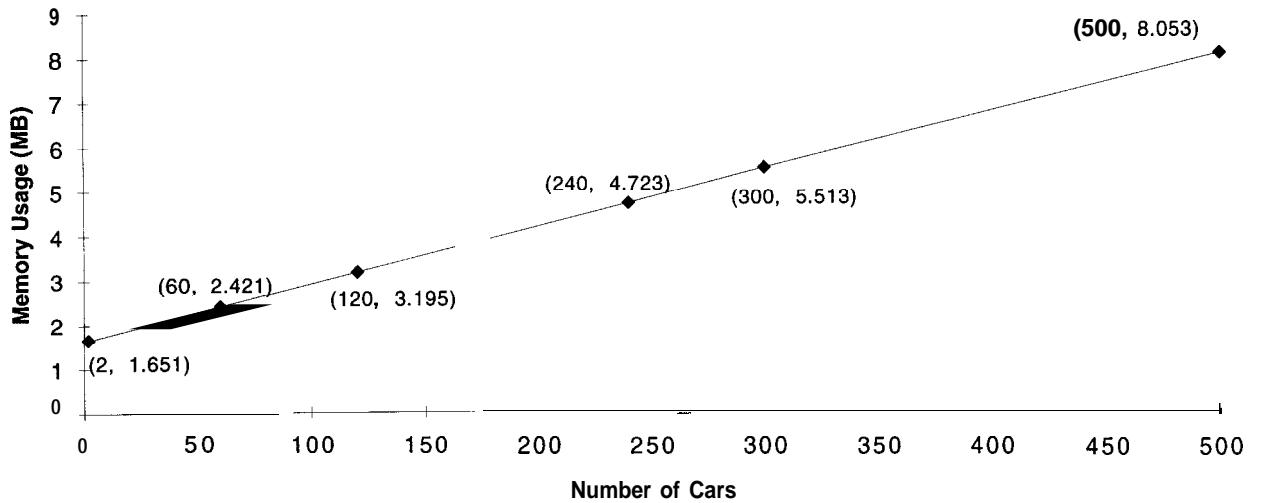


Figure 3.4: Memory Usage of SmartPath as a Function of Number of Vehicles in Simulation

Chapter 4

Parallelization and Distribution

In the previous chapter we saw that SmartPath can simulate 60 cars in real time. In a four-lane highway with an average flow of *2000 vehicles/lane/hour* and average speed of *100 kilometers/hour*, there are about *20 vehicles/kilometer/lane* and for the 4-lane highway *80 vehicles/kilometer*. If we consider the San Francisco Bay Area highway network, we have 300 kilometers of highway which amounts to a maximum of 24,000 vehicles on the highways at any time. To simulate 24,000 vehicles in SmartPath for one hour, the processor would spend 400 hours (16 days 16 hours), which is not convenient and may be impractical to use. There are two solutions to this problem. Either we aggregate and simplify the control layers and, as a result, lose the micro-simulation property of SmartPath in favor of timing efficiency, an approach followed in Netsim, Watsim, Intras and other traffic simulators, or we must allocate more processing power to the simulation.

The former approach is not followed in SmartPath, since one of its objectives is to exhibit the interaction of the controllers on the highway and in the vehicle.

More processing power can be allocated by employing a massively parallel machine (MPP) or distributing the AHS simulation over a network of workstations (NOW). Parallel machines are designed to have very low communication overhead and very high bandwidth among their processors and, therefore, are suitable for those applications that require high bandwidth and relatively large amounts of message passing. Their disadvantage is their high price and relatively slow processors compared with workstations. Good comparisons of parallel processors are found in [34] and [35]. On the other hand, workstations are widely available and have very fast processors. But the communication link between them, usually Ethernet, is very slow (the maximum bandwidth of Ethernet is 10Mbps with an effective bandwidth of about 1Mbps; the effective communication bandwidth of Thinking Machine

CM5 is 20Mbps), which makes them suitable for applications that require small amount of message passing [36]. For example, there are 90 workstations in the instruction labs in the EECS department of U.C. Berkeley, used mainly by undergraduate students for class assignments and e-mail. We can run SmartPath simulation on any one of these workstations (mostly DEC workstations with MIPS R3000 processors which are slower than the MIPS R4400 used for the performance analysis in the previous chapter) at the rate of 30 vehicles in real time, when the processor is lightly loaded.

We would like to use this pool of mostly idle processing power to simulate SmartPath. In the ideal case, we can simulate about 2700 (30 x 90) vehicles in real time, using the linear relationship between number of vehicles and the ratio of real time per simulation time of figure 3.2, which means that we can perform the one hour simulation of the 24,000 vehicles in the Bay Area in about 8.9 hours (24,000/2700), which is more convenient. However, this optimistic estimate is not achievable in a non-trivial simulation because of the overhead associated with the communication among the processors. The main objective of the distribution and parallelization of a simulation is to improve the performance by intelligent partitioning in order to reduce the amount of communication among the processors.

In this chapter we will discuss the distribution of SmartPath in a multi-processor environment which could be a MPP with distributed memory or a NOW. In section 4.1 we look at the distribution of AHS elements; in section 4.2, we develop an algorithm for synchronization of distributed simulation of multi-layered hybrid systems (with direct application to SmartPath); in section 4.3, the implementation of distributed SmartPath in a multi-platform environment (Sun, SGI, DEC, and IBM workstations) is discussed, and finally section 4.4 assesses the performance of the distributed simulation relative to the sequential simulation discussed in chapter 3.

4.1 Distributed Simulation of AHS

The metrics often used to evaluate the performance of a distributed system are speed-up, efficiency, and scaled speed-up.

Let

- C be the simulation scenario,
- $T_1(C)$ = the time needed to simulate scenario C in SmartPath when 1 processor is employed,

- $T_p(C)$ = the time needed to simulate scenario C in SmartPath with p processors,
- T_p^i = the time spent by processor \ast in a p processor environment on simulation only (not including the communication among the processors),
- $T_p(pC)$ = time needed to simulate the scenario C scaled by a factor of p ; in SmartPath the scaling of the scenario by p means increasing the number of vehicles in the simulation by a factor of p ,
- $C_p(i, j)$ = communication cost between two processors \ast and j ,
- $C_p(C)$ = the total communication cost among p processors.

We can see from the above notation that

$$T_1 = \sum_{i=1}^p T_p^i = pT_p^1,$$

where for the last equality we assumed that $T_p^i = T_p^j$ for all i and j , which is a valid assumption if all processors have the same load and processing power throughout the simulation.

Speed up S_p of the system is defined as

$$S_p(C) = \frac{T_1}{T_p(C)},$$

therefore, $0 \leq S_p \leq p$, since we can always guarantee that $T_1(C) \leq pT_p(C)$ by simulating p processors by 1 processor.

Efficiency E_p is defined as

$$E_p = \frac{S_p}{p}, E_p \leq 1.$$

Scaled speed up $R_p(C)$ of the system with p processors is defined as

$$R_p(C) = \frac{T_1(C)}{T_p(pC)},$$

therefore, $0 \leq R_p(C) \leq 1$.

T_p is essentially the time to simulate the system plus the time for the inter-processor communications. Therefore, the speed up S_p can be written as

$$S_p(C) = \frac{T_1(C)}{\frac{T_1(C)}{p} + C_p(C)}$$

and we can drop the dependence on the scenario C by using a fairly general scenario for simulation.

The cost C_p consists of two terms: the number and size of messages transmitted between the processors and the overhead associated with setting up the connection and transmission and reception of the messages.

Therefore, we have to employ a partitioning scheme that minimizes C_p ; such a scheme is the topic of this section. However, before we do that, we have to identify the entities and the interactions among them that might result in the message passing between processors.

4.1.1 Simulation Entities and Their Interactions

As described earlier in chapter 2, there are two control layers on the roadside, the network and link layer controllers, and two control layers inside each vehicle, the coordination and regulation layer controllers. The domain of operation of each layer is as follows:

- Network layer: the whole highway system.
- link layer: a section of the highway or multiple sections.
- coordination and regulation layer: the portion of the highway that the vehicle's longitudinal and lateral sensors can cover.

Now let us investigate the interactions among these controllers and other modules within SmartPath, the frequency of these interactions, and the number of controllers involved.

- The network layer transmits its routing table to each link layer controller. The routing table has size (*number-of -network-nodes* \times *number-of -AHS-exits*); each entry of the table is an array of integers, whose size can be approximated by the size of the network layers node structure. Each link layer controller in turn sends a message to the network layer; the message size depends on the controller, but the number of messages is on the order of number of sections in the highway network.

These transactions will take place periodically in the absence of emergency conditions; the period also depends on the controllers, but the network updates the routing table roughly every 15 minutes. Thus the period of these transactions is on the order of 15 simulation minutes.

- The link layer transmits its routing table to the coordination layer of the vehicles within its domain. The size of the table is (*number_of_lanes* \times *number-of -exits*) and

its entries are integers. Every vehicle also transmits the relevant state information to the link layer controller of the section it is traveling. The vehicle message contains its velocity, lane number, section number, and destination address. The size of the message depends on how the destination is coded, roughly the size of four integer values. The frequency of this message passing is a function of the time that it takes for a vehicle with the maximum allowable velocity to pass through a section, assuming that all sections have equal length. With a velocity of $25m/s$ and section length of 500 meters, the period of the message passing is 20 seconds.

- The coordination layer periodically receives sensor information which is local message passing within a car, but it also receives and transmits messages to other vehicles' coordination layers. The radius of this operation is limited by the range of longitudinal and lateral sensors; however, it is aperiodic and occurs only when the vehicle is involved in or in the process of initiating a maneuver. Communication between followers and their platoon leader can always happen, regardless of how many vehicles are in the platoon and the distance of the follower from the leader.

For every maneuver at least two messages (integer valued) pass between the coordination layer and the regulation layer, which again may be counted as local.

- The regulation layer also uses the sensors information periodically. If the vehicle is a follower, in addition to the sensor information, the regulation layer also receives periodically a message from the regulation layer of the vehicle ahead. This message contains control information such as velocity and acceleration of the transmitting vehicle.

The period of updates for sensor information and control information is presently 0.1 seconds.

Let us sum up what we have above. At every simulation time sample the vehicle and highway sensors scan the highway for vehicles and update the information in their respective controllers. Within a platoon each follower receives a message from the vehicle ahead and transmits a message to the vehicle behind, if any. Meanwhile, any vehicle may transmit a message to initiate a maneuver, if the required conditions are satisfied. The range of communications is at most equal to the detection range, unless the vehicles are part of the same platoon. Also, while the vehicle is traveling in a section, it will receive at least one message from the link layer, and it will send one message to the link layer controller of that section.

Given these requirements on the communication and sensors we have to devise a partitioning scheme which is optimum, i.e., if we export each element of the partition to a processor, we have the minimum possible communication among the processors.

4.1.2 Partitioning the AHS Simulation

The AHS simulation is a particle simulation in the sense that it simulates vehicles (particles) that are moving autonomously in a 2-dimensional (2-D) plane with strong interactions among them as listed in the previous section. There are two general partitioning method for distribution of particle simulation ([37] chapter 2):

- *particle (vehicle) partitioning*- In particle partitioning, a fixed number of vehicles is assigned to each processor. The advantages of this method are that it is perfectly load balanced, i.e, if there are 500 vehicles and 2 processors, according to this method we assign the simulation of 250 vehicles to each processor; also, there is no need to transfer the simulation of any vehicle to another processor. The disadvantage of this method is that it does not guarantee the locality of the vehicles with respect to each other, since each vehicle has its own travel plan, position, and velocity.
- *spatial (highway) partitioning*- In spatial partitioning, space is divided into sections and each section is allocated to a processor which simulates the particles in that section. In this method unlike in the previous one the coordinates of each partition with respect to the origin of the 2-dimensional plane are fixed, and interactions among processors are limited to the boundary regions; boundary regions are defined only if the interactions among the particles can be localized. In the case of AHS we can localize the interactions among the vehicles, since the interactions are limited by the detection range of the vehicle's sensors; i.e., if one vehicle can not detect another, it will not react to it; on the other hand, the controllers on the vehicle do not require the information about vehicles outside their detection range. Therefore, in order to minimize the communication among processors, it is enough to minimize the boundary regions. In the AHS simulation the minimum boundary is achieved by partitioning the highway on or parallel to the highway section divides. The disadvantage of this partitioning scheme is that there is extra overhead for transferring the simulation of vehicles from one processor to another as they move from one spatial partition to another. As a result of the vehicle transfer, the load on the processors may change from one time to another, and, therefore, the efficiency of the distributed simulation

may decrease.

In order to compare the two methods, we calculate in both cases the number of messages each processor should transmit, M^1 and M^2 , under each method. Let us assume that we want to simulate n vehicles on p processors (assume n is divisible by p), the average platoon size is S , each vehicle during its travel time in AHS performs q maneuvers, average number of messages for a maneuver is A , and we are simulating for T seconds with t as the simulation time increment.

For the *particle partitioning* method the communication costs among the processors are as follows. Each processor has to transmit the information regarding the position of each vehicle on the highway and the velocity of all vehicles in its domain to other processors; this information is used for sensors detections and we denote it by

$$M_d^1 = \frac{n}{p}.$$

Also, since each vehicle has equal probability to belong to any processor, the participants in a maneuver are likely to be in different processors; therefore, during a maneuver, there are extra message transmissions among the processors; the number of these messages is a function of the number of maneuvers in the highway and number of processors. As a result, the number of messages that are transmitted for maneuver purposes for each simulation step is

$$M_m^1 = \frac{p-1}{p} \frac{n}{p} Aq \frac{t}{T}.$$

The first term ($\frac{p-1}{p}$) is the probability that two vehicles are in different processors, and the second term is the number of vehicles in each processor.

To this number we should add the number of control messages that every follower should transmit to the car behind at every simulation time step,

$$M_c^1 = \frac{A-1}{A} \frac{n}{p}.$$

The first term ($\frac{A-1}{A}$) is the probability that a vehicle is not the tail car of a platoon, since the tail car will not transmit a message. Therefore, the total number of messages in the vehicle partitioning scheme is

$$M^1 = M_d^1 + M_c^1 + M_m^1 = \frac{n}{p} + \frac{A-1}{A} \frac{n}{p} + \frac{p-1}{p} \frac{n}{p} Aq \frac{t}{T}.$$

To calculate M^2 for the *spatial partitioning* method, note that if we substitute n with n' in M_d^1 where n' is the number of vehicles in the boundary regions of the processor,

we get M_d^2 . The number of messages required for maneuvers is equal to

$$\frac{2}{3}n'Aq\frac{t}{T}$$

where $2/3$ is the probability that the two vehicles belong to different processors, since a section can at most connect to two other sections.

To calculate the regulation layer control messages, we have to calculate the amount of time at least one member of a platoon is in a neighboring process. Let R be the flow of traffic in vehicles per second per lane and L the average length of a platoon with velocity v , then the time for a platoon to travel through the process boundary is

$$\frac{L}{v} \text{ sec.}$$

Therefore, the number of control messages per simulation step is

$$M_c^2 = \frac{L/v}{S/R} = \frac{LR}{Sv}.$$

In this method we also have to add the messages used to transfer the simulation of vehicles from one processor to another; the number of messages in the spatial partitioning case is

$$M_v^2 = tR$$

M_v^2 is calculated for a one-lane highway, and should be multiplied with the number of lanes for multi-lane highways. The total number of messages in the second case is

$$M^2 = M_d^2 + M_c^2 + M_m^2 + M_v^2 = \frac{n'}{p} + \frac{LR}{Sv} + \frac{2}{3}n'Aq\frac{t}{T} + tR$$

The communication costs for the distributed simulation of the Bay Area traffic (described at the beginning of this chapter) with 8 processors, average platoon size of 4 and platoon length of 24 meters (assuming length of the cars are 5 meters and the intraplatoon distance is 1 meter), average highway section length of 500 meters, and detection range of 60 meters are:

$$M^1 \simeq 2100 + 1575 + 1.63$$

$$M^2 \simeq 504$$

As we predicted the value of M^2 is much smaller than M^1 , even though M^2 is calculated for the worst case highway partitioning such that every section has a boundary region with another processor, which can be avoided, if the partitioning is done efficiently. The actual number of messages is on the order of 10 messages per simulation increments. If we increase the flow to 7200 vehicle/hour/lane in a three-lane highway, then we should expect that in every second 6 vehicles enter a new section; however in this case too, the main communication cost is in for the detection schemes (first term of M^2).

4.1.3 Partitioning the Highway for Minimum Communication Cost

In the previous section, we calculated the communication costs for detection, maneuvers, and control messages and did not consider the communication between link layer controllers and vehicles. However, this will not change the conclusion of the previous section that M^1 is significantly larger than M^2 , since the link layer communication can be considered local and will increase M^1 more than M^2 . Note that if the highway partitions are along the section boundaries, the value of M^2 will not increase. The reason is the locality of the link layer controller to its section.

In order to partition the highway, we will associate a directed graph $G = (N, E)$ to the highway section structure (see figure 4.1).

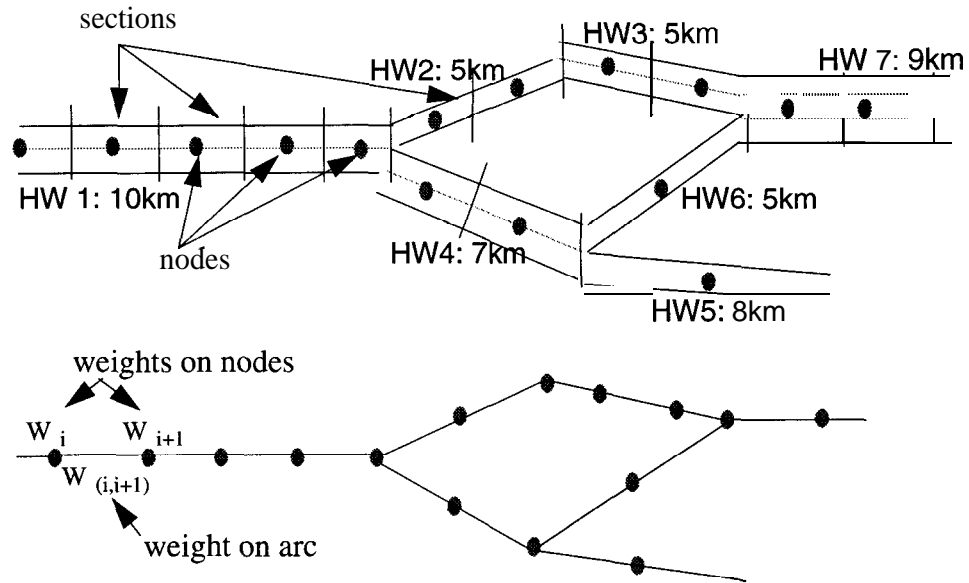


Figure 4.1: Highway and its Associated Graph

To every section of the highway a node is assigned, and there is an arc between any two nodes whenever the corresponding sections are connected through their lanes. The assignment of nodes to the sections enforces the indivisibility of the sections. The weight on node i , W_i , is the number of vehicles present in section i , since the cost of simulating section i is linear in the number of vehicles. The weight on the arc (i, j) , $W_{(i,j)}$ is the number N^2 calculated in the previous section.

From the above transformation, we see that partitioning the highway is equivalent to partitioning the graph G . If we think of a node $i \in N$ as representing a job to be performed, the weight W_i as the cost of job i , an arc $a = (i, j) \in E$ as the existence of some

data that should be transferred from job i to job j and vice versa (assuming undirected graph), and $W_{(i,j)}$ as the cost of the data transfer, then partitioning G means dividing N into the union of p disjoint set of nodes:

$$N = N^1 \cup N^2 \cup \dots \cup N^p,$$

where we assign node n to the processor i , if $n \in N^i$. This partition is subject to the following conditions:

- For all processors p and q ,

$$\sum_{j \in N^p} W_j \simeq \sum_{j \in N^q} W_j$$

which means that the load is approximately balanced across processors.

- For every processor p ,

$$\sum_{q \neq p} \left(\sum_{\substack{i \in N^p \\ j \in N^q}} W_{(i,j)} \right)$$

is minimized which means that the total cost of all messages communicated between different processors is minimized.

In this way we have transformed the question of the distribution of particles among several processors into a graph partitioning problem. Note that if all the highway sections are connected, the graph G is a connected graph. In order to facilitate graph partitioning, if the graph is not connected, we can introduce some auxiliary arcs with weight 0 such that the new graph G is connected.

Graph partitioning, in general, is an NP-complete problem, [38]; therefore, we resort to heuristics to find suboptimal solutions to our problem. There are two main approaches to graph partitioning, depending on the information available about the graph G : partitioning the graph with *coordinate information* and *coordinate-free partitioning*.

The *coordinate partitioning* scheme is useful for planar graph where we can assign (x, y) coordinates to each node. Typically, in planar graphs nodes that are spatially close together have edges connecting them. There are efficient algorithms that exploit this property, see [39]. In general, an algorithm in this family finds a line (or a plane if the graph is three dimensional) such that with respect to that line the nodes are equally divided and

assigns each set of the nodes to a processor. For more partitions, we can apply the algorithm recursively, bisecting each partition further until enough partitions are available.

The *coordinate-free* partitioning scheme assumes no identification of a node with a physical point in space, and clearly can also be applied to graphs with coordinate information. The most popular algorithms are the Kernighan/Lin algorithm [40] and recursive spectral bisection [41].

The **Kernighan/Lin algorithm** is a simple local descent algorithm which works well to improve an already available, reasonable partition. In this algorithm nodes are moved between sets in an effort to reduce the cost of the partition. There are extension of this algorithm for quadrisection [42] and partitioning to arbitrary number of sets [43].

The **Recursive Spectral Bisection** uses the second lowest eigenvector of the Laplacian matrix of the graph to divide the graph into two partitions. The second eigenvector is called the *Fiedler* vector. The (i, j) th element of the Laplacian matrix of the graph G is defined as

$$L_{i,j} = \begin{cases} -1 & \text{if } (i, j) \in E \\ d_i & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

where d_i is the degree of node i . The spectral bisection has been also extended to quadrisection and octasection, see [43].

To partition the highway graph we use an iterative method consists of three complementary iterative algorithms. The first algorithm creates a set of subdomains and attempts to improve the shape of subdomains by migrating the boundary nodes from one subdomain to another so that the radial distance from the center of the subdomain to the edges of the subdomain is minimized.

The second algorithm, devised by Song [44], performs load balance among partitions. This algorithm assumes that the workload consists of independent tasks of equal size and determines the number of nodes to migrate from a given processor to its neighbors. The nodes that are actually moved should also reduce the cut-edge weight.

The third algorithm is a localized version of the Kernighan-Lin algorithm which minimizes the communication cost. This iterative method is an order of magnitude faster than the spectral bisectioning with the equivalent quality of partitions measured by the number and weight of the cut-edges and load imbalance among the processors. A variation of the above scheme is originally proposed in [45] and is used in JOSTLE software tool.

4.2 Distributed Simulation of Hybrid System

As described in chapter 3, we can simulate a hybrid system in two phases: first simulate the continuous layer and then the discrete layer. The amount of the real time it takes to simulate each element is different from one layer to another and from one simulation time to another, and since the simulation of a large and detailed hybrid system (like AHS) is very time-consuming, we would like to distribute the execution of the layers in a multi-processor environment. We can also characterize the AHS simulation as a discrete-time (sampled-data) and discrete-event simulation. The sensors and regulation layer are discrete-time systems, and the coordination layer is a discrete-event system.

Assuming that the system is strongly coupled, i.e., there is a large penalty when the simulations in different processors are not synchronized, the distribution of the discrete-time simulation is traditionally done by using *barriers*. The barrier is a software block that disallows the progress of time, until the barrier is executed by all processors involved in the simulation. Barriers are implemented by broadcast communication messages in distributed memory systems and by flags in shared memory systems. However, given the special structure of SmartPath, the spatial partitioning method for distribution, and the locality of the discrete-time elements, we have implemented barriers by local communications among neighbors, which is much more efficient than broadcasting. We describe in the next section how we use these local barriers to synchronize the distributed partitions of SmartPath.

For the discrete-event simulation there are two main approaches to distribution: conservative and optimistic. In the conservative approach the events are timed, t_1 to t_n , where n is the total number of events generated throughout the simulation and are destined for another processor. We have $t_i \neq t_j$ whenever $i \neq j$. The conservative algorithm requires that events be transmitted in chronological order. The receiving end then waits until it receives all its input events, processes the event with the lowest time stamp, and increments its local clock to the value of the event just processed. In order to avoid deadlock, if a processor does not have an event to transmit, it sends a *null* message, which requires a *lookahead* capability for each processor, i.e., a processor should be able to look into the future for some strictly positive amount of time, and if it will not generate an event during this period, it should send a null message to all its output links. In [46] it is proved that there is no deadlock with this null message algorithm. Variations of the conservative algorithm have been developed, see [43] for more details and [47] for an excellent overview and comparisons of different conservative algorithms.

In the optimistic approach, or Time *Warp*, as it is called by Jefferson and Sowizral [48], every processor checks its input links and executes the event with the lowest time stamp. Contrary to the conservative approach it does not wait until an event has arrived at each of its input. As a consequence of this relaxation, it is possible for a processor to receive an event with a time stamp which is less than the processor's local clock value. If this happens, the processor should *roll back* its local clock, execute the event that has happened in the past, and nullify all the generated events and their effect subsequent to the new time after the roll back by sending anti-messages to other processors that received the erroneous events. The roll back capability requires storing the state of the system throughout the simulation time, which may require a large memory in the system. Other variations of Time Warp have been developed, see [49] for more details.

The conservative approach is ideal for fully connected, strongly coupled systems. It has the disadvantage that a slower or more loaded processor slows down the whole system, and so load balancing is of paramount importance in the conservative approach. On the other hand, the optimistic approach is ideal for loosely coupled systems with little communication among processors, and if the conditions are ideal, its efficiency tends to one. The drawback is that as the number of rollbacks increases, the efficiency may quickly degrade depending on the size of the system.

We argued in the previous section that the best way to partition the AHS is by allocating the simulation of the sections of the highway together with the vehicles in that section to different processors, so that each processor has a certain number of sections. This produces a connected graph, each node being a section. When a vehicle moves from one section to another, it may have to be moved from one processor to another, if the sections belong to different processors. With a large enough traffic flow (a flow of 7200 vehicle/lane/hour will cause inter-processor messages at the rate of two per second per lane) in the highway, we have a large demand on inter-processor communication. Also, for detection purposes, a message will be generated for each simulated vehicle in the boundaries of the processors at every time sample. Because of these reasons, we chose the conservative approach for the discrete event part of the simulation; however, we had to modify it in order to take into account the discrete-time part of the simulation, since we are simulating the continuous plant together with the discrete controllers in the same processor; therefore, we had to be careful about the synchronization between the discrete controllers simulated in different processors. The option of simulating the continuous plant separately from the discrete controllers is not efficient in our case because of multiple interactions that take

place between the discrete-time and discrete-event parts.

In this section, I describe an algorithm which functions as the supervisor for distributed message passing. This algorithm is needed since null messages alone will not guarantee the correctness of the simulation and lack of deadlock in a discrete-time simulation of hybrid environment. The algorithm developed in this section ensures the correctness of the simulation (which will be defined subsequently) by determining the sequences of events that should occur before the simulation clock can be incremented.

4.2.1 The Processor Model

We model each processors of our distributed system as follows, figure 4.2:

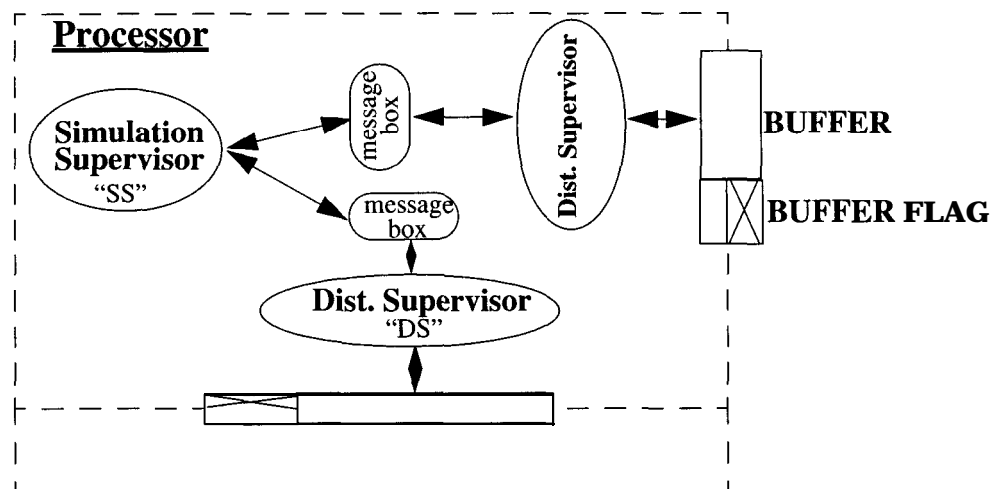


Figure 4.2: Modeling of a Processor

Within every processor there is a simulation supervisor, a set of distribution supervisors, buffers and buffer flags, and message boxes.

- The simulation supervisor (SS) consists of the main scheduler and simulates the multilayer system, the network and link layer controllers, FSMs for coordination and regulation layers, and the vehicle dynamics. Every event that is generated by a controller is marked by the simulation supervisor as an *internal* or *external* event. Internal events are those events whose recipients are also simulated within the same processor and supervised by the simulation supervisor; the recipients of external events are simulated in another processor, and the event should be sent to them.
- The message boxes (MB) are memory locations where the external events are stored until the simulation or distribution supervisor can serve them.

- Between every two neighboring processors, there is a buffer (B) which is a memory location readable and writable by both processors, and a buffer flag (BF) which models a toggle switch that points to only one of the processors at any specific time. The buffer flag is also readable and writable by both processors and initialized randomly to one of the two possible choices. By convention a processor can only access the buffer for reading and writing, if the buffer flag points to that processor. In this case the processor can also toggle the switch to the neighboring process, but before that it should have read the incoming messages from the buffer and written the outgoing messages into the buffer.
- The distribution supervisor (DS) supervises the message passing between two processors. It receives the external events from its own simulation supervisor and delivers them to the neighboring process by writing the messages into the buffer whenever it is allowed. The distribution supervisor also receives messages from the buffer and notifies the simulation supervisor of the messages. Our distribution algorithm resides in this entity.

Figure 4.3 shows the connections for a three-processor system. Note that there is one distribution supervisor for each neighbor, and therefore it is necessary for every processor to know how many neighbors it has. However, this number can change as the simulation progresses, which allows us to perform dynamic load balancing as we will see in the next chapter.

4.2.2 Simulation Supervisor (SS)

Figure 4.4 shows the sequence of events that takes place inside the simulation supervisor during one simulation time step.

The simulation supervisor has a list of all its neighbors which is initialized at the beginning of each simulation step. Since every processor has to simulate a fixed segment of the highway, the neighborhood list usually will not change, unless there is a repartitioning of the highway which may happen, if a load imbalance is detected (see chapter 5 for the definition of load imbalance). After the initialization of this list, the simulation supervisor simulates its own continuous plants in synchronization with other processors.

The basic idea behind the discrete-time simulation is the lock-step progress with the help of barriers. A *barrier* is a software block that when executed blocks that processor until all participating processors in the simulation executes the barrier. In a distributed

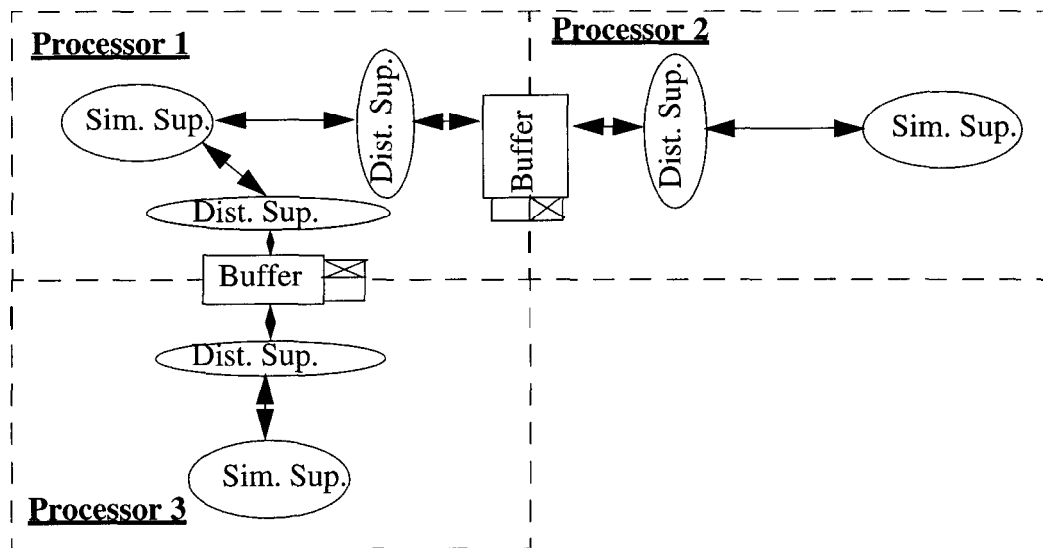


Figure 4.3: Modeling of Multiprocessor System

environment, barriers are implemented using the following two communication steps:

- When the barrier is executed, it sends a message to all processors involved in the simulation.
- Then, it waits until it receives the corresponding message from all other processors as well.

In SmartPath the relevant processors are limited to the neighboring processor, which reduces the number of messages and the resulting bottlenecks tremendously. However, there are two barriers involved in the discrete-time SmartPath simulation, one for moving the vehicles and another for detection purposes. Figure 4.5 shows the sequences of events for coherent completion of the discrete-time simulation.

The simulation supervisor first updates the position of all vehicles, and if some vehicles have passed the boundary of the processor, they are grouped according to their next host (processor). After the completion of the update, a message for each neighbor processor is created. The message contains both static and dynamic state information of the vehicles in the corresponding group. If there is no message for a neighbor, a message with no content is transmitted. A processor waits until it receives a message from each of its neighbors, and if any message has some content, the corresponding vehicle is created. For detection purposes, as was stated before, the information about the vehicles in the boundary regions should be transmitted to the processors that are connected to that boundary. As in

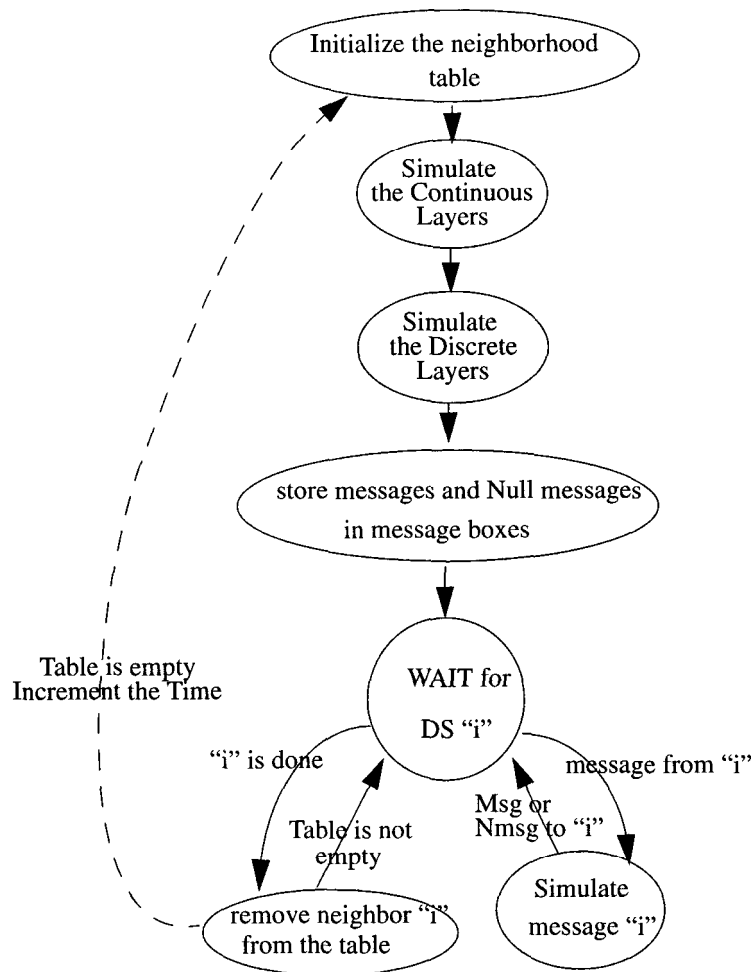


Figure 4.4: Modeling of the Simulation Supervisor

the first case, the processor groups the vehicles that are in the boundary region, and sends their position, velocity, and acceleration to the other processors, and waits until it receives messages from the neighboring processors.

Assuming that the simulation starts at time 0 on all processors, and the time is incremented after the second barrier, we can see that the time differential between any two neighbors is at most one time sample.

We return to figure 4.4. After completion of the discrete-time simulation, the simulation supervisor simulates the FSMs and other discrete controllers, which we call processes, since in SmartPath they are represented *as light weight processes*. Each controller may generate multiple internal or external messages that should be delivered to their respective processes within the simulation. All the internal events are delivered immediately, but each external event is put in the message box associated with the neighboring processor

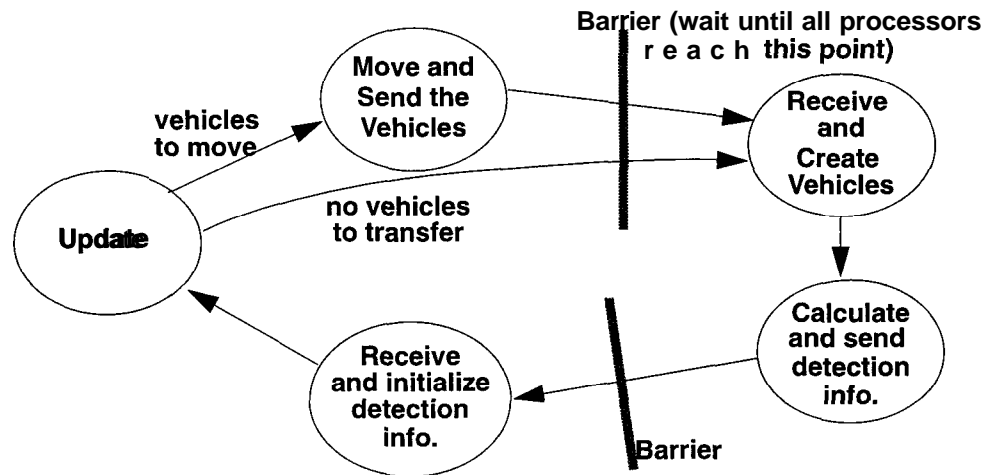


Figure 4.5: The Discrete-Time Simulation

that simulates the process to which the external event belongs. If the simulation supervisor can not find the location of the receiving process, the external event is copied into all message boxes. When the simulation of processes has ended, and none of the controllers is active, the distribution supervisor is activated; the simulation supervisor then moves to a wait state. If for some neighboring processors there is no external event, the simulation supervisor stores a null message in the corresponding message boxes.

The simulation supervisor stays in the wait state until one of the distribution supervisors notifies it of a message or generates a done event, which means that the simulation supervisor will not receive a message from that neighbor during this simulation time. In the former case, the message is delivered to the process it is intended for, if any, and after the simulation, the simulation supervisor either generates more external events for that distribution supervisor or stores a null message in the corresponding message box, effectively activating the DS. In the latter case (the generation of done event) the simulation supervisor marks that neighbor in its list of neighboring processors and checks whether all entries of the list are marked. If this is not the case, it again moves to the wait state, otherwise, the simulation of discrete-event elements of the system for that time step is completed and the simulation clock can be incremented.

In order to use automatic verification tools (COSPAN and HSIS), we had to specify how many neighbors every processor has and design the formal state machine for the simulation supervisor. The simulation supervisor's FSM for a processor with only one neighbor is shown in Figure 4.6.

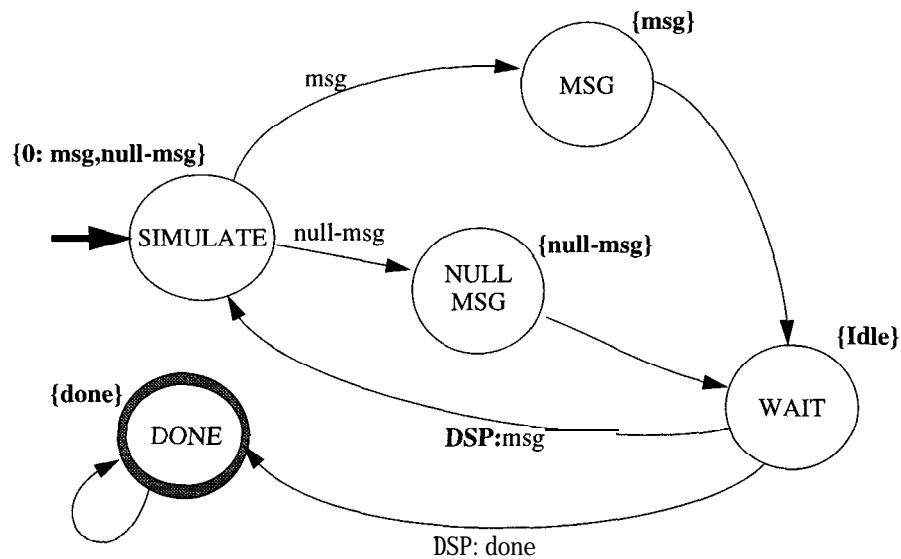


Figure 4.6: Formal State Diagram for the simulation supervisor

4.2.3 Distribution Supervisor (DS)

The distribution supervisor responds to two events, the *msg* or *null-msg* event from the simulation supervisor which indicates the generation or absence of a message, respectively, and the *my-turn* event from the buffer flag indicating the ownership of the buffer. The distribution supervisor in turn generates two events, *msg* and *done*, for the simulation supervisor and *toggle* for the buffer flag. The *msg* event indicates to the simulation supervisor the arrival of an external message that should be simulated. The simulation supervisor then checks whether the recipient of the message is simulated in its domain. For efficiency, if the process to which the message is to be delivered does not exist within the simulation domain, the message is ignored by the distribution supervisor. The *toggle* event indicates that the ownership of the buffer should be transferred to neighboring processor.

The DS will eventually move to the *DONE* state and generate the *done* event for the simulation supervisor, when the following two criteria are satisfied:

1. There is *null-msg* event from the simulation supervisor and given that the buffer flag points to its processor, there is no message in the buffer.
2. The buffer flag has already switched at least once, and the other processor had a chance to put its messages in the buffer.

The DS algorithm has four branches:

- there is msg from the simulation supervisor and no message in the buffer. In this case the DS will *toggle* the buffer flag. If the other processor already has the buffer, the DS generates the *done* event.
- there is $null-msg$ from simulation supervisor, and there is a message in the buffer. The DS will give the message to the simulation supervisor and wait for a message from it.
- there is a msg event from the simulation supervisor and no message in the buffer. The DS puts the simulation message in the buffer and generates the *toggle* event. It then waits until the buffer flag generates the *my-turn* event.
- there is a msg event from the simulation supervisor, the buffer flag points to its own processor, and there is a message in the buffer. In this case, the DS delivers the messages from the buffer to the simulation supervisor, puts the message it has received from the neighbor through the buffer, and generates the *toggle* event for the buffer flag. It then waits for either the msg or $null-msg$ from simulation supervisor.

The algorithm for the distribution supervisor is shown as formal FSMs in figures 4.7, 4.8, and 4.9, each depicting one branch starting from the beginning of the algorithm. The algorithm is not a tree (though it is drawn like one), since there are loops within each branch (shown by the dotted arrow) and connections from the states of the two right branches (figures 4.8 and 4.9) to the states of the left branches (figure 4.7, the connections are shown as labels $1A$ and $1B$). The states of the distribution supervisor are either $WAIT-SS-i$ or $WAIT-BF-i$ where i is an integer, $WAIT-SS$ indicates that the distribution supervisor is waiting for an event (either msg or $null-msg$) from the simulation supervisor, and $WAIT-BF$ indicates that it is waiting for the buffer flag to switch to its side.

The initial state of the DS is $WAIT-SS-1$. If the event $null-msg$ is generated by the SS, and the buffer is empty (B-idle), when the buffer flag generates the event *my-turn*, the DS just toggles the buffer flag and again waits for the *my-turn* event from the buffer flag after which if the buffer is still empty, the DS generates the *done* event for the simulation supervisor, since both criteria mentioned earlier are satisfied, and moves to the $DONE$ state. The second branch of the algorithm (in figure 4.7) starts from the $WAIT-BF-1$ or $WAIT-BF-2$ state on the condition that there is a message in the buffer. The message is delivered to the SS (by generating the $smsg$ event) and the DS waits for a response from the simulation supervisor. If the SS generates a message, the distribution supervisor generates the $B-msg$ event (a message for the buffer) and moves back to the $WAIT-BF-2$. In this

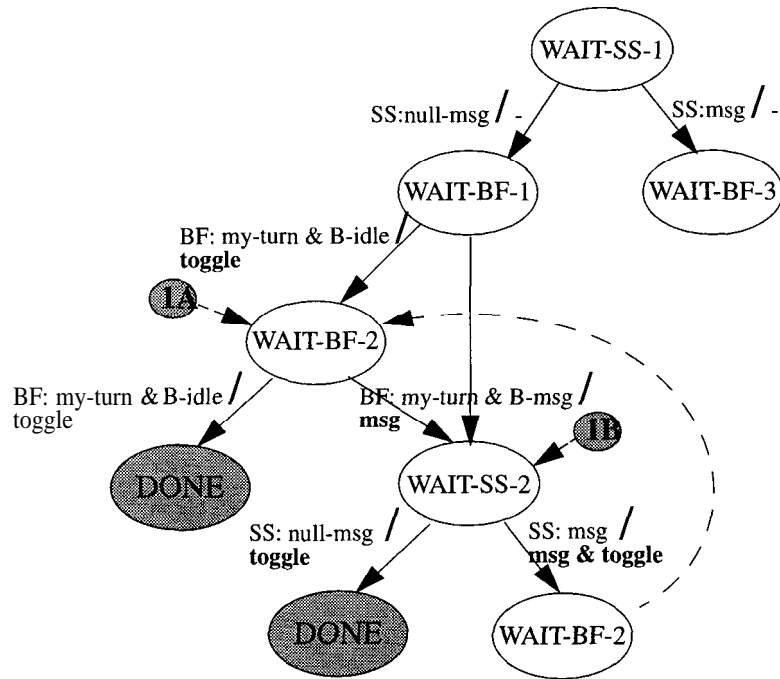


Figure 4.7: Formal State Diagram for the distribution supervisor- 1st and 2nd branches

branch, the sending and receiving of the messages between the two neighbors are essentially sequential, i.e., one sends a message and waits for the other to respond, while the other receives the message and waits for the simulation supervisor to simulate and generate a message or a null message.

The third branch of the algorithm, shown in figure 4.8, is created when at the beginning of the distribution the SS has generated a message and the buffer is empty. The DS then moves to the *WAIT-BF-4* after putting the simulation message in the buffer and toggling the buffer flag. If after the return of the buffer flag the buffer is still empty, the DS is done, otherwise the DS delivers the message in the buffer and moves to the *WAIT-SS-3*. The DS jumps to the first branch (*WAIT-BF-2* or *1A* label) if the SS has not generated any message.

The fourth branch of the algorithm (figure 4.9) is the most involved, since there is a message from the simulation supervisor and a message from the neighbor in the buffer. For efficiency, we deliver both messages and toggle the buffer flag; therefore, we allow both processors to continue simulation. For this reason, when the simulation supervisor does not generate a message, *null-msg*, the DS still has to wait for the buffer flag and wait for any messages that may be generated from its earlier SS message by neighboring processor.

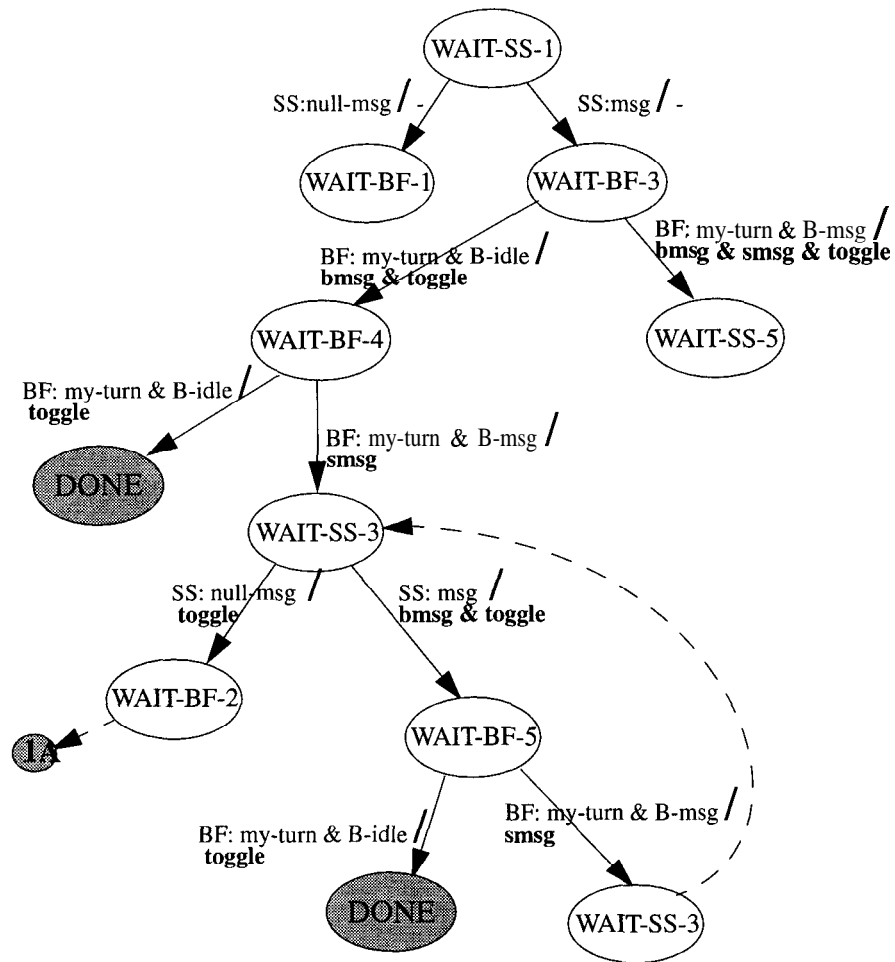


Figure 4.8: Formal State Diagram for the distribution supervisor- 3rd branch

In the next section, I discuss the criteria for the correctness of the algorithm.

4.2.4 Assumptions and Correctness of the DS Algorithm

We want to show that the behavior of the distributed system is the same as in the single processor simulation. So in order to prove the correctness of the distribution we have to show that:

- A. The DS will eventually terminate, i.e., it will go to its *DONE* state.
- B. The DS will not generate any message (*B-msg* in the DS FSM) to the buffer while the distribution supervisor of a neighbor is in the *DONE* state.

As we note from the DS state diagram, it is possible that the DS remains in the *WAIT-SS-i* or *WAIT-BF-i* indefinitely, or makes infinite transitions from *WAIT-SS-i* to *WAIT-BF-i*,

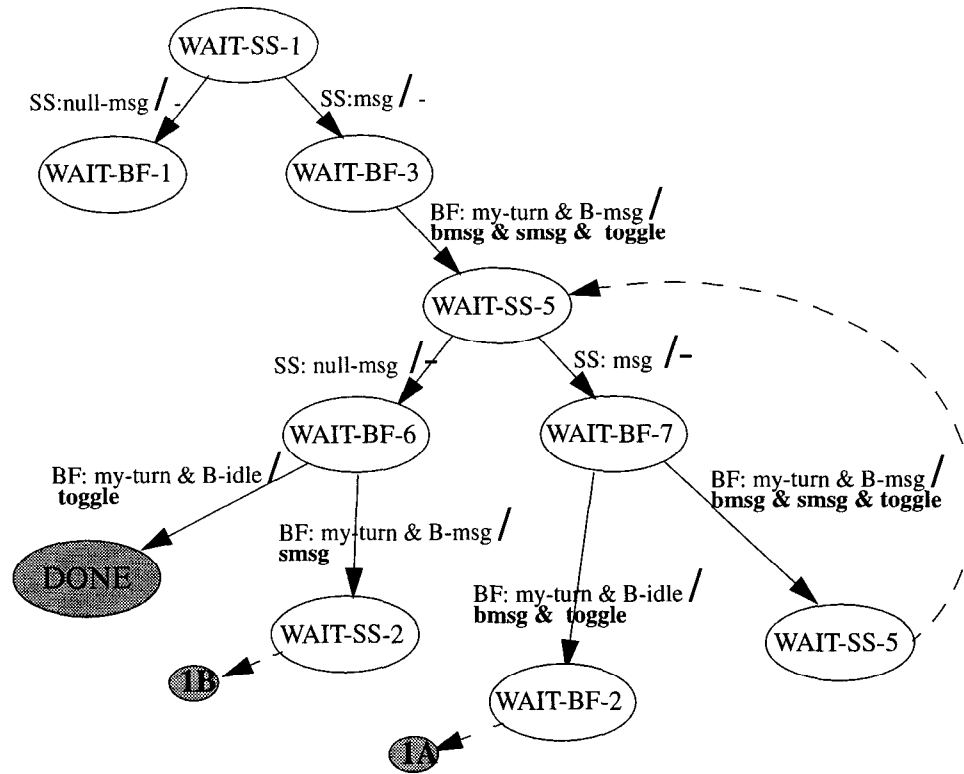


Figure 4.9: Formal State Diagram for the distribution supervisor- 4th branch

and vice versa. So in order to prove the eventual termination of the algorithm, we make four assumptions:

1. Each processor has a finite pre-defined number of neighbors.
2. The simulation phase of the simulation supervisor always terminates in finite time.
3. There will not be an infinite message passing loop between the neighboring processes.
4. A message from a neighbor will not generate another message to another neighboring processor other than the original generator of the message. However, a message may generate some internal messages. We call this assumption the *independence assumption*.

Under these assumptions, we are able to conclude that all simulation supervisors receive the correct set of messages for every simulation time step without deadlock. The first assumption is clearly justified. We can justify the second and the third assumption on the ground that if they do not hold, the uni-processor simulation also will not proceed to

the next time sample, so the behavior of the distributed system will be the same as the uni-processor simulation.

The last assumption may not be true for specific applications and has to be checked separately. In the AHS simulation, if the length of the section is large enough then the vehicles in the different boundaries are independent and will not interact; therefore, under these conditions this assumption holds. We will show later, how to augment the algorithm, when the fourth assumption can not be validated.

Proposition 4.1 *Under assumptions 2 and 3 above, the DS eventually will terminate.*

Proof: If the DS does not go to the *DONE* state, either it is indefinitely in a wait state, or it makes infinitely many transitions within a loop.

1. There are two general wait states: wait for the simulation supervisor, *WAIT-SS-i*, and wait for the buffer flag, *WAIT-BF-i*.

The DS cannot be in the first wait state because of assumption 2.

The DS cannot be in the second wait state indefinitely for the following reason. If the DS is waiting for the buffer flag, it means that the DS in the neighboring processor has the buffer flag. The neighboring DS cannot be in the *DONE* state, since before going to the *DONE* state, the DS switches the flag, so it should itself be in a wait state. The neighboring DS cannot be in the wait state for the buffer flag, since the buffer flag has only two states and always points to one of the two processors. Therefore, the neighboring DS should be in a wait state for the simulation supervisor, and because of the second assumption, it cannot remain in this state indefinitely and eventually will return the buffer flag to the neighboring DS.

2. There are three possible infinite loops: *WAIT-BF-2-WAIT-SS-2* (in figure 4.7), *WAIT-SS-3- WAIT-BF-5* (in figure 4.8), and *WAIT-SS-5— WAIT-BF-7* (in figure 4.9). Each is possible only if infinitely many messages passes between the two neighboring processors, which violates the third assumption.

Therefore, we conclude that the DS will eventually make a transition to the *DONE* state.

□

Proposition 4.2 *Under assumptions 2, 3, and 4 above, the DS will not send a message to the neighbor DS when the neighbor DS is in the *DONE* state.*

In other words, whenever a distribution supervisor makes a transition to the *DONE* state, the neighboring DS either has already moved to its *DONE* state, or it will move in the next transition. By proving this proposition, we can verify that there will not be any message from one simulation time step to the next time, i.e., the simulation correctly simulates the behavior of the system in the distributed environment.

Proof: The DS will go to the *DONE* state only after it passes one of the following states which we call primary *states*: *WAIT-BF-2*, *WAIT-SS-2*, *WAIT-BF-4*, and *WAIT-SS-5*, which means that the other DS is also past the primary states, since the buffer flag has been toggled. After the primary states, a DS moves to the *DONE* state only after a *null-msg* and a *B-idle*. Since the above events are symmetric, i.e., the *null-msg* in one DS is equivalent to the *B-idle* in the neighbor, and the fact that the DS returns the buffer flag before the transition to the *DONE* state, we can conclude that as soon as one DS moves to the *DONE* state, the neighbor DS will follow. In other words, if there was a message generated after the primary states, it should have been as a result of an earlier *B-msg* event, which could not be generated. \square

Now that we shown the DS will terminate and the consistency of the transactions between any two DSs, we can state our main theorem.

Theorem 4.1 *In the finite distributed environment under assumptions 1-4, all processors receive the correct set of events and eventually move to the DONE state.*

Proof: By construction, between any two processors there is a buffer, a buffer flag, and every processor has a specific DS for each of its neighbors. From proposition 4.1 every DS will eventually terminate, and so in a finite time all the DSs of a processor will terminate. Therefore, each processor will eventually move to the *DONE* state.

From proposition 4.2 when a DS is terminates it will not receive a message, and from assumption 4 (the independence assumption), the simulation supervisor will not generate a message for a terminated DS as a by-product of a message received from another DS. Therefore, the simulation is correct. \square

We programmed the above model in COSPAN [13] and verified the DS algorithm for two-processor and three-processor systems. The “monitor” chosen for the verification was to check that all the simulation supervisors are in the *DONE* state, and the message boxes and buffers are in the *IDLE* state. The DS chosen for the verification was designed to

make a transition from its *DONE* state to the *WAIT-SS-1* again, so it can make a transition the next sample time, too.

4.2.5 Extension to the DS Algorithm

We have described in the previous section the algorithm that can be used to monitor the distributed simulation of a hybrid system. This algorithm should be employed on top of the conservative approach to the distribution, which uses null messages to avoid deadlocks. In our case, the null messages generated by the simulation supervisor are used to indicate to the distribution supervisor that the simulation has already been performed and it did not generate any external events. We also proved that the algorithm will terminate regardless of the number of processors engaged in the simulation, as long as the four assumptions mentioned before are satisfied. However, the independence assumption is not well-justified and can be violated in some applications. The independence assumption basically decouples the boundaries of a processor; this allows for the simulation supervisor to increment a counter for the “done” distribution supervisors and as soon as the counter is equal to the number of neighbors the processor has, it moves to its *DONE* state. Let us look at an example that violates the independence assumption.

In the PATH-AHS design, the maximum number of vehicles in the platoon is 20 and with 5 meters as average length of a vehicle and 2 meters intra-platoon distance, the length of a platoon can be as large as 140 meters. Assuming that a longitudinal maneuver, i.e., a maneuver whose participants are all in the same lane, involves only 2 platoons, and the sensor range of a vehicle is 60 meters, then in order to have the 2 platoons in at most 2 processors, the minimum length of a section can not be less than 300 meters ($2 \times 140 + 60$), otherwise there is a possibility that a message generated from the tail follower of a platoon in a processor generates a message to the leader of another platoon in another processor which is not its direct neighbor.

The easiest solution is to use a global barrier as the synchronization point among all the processors. The simulation supervisor after all of its distribution supervisors has moved to the *DONE* state, moves to an intermediate state, waiting for other processors to reach that state, after which it moves to its *DONE* state. But global synchronization is very expensive in terms of communication costs, since every processor has to broadcast its state to other processors. Another solution is to use a variation of the optimistic approach in the simulation supervisor to augment the DS algorithm. This approach has merit, if the probability of an external events to interfere with processors other than the processor

for which it is generated, is small. This solution will require more memory to save the simulation state for possible roll backs.

If the highway graph is connected, the processor graph is also connected, since it is only an aggregation of the highway graph. Let us define the *distance* between any two processors as the minimum number of processors that are between the two over all paths that connect the two. Also, define the *graph diameter* as the maximum distance that exists on the graph. The significance of the graph diameter is that it limits the number of simulation steps that should be saved by every simulation supervisor. I remind the reader that the processors are pairwise synchronized for the purpose of moving the vehicles and vehicle detection (the continuous elements of the the hybrid system). Therefore, it is not possible for any processor to be ahead of the neighbor by more than one simulation step time. As a result, if one processor is blocked, its neighbors will be blocked at most in the next clock cycle and thus, the simulation clock value of no processor can be more than the graph diameter away from the blocked processors. So no simulation supervisor can receive a message with time stamp less than $simulationTime - graphDiameter$, and there is no need to save the state of the simulation for more than the graph diameter time.

The algorithm for the simulation supervisor, then, will be augmented as follows:

1. To every external event append the simulation time.
2. After a DS is in *DONE* state, and an external event for that DS is generated, activate the DS again.
3. Save the state of the simulation for the amount of time equal to the graph diameter.
4. If an incoming message has a time stamp earlier than the simulation clock, restore the saved simulation state, activate the distribution supervisor for that boundary, and deliver the message to the entity.

4.3 Implementation of the Distributed SmartPath

Our goal in implementing the Distributed SmartPath is to provide a consistent interface for interconnecting different platforms. The Distributed SmartPath can be executed in IBM RS6000, SUN SPARC, DEC, and SGI workstations. The distributed platform can also incorporate a mixed set of workstations with the exception that the animation should run on the SGI workstations. For inter-processor communication, we use standard TCP

sockets. An earlier version of the distributed SmartPath was ported to a 64-node Thinking Machine CM-5 to acquire some estimates of the speed up (discussed in the next section). The preliminary results was on the order of $0.9p$ speed up where p is the number of nodes in the simulation. However, since we used split-C language (instead of C) for message passings among processors, that version was not portable to a workstation.

The workstation in which the user executes the SmartPath program is the main server for the distributed simulation. The server initiates the SmartPath program in the user-determined clients. Every client is given the list of all clients, so that they can create connections to others. The end result is that every processor is connected to all others. If the user does not specify any client, the server initiates a client process in the same workstation. If the simulation is specified to be interactive, the server waits until it receives a message from the animation platforms. The address of the animator process is then sent from the server to all clients, so that they can establish connections with the animator.

To implement the discrete-time and discrete-event distributed simulation, we added a distribution layer and changed the scheduler of SmartPath. Recall that SmartPath's scheduler after initialization of all layers has the following loop: the regulation layer wake-up, update the position of all vehicles, wake-up for the coordination, link, and network layer controllers, and increment the simulation clock by the simulation time step. We modified the above scheduler as I explain next. Note that in the description that follows the "process" is the SmartPath simulation which is executed as a Unix process in the workstation.

- The initialization of the distribution layer is added to the initialization routines of the scheduler.

If the process is the server process, it creates the highway graph by reading the input file provided by the user, determines the number of clients, assigns a consecutive number to each client (server is processor number 0), partitions the graph so that number of partitions is equal to the number of clients and all partitions have approximately equal weight, and transmits the relevant information, which I will shortly describe, regarding the partitions to the clients. After transmission of the partitions, the server waits until it receives an *all-done* message from every client and then terminates.

If the process is a client process, it receives the communication address of the server, its processor number, and the user-provided input file as part of the initialization argument. The client connects to the server after reading and initializing the highway graph structures from the input file. The client receives the list of all other clients,

creates a connection to every client, and waits until it receives the partition allocated to it by the server. Every partition consists of a set of nodes and edges. Every node contains the id of a highway section (as described in 4.1.3), the processor number to which it belongs, and a list of vehicles which is initialized and updated by the client process.

Since every edge structure contains the source node (the upstream node) and the destination node (the downstream node) of the edge, the client sets up its table of the neighboring processes, searching the edge structures and if either the source or destination node does not belong to it, it inserts the processor number of the node in its table and marks the node as a boundary node. The client process also creates a memory location, a *message box*, for every neighbor that it uses for storing the messages and information regarding the vehicles that should be transferred to that neighbor.

- The regulation layer wake-up routine is executed as in the sequential simulation; however, since it is possible that part of the platoon is in another client, the regulation layer after executing the controller for those platoons that are wholly within its domain, waits for controller messages to arrive from other clients. As soon as the messages have arrived, the followers are activated and the regulation layer returns the control to the scheduler.
- To update the position of the vehicles, the client process traverses the list of vehicles in its nodes, and calculates the next position of the vehicles. If some vehicles are moving out of the section to another section, they are placed in the vehicle list of the downstream node, if the downstream node belongs to the same client; otherwise, the moving vehicles are placed in the message box of the corresponding neighbor. After all vehicles are updated, the client creates a message for every neighbor from its message box, and terminates the simulation of the transferring vehicles. If the message box of a neighbor is empty, a *NULL* message is sent to that neighbor. The client then waits until it receives a message from each of its neighbor and creates the vehicles that are moved to its processor, if any. If the vehicles that are transferred to another client have followers, the simulation sets a flag for the regulation layer and waits for the regulation layer control messages (infra-red messages) for the followers from the neighboring clients.

If the simulation is in the interactive mode, the updated information of the vehicles

are sent to the animation host, which in turn transmit an acknowledgement after reception of the data message. The animation program may also transmit a set of commands-store and restore-for the whole simulation and acceleration and steering angle for a particular vehicle in the manual mode. See section 3.2.2 for store and restore commands, and 3.2.3 for the description of the manual mode.

After receiving the transferred vehicles and the animation acknowledgement, if applicable, the client process stores the sensor information of the vehicles which are in the boundary zone (sensor range) of the boundary nodes in the message boxes of the neighbors. The client process transmits this information to the neighbors, and waits for the sensor messages from the neighbors. At this point the discrete-time part of the simulation has been performed.

- After the execution of the wake-up routines for the control layers, the scheduler activates the distribution supervisor. The distribution supervisor is implemented according to the algorithm described in section 4.2.3.

In the implementation of the distribution supervisor, we assume that the section length is large enough so that the vehicles in different boundaries remain independent of each other, and we have not yet implemented the augmented version of the distribution supervisor.

4.4 Performance of the Distributed SmartPath

We saw earlier in section 3.3 that the performance of SmartPath simulation is a linear function of the number of vehicles being simulated. Our goal in the distribution was to preserve the linearity of the simulation performance with respect to the number of vehicles and have the speed up close to the ideal case by reducing the communication cost among the processors.

Experiments were performed on up to four SGI INDIGO II workstations with MIPS 4400 processors (200MHz) and 32 MB of memory. The graphs of figure 4.11 and 4.12 are the results of a set of experiments with the SmartPath3.0. The highway for the experiments, shown in figure 4.10, is a double “race track” consisting of twelve 300-meters sections and four 1000-meters sections. All sections have three lanes except the middle sections which has six lanes.

For the speed up graph, the highway is populated with 1260 vehicles. To simulate this many vehicle in one processor, it takes about 20 times the simulation time (i.e., for 1

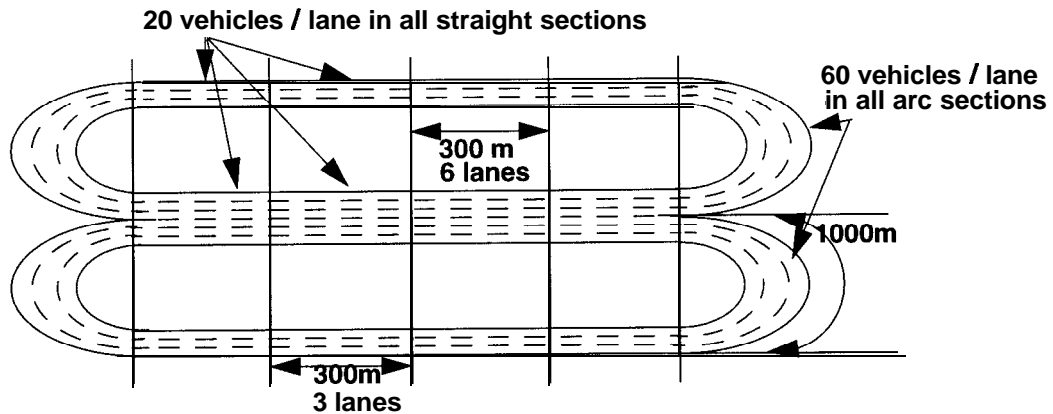


Figure 4.10: The Test Highway for Load Balancing

second of the simulation time, 20 seconds of the real time is needed). In a two processors system we could achieve the speed up of 1.7 on average (i.e. for 1 seconds of simulation time, 12 seconds of real time is needed), and in a four processor system, we have the speed up of 3.4. Figure 4.11 shows the speed up of the distributed simulation as a function of the number of processors.

$T1/Tp$

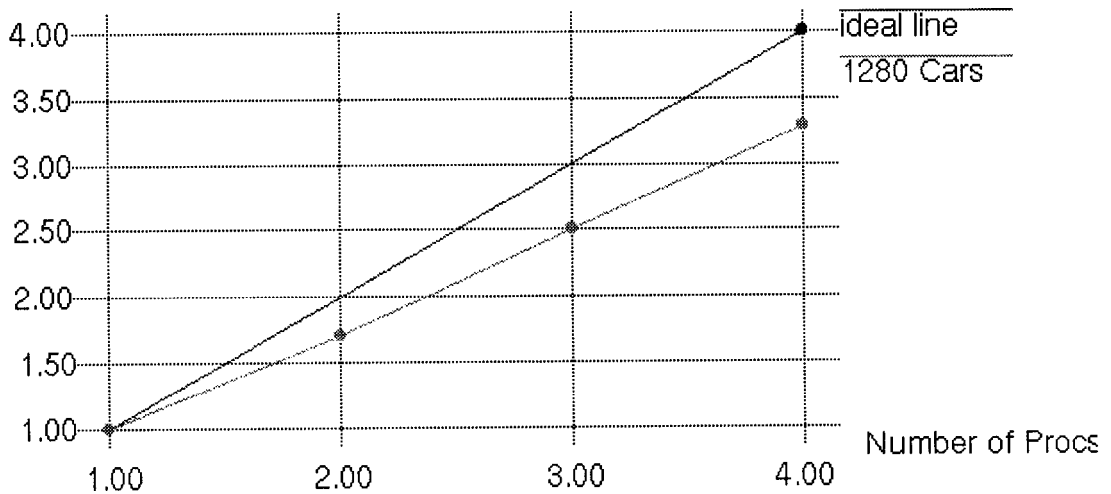


Figure 4.11: Speed up of the Distributed Simulation

Our experiments also showed us that to simulate few vehicles (on the order of 100), it is much faster to simulate the system on a single processor. The reason is that we are using TCP-IP protocol stack for the inter-processor communication, and there is a fixed overhead cost associated with sending a message. When few vehicles are simulated,

the communication cost is dominant and diminishes the benefits of the distribution.

The next performance measure is the scaled speed up, which is shown in figure 4.12. For this graph we used the same highway as above, but we increased the number of the vehicles as we increased the number of processors. For this experiment we added 270 vehicles to the system for every processor. For one processor to simulate 270 vehicle corresponds to about 4.5 times the simulation time. The ideal situation is that it takes the same amount of time as we increase the number of processors and vehicles, which translates to the scaled speed up of 1. However, we cannot achieve this scaled speed up because of the inter-processor communication. As shown in figure 4.12, for two processor the value of scaled speed up is 0.84, for three processors it is 0.75 and for four processors it is 0.72. The value of the scaled speed up drops noticeably from two processors to three processors, since in a two-processor system a processor can have at most one neighbor, but in a three-processor system the number of neighbors can be two, as was actually the case in the simulation. The scaled speed up did not decrease more than 0.72.

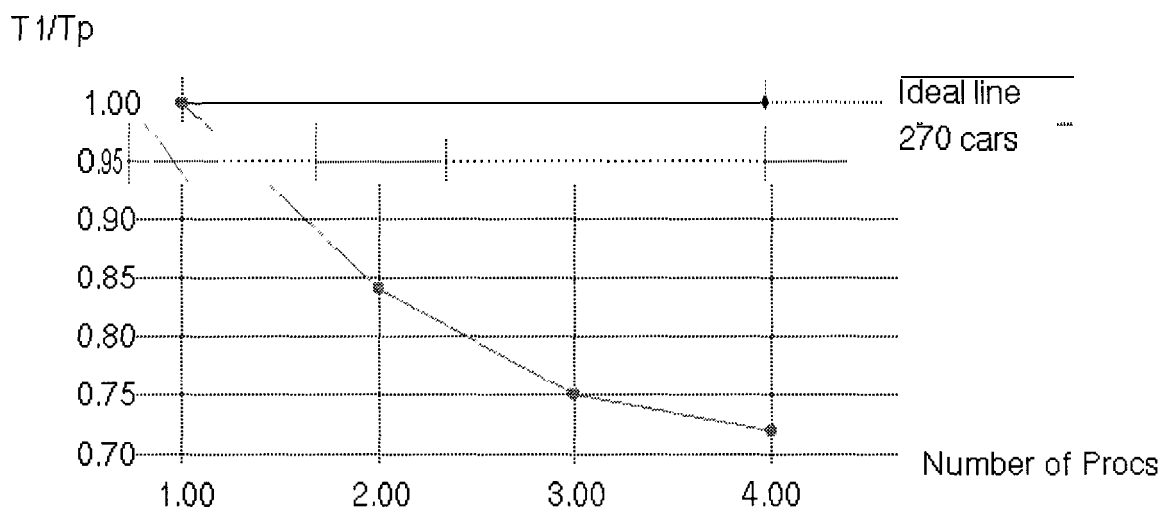


Figure 4.12: Scaled Speed up of the Distributed Simulation

Chapter 5

Dynamic Load Balancing

5.1 Introduction

The disadvantage of the spatial partitioning of the highway graph for the purpose of distribution is the possibility of load imbalance among the processors. For a P processor system, let $W_i(t)$ be the work that a processor is performing at simulation time t in terms of real time, and $W(t) = \frac{\sum_{i=1}^P W_i(t)}{P}$ be the average amount of work, then we define the load imbalance AL as

$$\Delta L(t) = \frac{\max_i(W_i(t)) - W(t)}{W(t)}$$

The load imbalance is the result of unavoidable non-uniform movement of the vehicles on the highway, even if the initial distribution of the vehicles on the highway is uniform, i.e., all nodes and edges have the same weights at the simulation time 0. Intuitively, if the flow of traffic is regular, and the initial distribution of the traffic is uniform among the lanes and sections, then it will remain so for the duration of the simulation. However, any disturbances on the highway flow, which may have been caused by vehicles changing lane will unavoidably cause imbalance among the processors. The reason is that the change lane maneuver is accompanied by degradation of the speed in the target lane of the vehicle. Minor and major incidents that may cause the reduction of the velocity of the vehicles in a lane or a section, also contribute to the load imbalance among the processors.

We devised the following experiments with the distributed version of SmartPath with fully implemented PATH-AHS control hierarchy (second constructs) in order to observe the degree of the load imbalance that the change lane maneuver and minor incidents may cause.

The highway is a double “race track” (shown earlier in figure 4.10). The track is

| | | | | | | | | | | | | | | |
|---------------------|---|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Load Imbalance (%) | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | 65 | 70 |
| Simulation Time (%) | 7 | 9 | 8 | 10 | 13 | 7 | 10 | 12 | 7 | 9 | 5 | 3 | 1 | 0 |

Table 5.1: Average Load Imbalance

populated by vehicles which are grouped in platoons of average size of 5 *vehicles/platoons*. The total number of vehicles is 1680 vehicles, uniformly distributed on the highway. The initial velocity of all vehicles is 25 *meters/sec*; therefore, the flow of traffic is 5724 *vehicles/lane/hour*. The platoons can merge, and the vehicles within the platoons are randomly selected for change lane maneuvers.

Table 5.1 shows the result of the simulation. The top row of the table is the percentage of the load imbalance ($\Delta L \times 100$) and the bottom row shows the percentage of the simulation time with that imbalance. For example for 13 percent of the simulation time, the system was experiencing 25 percent imbalance. It is clear from the result 5.1 that we need to investigate the dynamic load balancing option for the simulation.

In this chapter I study the dynamic load balancing problem using dynamic programming and show that under suitable conditions the optimal behavior is of a threshold type, that is if the load imbalance exceeds a certain threshold, it is beneficial to perform a load balancing operation on the processors. In section 5.2 we present our formulation of the problem and the conditions for a closed form solution. In section 5.3 we discuss implementation issues and the cost of the dynamic load balancing. Section 5.4 evaluates the performance of the simulation, when the load balancing option is active.

5.2 Problem Formulation and Structure of Optimal Policy

The SmartPath distributed simulation a process starts at time 0 and evolves in discrete time increments. At every step the simulation moves from one state to another, and at every state of the simulation we should decide whether to perform the load balancing operation or to let the simulation progresses to the next time step unchanged. Clearly the decision is based on the cost of the load balancing operation and the state of the simulation. Our goal in this section is to design a policy, that minimizes the real time spent by the distributed system in simulating an AHS scenario. To design the policy we shall need some preliminaries about dynamic programming which I present next.

5.2.1 Dynamic Programming

Dynamic programming is a recursive technique used to calculate optimal sequential decisions sequentially. Every decision has a cost associated with it, and since it may change the state of the system, it may affect the subsequent decisions. Therefore, the objective of dynamic programming is to properly balance the decisions that incur small short term cost with the possibly high cost in the long run. If the decision is only based on the current state of the simulation, then the process is a Markov decision process.

Let

- X_i be the state of the system at time i with X_0 the initial state,
- $\pi = \{a_1, a_2, \dots\}$ be the policy and a_i be action chosen by the policy π at time i ; a policy is called a stationary policy, if the action chosen at time i is non-randomized and only depends on the state of the system,
- $C(X_i, a_i)$ be the cost of simulating the system when it is in state X_i and the action a_i is chosen,
- $V_\pi(i)$ be the expected discounted cost of simulation when the initial state of the system is state i and policy π is chosen, i.e.,

$$V_\pi(i) = E_\pi \left[\sum_{n=0}^{\infty} C(X_n, a_n) \alpha^n \mid X_0 = i \right] \quad (5.1)$$

and

$$V_\alpha(i) = \min_{\pi} V_\pi(i) \quad (5.2)$$

- $\phi_\pi(i)$ be the expected average cost of the simulation when the initial state of the system is state i and policy π is chosen, i.e.,

$$\phi_\pi(i) = \lim_{N \rightarrow \infty} \frac{1}{N} E_\pi \left[\sum_{n=0}^{N-1} C(X_n, a_n) \mid X_0 = i \right] \quad (5.3)$$

Our goal is to find the policy π^* such that

$$\phi(i) \stackrel{\text{def}}{=} \phi_{\pi^*}(i) = \min_{\pi} \phi_\pi(i) \quad (5.4)$$

From the *principle of optimality*, which says that *an optimal policy has the property that whatever the initial state and initial decisions are, the remaining decisions must*

constitute an optimal policy with regard to the state resulting from the first decision [50], equation (5.2) can be written as:

$$V_\alpha(i) = \min_a (C(i, a) + \alpha \sum_{j=1}^{\infty} P_{ij} V_\alpha(j)) \quad (5.5)$$

where $P_{ij} = P(X_k = j | X_{k-1} = i)$ is the probability of moving from state i to state j .

Let

$$h_\alpha(i) = V_\alpha(i) - V_\alpha(0)$$

for some fixed state which we call state 0; then, we can write equation (5.5) as:

$$(1 - \alpha)V_\alpha(0) + h_\alpha(i) = \min_a (C(i, a) + \alpha \sum_{j=1}^{\infty} P_{ij} h(j)) \quad (5.6)$$

Now, if for some sequence $\alpha_n \rightarrow 1$,

$$h_{\alpha_n}(i) \rightarrow h(i), \quad (5.7)$$

and

$$(1 - \alpha)V_\alpha(0) \rightarrow g \quad (5.8)$$

for some constant g , then we have

$$g + h(i) = \min_a (C(i, a) + \sum_{j=1}^{\infty} P_{ij} h(j)) \quad (5.9)$$

the function h inherits the characteristics of V_α in the sense that, if V_α is convex and increasing (or non-decreasing), h is convex and increasing (or non-decreasing). It can also be shown that if (5.9) has a solution (which is based on the existence of the limit functions (5.7) and (5.8)), then $g = \phi(i)$, and the stationary policy that minimizes (5.9), also minimizes (5.4).

For conditions (5.7) and (5.8) to hold, it is sufficient to show that for some fixed state (say state 0), there exists a constant $N < \infty$ such that

$$|V_\alpha(i) - V_\alpha(0)| < N$$

for all α and i .

The results that I described above are the tools necessary to formulate the dynamic programming problem for SmartPath. There is a vast literature dealing with the theory and application of dynamic programming. Bellman's book [51] is an excellent reading. For applications of dynamic programming see [50]. Most of the above results are from [52] and [53].

5.2.2 Dynamic Load Balancing: Problem Formulation

For the distributed SmartPath simulation let X_i^j , the state of a processor j at time i , be the number of vehicles simulated in that processor, and let the state of the system be

$$X_i = \|X_i^j\|_\infty = \max_j X_i^j$$

Assuming that the vehicles are uniformly distributed along the lanes and along the sections, the cost of simulation is directly proportional to X_i . Also, the decision at every time step is either to perform the load balancing operation or do nothing. After load balancing is performed the state of the simulation is always state 0, which is the state with no load imbalance among the processors, i.e., each processor has $\frac{\sum_{j=1}^P X_j}{P}$. Let

$$a_i = \begin{cases} 0 & \text{if no load-balance} \\ 1 & \text{if load-balance} \end{cases}$$

which gives the one-step cost of the simulation to be

$$C(X_i = j, a_i) = \begin{cases} K + C(0) & \text{if } a_i = 1 \\ C(j) & \text{if } a_i = 0 \end{cases}$$

where K is the cost of load balancing, $C(0)$ is the cost of simulation when every processor has an average number of vehicles, and $C(j)$ is the cost in state j . From equation (5.2), we can write

$$V_\alpha(i) = \min_a (C(i, a) + \alpha \sum_j P_{ij} V_\alpha(j)) = \min(K + C(0) + \alpha \sum_j P_{0j} V_\alpha(j), C(i) + \alpha \sum_j P_{ij} V_\alpha(j)) \quad (5.10)$$

We can see that

$$V_\alpha(i) \leq K + C(0) + \alpha \sum_j P_{0j} V_\alpha(j) \leq K + V_\alpha(0)$$

The last inequality follows from the fact that equation (5.10) for $i = 0$ is

$$V_\alpha(0) = \min(K + C(0) + \alpha \sum_j P_{0j} V_\alpha(j), C(0) + \alpha \sum_j P_{0j} V_\alpha(j)) = C(0) + \alpha \sum_j P_{0j} V_\alpha(j)$$

Also, since $V_\alpha(i)$ is increasing in i (to be proved later), $V_\alpha(i) - V_\alpha(0)$ is non negative, and we can write:

$$|V_\alpha(i) - V_\alpha(0)| < K$$

Therefore, there exist a constant g , and a bounded function $h(i)$ such that

$$g + h(i) = \min(K + C(0) + \alpha \sum_j P_{0j} h(j), C(i) + \alpha \sum_j P_{ij} h(j))$$

where g is the value of the average cost function shown in (5.4). To solve the above equation and find the optimal policy, we can use the method of successive approximation and policy iteration [50]; however, with certain assumptions we can describe the structure of the optimal policy more clearly.

5.2.3 Dynamic Load Balancing: Structure of the Optimal Policy

In order for $V_\alpha(i)$ to be an increasing function of i we need the following assumptions:

- $C(i)$ is an increasing function of i .
- $\sum_{j=k}^{\infty} P_{ij}$ is an increasing function of i . This assumption is sometimes called the stochastic ordering of states.

Since the state of distributed SmartPath is based on the load imbalance among the processors, the first assumption is justified. The second assumption means that it is easier to move to a state with higher cost from a state that has a high cost than a state with low cost, which is generally true in the highway; a congestion is more likely to occur in a dense section than in a section with a small number of vehicles. However the assumption also says that there is a better chance to move to a “good” state from a “good” state than from a “bad” state. This may not be true if the controllers (specifically the link layer controllers) are designed such that they wait till the state of the system becomes worse than some threshold and then starts functioning, for instance by opening a new lane, which means that if we allow the highway traffic to reach that congestion level, there is a better chance of moving to a “good” state.

It is easy to verify that if $f(i)$ is a non-decreasing function, then $\sum_j P_{ij} f(j)$ is also non-decreasing. The next step is to show that $V_\alpha(i)$ is a non-decreasing function of i .

Proposition 5.1 *Under assumptions 1 and 2,*

$$V_\alpha : i \mapsto \min(K + C(0) + \alpha \sum_j P_{0j} V_\alpha(j), C(i) + \alpha \sum_j P_{ij} V_\alpha(j))$$

is a non-decreasing function.

Proof: The proof is by induction. Let us assume that the simulation time is N and $V_n(i)$ is the value function when n steps remain in the simulation.

$$V_n(i) = \min(K + C(0) + \alpha \sum_j P_{0j} V_{n-1}(j), C(i) + \alpha \sum_j P_{ij} V_{n-1}(j))$$

and

$$V_0(i) = \min(K + C(0), C(i))$$

Thus, from the first assumption, V_0 is a non-decreasing function. Now, assume that for $j = 0, \dots, n-1$, V_j is non-decreasing. Now

$$V_n(i) = \min(M, f(i))$$

where

$$M = K + \mathbf{c}(\mathbf{0}) + \alpha \sum P_{0j} V_{n-1}(j)$$

and

$$f(i) = C(i) + \alpha \sum P_{ij} V_{n-1}(j)$$

note that from condition 2, $\sum_j P_{ij} V_{n-1}(j)$ is a non-decreasing function; therefore, $f(i)$ is a non-decreasing function, which makes $V_n(i)$ a non-decreasing function. Since

$$V_\alpha(i) = \lim_{n \rightarrow \infty} V_n(i)$$

then, $V_\alpha(i)$ is a non-decreasing function of i . \square

We know from the previous section that $h(i)$ inherits the properties of $V_\alpha(i)$, thus, $h(i)$ is a non-decreasing function of i and $\sum_{j=0}^{\infty} P_{ij} h(j)$ is a non-decreasing function of i . Therefore, the optimal policy is of a threshold type. It will perform a load balancing at state i if

$$K + C(0) + a \sum_{j=0}^{\infty} P_{0j} h(j) < C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} h(j)$$

let

$$I = \{i : C(i) + \alpha \sum_{j=0}^{\infty} P_{ij} h(j) < K + C(0) + a \sum_{j=0}^{\infty} P_{0j} h(j)\} \quad (5.11)$$

then if $i \notin I$, we should activate the load balancing operation.

The above can be summarized in the following proposition:

Proposition 5.2 *Under assumptions 1 and 2, there exist $\hat{i} > 0$, such that whenever $i > \hat{i}$, the policy that chooses to perform the load balancing operation is the average cost minimized policy. Furthermore*

$$\hat{i} = \min\{i > 0 : C(i) + a \sum_{j=0}^{\infty} P_{ij} h(j) > K + C(0) + a \sum_{j=0}^{\infty} P_{0j} h(j)\}$$

5.3 Implementation of the Load Balancing Scheme

To implement the load balancing scheme and to find the corresponding threshold, the server requires periodic transmission of the workload from each client. The workload of each client is estimated by the number of vehicles the client is simulating. The transmission period should be short enough so that variations of the workload are correctly conveyed to the server; however, as the period decreases the cost of load balancing due to the high cost of communication increases. We found it more efficient to allow the server to set the next transmission time of the workloads from the clients.

The message that the clients transmit to the server contains the number of vehicles in each section and the average velocity of vehicles on that section. From this information, the server decides whether to start a new partitioning of the highway graph or continue the simulation as it is. The decision to load balance is based on the load imbalance experienced by the system. In section 5.2.1 we proved that if the distributed system is in state i such that $i \notin I$ where I is the set defined in equation (5.11), then the server should perform the load balancing operation. We can rewrite equation (5.11) as

$$I = \{i : \mathbf{C}(i) < K + \mathbf{C}(0) + a \sum_{j=0}^{\infty} P_{0j}h(j)\} - a \sum_{j=0}^{\infty} P_{ij}h(j)$$

$$I = \{i : \mathbf{C}(i) < K + \mathbf{C}(0) + \alpha \sum_{j=0}^{\infty} (P_{0j}h(j) - P_{ij}h(j))\}$$

Since $\sum_{j=0}^{\infty} P_{ij}h(j)$ is increasing in i , $\sum_{j=0}^{\infty} (P_{0j}h(j) - P_{ij}h(j))$ is always non-positive; therefore, if there exist a state j , such that $K + \mathbf{C}(0) < \mathbf{C}(j)$ then $j \notin I$. However, there is a possibility that there are other states which are less than j and load balancing is still beneficial.

The implementation of the load balancing operation in SmartPath is as follows:

- The server after initializing the clients, the highway graph, and transmission of the partitions to the clients, calculates the next reporting time for clients and waits until it receives the information regarding their workloads. After receiving the information, it calculates the value of imbalance and decides whether to perform load-balancing or not. If the decision is to balance the load, the server generates a set of new partitions from the highway graph according to the new node and edge weights acquired from the clients and transmits the new partitions to the clients. It then calculates the next reporting time and waits for the next message.

- The simulation scheduler on the clients has been changed to accommodate the load balancing operation. Every client after incrementing the simulation time and before updating the vehicles position on the highway, checks the simulation time. If it is the reporting time, it sends the number of vehicles and average velocity of vehicles on every section in its domain to the server. For efficiency, the client then continues the simulation for the next time, after which it waits for the decision from the server. The assumption is that the state of the traffic does not change during one simulation time sample which is accurate if the sample time is small. Therefore, while the calculation on the server is progressing, the clients are simulating the vehicles. The message from server to a client contains the new nodes that it should simulate and the nodes that it should transfer to another client. The client then transmits the vehicle information for the nodes that have to be transferred and waits until it receives the vehicles on its newly acquired nodes. After the nodes are transferred and the vehicles on the new nodes are created, the simulation continues.

5.4 Performance of the Load Balanced SmartPath

We simulated a series of AHS scenarios in SmartPath environment to investigate the following points:

Cost of workload transmission from clients to the server The cost of workload transmission is a function of how often the messages are transmitted. When the message transmission is at every simulation time step (0.1 seconds), the simulation time increased by 5%, and for message transmission at every second, the increase in the simulation time was 0.5%. Note that the clients do not wait for a response from the server until the next sample time. Furthermore, with the load balancing option, this cost is unavoidable. For long simulations (on the order of 1 hour), we set the transmission period to be equal to the time required for a vehicle to move from one section to another, which is calculated by the server' and sent to the clients. This period is also used in the link layer controller for broadcasting the routing information.

Cost of load balancing The cost of load balancing depends on the scenario, the magnitude of the load imbalance, the number of sections that have to be transferred from

Every client sends the average velocity of every section in its domain. The server calculates the $\min_i \frac{V_i}{L_i}$ where V_i and L_i are the average velocity and length of section i , respectively, and send it to the clients. The period is usually between 13-20 seconds for 300-meter sections.

one client to another, and the number of clients involved in the load balancing operation. Usually, if the congestion is the result of an accident or velocity degradation in one section or in sections that are sufficiently far from one another, the clients are involved pairwise in the load balancing operation. In this case cost of load balancing on average is 0.14 seconds per occurrence.

Threshold for load balancing operation. From chapter 4 we know the distributed simulation can simulate about 50 vehicles per processor in real time, and the simulation time is linear with respect to the number of vehicles. With the simulation sample time of 0.1 seconds and the load balancing cost of 0.14 seconds, the state j such that $0.14 + C(0) < C(j)$ is the state that the imbalance in the system is at least 70 vehicles ($\frac{0.14}{0.1}50$). As I mentioned before, it may be beneficial to initiate the load balancing operation with smaller imbalance. The experiments with SmartPath showed us that the simulation time is reduced when the threshold is set to be larger than 50 vehicles. The magnitude of the simulation time reduction depends on the scenario and the nature of the incident.

In one scenario we reduced the velocity of one section to 80% of the normal velocity (from 25 to 20 m/s) for different time durations. By activating the load balancing option, we could decrease the simulation time by about 10%. In another scenario we blocked one lane in a section by stalling a vehicle in that lane for the duration of the simulation; the decrease in the simulation time was in the order of 15%. In these scenarios we used a closed track with a fix number of vehicles.

In the third scenario we introduced an entrance (a traffic source) in one of the sections with a constant inflow to that section and an exit (a traffic sink) in another section. When the traffic flow from the entrance was low, no load balancing happened, and the performance of the simulation was just slightly lower than the distributed simulation without load balancing (due to the load balancing overhead). As soon as the traffic flow increased to 1 vehicle per second, we had several load balancing operations, but the net result was that the the simulation time increased. The reason was that the traffic was flowing regularly and we had a wave of vehicles moving smoothly in the highway. Since the load balancing operation was only checking the magnitude of the imbalance and not the dynamic of the imbalance, it was performing the load balancing operation at every reporting time which was inefficient and increased the simulation time. We solved the problem by calculating the average number of vehicles in every

section during the next time interval (from the current time to the next reporting time) by considering the in-flow and out-flow to that section.

The load balancing operation is costly and inefficient when the simulation time is short or a few vehicles are simulated. Our experiments with SmartPath have shown us that for large scale simulation, the above load balancing technique can reduce the simulation time by as much as 20%.

Chapter 6

Conclusion and Future Extensions

We have presented in this dissertation the structure of PATH-AHS proposal. The proposal is based on a four-layer control hierarchy, and although intended for fully automated vehicles and highways, it is rich enough to encompass proposals based on different strategies. We described the interfaces and the information that should be exchanged among different control layers. The next step was to present the controllers for the control layers. We followed the PATH-AHS proposal of platooning as the mean to reduce the congestion and increase highway capacity. A platoon is a group of vehicle traveling in close proximity of each other. The distance between adjacent vehicles in a platoon is small, and the distance between platoons is large. We describe the internal structure of each control layer and the interfaces among them. The PATH-AHS proposal requires the following maneuvers: join, split, and change lane. In the join maneuver two platoons become one platoon. In the split maneuver two platoons are formed from one platoon. In the change lane maneuver a single vehicle changes lane. Since the vehicles are under automatic control, each maneuver use a communication protocol to acquire permission from the surrounding vehicles to perform the maneuver. The communication protocols are designed to ensure the safety and efficiency of the maneuvers.

In order to observe the behavior of any specific proposal and to test, evaluate, and compare different proposals, we simulated the design. We developed the SmartPath simulation tools for AHS scenarios. SmartPath demonstrated that our modeling approach is sound. SmartPath is a visual simulation package. It provides a graphical interface to view the simulated data (vehicles and highway) in a natural way. SmartPath is a micro-simulation: the behavior of each functional element of the vehicle and highway is individually modeled and simulated. SmartPath is also a distributed simulation, so that different

sections of the highway network can be simulated in different processors. The distributed SmartPath is designed for distributed memory systems and uses the TCP-IP protocol stack for communication purposes.

The future work in the area of control and communication design is as follows:

- **Network and Link layer Controllers** The controller designs for the network and link layer controller are in various stage of developments. The network layer controller is still in the process of development; however, the link layer controllers proposed in [10] and [12] are being integrated in the SmartPath simulation and the testing will begin soon.
- **Communication and Sensor Design** The work on vehicle-to-vehicle communication is in its infancy. Communications among the vehicles within a platoon is carried by radio under the token-bus protocol. The communication among vehicles in different platoon is on the planning phase, see [54] and [55].

The research on sensors is mostly directed toward vision-based sensors. However, the field trials are not completed yet.

- **Degraded Mode** At this stage we have designed and simulated the PATH-AHS proposal assuming that all components of the system perform as expected, which we call the *normal mode*. The next step is to design the controllers for the *degraded mode* of operation, i.e., when there is one or a multitude of device and/or controller failures in the AHS. In [56] and [57] the faults are classified, and a research group is already formed under PATH that actively pursues the design of the controllers for the degraded mode. SmartPath simulation is used as the primary tool for testing and integration of the controllers.

Bibliography

- [1] P. Varaiya, "Models, simulation, and performance of fully automated highways," tech. rep., PATH Research Report UCB-ITS-PRR-94-21, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1994.
- [2] H. Al-Deek, M. Martella, A. May, and W. Sanders, "Potential benefits of in-vehicle information systems in a real freeway corridor under recurring and incident induced congestion," tech. rep., UCB-ITS-PRR-88-2, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1988.
- [3] H. Al-Deek and A. Kanafani, "Some theoretical aspects of the benefits of en-route vehicle guidance," tech. rep., UCB-ITS-PRR-89-2, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1989.
- [4] J. Bender, "An overview of systems studies of automated highway systems," *IEEE Transactions on Vehicular Technology*, vol. 40, pp. 82-99, February 1991.
- [5] C. Thorpe, M. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the Carnegie-Mellon Navlab," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, pp. 362-373, May 1988.
- [6] D. Koller, T. Thorhallson, and H.-H. Nagel, "Model-based object tracking in traffic scenes," in *European Conference on Computer Vision*, (S. Margherita, Italy), 1992.
- [7] P. Varaiya and S. Shladover, "Sketch of an IVHS systems architecture," in *Proceedings of the Vehicle Navigation and Information Systems Conference*, (Dearborn, MI), pp. 909-922, October 20-23 1991.
- [8] P. Varaiya, "Smart cars on smart roads: Problems of control," *IEEE Transactions on Automatic Control*, vol. 38, no. 2, 1993.

- [9] D. Bertsekas and R. Gallager, *Data Networks*, vol. 2nd edition. Prentice Hall, 1992.
- [10] M. Broucke and P. Varaiya, "A theory of traffic flow in automated highway systems." TRB Annual Meeting, Washington, D.C., 1995.
- [11] B.S.Y. Rao and P. Varaiya, "Roadside intelligence for flow control in an IVHS," *Transportation Research Journal, part C*, vol. 2, no. 1, pp. 49-72, 1994.
- [12] P. Li, R. Horowitz, L. Alvarez, J. Frankel, and A. M. Robertson, "An AVHS link layer controller for traffic flow stabilization," tech. rep., PATH draft note D95-15, Institute of Transportation Studies, University of California, Berkeley, CA 94720, March 1995.
- [13] Z. Har'El and R. P. Kurshan, *COSPAN User's Guide*. AT&T Bell Laboratories, Murray Hill, NJ, 1987.
- [14] A. Aziz, F. Balarin, S. T. Cheng, and R. Hojati, "HSIS: a BDD-based environment for formal verification," in *Proceedings of 31st ACM/IEEE Design Automation Conference*, (San Diego, CA), June 1994.
- [15] J. Frankel, L. Alvarez, P. L. R. Horowitz, and A. M. Robertson, "Safety oriented control for AVHS maneuvers," tech. rep., PATH report DR54-94, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1994.
- [16] D. Godbole and J. Lygeros, "Longitudinal control of the lead car of a platoon," tech. rep., PATH Memorandum 93-7, Institute of Transportation Studies, University of California, Berkeley, CA 94720, July 1993.
- [17] J. Hedrick, D. McMahon, V. Narendran, and D. Swaroop, "Longitudinal vehicle controller design for IVHS system," in *Proceedings of the 1991 American Control Conference*, (Boston, MA), pp. 3107-3112, June 26-28 1991.
- [18] W. Chee and M. Tomizuka, "Lane change maneuver of automobiles for the intelligent vehicle and highway systems," *Proceedings of American Control Conference*, pp. 3586-3587, 1994.
- [19] S. Sheikholeslam and C. A. Desoer, "Longitudinal control of a platoon of vehicles," tech. rep., UCB-ITS-PRR-89-3,6, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1989.

- [20] S. E. Sheikholeslam, *Control of a class of interconnected nonlinear dynamical systems: The platoon problem*. PhD thesis, University of California, Berkeley, 1991.
- [21] A. Hitchcock, "Organization of exit and entry on an automated freeway." PATH Technical Report MOU 19 Occasional Papers, 1993.
- [22] D. N. Godbole, F. Eskafi, E. Singh, and P. Varaiya, "Design of entry and exit maneuvers for AHS," *Proceedings of American Control Conference*, pp. 3576-3580, 1995.
- [23] I. P. Hsu and J. Walrand, "Communication requirements and network design for IVHS," tech. rep., PATH working paper ; UCB-ITS-PWP-93-18, Institute of Transportation Studies, University of California at Berkeley, 1993.
- [24] K. Chang, W. Li, A. Shaikhbahai, and P. Varaiya, "A preliminary implementation for vehicle platoon control system," in *Proceedings of the 1991 American Control Conference*, (Boston, MA), pp. 3078-3083, June 26-28 1991.
- [25] B. Foreman, "An infrared inter-vehicle communication system." to appear as PATH Research Report, 1996.
- [26] F. Eskafi, K. Nassiri-Toussi, and G. Liu, "An adaptive baud protocol for wireless communication." to appear as PATH Research Report, 1996.
- [27] F. Eskafi and D. Khorramabadi, "SmartPath User's Manual," tech. rep., PATH Technical Note UCB-ITS-94-2, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1994.
- [28] F. Eskafi and D. Khorramabadi, "SmartPath3.0 User's Manual," tech. rep., to appear as PATH Technical Note, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1996.
- [29] H. Schwetman, *CSIM17 Reference Manual*. Mesquite Software, Inc., 3925 West Braker Lane, Austin, TX. 78759-5321, 1994.
- [30] J. Lygeros and D. Godbole, "Longitudinal control of the lead car of a platoon," *IEEE Transactions on Vehicular Technology*, vol. 43, no. 4, pp. 1125-1135, 1994.
- [31] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, "Protocol design for an automated highway system," *Discrete Event Dynamical System*, vol. 2, pp. 183-206, XX 1993.

- [32] F. Eskafi, D. Khorramabadi, and P. Varaiya, "An automated highway system simulator," *Trusporation Research -C*, vol. 3, no. 1, pp. 1-17, 1995.
- [33] B.S.Y. Rao, P. Varaiya, and F. Eskafi, "Investigations into achievable capacity and stream stability with coordinated intelligent vehicles." Transportation Research Board, 1993.
- [34] R. W. G. Fox and P. Messina, *Parallel Computing Works*. Morgan Kaufmann Publishers, 1994.
- [35] S. Saini and D. H. Bailey, "Nas parallel benchmark results 12-95," tech. rep., Report NAS-95-021, Numerical Aerospace Simulation Facility, NASA Ames Research Center, Mail Stop T 27A-1 Moffett Field, CA 94035-1000, USA, December 1995.
- [36] T. E. Anderson, D. E. Culler, and D. A. Patterson, "The berkeley networks of workstations (NOW) project," tech. rep., Digest of Papers. COMPCON '95. Technologies for the Information Superhighway, San Francisco, CA, March 1995.
- [37] I. Foster, *Designing and Building Parallel Programs*. Addison-Wesley Inc., 1995.
- [38] M. R. Garey and D. S. Johnson, *Computers and Interactability*. W.H. Freeman and Company, 1979.
- [39] J. Demmel, "Applications of parallel computers." U.C. Berkeley CS267, Available via WWW URL from <http://HTTP.CS.Berkeley.EDU/~demmel/cs267/>, 1995.
- [40] B. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal*, vol. 29, pp. 291-307, 1970.
- [41] H. D. Simon, "Partitioning of unstructured problems for parallel processing," *Proceedings of the Conference on Parallel Methods on Large Scale Structural Analysis and Physics Applications*, 1991.
- [42] P. Suaris and G. Kedem, "An algorithm for quadrisection and its application to standard cell placement," *IEEE Trans. Circuits and Systems*, vol. 35, pp. 294-303, 1988.
- [43] B. Hendrickson and R. Leland, "Multidimensional spectral load balancing," *Tech. REp. SAND 93-0074*, 1992.
- [44] J. Song, "A partially asynchronous and iterative algorithm for distributed load balancing," *Parallel Comput.*, vol. 20, pp. 853-868, 1994.

- [45] C. H. Walshaw, M. Cross, and M. G. Everett, "A localized algorithm for optimizing unstructured mesh partitions," *International Journal of Supercomputer Applications and High Performance Computing*, vol. 9, pp. 280-95, Winter 1995.
- [46] J. Misra, "Distributed discrete event simulation," *Comp. Surveys*, vol. 18, pp. 39-65, March 1986.
- [47] R. Righter and J. C. Walrand, "Distributed simulation of discrete-event systems," *Proceedings of the IEEE*, vol. 77, pp. 99-113, January 1989.
- [48] D. R. Jefferson, "Virtual time," *ACM Trans. Prog. Lang. and Sys.*, vol. 7, no. 3, pp. 404-425, 1985.
- [49] V. K. Madiseti, J. C. Walrand, and D. G. Messerschmitt, "Asynchronous algorithms for the parallel simulation of event-driven dynamical systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 1, pp. 244-74, July 1991.
- [50] R. A. Howard, *Dynamic programming and Markov processes*. Technology Press of Massachusetts Institute of Technology, 1960.
- [51] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*. Princeton University Press, 1962.
- [52] S. M. Ross, *Applied probability models with optimization applications*. Holden-Day, 1970.
- [53] S. M. Ross, *Introduction to stochastic dynamic programming*. Academic Press, 1983.
- [54] S. Sachs and P. Varaiya, "A communication system for the control of automated vehicles." PATH Technical Memo 93-5 Institute of Transportation Studies, University of California, Berkeley, CA, September 1993.
- [55] S. Streisand, "A communications architecture for IVHS," Master's thesis, Department of Electrical Engineering & Computer Sciences, University of California, Berkeley, 1992.
- [56] D. N. Godbole, J. Lygeros, E. Singh, A. Deshpande, *et al.*, "Design and verification of communication protocols for degraded modes of operation of AHS," *Proceedings of the 34th IEEE Conference on Decision and Control*, pp. 427-32, Dec 1995.

- [57] J. Lygeros, D. N. Godbole, and M. E. Broucke, "Design of an extended architecture for degraded modes of operation of IVHS," *Proceedings of the 1995 American Control Conference*, pp. 3592-6, June 1995.