SIMON HALLÉ

AUTOMATED HIGHWAY SYSTEMS: PLATOONS OF VEHICLES VIEWED AS A MULTIAGENT SYSTEM

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en Informatique
pour l'obtention du grade de Maître ès sciences, (M.Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

JUIN 2005

Résumé

La conduite collaborative est un domaine lié aux systèmes de transport intelligents, qui utilise les communications pour guider de façon autonome des véhicules coopératifs sur une autoroute automatisée. Depuis les dernières années, différentes architectures de véhicules automatisés ont été proposées, mais la plupart d'entre elles n'ont pas, ou presque pas, attaqué le problème de communication inter véhicules.

À l'intérieur de ce mémoire, nous nous attaquons au problème de la conduite collaborative en utilisant un peloton de voitures conduites par des agents logiciels plus ou moins autonomes, interagissant dans un même environnement multi-agents: une autoroute automatisée. Pour ce faire, nous proposons une architecture hiérarchique d'agents conducteurs de voitures, se basant sur trois couches (couche de guidance, couche de management et couche de contrôle du trafic). Cette architecture peut être utilisée pour développer un peloton centralisé, où un agent conducteur de tête coordonne les autres avec des règles strictes, et un peloton décentralisé, où le peloton est vu comme une équipe d'agents conducteurs ayant le même niveau d'autonomie et essayant de maintenir le peloton stable.

Abstract

Collaborative driving is a growing domain of Intelligent Transportation Systems (ITS) that makes use of communications to autonomously guide cooperative vehicles on an Automated Highway System (AHS). For the past decade, different architectures of automated vehicles have been proposed, but most of them did not or barely addressed the inter-vehicle communication problem.

In this thesis, we address the collaborative driving problem by using a platoon of cars driven by more or less autonomous software agents interacting in a Multiagent System (MAS) environment: the automated highway. To achieve this, we propose a hierarchical driving agent architecture based on three layers (guidance layer, management layer and traffic control layer). This architecture can be used to develop centralized platoons, where the driving agent of the head vehicle coordinates other driving agents by applying strict rules, and decentralized platoons, where the platoon is considered as a team of driving agents with a similar degree of autonomy, trying to maintain a stable platoon.

Avant-propos

Je voudrais remercier tous ceux qui ont rendu possible l'aboutissement des recherches effectuées à l'intérieur de mon projet de Maîtrise. J'aimerais tout d'abord remercier mon directeur de recherche, M. Brahim Chaib-draa, pour sa grande disponibilité, son soutien, ses idées, ainsi que ses précieux conseils.

J'aimerais ensuite remercier les personnes qui ont travaillé avec moi sur le projet Auto21, sans qui nous n'aurions pu achever tous les délivrables de ce projet. Un merci spécial à Phil, Vince et Charly, pour leur travail intense sur le simulateur, l'équipe de l'Université de Sherbrooke, ainsi que Julien, qui a mis tous ses efforts à comprendre mon québécois pour me donner un support bien apprécié dans mes recherches.

J'aimerais de plus remercier le personnel de département d'IFT-GLO: Lynda avec qui tous les problèmes se règlent en deux temps trois mouvements, Gilles qui peut nous construire un meuble de bureau en quelques minutes et tous les autres.

Ensuite, je ne peux passer à côté de tous les membres du Damas, avec qui on peut échanger sur nos problèmes, tout en prenant une petite pause question de se reposer l'esprit. Merci à ceux qui ont organisé des activités et à ceux qui sont venus à celles que j'ai organisées, on s'est toujours bien amusé. Et puis que serait le Damas, sans ses machines à cafés qui transforment un lendemain de veille en une journée de travail productive, alors merci à: Mr. Coffee I, II et III. En plus des membres du Damas, merci à mes ami(e)s Jeff, Jean-Seb, les anciens du bac, enfin tous ceux sur ma liste de contacts msn, et merci au pub de l'Université pour ses 5 à 7, se transformant en 5 à 2.

Finalement, j'aimerais dédier ce mémoire à mes parents. Claude, mon père et Lisette, ma mère, qui m'ont soutenu autant moralement que financièrement et qui m'ont toujours encouragé à foncer et aller plus loin. Merci pour tout, cela n'aurait pas été possible sans vous.

Simon Hallé

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

During the past decades the amount of vehicles driven on North America's major cities' road system has increased at such a rate that we can not keep up by always adding more lanes to our highways [Randall et al., 2000]. Indeed, it is expected that the demand on America's roadways will double by the year 2020 [Network, 2004], meaning that this increase in road users will bring along more pollution, accidents, stress, waste of time. Although the number of accidents are slowly decreasing with the increasing number of drivers in Canada [Gutoskie, 2001], it is estimated that more than 90% of all driving accidents are caused by human errors such as fatigue, inattention, or intoxicated driving [Smiley and Brookhuis, 1987]. Apart from having to face these problems, tomorrow's Canadian driver is always trying to keep up with time by talking on the phone, putting on her makeup or reading while driving, and thus, could use some help to drive his or her car. An answer to our twenty-first century driver's prayers could then be found in a new technology, gaining popularity all over the world, known as Intelligent Transportation Systems (ITS). These systems can be seen as a complex set of technologies that are derived from information and computer technologies, as well as applied to transport infrastructure and vehicles [Lin and Leung, 2002].

It is shown that ITS may provide potential capacity improvements as high as 20 percent [Stough, 2001], which would also lead to fewer pollution caused by both the reduction of time the vehicles spend on the highway and a low emission intelligent driving model based on smooth speed and acceleration changes [Barth, 1997]. Other benefits of ITS include: (i) increased safety; (ii) time savings; (iii) reliable transportation system; (iv) enhanced productivity for the domain of transportation [Gillen et al., 2000], etc. However, in order to benefit the most from ITS, all the technologies must be used as whole and they should collaborate to gain maximum efficiency.

At the moment, ITS is being used as part automated controllers in luxury vehicles including Intelligent Cruise Control (ICC)[1] and warning or collision avoidance systems. Other services relating to the road infrastructures are also being offered in the form of communicated information relating to maps, service locators or anti-theft systems. In addition, the communication systems relating to ITS are also being improved radically through research, on the road infrastructure and vehicle level, to create information networks through a dynamic intranet based on moving vehicles. Following the same path, the Canadian government is now looking for a newer approach to its transportation system, which would include more and more ITS technologies [ITS, 1999].

Being aware of all those facts, we propose to use all the current technologies relating to communicated traffic information and automated vehicle controllers inside a Collaborative Driving System (CDS)[2] A CDS is a system, based on inter-vehicle communications, in which vehicles collaborate by exchanging information or request, in order to drive autonomously (more or less) in formations of vehicles. Thus, CDS is the ultimate form of ITS where the driver can be removed at some point and it is also a logical achievement of all the services offered by the ITS.

In this thesis we describe the research process that led us to the elaboration of a CDS prototype for the Auto21 project. This first chapter begins with a detailed representation of the problems we addressed (Section 1.1). The following sections depict the reasons that motivate us to address a problem relating the Intelligent Transportation Systems (Section 1.2) and the reasons that motivate the development of a Collaborative Driving System (Section 1.3). Finally, the objectives of this thesis are described in Section 1.4, followed by a presentation of the thesis organization in Section 1.5.

1.1 Problem Description

Out of all the technological equipments people use in their every day life, the automobile is probably the most complex one. Indeed, technological advances available for vehicles are growing rapidly, often helping to reduce the negative effects of transportation systems, such as pollution, traffic and safety. Among the technologies making vehicles "intelligent", the ones relating to driver's assistance or autonomous driving are very complex issues. This type of system has to respond to real time critical situations as car malfunctions or cars suddenly braking in front of you. As more driving tasks are

[1] Also known as Adaptive Cruise Control (ACC), ICC uses sensors to automatically maintain following distances.

[2] CDS is sometimes called Co-operative Driving System.

being handled by the vehicle itself, and more gadgets are appearing in your car, two problems arise: conflicts between these automated tasks and the human driver's acceptance or disturbance from these technologies. For example, automated route finders, using a digital map and a Global Positioning System (GPS), often have disturbing effects on most drivers that cannot always stay focused on both the map and the road. Therefore, making a vehicle more "intelligent" by adding new technologies is one thing, but making the vehicle easier to drive for a human is another, which results in a problem that can be addressed through a system incorporating all these technologies, like the CDS.

As many countries are battling in the race for autonomous vehicles, Canada is covering many aspects through the Auto21 network. This network and the definition of the project covering the autonomous vehicles aspects are described in Section 1.1.1. Following this description, the problem of autonomous driving systems is presented in Section 1.1.2. Then, the problems concerning the current ITS technologies are detailed in Section 1.1.3, while the difficulties in choosing the right test environment for our CDS are presented in Section 1.1.4.

1.1.1 Auto21 Project

The Canadian government through the Canadian Networks of Centres of Excellence (NCE)[3] and the help of more than 120 industry, government and institutional partners supports a network called Auto21 [Auto21, 2004] [DAMAS-Auto21, 2004], which brings together most of the Canadian researcher relating to the automobile domain. As its main goal, this networks aims to strengthen the competitive position of Canada in the automotive industry, our most important industry. The Network currently supports over 230 researchers working at more than 35 academic institutions, government research facilities and private sector research labs across Canada and around the world. The researches on Collaborative Driving System (CDS) are done within the network as part of one of the six themes which is called *Intelligent Systems and Sensors* [Auto21, 2004]. The CDS project is led by Dr. François Michaud[4] and involves the University of Sherbrooke, Calgary and Laval University. As its main goal, this project aims at creating the prototype of a system allowing vehicles to coordinate in high-density highway traffic (CDS presented in Section 1.3). To do so, each university has different fields of expertise that are required to complete this multi-disciplinary application and which are briefly described here:

[3]For more information, visit http://www.nce.gc.ca
[4]For more information, visit http://www.gel.usherb.ca/michaudf/

- The University of Calgary, being specialized in the field of telematics with the department of Geomatics engineering, has been assigned the research on *Intelligent Sensors for Vehicle Perception and Navigation*.

- Laval University, through Dialog, Automatic Learning and Multiagent Systems (DAMAS)[5] Laboratory, is involved in the *Coordination and Communication Architectures* sub-project that is detailed in this thesis.

- Sherbrooke University, through the Research Laboratory on Mobile Robotics and Intelligent Systems (LABORIUS)[6], is involved in the two previous sub-project as well as the *Integrated Navigation, Guidance and Control* sub-project.

1.1.2 Autonomous Driving

The previous section described the Auto21 research project that focuses on the collaborative driving research domain, which aims at creating automated vehicles which collaborate in order to navigate through traffic. In this sort of driving, one generally form *a platoon*, which is a group of vehicles whose actions on the road are coordinated using communications. The first vehicle of a platoon is called the platoon *leader* and its role is to manage the platoon and guide it on the road at an undefined level of authority. The other vehicles are called *followers* and their main goal is to maintain a specific distance in time with the preceding vehicle using information from sensor(s). Figure 1.1 shows a real platoon of vehicles formed of automated vehicles from the PATH project[7], evolving on an Automated Highway System (AHS). Within the previously defined Auto21 project, different levels of system functionality have been defined to extend the project through time and needs. As it is the basic need of this project, all the different levels need to support and maintain a platoon structure during such events as vehicle leaving or entering their platoon through different possible emergencies. Three levels of autonomy specify the leader and the followers' tasks and roles inside our project:

- In the first level of autonomy (autonomous longitudinal control), only the relative distance and velocity of the cars are actively controlled in a type of generalized and distributed "cruise control system", although drivers still steer their vehicles manually. A possible usage of the system at this level could be achieved using a driver assistance interface providing steering actions orders to the driver as a lane changes for example. This way, the system could control the vehicle's brake and gas as long as the vehicle is part of, or switching from different platoons.

[5]For more information, visit http://www.damas.ift.ulaval.ca
[6]For more information, visit http://www.gel.usherb.ca/laborius/
[7]For more information on California PATH, at UC Berkeley, visit http://www.path.berkeley.edu

- In the second level of autonomy (semi-autonomous longitudinal-lateral control), the lateral and longitudinal motion of each vehicle are autonomously controlled relatively to the one preceding it, all the way to the first "lead car", in a form of generalized car-train with a specially equipped lead car and trained driver. Again, a possible immediate use of the second level could be done using such lead cars that would co-exist in a given urban center, each with its own generic destination, much like a conventional train or bus, but with the added freedom of "getting off the train with your car".

- Finally, in the third level of autonomy (fully autonomous longitudinal-lateral control), the addition of cooperative steering, using the road and the telematic infrastructure as a guide for absolute motion control, will provide autonomous road-following capabilities. Thus, each vehicle in the third autonomy is able to take the role of a leader, meaning that platoons are completely and autonomously manageable. This final level complicates the problem of coordination, as we eliminate the master entity called the leader, which centralized the coordination for a platoon formation. On the other hand, giving autonomy over collaborative issues to each vehicle allows for much more flexibility, which is crucial for platoons evolving in dense traffic.

Figure 1.1: Platoon of automated vehicles on an Automated Highway System, developed by the PATH project [Hedrick et al., 1994].

At the moment only the first level has been developed and represents the research described in this thesis. The various achievement levels that were just mentioned guarantee that a functional system will be available at every step of this long-term project. To achieve this, the different functional systems should also evolve through software simulation, simulation with robots and finally, using real vehicles on a test road.

For a better understanding of the problems that must be resolved to maintain a stable platoon formation, the two main disturbances in the platoon formations are described below. Those two "disturbance", called *split* (vehicle exiting the platoon) and *merge* (vehicle entering the platoon), are represented in Figure 1.2 and can be detailed as follows:

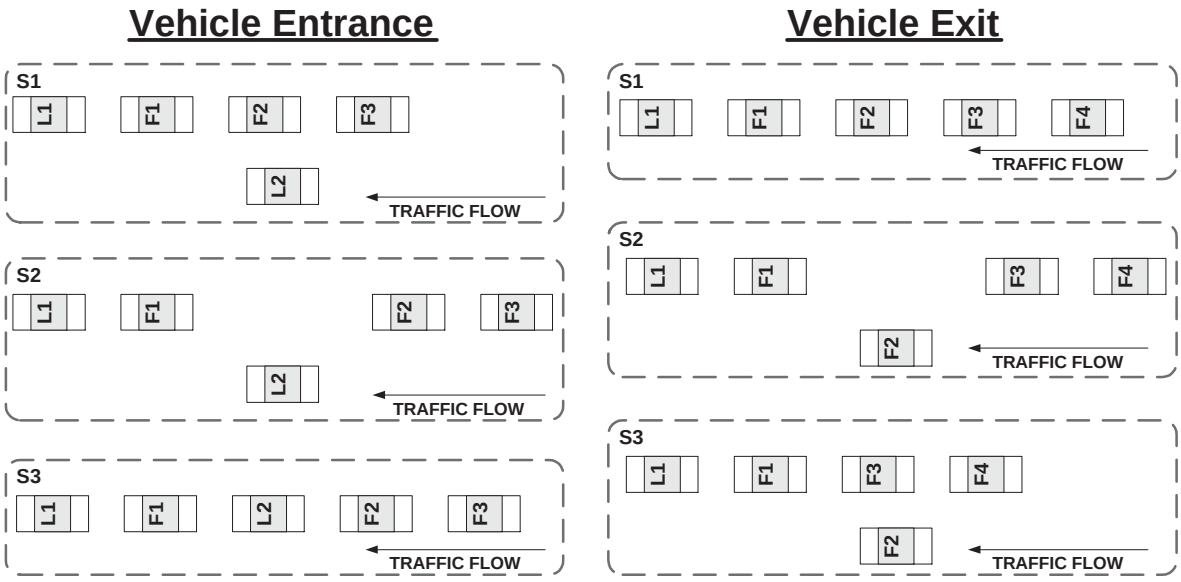

Figure 1.2: The three steps of the removal (split) and insertion (merge) of a vehicle in the platoon.

A Vehicle splitting happens when a vehicle member of a platoon decides to leave it, thereby forming two non-empty platoons. To execute this manoeuvre, the splitter (*F2* in Figure 1.2) must communicate its intention of leaving the platoon, so the platoon formation modifies the distances at the front and rear of the splitting vehicle as shown in *step 1* (*S1*) of Figure 1.2. When this new formation gains stability, the splitting vehicle *F2* can change lane, while the rest of the platoon followers keep the same distances. When the splitting vehicle safely left the platoon (*S2*), the gap created for its departure can be closed, thus forming back the precedent platoon, minus one vehicle (*S3*).

A Vehicle merging is the exact opposite of a split manoeuvre: two non-empty platoons merge together to become one. This manoeuvre requires a platoon formed of only one vehicle, which is *L2* in Figure 1.2, to communicate to another platoon its will to join it. Moving from *S1* to *S2*, the latter platoon reacts by creating a safe space and communicating to the merging vehicle the dynamic position of this space in its platoon. The merging vehicle modifies its velocity to join the meeting point, verifies if it is safe to merge and changes lane to enter the platoon formation and leave *S2* to go to *S3*. Once the merged vehicle has stabilized its inter-vehicle distance, the platoon can reach its precedent formation plus one vehicle, by diminishing the distances with the new

vehicle. Although, the steps of the merge manoeuvre may differ from one coordination approach to another, this represents the general pattern of the merge manoeuvre.

Within the CDS project of Auto21, the problem of platoon driving within the three levels of autonomy presented above has been separated in three different sub-projects. Each sub-project has a common goal of creating a CDS that supports platoons of autonomous vehicles, but they attack very particular aspects of this problem.

- *Intelligent Sensors for Vehicle Perception and Navigation* is a sub-project that studies different types of sensors to provide a higher level of information on the vehicles' surrounding environment. To do so, optimal sets of sensors have to be developed and tested on vehicles. Algorithms for data fusion and filtering should also be developed considering the information required by the deliberative system. The information that the navigation system should provide includes: detection of obstacles, detection of neighboring cars, measurement of relative motion, absolute positioning, etc.

- *Integrated Navigation, Guidance and Control* sub-project has to develop a system in charge of determining the desired dynamical state of the vehicle and applying actions relating to lateral and longitudinal motion in order to acquire those states. At a lower level, this project has to develop controllers, for both the steering wheel and gas and brake pedals, that act in collaboration with the guidance system, using low-level data from the sensors. For the guidance system, such things as the platoon configuration, vehicle's destination and safety issues are taken into account to specify the desired vehicle states to the controllers. The guidance algorithms also need to resolve problems as the platoon's string stability and focus on the robustness of this driving architecture.

- *Coordination and Communication Architectures* is the third sub-project, working in relation with the navigation system by using the information it provides on the vehicle's environment in order to reason about collaborative driving issues in a platoon configuration. By coordinating its vehicle's actions, the communication architecture advises the guidance system, developed in the previous sub-project. The main coordination aspects that are handled are: the support of a vehicle's entrance and exit from the platoon, a vehicle's lane changes and entrance or exit from the highway, and the maintenance of stability with other vehicles member of the same platoon.

The first two sub-projects mainly concern the university of Sherbrooke and Calgary, while the last sub-project relates to the research detailed in this thesis.

1.1.3 Intelligent Transportation Systems & Artificial Intelligence

As the name Intelligent Transport Systems invokes it, these systems must provide AI-based responses to their users. Although this should be true, most of the Intelligent Transportation Systems (ITS) technologies we have seen so far are based on reactive systems. This type of system does not reason or deliberate using exhaustive planning with up to date knowledge bases and elaborate communications, so their actions are very limited.

ITS have mostly focused on the reactive control aspects of automated vehicles, but very few research has been done on the cooperation and coordination of these tasks. In fact, at the vehicle level, Artificial Intelligence (AI) has mostly been used for applications relating to longitudinal control such as Adaptive Cruise Control [Winner et al., 1996] and its enhanced versions, like the Semi-Autonomous Adaptive Cruise Control (SAACC) [Rajamani and Zhu, 2002] detailed in Section 4.1. On the other hand, the vehicle's lateral control is a research area for which applicable solutions have only started to emerge and some have been successful in experimental conditions [Rajamani et al., 2000]. But this technology has to succeed in many more test scenarios since the two dimension control environment is much more complex than the one dimensions of the longitudinal control. In addition, the lane change control algorithms will require severe logical checks to ensure its functionality in uncertain situations. For this reason, automated lateral control, as lane following will not be applicable to commercialized cars in the years to come, so we do not address this problem in our research.

Apart from lower-level control functions, much more complex applications have also been developed for the platoon architectures and were used in real life situations. Such a demonstration was done during Demo 2000 in Japan, by car constructors and projects as the University of California at Berkeley's PATH project [Hedrick et al., 1994] and Japanese research programs as ASHRA[8] and Tsugawa et al. [2001]. Although the communicative and guidance approaches presented in those projects proved to be successful, they did not address the problem of the vehicle formation's flexibility and their communications' efficiency as important issues. As a result, their coordination models have mostly been using communication protocols based on hard coded platoon states, instead of generic states that would be able to respond to any situation. From the information that these projects made available, their architecture is centered on the platoon, seen as a static formation and, most of the time, on its leader, thus restricting the

[8]For more information on Japan's Advanced Cruise-Assist Highway System Research Association, visit http://www.ahsra.or.jp

collaboration possibilities. Moreover, their collaboration models were kept simple and required scripted plans to react to unforeseen situations. To conclude these remarks, it must be mentioned that most of the national projects as the American and Japanese ones presented the fully autonomous platoon architecture as the highlight of their respective project [PATH, 2004], using a demo version developed at the beginning of the project, which was then left aside. Hence, for the past years, researchers concentrated their efforts on the longitudinal and lateral control to be used as intelligent cruise control and left enormous amount of work for the collaborative/networking part of their architecture.

As it has been shown, the problem of longitudinal control is well defined and solutions are now being applied to luxury vehicles as some Mercedes-Benz models, shown in Figure 1.3, which is now available with a distronic adaptive cruise control capable of maintaining inter-vehicle distances. But a great amount of research has yet to be achieved to combine these vehicle control systems with the available route planning systems, in the most efficient manner. As it is represented in Figure 1.4, ITS infrastructures offer multiple services that are very complex and require more coordination among them. Following from these facts, a "merging" system incorporating both the vehicle control and traffic information systems, through a Multiagent approach, seems inevitable. Accordingly, the problem of modelling a Collaborative Driving System as a Multiagent System (MAS), focusing on inter-vehicle coordination, was defined as the most propitious problem to address as part of this thesis and for the Auto21 project's research at DAMAS laboratory.

Figure 1.3: Distronic Adaptive Cruise Control in a Mercedes-Benz [Mercedes-Benz, 2004].

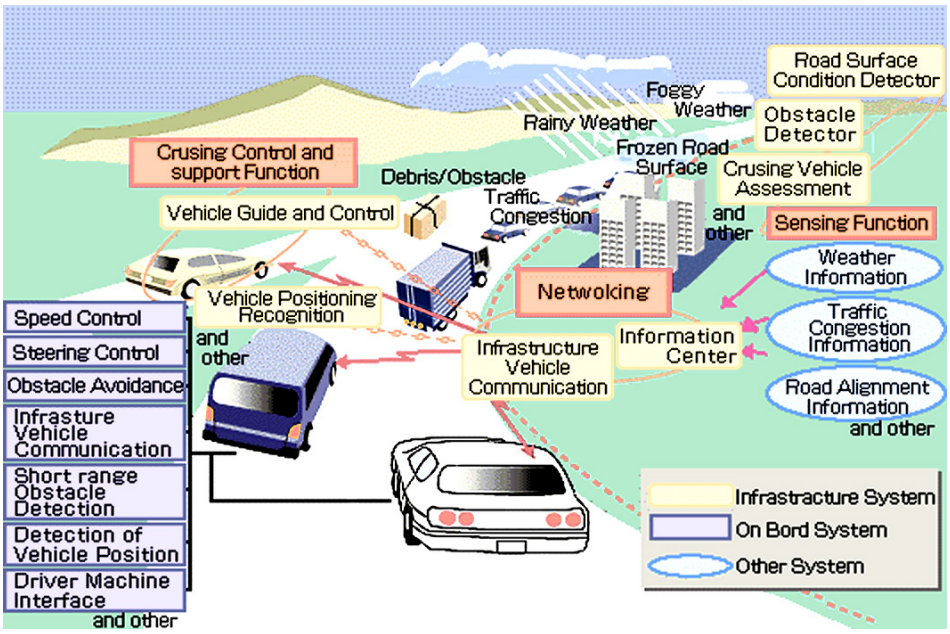

Figure 1.4: ITS architecture of the ASHRA association [AHSRA, 2004].

1.1.4 Intelligent Transportation Systems Simulation

The last problem to be addressed as part of this thesis is the one of finding a suitable environment to test our autonomous driving system. As possible test environments, the following options can be considered: (i) a group of real vehicles; (ii) mobile robots; (iii) or a simulation software. Real vehicles have the advantage of representing the real environment in which the system will evolve and thus, being a good proof of the system feasibility. On the other hand, real vehicles are very expensive, they must be used on a dedicated highway and they may be harmful to people, depending on the type of tests that are executed. Mobile robots are also expensive and, as it is the case for real vehicles, they must be equipped with the necessary sensors and communication devices, which may also be very expensive.

On the other hand, a software simulator does not have the previous disadvantages, although it never reaches the same level of reliability as real vehicles. A software simulator is easier to use, since it does not require a dedicated space, expensive resources and people to manoeuvre, and it can be used in a running loop on customizable time frames. However, intelligent vehicles software simulators may be expensive and require expensive computers, depending on the level of detail of the vehicle model and the scale of the traffic simulation.

1.2 Motivations relating to Intelligent Transportation Systems

ITS have been introduced to the domain of transportation as an answer to many problems that they are already starting to resolve. Indeed, this technology has been very useful in such aspects as: (a) traffic on highways; (b) safety while driving; (c) effects of vehicle on the environment; (d) efficiency in transportation industries; (e) and different social aspects relating to drivers. As these are probably the most important considerations relating to vehicle transportation, their enhancement through ITS constitutes a major motivation to the development of a Collaborative Driving System.

1.2.1 Traffic

As it was mentioned in this chapter's introduction, the vehicle traffic on roads is a major problem that can be resolved using ITS. Up until now, the only solution was to build more roads, and from 1990 to 1995, the overall road system length has increased by more than 13,000 kilometers in Canada [Randall et al., 2000]. Furthermore, the highways formed of more than three lanes, which constitute the type of road usually extended to increase traffic capacity in major cities, grew by almost seven percent (even more for the United States and Mexico) in only five years [Randall et al., 2000]. Taking these facts into account, it is obvious that Canada has an urgent need for an alternative solution to its traffic problems.

As mentioned earlier, ITS have a proven capacity to improve traffic flow, and this is especially true when using technologies as the Adaptive Cruise Control (ACC) and the platoon model. Considering the fact that the instability in the traffic flow is the first cause of the usual traffic jams, stabilizing the traffic flow using ACC within a platoon will ultimately improve the highways capacity [Liang and Peng, 2000]. Indeed, studies on the platoon string stability using inter-vehicle communications showed a great improvement in the flow of vehicles that were used in platoon formations [Darbha and Rajagopal, 1998]. Moreover, using a dedicated highway for automated vehicles, the traffic equilibrium can be reached and maintained more easily at the highway level, increasing even more the benefits of this technology. Apart from the traffic stability the capacity of existing highways can also be increased using this form of automated driving. This can be done by reducing distances between vehicles until they reach the minimal safe distance, considering the controllers' capabilities and the road condition.

1.2.2 Safety

Safety being a very important issue, it is a great incentive to improve automated driving tasks and vehicle emergencies systems, which are directly linked to our CDS. Although Canada's fatality rate per 10,000 motor vehicles registered decreased from 1.79 in 1996 to 1.63 during 1998, its international ranking among Organization for Economic Co-operation and Development (OECD) member countries decreased to 9th from 8th during 1996 and 1997, according to Gutoskie [2001]. According to the same author, the national target calls for a 30% decrease in the average number of road users killed and seriously injured for the 10 years to come.

These numbers show that actions must be taken and automated driving systems could be very effective in resolving this problem. Indeed, technologies precursive to CDS, as the Collision Warning Systems (CWS) have improved the driver's reaction time to emergencies and thus, lowered accidents. It has been argued that if an extra half a second of warning time, is provided to a driver, 60% of collisions can be avoided and with one second of warning time this portion increases to 90% [Woll, 1997]. Thus an important amount of collisions could be avoided using CDS, as this system's sensors are directly linked to the effectors and have a faster reaction time to warnings than humans. Moreover, Touran et al. [1999] showed that the probability of a rear-end collision between a lead car and a car equipped with Autonomous Intelligent Cruise Control (AICC) is significantly lower compared to unequipped cars.

Finally, if we consider the arguments on the string stability, presented in the latter Section 1.2.1, safety enhancement can also be derived from the same facts. Hence, a stable vehicle formation lowers the acceleration and deceleration of each vehicle, thus lowering the possibility of collisions, often caused by unstable traffic flow leading the drivers to apply high decelerations.

1.2.3 Environment

Another major societal issue relating to the use of automobiles is its environment impacts. Following from new Canadian regulations, the automobile industry had to improve the negative impacts of their vehicles, but there are still a lot of efforts left to be done. Indeed, the automobile remains a major source of pollutants as the Canadian emissions of CO2, and contributes to increasing the concerns of global warming [Smith, 1993]. The Transportation sector (excluding pipelines) represents one of the largest sources of emissions in Canada, accounting for 24.7% of total emissions in 2001

(177 Mt) [Jaques, 2003]. For Canada's major cities as Vancouver, Calgary, Toronto, Ottawa, Montreal, and Quebec City, cars have a very important influence on the air quality which influences the citizens' well-being.

Liang and Peng [2000] proved that an Adaptive Cruise Control (ACC) could reduce the average acceleration level of a vehicle, which in return lowers the vehicle fuel consumptions and emissions. Furthermore, Bose and Ioannou [2001] showed that as much as 60% reduction in the air pollution could be achieved, if 10% of the current vehicles would be equipped with ACC. Because vehicles part of a CDS can easily maintain a stable velocity, as well as a close distance with their preceding vehicle, they can reduce the wind resistance applied on their vehicle. Therefore, platoon members have lower fuel consumptions, which is another important factor motivating the use of CDS.

1.2.4 Efficiency

By enhancing highways capacity and providing optimal route planning, ITS traffic management systems can greatly improve the efficiency of Canada's road network. This kind of benefit results from the operational efficiency gained by larger organizations making a wide use of the road network. Thus, city or national bus transit, freight transportation companies, emergency vehicles and many more could profit from advances in ITS, at the productivity level. For both users relating to carriers and state agencies, there is a positive impact on the productivity when using ITS related to guidance or traffic management systems, as they provide significant cost savings and improved service [Proper, 1999].

In particular, ITS technologies relating to vehicle location systems have great impact on fleet management based on a vast road network. As shown in Gillen et al. [2000], Automatic Vehicle Location (AVL) applied to public transit allows transit managers to better utilize resources and generate cost savings. In addition, vehicle navigation devices, when deployed in a wider area, can greatly improve travel time, a well as travel planning time, as opposed to the use of standard maps [Inman et al., 1996]. At last, preliminary analysis revealed that the throughput generated using AHS was increased of 300% for autonomous driving vehicles in platoon formations, and 200% for non-platooned vehicles. Moreover, analysis based on freeways in Long Island and Washington DC, predicted that these capacity improvements could reduce travel time by 38% to 48% [Stevens, 1995]. AHS and the different navigation technologies relating to ITS thus have tremendous impact on the economy and this fact is enhanced when using CDS to form platoons.

1.2.5 Social Aspects

Social issues represent aspects of driving that may not be as obvious as the previous points, but that still need to be improved. Road rage is a rising problem, broadly publicized lately, which is usually caused by stressed drivers or unrespectful driving behaviour from other drivers. In fact, impaired driving and road rage constitute the most important anti-social behaviours and they even seem to be on the rise [Elliott, 1999]. In addition to the current societal problems, the drivers' possible distractions are growing since the appearance multimedia technologies available in cars. These technologies, along with cellular phones, are now broadly used and they constitute a major cause of risky driving behaviours' growth [Beirness et al., 2002].

As a solution to these problems, using an automated driving system can reduce the driver's mental workload to a certain degree [Stanton and Young, 1998], which should improve the driver's behaviour while driving. As the driver gets used to the system, a greater predictability and smoothness of the vehicle handling reduces, in most cases, the driver's stress. As stress should be reduced and distracting technologies should affect less drivers with automated driving vehicles, anti-social driving behaviours would decrease. Moreover, using a fully automated driving system, the problems of impaired driving would be history, considering that humans would not have to drive anymore.

1.3 Motivations relating to Collaborative Driving System

The motivation for the development of a Collaborative Driving System comes in part from the previous motivations of ITS, but also from the benefits surrounding the development of such a system. Canada has far more to go to meet the advances, in the domain of intelligent vehicles, that some countries as the USA and Japan have reached. In addition, ITS infrastructures are now being developed across the nation and the Collaborative Driving System (CDS) proposed in this thesis would constitute their smartest and most efficient use. A collaborative system embracing the currently available vehicle controllers would resolve the previous problem of conflicts between the different vehicles' automation systems. Such a system would ultimately lead to a fully autonomously driven vehicle which would also resolve problems relating to the human driver's acceptance, currently being studied.

More specifically, the motivations for our research on CDS relate to the impact of

our work inside the sub-project of *Coordination and Communication Architectures*. Our work first results in an application that can be reused for other similar problems, which is a great source of motivation, as shown in Section 1.3.1. Another source of motivation is based on the communication and cooperation infrastructure that will be developed as part of our CDS, as explained in Section 1.3.2. A final motivation for this project is presented in Section 1.3.3, which describes the motivation in building our own software simulator.

1.3.1 Possible Deriving Applications

Within the CDS described in this thesis, the problem of maintaining a stable platoon of automated cars on the highway is being resolved. However, the methodology that is used to resolve this problem may also resolve similar problems part of different domains. In fact, various dynamic systems that evolve in a transportation related environment share several needs and goals, and could also be automated at a certain degree, as we plan to do with cars. Girard et al. [2001] presented four similar applications of networked multi-vehicle systems: (1) Mobile Offshore Base (MOB); (2) Automotive Applications (AA) using platooning strategies; (3) Unmanned Combat Air Vehicles (UCAV); (4) Autonomous Underwater Vehicles (AUV). As these applications all relate to the control of a certain group of vehicles equipped with sensors and effectors, that communicate to coordinate their actions, they can share a common agent-based generic architecture. Thus, considering that abstract manoeuvring and communication behaviours can be shared among these applications, it is possible to share a common application core.

If we focus on the domain of automobile, many applications, using a CDS to form platoons, can be outlined. In the public sector, CDS can be used for public transit as the PATH project has done using platoons of buses. Then, emergency vehicles could also find great improvements in their efficiency, as automated driving would enable them to get prepared or perform rescue tasks while driving. As mentioned in Section 1.2.4, freight transportation companies would be more efficient by regrouping their trucks in platoon formations using automated guidance systems. Other derived applications could be developed, as for example, the use of CDS in small electric vehicles, within retirement villages where elders, which often suffer from different disabilities, could be moved around more easily. Moreover, an autonomous driving vehicle could also be used by rental cars companies or for valet parking services, since the unmanned vehicles could return back to a specific destination autonomously. Finally, different applications relating to militaries are foreseen. For instance, the creation of unmanned military convoys transporting goods during dangerous missions could be based on a CDS and it would result in no human losses.

1.3.2 Communication and Cooperation in ITS

As mentioned in Section 1.1.3, recent advances in ITS are starting to build a communication infrastructure dedicated to vehicle guidance and traffic management. Private companies are also part of this technologic boom, as new models for communication networks using vehicles are being proposed by companies like Nortel and Siemens. Major car manufacturers are using more and more communication devices in their vehicles and have plans, in association with telecommunication corporations, for vehicle-oriented Wireless Local Area Network (WLAN) [Holfelder, 2003]. Thus, as the communication structure is being built, information from road-side sensors are also starting to be exchanged for navigation and traffic management purposes.

Furthermore, vehicles should also take part in this network by both sending information about their state and intentions, and receiving the same information from others, in order to plan driving actions. Communication among vehicles using CDS or ACC is very profitable for automated driving in formation of many vehicles. Swaroop et al. [1994] demonstrated that a constant spacing platoon is stable only if certain types of vehicle-to-vehicle communication are available. Xu et al. [2002] also shown the benefits of communication as an addition to standard ACC, which resulted in faster response time and a smoother and safer reaction, resulting in a more comfortable ride.

Most of the result on the use of inter-vehicle communications related to cruise control technologies and not much to complex guidance and control systems as the CDS. Using complex communication messages, control and guidance systems could communicate information about their driving actions through protocols relating to intra or inter-platoon tasks. A convenient system model in which we could include the inter-vehicle guidance and coordination issues could be the Multiagent System (MAS). Indeed, agent based transport logistics systems have been used to analyze the data provided by road-based sensors, and they proved to be useful in representing and managing traffic information [Davidsson et al., 2004]. MAS have also been used in the domain of transportation for applications such as real-time traffic lights control and presented much more efficient results [Dresner and Stone, 2004], and in some cases safer results [Conde et al., 2004], than the current reactive systems.

Wada et al. [2004] have shown that for their prototype of autonomous vehicle, one of the major requirement was for the vehicle to support additional resources without administrative overhead and to offer in-vehicle networks that are flexible and scalable. This relates to what was mentioned in Section 1.1.3 about the current ITS flexibility needs which could be resolved through a Multiagent System. Such system provides a wrapping layer over the multiple sensing and actuating technologies as well as interop-

erability through the Agent Communication Language (ACL). This type of wrapper is detailed in Chapter 2, which presents different agent-based architectures along with agent-based coordination techniques. These techniques have been used in many environments, to handle distributed autonomous applications, as it is the case for our platoon of vehicles. As some examples, we can point out successful uses of a MAS that motivate its use in our CDS:

- Agents for industrial systems management, as the ARCHON project which led to many applications as a power distribution system in Spain [Jennings et al., 1995].
- Agents for spacecraft control consisting in a real-time in-flight diagnosis application [Georgeff and Lansky, 1987].
- Frigates resource management and positioning in a real-time combat environment, as the NEREUS project [Morissette et al., 2004].
- Management of different rescue teams acting in a large simulated urban disaster, to save lives and minimize building damages [Paquet et al., 2004].
- Multi Agent Based Simulation used to synthesize social behaviours of humans or any dynamic objects [Moss and Davidsson, 2001].
- Agents for workflow and business process management as the ADEPT system [Jennings et al., 1996].
- Tactical air traffic controller agents that help alleviate air traffic congestion, as the OASIS system [Ljungberg and Lucas, 1992].

1.3.3 Collaborative Driving System Simulation

As mentioned in Section 1.1.4, considering the different complications relating to the use of real cars or robots to test our CDS, it was more advisable to use a software vehicle simulator, at least for the initial phases of development. The choice of a software simulator was motivated by the following aspects, which also drove our choice for the right simulator: (a) a low cost; (b) reasonable computing power needs; (c) reliability; (d) respond to our ITS needs; (e) easy to use and extend; (f) able to interface with the Java language. More specifically, the chosen software simulator had to support the following requirements, relating to CDS: (i) simulate the vehicle dynamics with a complete vehicle model (with a certain degree of details); (ii) simulate internal and external vehicle sensors; (iii) simulate different types of inter-vehicle communication systems; (iv) simulate manageable vehicle models; (v) manage platoon scenarios in

batch testing with possible uncertain events; (vi) keep a simulation log on vehicle and driving agents aspects.

Given those needs, the two main possibilities are to either build our own simulator or buy one. Building its own simulator requires more time and software simulation knowledge, but gives total control over the simulator's source code. In fact, a CDS simulator can be built over an existing open source simulator project or using available libraries, and this can reduce the programming task. On the other hand, if there is a simulator available on the market, that exactly suits your needs, it may be the best option. Thus, considering that we had software developer resources available at the DAMAS laboratory, it was more advisable to either choose a simulator we could easily extend or to build our own simulator.

As a first glance at the available simulators relating to autonomous vehicles, we looked at simulators from two similar projects: California PATH's Smart AHS Simulator [Kourjanski et al., 1998] and Carnegie Mellon's Simulated Highways for Intelligent Vehicle Algorithms (SHIVA) [Sukthankar et al., 1998]. PATH's simulator answers to most of our requirements as it was built to test autonomous vehicle platoon formations, but it was programmed using a new language called SHIFT and it requires Silicon Graphics supercomputers. SHIVA simulator was programmed in C++, which can be easily interfaced with Java, but it requires Sun Sparc Stations to run, it has a poor documentation and it does not offer licensing possibilities with source code access. Other free vehicle simulators and some open source ones are also available through the Internet, but most of them are either too abstract or oriented for gaming purposes. Detailed vehicle simulators, like CarSim[9], are also available, but usually at expensive cost and they cannot be extended for the autonomous driving implementation needs, since their source code is not available. Finally, we also analyzed traffic simulators that mainly simulate vehicles at a higher level, like the microscopic traffic simulation package Paramics, developed by Quadstone[10]. Similar simulators that can simulate road network's through a social network at a very high macro-level of traffic representation are available [Balmer et al., 2004], but all of them suit traffic management needs, so they were not considered in our final choice.

Considering all these options, we concluded that the best choice was to build our own in simulator using some available libraries to lower the programming task. Using Java programming language and its variety of free libraries and open source code, we knew that our simulator could easily interface with our driving agents, also programmed in Java. Therefore, the choice of building our own Java-based simulator responded to

[9]For more information, visit http://www.carsim.com
[10]For more information, visit http://www.paramics-online.com

our motivations in having a simulation software that respects our specific needs, with total control over the source code.

1.4 Thesis Objectives

According to the Auto21 project's initial tasks decomposition for the theme F: *Intelligent Systems and Sensors* [Auto21, 2004], different milestones have been established and divided within working groups according to research specialization and university affiliation. The Collaborative Driving Systems project's objectives have been briefly summarized in these lines:

> This research project aims at developing a prototype of Collaborative Driving System (CDS) that can be used as part of a Canadian Automated Highway System (AHS). The objectives of this research should be to provide a recommendation and description for an intelligent navigation, guidance and cooperation system. The description of the different architectures should be merged together into one fully operative CDS that should be fault tolerant, efficient and robust, while focusing on safety. The CDS prototype should be developed and tested using software simulation, robots and vehicles to validate the system's motivations that were presented in Section 1.2 and more importantly, demonstrate the system's robustness. As presented in Section 1.1.2, the first model should be an answer to the vehicle's longitudinal control in a model relating to an Adaptive Cruise Control (ACC) system, used in a platoon formation.

Furthermore, the sub-project assigned to DAMAS (*Coordination and Communication Architectures*) has been defined as follows:

> This project has to conduct a survey about potential architectures incorporating the main components of the CDS (sensing, control and communication) as one system that can be used in each vehicle, to cooperate with the highway's infrastructures. Within this architecture, the research should focus on the usage of communication systems and possible coordination techniques. In a first phase, this project should resolve the problem of coordination inside a same platoon (intra-platoon) and in second phase, between neighboring platoons (inter-platoon). The coordination system should demonstrate its ability to handle events such as lane changes and

vehicle merging and leaving platoons, and it should maintain the platoon stable through different possible disturbances. As part of our objectives, the complexity of the approaches, the ability to handle unanticipated event and the amount (in quantity and from/to whom) of communication required to reach an efficient level of coordination are examined.

Accordingly, the objective of our research project mainly consists in studying different architectures related to automated transport systems and the coordination of agents. This study should then lead to the conception of a MAS suiting most of our needs, and to the conception of a simulated system to prove our affirmations. The tasks required to meet these objectives can be summarized in the following points:

- Study different architectures and techniques used for automated vehicles.
- Design a flexible architecture, specific to this project, suiting the different sub-projects needs: *Perception and Navigation, Guidance and Control, Coordination and Communication*.
- Analyze and develop coordination techniques within the previous architecture, to point out the pros and cons of each one.
- Design and develop the selected coordination technique(s) into a collaborative driving application, where agents guide the vehicles to respond to the platooning problematic.
- Analyze and design a highway simulator with specific vehicle technology requirements, to test and evaluate the developed Collaborative Driving System (CDS).
- Implement and develop the previous simulator, according to gradual simulation needs.
- Analyze the coordination and communication aspects of the platooning manoeuvres under various simulated scenarios and conclude on the performances of the proposed CDS.

Although a fully autonomous platoon architecture may seem out of reach, considering the current available technology, a gradual application of the collaborating system, as presented in the problematic is feasible. Hence, even though the lateral controllers are not currently applicable to commercialized vehicles, our coordination model will be usable by assigning tasks that cannot be safely automated, directly to the human driver. Using an interface between the driver and the CDS, as a communication system through the vehicle's speakers, the CDS could request tasks as change lane to the driver. This way, the inter-vehicle tasks coordinated through our system could be applied, at

any time within the development phases, by either using an automated controller or the human driver, for different vehicle control issues. Accordingly, to set the basis of our CDS, we decided to focus on developing a functional demo supporting the automated platoon formation. Since we aim to study approaches for collaborative driving, this research project mostly focuses on the collaboration between platoon members.

As mentioned earlier, the leading vehicle also includes sensing and communication devices, but it is driven by a human for the initial phase. Thus, every platoon members, including the leader, are considered as agents, which can sense and communicate with each others. The only difference between those agents is the actions they can perform, as the leader only communicates, while the followers can also control the gas and brake pedals, and later, the steering wheel.

1.5 Thesis Organization

This thesis is organized in such a way that agent and Multiagent general architectures are first presented in Chapter 2, along with different Multiagent coordination techniques, by putting the emphasis on Teamwork for agents. Before describing the MAS we developed for Auto21, the simulation environment in which our driving agents evolve is presented in Chapter 3, which details every modules of our simulator and how this simulator supports the test scenarios presented as part of our results. Afterwards, Chapter 4 presents the architecture we developed for the Collaborative Driving System of Auto21 and details the coordination and control aspects, as well as the implementation process of this architecture. The inter and intra-platoon coordination models that enable our vehicles to collaborate are then described in Chapter 5, which focuses on centralized and teamwork intra-platoon coordination models. Finally, the development of our agents inside our simulator, according to the previous architecture, is explained in Chapter 6, which also focuses on the centralized and teamwork coordination models. This chapter ends with a presentation and a discussion on the simulation results relating to intra-platoon coordination test scenarios. To conclude, Chapter 7 summarizes our different achievements and results, and ends by detailing possible future works for the CDS project of Auto21.

Chapter 2

Agents and Muliagent Systems

From its debut, around 1943 [Russell and Norvig, 2003], Artificial Intelligence (AI) has been a leading research area of computer science, which proposes a rational approach to realize a given task. In order to use techniques relating to AI in different environments and provide a reasoning system closer to the "human" model, the agent paradigm has widely been used in the past decades. The term "agent" has been used for many purpose and may be a little confusing at that point. For this reason, this chapter clarifies the different possible architectures of agents that relate to different types of environment and levels of complexity. A given environment has specific properties that can be seen as a level of challenge for an agent. If the environment in which our agent evolves forces it to reason about its goals before acting, a deliberative architecture should be considered. In simpler environments, a reactive architecture is more appropriate, while other environments may necessitate reactive and deliberative behavior, thus requiring a hybrid agent architecture. Finally, in other types of environment, an agent must interact with other agents in order to collaborate or compete for a given goal. In this context, the term Multiagent System (MAS) is used to qualify a system in which distributed agents must interact with each others. In this thesis, a MAS is used to model the automated driving system installed on each vehicle to autonomously drive a vehicle and collaborate with others.

This chapter does not propose a vast introduction to intelligent agents, since this subject has been covered many times, so the reader should refer to Russell and Norvig [2003] for a complete introduction. Instead, only the agent and MAS aspects relating to our Collaborative Driving System (CDS) and autonomous driving agents are covered in the following sections. First, Section 2.1 describes the agent paradigm and presents the major architectures that can be used to develop an agent. Then, Section 2.2 describes the problems that are addressed by a Multiagent System and presents different

architectures that can used to coordinate agents.

2.1 Single Agent Architectures

An agent is usually described as anything that can perceive its environment through sensors and act upon that environment through effectors (actuators) [Russell and Norvig, 2003]. Another definition given by Wooldridge [2002] formally and briefly defines an agent as a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives. More applicatively, an agent is an autonomous entity that is usually goal driven, meaning that all of its actions will be done in accordance with a specific goal, pursued by the agent. The actions applied by the agent are therefore a response to a percept or a sequence of percept that defines the agent's view of its environment.

The general representation of an agent is shown in Figure 2.1, where the question mark represents the agent's reasoning system. The reasoning system defines the agent's rationality, which in turn defines the agent's autonomy. At the lowest level of autonomy, an agent mainly relies on its built-in knowledge, while at the highest level, the agent's experience (beliefs acquired through time) is also determining on its behaviour. These different levels of autonomy can be acquired through different categories of agent architectures, going from a simple reactive agent to a complex deliberative agent. The following sections present those architectures, starting with the simple reflex (purely reactive) agent in Section 2.1.1, followed by the deliberative agent (agent with states) in Section 2.1.2 and the Belief Desire Intention (BDI) agent in Section 2.1.3.

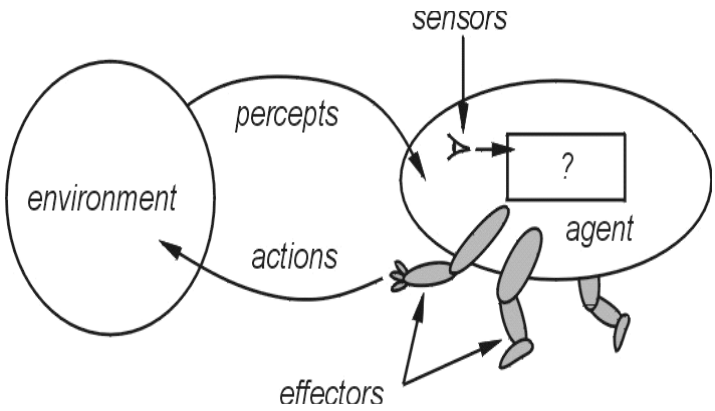

Figure 2.1: The general model of an agent interacting with the environment through sensors and effectors, from Russell and Norvig [2003].

2.1.1 Reactive Agents

A reactive agent is an agent that chooses an action without any reference to its history. Its decision is only based on the present state of the environment and therefore, its decision function can be represented as:

$$action : Per \rightarrow Ac$$

Where the agent's action Ac is executed in reaction to a percept Per from the environment.

2.1.2 Deliberative Agents

A deliberative agent, on the other hand, uses a decision function based on a sequence of environment states represented by the agent's internal state. In this case, the agent's internal state I is determined by its previous state and its new percept Per: $I \times Per \rightarrow I$. Using its internal state, the decision function of a deliberative agent can be defined as:

$$action : I \rightarrow Ac$$

This decision function applies the mapping between states and actions considering two possible reasons: a goal or a utility. In the case of a "goal-based agent", a goal is defined as a state of the environment and the agent must apply a series of actions leading to this state. The agent needs knowledge about the results of its actions to apply the right action, leading to the right goal. In the case of our autonomous driving agents, a goal may be defined as a lane change and the agent will apply a steering action for a certain time, to achieve the goal of being in the other lane.

In contrast, a "utility-based agent" does not have a specific goal, but it uses a utility function instead. This function maps a state with a real number that defines the value of the state or the desire the agent should have toward this state. Again, by having a knowledge about the results of its actions, the agent can choose which action can lean him to the highest value state. For instance, if an autonomous driving agent just entered the highway, its utility function will assign a high value to the state of being in the leftmost lane, thus leading the agent to change lane.

2.1.3 BDI Agents

The BDI agent is probably one of the most popular agent architecture, since it is versatile and flexible, and it can be seen as a combination of a goal-based and a utility-based agent. This architecture originated in the work of the Rational Agency project at Stanford Research Institute in the mid-1980s. Afterwards, the architecture has been extended in many different ways considering application or programming languages requirements. However the general concept based on beliefs, desires, and intentions has always been the same.

Figure 2.2 presents the reasoning process of a BDI agent starting from the entrance of new percepts to the execution of a new action, as presented in Wooldridge [1999]. This architecture regroups seven major components, which can be described as:

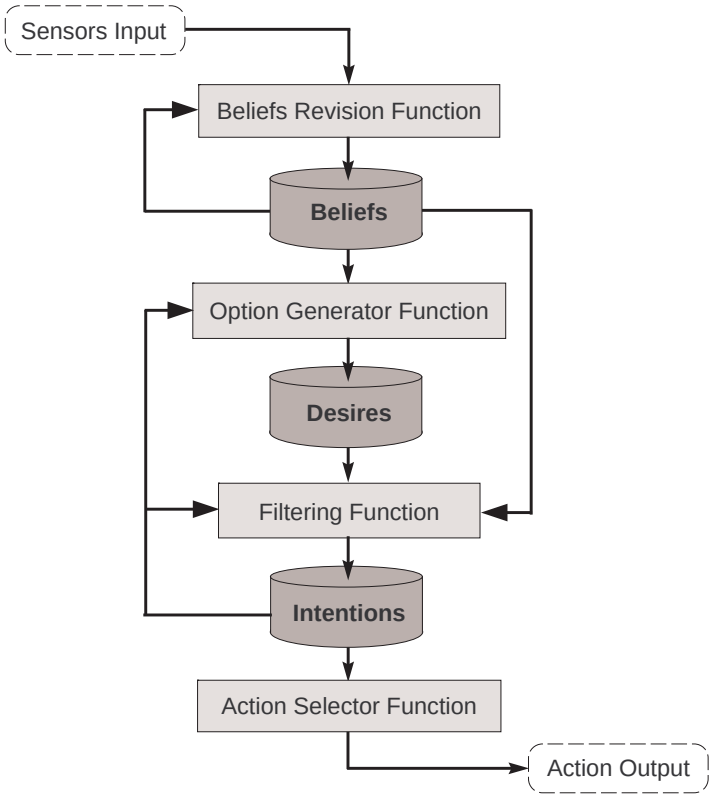

Figure 2.2: Reasoning process of a BDI agent, from Wooldridge [1999].

Beliefs Revision Function: A function using the new percepts as its entrance, to determine the agent's new beliefs considering its current beliefs.

Beliefs: The information or knowledge that the agent currently has about its environment.

Option Generator Function: A function that determines the agent's possible *options* (i.e. desires), considering the agent's current beliefs and intentions.

Desires: The desires that are currently held as possible *options* for the agent.

Filtering Function: A function representing the agent's deliberative process. This function determines the agent's intentions considering its desires, the current belief state and its current intentions.

Intentions: The agent's intentions characterize its current state of mind. They can be seen as persistent goals and once an agent adopts a new intention, this intention constrains its future deliberative process.

Action Selector Function: A function that determines the actions this agent should execute, in order to act in accordance with its intentions.

For a better understanding of the action generation process of a BDI agent, algorithm 1 presents Rao and Georgeff [1995]'s **BDI-interpreter**. In this algorithm, event-queue (sensor inputs), beliefs, desires, and intentions are considered as global structures. The *option-generator* determines the agent's new desires, which are used to determine new intentions using the *update-intentions* function. Using these new intentions, the agent executes the proper actions with the *execute* function. New sensor inputs are then retrieved with the *get-new-external-events* function, to update the event-queue. Finally, the satisfied or impossible desires and intentions are removed using the *drop-successful-attitudes* and *drop-impossible-attitudes* functions.

Algorithm 1 BDI-interpreter

```
initialize-state();
loop
  options := option-generator(event-queue);
  selected-options := deliberate(options);
  update-intentions(selected-options);
  execute();
  get-new-external-events();
  drop-successful-attitudes();
  drop-impossible-attitudes();
```

BDI extensions

As mentioned earlier, the BDI architecture has been used and extended for different purposes over the past years. The JACK Intelligent Agents^{TM}agent oriented programming language, presented in Section 6.2.1, is an example of such an extension for a

programming language. JACK has been used to develop our driving agents, for the Auto21 CDS project and therefore, our agent architecture, detailed in Chapter 6 is an extension of the BDI architecture. Other extensions of BDI have been realized for different applications and one of these applications is presented below.

The Procedural Reasoning System (PRS) architecture [Georgeff and Ingrand, 1990] is considered as an extension of the BDI model and it has also inspired JACK's agent-oriented language. PRS is a general purpose hybrid architecture that is well suited for real-time control environments like the CDS domain. This architecture has been experimented in the Reaction Control System (RCS) of a space shuttle that had to be "fault resistant". This means that PRS is able to respond to any malfunctions and it corresponds to a robust model, appropriate for the control of vehicles.

PRS defines a complete agent reasoning system by interconnecting the agent's belief database with its goals and by using a library of plans called Knowledge Areas (KAs) that represent the agent's intentions. The core of the agent's reasoning is based on an interpreter or inference mechanism that selects the appropriate plans considering the current beliefs and goals. In PRS, a plan is considered as a recipe of actions to execute in time or according to specific states (similar to JACK). Once a plan is selected, it is executed by placing the plan in an intention set, controlled by a task manager.

2.2 MAS Architectures

The previous architectures focused on developing an individual agent, but did not address the problem of multiple agents interacting in the same environment. This issue is examined by a Multiagent System (MAS) architecture, which regroups distributed agents seeking a different or similar goal, together or in competition. A MAS can be described through the definition of a Multiagent environment, which was summarized by three major characteristics by Huhns and Stephens [1999]:

- Multiagent environments contain agents that are autonomous and distributed, and may be self-interested or cooperative.
- Multiagent environments provide an infrastructure specifying communication and interaction protocols.
- Multiagent environments are typically open and have no centralized designer.

A generic example of a MAS is given in Figure 2.3, which is Jennings [2000]'s illustration of different groups of agents evolving in the same environment. The sphere of influence

represent the relation of dependency that agents may have among themselves. If we relate to our CDS, each agent is a vehicle driver and the organizations are platoons of vehicles. The environment is the highway and each vehicle influences its neighboring vehicles. The interaction among platoon members (intra-platoon) and among different platoons (inter-platoon) is defined by the coordination models that are presented in Chapter 5.

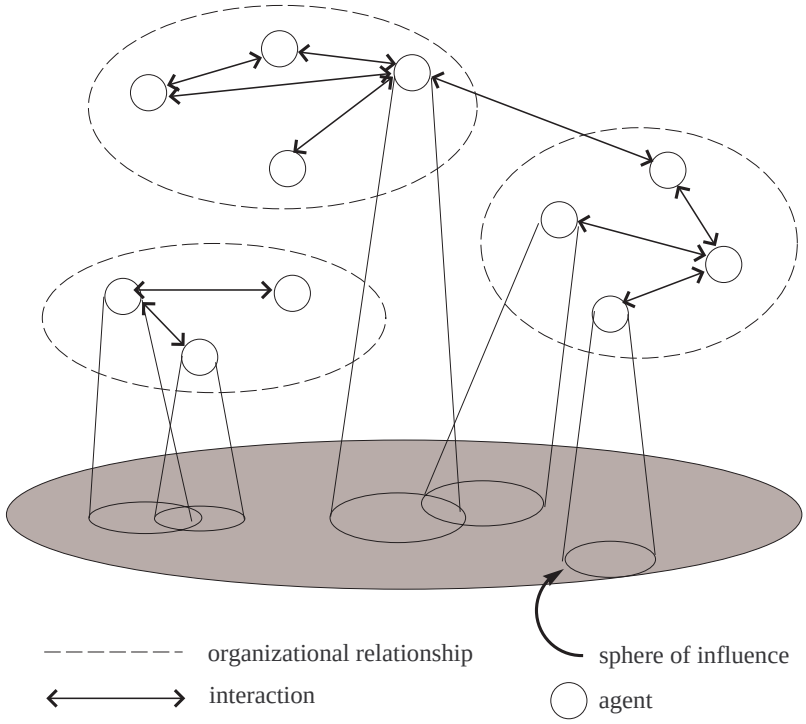

Figure 2.3: Typical structure of a Multiagent System, from Jennings [2000].

To resolve the interactions between the agents of our system, a coordination mechanism must be deployed. As part of the research on MAS, different coordination architectures have been proposed and this section presents the ones that have influenced the choice of the coordination architecture of our CDS. Section 2.2.1 presents a coordination model based on social laws, which controls the agent's communications using a set of laws. Then, the coordination models presented in the following sections can be seen as extensions of the previous BDI and PRS architectures, adapted to resolve problems in groups of agents. First, Section 2.2.2 shortly presents the joint intentions coordination model, which is an introduction to the concept of teamwork. Then, Section 2.2.3 defines coordination models that are based on distributed planning or Shared Plans (SP). Finally, Section 2.2.4 describes the most important coordination model of this thesis: the Multiagent teamwork coordination.

2.2.1 Social Laws

Social laws can be seen as conventions on the communicative behaviors of autonomous agents. In our CDS, social laws are proposed as one of our intra-platoon coordination models, presented in Section 5.2.2.

The use of social laws to restrict agents' actions in a group has been done to regulate cooperative actions without necessitating additional communications. By determining a pattern of behavior either manually, by its designer or through learning from the agents, laws can be defined to regulate the agents' behavior. The emergence of the laws through learning can be realized using a number of strategies.

Simple Majority: Agents will define laws according to their observation of similar patterns of execution from other agents.

Simple Majority with agent types: This corresponds to the previous update function, but the agents are regrouped by similar types that can observe each others and communicate to share knowledge and favor the emergence of common laws.

Simple majority with communication on success: This strategy is attractive to environment with limited communications, as agents only communicate strategies when these strategies have proven to be successful throughout the agent's experiences. This is also the most interesting approach for social laws emergence for an application like CDS, since it should minimize sharing of noisy or unstable policies, as well as the overall communications.

To avoid unexpected behaviors, the social laws can be defined offline, which relates to the design of mechanisms. Using constraints defined as a pair of action with a relative environment state, a set of laws can be developed to constrain such actions from being executed. In our application, we chose the offline approach and we based our coordination model on Shoham and Tennenholtz [1995]'s social laws formalism. This formalism defines a social agent using a tuple $(S, \mathcal{L}, A, SL, T)$ where

- $s \in S$ is the state of the environment.
- $\varphi \in \mathcal{L}$ is a first order sentence of a language relating to the previous state.
- $a \in A$ are the agent's possible actions
- $sl \in SL$ are the agent's social laws that, for a given law sl_1, restrict different actions considering an agent state $(a_i, \varphi_i) \in sl_1$.
- T is the total transition function defined as $T : S \times A \times SL \times \rightarrow 2^S$

In their model, Shoham and Tennenholtz [1995] use a transition function T to do the mapping between states, actions and social laws. As mentioned before, social laws are used to *restrict* the actions of an agent. Thus, when an agent is in a state corresponding to a social law, a transition will be realized to prohibit the agent from executing a specific action. More specifically, if the agent's state satisfies a sentence in the language (denoted as $s \models \varphi$), the transition function verifies if the sentence φ refers to a social law in SL. If a social sl for which $(a, \varphi) \in sl$ is found, then the transition $T(s, a, sl) = \emptyset$ is applied, and the agent is not allowed to execute the action a. In our application, the set of actions A refers to communicative actions and the social laws allows our driving agent to determine when they should communicate (execute an action in A), without requiring further communications.

2.2.2 Joint Intentions

The joint intentions model is another form of coordination, which uses the intentions from the previous BDI agent model, instead of laws, to coordinate the actions of agents that collaborate to achieve a common goal. The joint intentions model is at the origin of the teamwork coordination model, presented in Section 2.2.4. In this model, agents share their individual intentions (determined inside the BDI model) and try to find common intentions, to coordinate their efforts on achieving the goal relating to this intention.

The notion of Joint Persistent Goal (JPG) was proposed by Levesque et al. [1990] to complete the joint intentions model and ensure that agents commit to their intention until the goal has been achieved. In a JPG, a group of agents have a collective commitment about some goal φ, which refers to an intention ψ. Therefore, once a group of agents determine that they have a similar intention ψ, they share a Joint Persistent Goal φ, which guides their individual goals as follows:

1. Initially, every agent does not believe that the goal φ is satisfied;
2. Every agent i then has a goal of φ until the termination condition is satisfied (4);
3. Until the termination condition is satisfied:
    - if any agent i believes that the goal is achieved, then it will have the goal that this becomes a mutual belief;
    - if any agent i believes that the goal is impossible, then it will have the goal that this becomes a mutual belief;

- if any agent i does not have ψ as a local intention anymore, then it will have the goal that this becomes a mutual belief;

4. The termination condition is that it is mutually believed that either:
   - the goal φ is satisfied;
   - the goal φ is impossible;
   - the intention ψ for the goal is no longer present;

2.2.3 Distributed planning

Another coordination model which has been reused inside the teamwork model is the distributed planning. Distributed planning can be achieved in many different ways, but this section only details a model that relates to the coordination of plans inside a team. Before detailing this particular model, the three major forms of distributed planning in a MAS are described by relating to the definition of Durfee [1999]:

Centralized planning for distributed plans: In this model, a centralized planning system develops a plan for a group of agents, in which the division and ordering of labor is defined. This "master" agent then distributes the plan to the "slaves", who then execute their part of the plan.

Distributed planning: In this model, a group of agents cooperates to form a centralized plan. Typically, each agent is a "specialist" in a specific aspect of the overall plan and will contribute to a part of it. However, the agents that form the plan will not be the ones to execute it, since their role is merely to generate the plan.

Distributed planning for distributed plans: In this model, a group of agents cooperate to form individual plans of actions, dynamically coordinating their actions along the way.

The third form of distributed planning is the most complex one, but it is also the form that relates the most to the coordination of autonomous driving agents, inside a CDS. The distributed planning for distributed plans can be achieved through different models of Shared Plans (SP) as: the Partial Global Planning (PGP) [Durfee and Lesser, 1987]; and the Partial Shared Plans (PSP) [Grosz and Kraus, 1996, 1999].

The PGP can be summarized by three iterated stages an agent goes through, when using this planner:

1. Each agent decides what its own goals are, and generates short-term plans in order to achieve them.
2. Agents exchange information to determine where plans and goals interact.
3. Agents alter local plans in order to better coordinate their own activities.

The PSP is similar to the PGP, but it is based on joint intentions, presented in Section 2.2.2, and it is usually used by teamwork models to execute the plans of a team of agents. In this model, shared intentions and plans are well distinguished from individual ones within respective execution, although they have hierarchical relations with each others. Therefore, when agents come to a joint intention towards a goal, the PSP constrains each agent to execute local plans that are compatible with this joint intention. Note that PSP is defined as a complete architecture in Grosz and Kraus [1996], but for the purpose of this thesis, we will not go further in its description. Nevertheless, Section 2.2.4 gives more details on the plan (operator) hierarchy and shows the relation between shared and individual plans.

2.2.4 Multiagent Teamwork

The teamwork coordination model is based on the joint intentions model and the Partial Shared Plans (PSP) model, which were presented earlier. In the teamwork model, agents are regrouped in teams, in which each agent has a specific role that guides its actions. Teamwork for agent has been a very popular subject for the past years and many different architectures have been developed: PTS [Stone and Veloso, 1999], DTCP [Zhang et al., 2004], RETSINA [Giampapa and Sycara, 2002], etc. For our CDS, we selected the STEAM architecture, defined by Tambe and Zhang [2000], since it suited all of our needs. Therefore, this section details the STEAM architecture, which is presented inside our application later in Section 5.2.3.

STEAM stands for Shell for TEAMwork, it is based on the joint intentions theory (Section 2.2.2) and it uses a planning system based on the Shared Plans (SP) theory (Section 2.2.3). Accordingly, STEAM plans, which are called operators, are organized in a hierarchy similar to the PSP of Grosz and Kraus [1996]. Another similar, but not necessarily related hierarchy is applied to the agent roles included inside a team formation. Thus, a team structure in STEAM contains one or many different roles that must be filled by one or many agents, in order to form a joint intention and begin the execution of the team's tasks.

STEAM Team Structures

In STEAM, a team may have a flat or a hierarchical organization where a team may be recursively composed of subteams. Each team is composed of roles (and/or sub-teams), which can be of two types:

Persistent roles: These are long-term assignments of roles to the individuals or sub-teams in the organization. For example, in our CDS, the persistent roles are the ones of the leader and the followers, as shown in Section 5.2.3. Typically, this role assignment will not change in the short term.

Task-specific roles: These are shorter-term assignments of roles based on the current task and situation. For instance, in our CDS, the roles that are required to execute a merge or split manoeuvre in the platoon are task-specific roles.

An example of the roles that may be required for a team in a different domain than CDS was presented in Tambe [1997]. This team is presented in Figure 2.4, which shows a team of attack helicopters on a mission. In this team, *transport* helicopters have static roles that are persistent since they relate to the type of helicopter. However, the *escorts* surrounding the transports may become *attackers* at some point during the mission, so they are task-specific roles.

The assignment of roles to agents or subteams is based on their capabilities and their current state. However, this assignment may not be provided ahead of time, so individuals may need to volunteer or be requested to fill in the roles. For instance, in our CDS, agents that are not leaders are defined as followers at their initialization. On the other hand, the roles involved in a manoeuvre of merging or splitting from a platoon will be assigned to agents considering their position in the platoon.

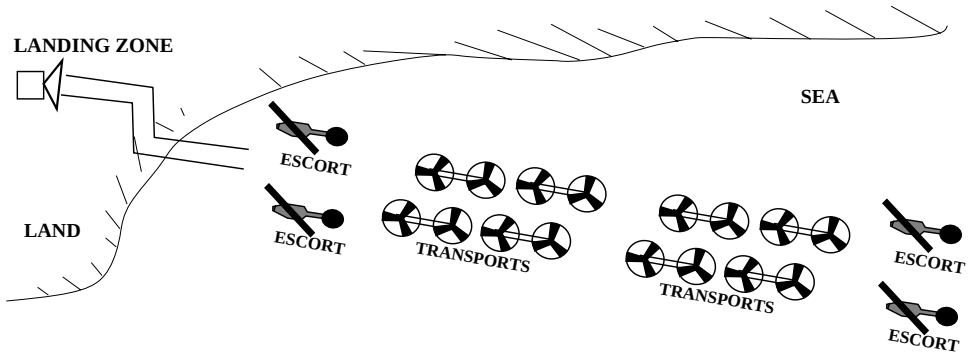

Figure 2.4: Roles involved in a team of attack helicopters, from Tambe [1997].

STEAM Teams and Joint Intentions

Before assigning roles to agents, a team must be formed and this requires all team members to have a common goal. As presented in Section 2.2.2, the joint intentions allow agents to communicate their individual intentions and form groups that stick to this intention until the goal is either achieved or impossible, in a Joint Persistent Goal (JPG).

By using the joint intentions model, a team in STEAM ensures that each member shares the same goal and that they will not deviate from this goal until its teammates all agree about it. In our CDS, the joint intention of inserting a vehicle in the platoon (execute a merge task) arises when the members of a merge task-team A mutually believe that a vehicle i wants to merge the platoon. Thus, in the joint intention model of Levesque et al. [1990], this situation could be denoted as $JPG(A, \text{[Insert Vehicle]})$, meaning that the task team A has the joint persistent goal of achieving the team action of inserting a vehicle in the platoon. In the precedent context, the joint intention model also defines the precondition of the team plan [Insert Vehicle] (detailed in the next section), which is: the vehicle i is not currently in the platoon; and its postcondition, which is: the vehicle i is in the platoon.

The joint intention model also specifies a protocol to establish mutual belief, known as the request-confirm protocol [Smith and Cohen, 1996]. But considering that in our case, the mutual belief comes from a broadcasted communication from the task initiator (merger or splitter), we do not need such a protocol and it is not presented in this thesis.

STEAM Domain-Level Operators

The domain-level operators relate to the agents' individual plans and the team's Shared Plans (SP) that are particular to the agent's domain of application. In STEAM, domain-level operators are structured inside an operator hierarchy in the same way as the agent plan architecture of [Rosenbloom et al., 1991].

As with individual operators, team operators also consist of: (i) precondition rules to help their activation; (ii) an operator application rules to apply active operators; and (iii) termination rules to terminate active operators. However, while an individual operator applies to an agent's private state (its private beliefs), a team operator applies to an agent's team state. A team state is the agent's abstract model of the team's mutual beliefs about the world. The mutual beliefs may be synchronized through

communication, using the SC operator presented below or by making the assumption that a certain belief was perceived by all team members.

For a better understanding of the domain-level operators, Figure 2.5 illustrates an example of the individual and team operators present in the domain of attack helicopters, detailed in Tambe and Zhang [2000] . The key novelty in the STEAM operator hierarchy is the addition of team operators in the tree. Thus, operators shown in brackets (i.e. []), such as [Engage] are team operators and others are individual operators, which express an agent's own activities. At any time, only one path through this hierarchy is active in an agent. This fact is ensured by our plan execution framework, presented in Section 6.2.3. Note that the same operator hierarchy tree was defined for our CDS, but it is only presented in Section 5.2.3, since this is a introduction to STEAM.

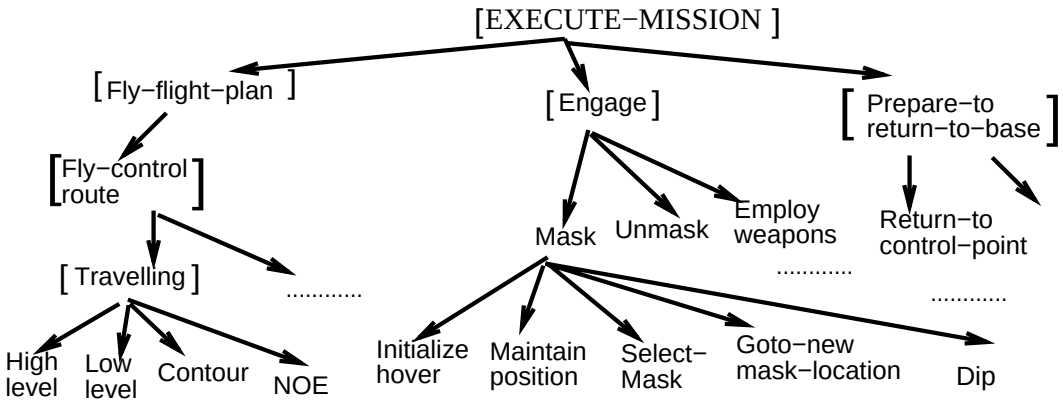

Figure 2.5: Domain level team operators in an example of the attack helicopter domain from Tambe and Zhang [2000].

As the SP model of Grosz and Kraus [1996], the joint intentions relating to a team operator define the subset of plans that can be executed by an agent in a specific role. More specifically, a role in STEAM contains a specification that constrains its agent to a subset of individual operators inside the team operator. For instance, if an agent in the domain of the attack helicopter fills the *transport* role and its team initiates the [Engage] team operator, it will not be allowed to execute the "Employ Weapon" individual operator because of its role definition (a transport unit).

STEAM Operators

To complete STEAM's team-oriented framework, three categories of operators have been defined: CP, MR, and SC. These operators have the advantage of being handled

by the framework, so they do not require any additional development by their users (agent developer). Once a team is formed, through a joint intention, these operators monitor specific dynamic aspects of the team and they ensure the coherence of the activities of the team. To preserve coherence, each operator can communicate with other team members and "automatically" coordinate with others when new aspects arise during the execution of the team's task.

First, the Coherence Preserving (CP) operator's function is to ensure coherent initiation and termination of team operators. A CP action can be seen as a communicative act to inform others, if the agent discovers that the currently selected team operator is either achieved, unachievable, or irrelevant. Consequently, this operator maintains the coherence of the joint intention, as it was shown in Section 2.2.2.

Second, the Monitor and Repair (MR) operator's function is to detect if a team task is unachievable due to unexpected member failure. In other words, this operator preserves the roles constraints within the team. When defining team formations, the required roles and the amount of agents that must fill each role must be specified. In addition, the relation between these roles must also be specified to ensure that the team is always "logically coherent". For this purpose, STEAM supports the definition of AND ($\bigwedge$), OR ($\bigvee$), and Role dependency ($\Longrightarrow$) relations among its teams' roles. Thus, if two roles are in an AND ($\bigwedge$) relation, the failure of one role will result in the execution of a MR operator that finds another agent to fill this role. In an OR ($\bigvee$) relation both roles must fail and in a Role dependency $R_1 \Longrightarrow R_2$, only the failure of role R_2 will be critical.

Third, the Selective Communication (SC) operator's function is to communicate for the team operators' synchronization and termination. This is probably the most useful operator of STEAM since it integrates decision theoretic communication selectivity and provides a manageable way to ensure mutual beliefs inside the team. In a few words, this operator monitors the agent's local knowledge and compares it with the team's mutual beliefs to decide whether or not the agent should communicate its new knowledge to other team members. The SC operator is close to Tambe's latest infrastructure: COM-MTDP (**Com**municative **M**utiagent **T**eam **D**ecision **P**roblem) [Pynadath and Tambe, 2002], in which communication selection's optimality is considered, as opposed to STEAM, which uses decision-theoretic communication selectivity. SC operators thus verify if a communication within a team must be done, according to the domain's communication costs and benefits. Not only that, but the selective communication also verifies the likelihood that the information it wants to communicate, is already common knowledge.

Figure 2.6 shows a decision tree representing the Selective Communication (SC) decision the agent must take, as part of the STEAM framework [Tambe and Zhang, 2000]. The first 2 branches represent the choice of communicating or not, which are then divided by the probability ρ that the information (belief) the agent wants to communicate is not known by its teammates. Furthermore, a third pair of branches is added to specify the probability σ that this information opposes a threat to the execution of the current team operator. This last probability represents the significance of the new information compared to the current mutual belief, for the current team operator. To make a decision on the communication of a new belief, a team member must verify if the expected utility of making this communication $EU(C)$ is higher than the expected utility of not communicating $EU(NC)$. $EU(C)$ is defined as:

$$EU(C) = \sigma * S - (C_c + (1 - \sigma) * C_n)$$

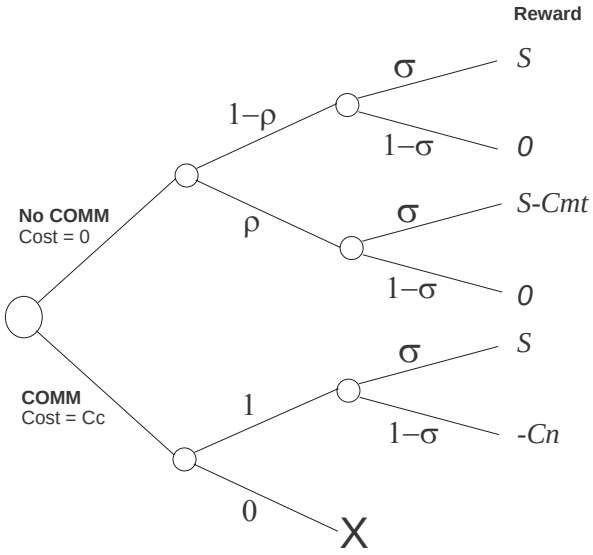

Figure 2.6: Decision tree with probability and rewards for communicative acts in STEAM [Tambe and Zhang, 2000].

Within this definition, the EU is a reward S for the synchronization of the team's belief during the execution of a team operator, minus the Cost of communication C_c and the Cost of nuisance C_n. C_n is used in probability $1 - \sigma$, which is the probability that the information to communicate opposes a threat to the current team operator. $EU(NC)$ is defined in the same way:

$$EU(NC) = \sigma * S - (\rho * \sigma * C_{mt})$$

Where the Cost of nuisance is replaced by C_{mt}: the cost for miscoordination. These two definitions give us the equation to make a decision on the Selective Communication (SC), thus communicating when the $EU(C) > EU(NC)$, i.e., iff:

$$\rho * \sigma * C_{mt} > C_c + (1 - \sigma) * C_n$$

STEAM Evaluation

As shown in Tambe and Zhang [2000], STEAM's flexibility and reusability are some of the major advantages of using this team framework. Indeed, by providing three types of team operators ensuring the team coherence and automatically managing the team's communications, STEAM presents flexible framework to develop a MAS. As it will be shown in Section 6.4, using STEAM makes the development of agents and their plans a lot easier since many communication plans are already developed as part of the framework.

Another important advantage of STEAM is its selective decisions on communications. Tambe and Zhang [2000] showed that STEAM allowed agents to perform well with fewer messages than other messaging theories. However, better results on communication selectivity could probably be achieved by using an architecture as COM-MTDP [Pynadath and Tambe, 2002], but for the moment, STEAM provides the right level of selectivity for the needs of our CDS.

Chapter 3

Agent Oriented Driving Simulator

Most software development projects in computer science require a testing environment to ensure a robust and fault tolerant product. In the case of our application such an environment is required to test the Multiagent System (MAS) we built to support the Collaborative Driving System (CDS) of the Auto21 project, in accordance with the theory on intelligent agents presented in the previous chapter. Recent publications in agent oriented project management [Knublauch, 2002] proposed a series of test cases and unit tests that should be applied through the development process of the agent's behaviors. Thus, to perform those test and support our development phase, an agent oriented driving simulator was built and will be described in this chapter.

As mentioned in Chapter 1, we had to build our own simulator for the Auto21 project because most of the simulators currently available on the market did not suit our specific needs. Indeed, our simulator requires a lower level of micro traffic simulation and a higher level of vehicle dynamics and sensory system simulation. In order to facilitate the development process of our simulator, we started its development from a frame of simulator available at Dialog, Automatic Learning and Multiagent Systems (DAMAS) laboratory [NEREUS, 2004], which was used as an event clock. To extend this frame to the simulation of a CDS including detailed vehicle dynamics, we used Java 3D API™, which provides a three dimension environment in which we could include 3D shapes of vehicles. Then, we adapted the Java3D environment considering the requirements of automated driving systems, by providing different hierarchical levels of simulated objects along with their dynamics and the required sensory, actuation and communication systems. Such a vehicle simulation model including sensing, actuation and communication functionalities enabled us to create an interface that any type of driver can use to drive the vehicle. Therefore, our simulator provides an interface with the simulated vehicles, so the intelligent driving agents that are presented in following

chapters can easily be added to our simulator environment and drive these vehicles.

A development process of about two years at our laboratory resulted in a simulator called Highway Environment Simulator for Travelling Intelligent Agents (HESTIA)[1], which is represented by a screen-shot in Figure 3.1. This screen-shot shows the main 3D environment Graphic User Interface (GUI) including a platoon of vehicles moving on a straight road, along with the simulation results graphics at the bottom of the screen. This simulator can be used to simulate a driving system ranging from a simple Adaptive Cruise Control (ACC) to a complete CDS, and it offers to developers the possibility to extend the simulator, as we have complete control over the source code. The following sections describe the HESTIA simulator starting with an overview of its engine in Section 3.1, the 3D environment in Section 3.2, and all the components simulated inside each vehicles in the following sections. To end this chapter, Section 3.6 presents the interface between our drivers (driving agents) and the simulated vehicle, while Section 3.7 describes the system developed to automatically execute collaborative driving scenarios.

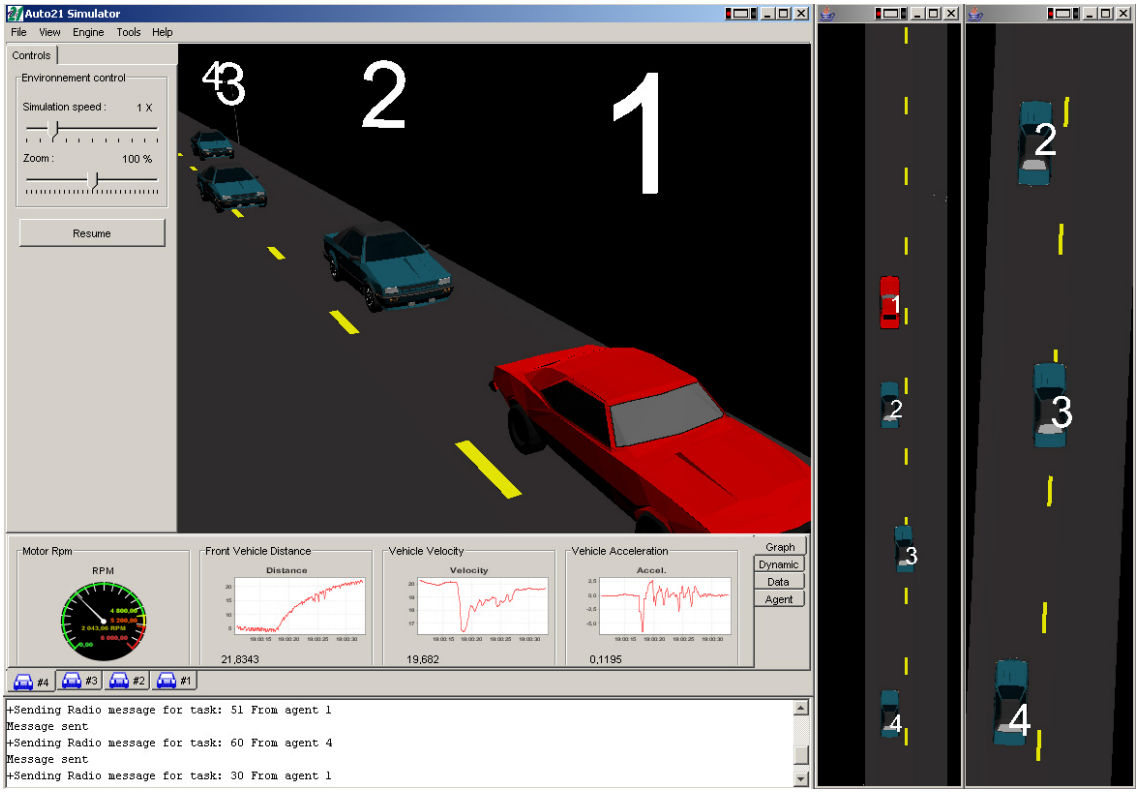

Figure 3.1: Screen shot of a merging vehicle inside the HESTIA 3D simulator.

[1]Hestia is the Greek goddess who symbolized the alliance of the Metropolis with the smaller settlement, which can be seen as the source of the road networks.

3.1 Simulator's Engine

The simulator's engine is the core of the simulator, which controls the time steps of the main running loop. In our case, the simulation runs in continuous time and it is controlled by time step events, which notify the simulated objects of new time steps. Each time step has been set to a value of $20ms$, which was considered as the highest value we could use to receive substantial results from the differential equations of the vehicle dynamics. Moreover, the choice of events to regulate time inside the simulator's engine allows the HESTIA simulator to run in both discrete (if required) and continuous time, which is very useful.

For a better understanding of the simulator's running loop, Figure 3.2 shows the relation between the simulator's engine and the simulated objects. The engine sends time step events to a "Simulated Object" at a time sequence specified in a configuration file and these time step events refer to a time quantum value specified in the same file. A "Simulated Object" can be any type of moving 3D object ranging from a bicycle to a boat, which can run inside our simulator. Note that in the current version of our simulator, only the "Simulated Vehicle" object, representing a car, extends the "Simulated Object" and therefore, only cars run in our simulator. The "Simulated Vehicle", which is detailed in the following sections, includes the vehicles' sensors, actuators, communication system and an interface with its driver.

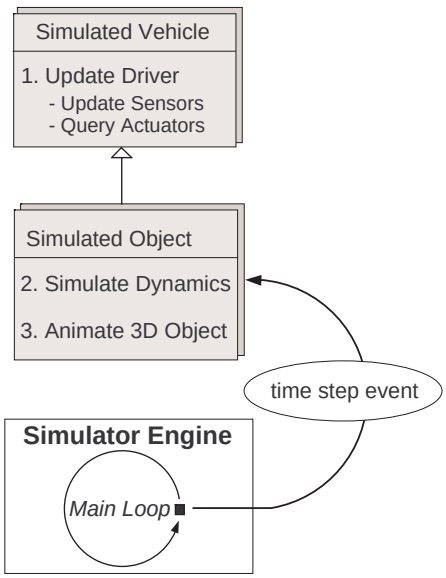

Figure 3.2: The simulator's engine main loop flow.

As Figure 3.2 shows it, once an "Simulated Object" receives a time step event, the

"Update Driver" method of "Simulated Vehicle" is called (action 1. in Figure 3.2). Action 1. results in the update of the vehicle sensors, if necessary, and the query of new actuation commands through the driver's interface. Then, the simulated object dynamics are updated in action 2. according to the actuation command queried in action 1. At last, the 3D object is modified (translations and rotations, in action 3.) considering the time step and the motion calculated by the dynamics simulation model in action 2.

3.2 3D Environment

The 3D environment of our simulator is composed of a straight road on which automated vehicles drive in a longitudinal manner as presented in the example of Figure 3.1. The current vehicle dynamics only support two dimensions so the third dimension (z axis) is only used by sensors that recognize the vehicles' different shapes in the three dimensions. In this 3D environment, objects like our 3D vehicles can be assigned different behaviors, which allows us to implement sensors that retrieve vehicles' information, as it is shown in Section 3.4. Since the road is also a 3D object, it can be assigned properties like its friction coefficient, which can be retrieved to modify the vehicle tires' behavior.

Besides the previous 3D objects' features, the choice of Java 3D was motivated by the facts that:

- Java 3D provides a high-level object-oriented programming paradigm that enables a rapid deployment of new 3D features on our simulator, while reusing our code.
- Java 3D's support, through Java's developers community, brings a wide variety of tools and extensions to Java 3D, often in open source code.
- Java 3D also provides support for runtime loaders enabling us to import models of vehicles from practically anyone, like vendor-specific CAD formats.
- Java 3D draws its ideas from existing graphics APIs and from new technologies. Java 3D's graphics constructs synthesize the best ideas found in low-level APIs such as Direct3D, OpenGL, QuickDraw3D, and XGL
- The agent-oriented tools we are using (like JACK Agent) are also Java APIs and most of our programmers are acquainted with Java.
- Java 3D is part of the JavaMedia suite of APIs, making our simulator available on a wide range of platforms.

We developed our simulator's 3D environment and the vehicle simulation model according to a general model that is presented in Figure 3.3. This model shows how the simulated environment is being used by our driving agents that act on the simulated inter-vehicle communications, presented in Section 3.5 and the vehicle's controller, supported by the driving system interface, described in Section 3.6. The vehicle controllers are directly related to the simulated vehicle dynamics, detailed in Section 3.3. In addition, the agents in our CDS can use one or many sensors that are implemented using Java 3D tools, which enable the simulated sensors to recognize 3D vehicle shaped and retrieve the associated vehicle data, as it is shown in Section 3.4. Finally, a *Scenario Manager* and a *Log Manager* have been developed to execute collaborative driving scenarios and keep a log of the details of their execution, as it is shown in Section 3.7. All the previous components are described in this chapter by defining the specifications we used to create their simulation model and by defining the software engineering model we used to develop them inside the simulator software. Figure 3.3 is therefore referenced in each software engineering section to highlight the relation between the different components.

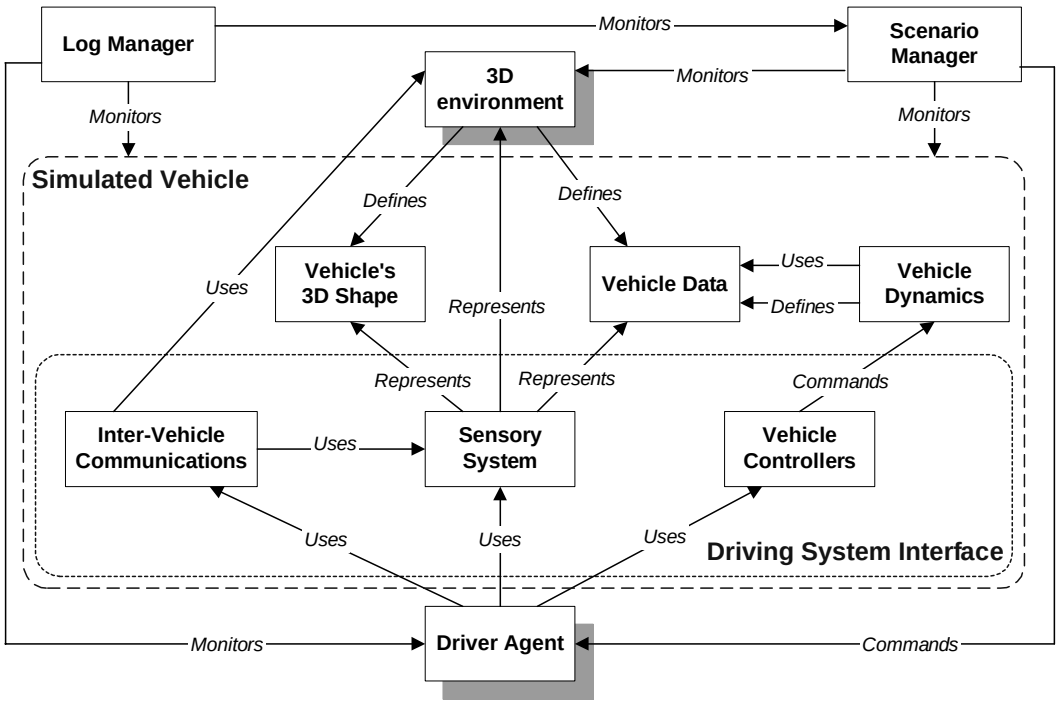

Figure 3.3: The general model of the vehicle simulation environment.

Since different types of vehicles should be tested inside this simulator, we made it easier to define car specifications for the simulator's users. Hence, an XML file, representing a vehicle model class, can be edited to specify the 3D model of the vehicle and the vehicle's specification data, like its weight, length, the properties of its engines, etc. In addition, the different devices used by the automated driver can be customized

in this file to include different sensor classes or communication transceiver/receiver classes. The XML files is loaded at startup to make the vehicle model available for the platoon scenarios or to manually add specific vehicles on the highway.

3.3 Vehicle Dynamics

A vehicle dynamics simulation can get very complex when it is being detailed at a high level, so we decided to use a simplified version of the vehicle's components dynamics, which makes the simulated vehicles less "process intensive" and still offers relatively good dynamic behaviors. This means that our level of details is much lower than other single vehicle car simulators like CarSim[2], a costly simulator widely used to analyze the behavior of four-wheeled vehicles under every aspect of their mechanical components. Our simulator uses an abstracted dynamics model to allow us to run multiple cars or car platoons in real-time, in the simulator. Nevertheless, our simulated vehicle model includes longitudinal and lateral vehicle dynamics, wheel model dynamics, engine dynamics, torque converter model, automatic gear shifting and throttle/brake actuators. The engine and transmission torque converter and differential were translated from a model developed under MATLAB/SIMULINK by our partners at Sherbrooke University [Huppe et al., 2003]. The wheel model and vehicle's lateral and longitudinal dynamics have been developed according to a single-track model, as well as the theory on the chassis' motions models, described in Kiencke and Nielsen [2000]. The detailed theory on the vehicle dynamics simulation model is described in Section 3.3.1, while Section 3.3.2 presents the implementation of the dynamics model in the simulator infrastructure.

3.3.1 Dynamics Specifications

In order to present a wide overview of the theory on the vehicle dynamics, this system can be divided in three main categories: (i) the driveline; (ii) the wheel model; (iii) the vehicle model. For each category and throughout this section, different variables relating to a vehicle dynamics are being used according to current definitions:

[2]For more information, visit http://www.carsim.com

v_{Ds}	:	*angular velocity of the drive shaft*
v_{Tu}	:	*angular velocity of the torque converter's turbine*
$Ratio_{Tr}$	:	*transmission ratio*
$Ratio_{Di}$	:	*differential ratio*
$\dot{v}_W$	:	*angular acceleration of the wheels*
R_W	:	*radius of the wheels*
T_{Ds}	:	*torque of the drive shaft*
T_{Brake}	:	*torque of the brakes*
F_W	:	*traction force of the wheels*
I_W	:	*inertia of the wheels*
v_W	:	*rotational equivalent wheel velocity*
v_{WGC}	:	*wheel ground contact point velocity*
v_{CoG}	:	*velocity at the vehicle's Center of Gravity*
v_{wind}	:	*environment's wind velocity*
$\dot{\psi}$	:	*yaw rate*
ψ	:	*yaw angle*
s_L	:	*longitudinal slip*
s_S	:	*lateral (side) slip*
s_{Res}	:	*resultant wheel slip*
α_R	:	*rear tires side slip angle*
α_F	:	*front tires side slip angle*
β	:	*vehicle body side slip*
δ_W	:	*front wheels turn angle*
l_R	:	*distance from Center of Gravity to rear axle*
l_F	:	*distance from Center of Gravity to front axle*
μ	:	*friction co-efficient*
$\underline{T}_{UIn}$	:	*undercarriage to inertial co-ordinates transformation matrix*
m_{CoG}	:	*mass at the Center of Gravity*
$\ddot{x}_{In}, \ddot{y}_{In}, \ddot{z}_{In}$	:	*acceleration in the inertial co-ordinate system*
F_{WS}	:	*lateral (side) friction force of the wheels*
F_{WL}	:	*longitudinal friction force of the wheels*
F_{Xij}, F_{Yij}	:	*traction forces from the rear and front, left and right wheels*
F_{ZCij}	:	*vertical chassis forces on the rear and front, left and right wheels*
F_R	:	*friction between the tire and road surface*
F_{wind}	:	*wind (drag) resistance*
F_G	:	*gravitational force*
C_{aer}	:	*co-efficient of aerodynamic drag*
$J_{Z\ddot{\psi}}$	:	*torque around the Center of Gravity's z axis*

Driveline Model

The model of our simulated vehicle's driveline is described in Figure 3.4, which shows the flow of torque created by the main components inside the vehicle's driveline. Hence, the driveline starts with the torque created by the engine's combustion, which is subtracted from its internal friction torque and the external load created by the clutch torque, to get the engine's revolution. Then, the engine causes the rotation of the flywheel which is the entrance of the torque converter. Basically, the flywheel works on the pump that makes the torque converter's turbine spin. This creates the turbine torque that directly acts on the transmission, which applies a transmission ratio depending on the current gear, to turn the propeller shaft. Notice that we are using an automatic transmission (which explains the torque converter) in the driveline model. The automatic gear shifting is done according to two reference tables, given by the car constructor, which return a downshift and an upshift velocity according to the current gear and gas throttle.

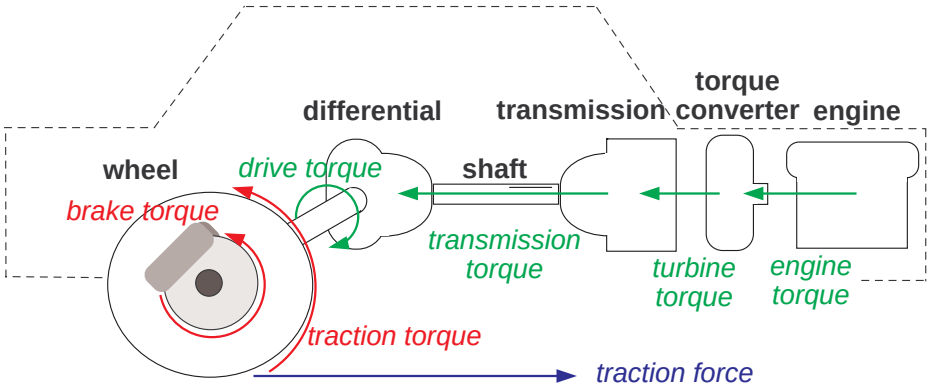

Figure 3.4: Abstract model of a car driveline dynamics.

Another conversion ratio is finally applied on the propeller shaft's angular velocity to get the drive torque. This ratio relates to the differential's action and its resulting torque enables us to get the angular velocity of the drive shaft. The final part of the driveline that we just described can be summarized as:

$$v_{Ds} = \frac{v_{Tu}}{(Ratio_{Tr})(Ratio_{Di})} \tag{3.1}$$

Where v_{Ds} represents the angular velocity of the drive shaft, given by the angular velocity of the torque converter's turbine, v_{Tu}, divided by the product of the transmission and differential ratios. The resulting rotation of the drive shaft determines the rotation of the wheels, by subtracting the brake and traction torque. More specifically, the brake torque results from the pressure applied on the brake pedal, while the traction torque results from the traction force that makes the vehicle move on its longitudinal

axis. This can be summarized by the following equation, which determines the wheel's angular acceleration $\dot{v}_W$:

$$\dot{v}_W = \frac{T_{Ds} - T_{Brake} - 2 \cdot R_W \cdot F_W}{I_W} \tag{3.2}$$

In equation 3.2, the traction torque is referenced by the wheel radius R_W times the traction force on the wheel F_W, which is multiplied by 2, for the two wheels. The three torques are then summed to get the total torque, which is divided by the inertia value of the rear wheels and the drive shaft to get the wheels' angular acceleration, source of their angular velocity. The traction force is detailed later on in the wheel model, but it basically represents the friction force resulting from the wheels' velocity and angle, the type of tire and the road's condition. As it will be shown in the description of the vehicle model, the traction force makes the vehicle move and it also represents the negative charge that the engine must fight to make the wheels turn.

Wheel Model

The wheel model that has been developed in the HESTIA simulator uses the wheels' current angular velocity and turn angle as inputs to deliver the vehicle's front and rear traction forces. As a first step, the wheel model must determine the differential angles in the movement of the vehicle's Center of Gravity (CoG) with the front and rear wheels. Following from this, the tire side slip angle can be found by using a single-track model [Kiencke and Nielsen, 2000]. This angle is then used along with the wheel axle velocity vector and the wheel ground contact point vector to determine the derivation of the wheels' slip. Then, the frictional forces of the wheels can be determined using an estimate of the friction co-efficient and the previous wheel slip information. The traction force is finally given by the transformation of the frictional forces vectors from the wheel system to the vehicle's undercarriage system, by considering the wheels' turn angle (front wheels).

The previous representation of the wheel model was rather brief, so the parameters' calculation are now detailed a little bit more. The wheel slip used to measure the frictional forces can be described as the difference between the rotational equivalent wheel velocity and the CoG velocity (wheel ground contact point velocity). The first velocity is referenced as v_W and the latter velocity as v_{WGC}, in Figure 3.5. The wheel slip is calculated as the difference between these two vectors, projected on the vector v_{WGC}, which represents the real wheel velocity. This way, longitudinal slip s_L occurs when the engine makes the drive shaft turn faster than the current rotation of the wheels, while lateral slip, or side slip s_S, occurs when the wheels are being turned at an

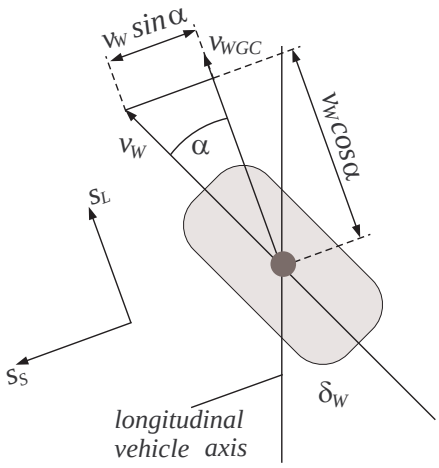

Figure 3.5: Wheel slip calculation using Burckhardt method.

angle different from the current rotation angle of the vehicle CoG. To find the lateral slip, we must calculate the angle α between vectors v_W and v_{WGC}, for both the front and rear sets of wheels.

The rear tires side slip angle α_R is caused by the lateral force of the vehicle's CoG, which creates the vehicle body side slip angle β and the yaw rate $\dot{\psi}$. $\dot{\psi}$ represents in a way, the angular velocity of the vehicle on the z axis, as shown in Figure 3.6. The value of α_R can be formalized by the following equation:

$$\alpha_R = -\beta + \frac{l_R \cdot \dot{\psi}}{v_{CoG}} \tag{3.3}$$

On the other hand, the yaw rate $\dot{\psi}$ has the inverse effect on the front wheels' slip angle α_F. This is shown in Figure 3.5, where the front wheel is also affected by its turn angle δ_W. More specifically, this fact is represented by:

$$\alpha_F = -\beta + \delta_W - \frac{l_F \cdot \dot{\psi}}{v_{CoG}} \tag{3.4}$$

Once, we have the side slip angles α, the difference between vector v_W and v_{WGC}, for both the rear and front wheels, enables us to find the side and longitudinal slip (s_S and s_L). v_W is determined from the rotation of the wheels on the drive shaft (from $\dot{v}_W$ in equation 3.2) and v_{WGC} is determined from the vehicle's body velocity. In our case, we simplified the equations for v_{WGC} and did not take in consideration the yaw rate body side slip angle to determine this velocity, like it was proposed in Kiencke and Nielsen [2000]. Moreover, since we are using a traction on the rear wheels only, the front wheels' velocity v_W is the same as the wheel's real contact point velocity v_{WGC}, but the difference comes from the side slip angle α, as shown by the equations in Table 3.1.

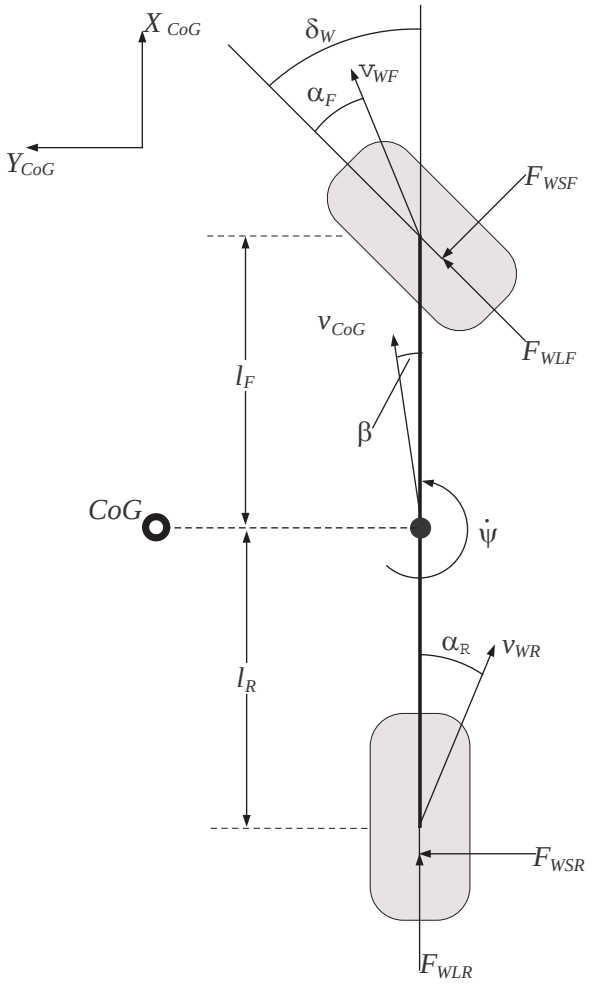

Figure 3.6: Tire side slip angle calculation using the single-track model.

	Braking $v_W \cdot \cos\alpha \leq v_{WGC}$	**Driving** $v_W \cdot \cos\alpha > v_{WGC}$
Longitudinal slip	$s_L = \frac{v_W \cdot \cos\alpha - v_{WGC}}{v_{WGC}}$	$s_L = \frac{v_W \cdot \cos\alpha - v_{WGC}}{v_{WGC} \cdot \cos\alpha}$
Side slip	$s_S = \frac{v_W \cdot \sin\alpha}{v_{WGC}}$	$s_S = \tan\alpha$

Table 3.1: Equations for the longitudinal and side wheel slip.

From the two slip values, we can calculate the resulting wheel slip, which is simply $s_{Res} = \sqrt{s_L^2 + s_S^2}$. The resulting slip value allows us to calculate the friction co-efficient μ, by using the method of Burckhardt [1993], defined as:

$$\mu(s_{Res}) = c_1 \cdot (1 - e^{-c_2 \cdot s_{Res}}) - c_3 \cdot s_{Res} \tag{3.5}$$

Where c_1, c_2, c_3 are constant given for specific types or road. Thus, a vehicle's low-level controller that needs to get a maximum acceleration could use this formula to find the wheel slip at which it gets the highest friction coefficient μ. Having this μ value, the controller could modify the gas or brake percentage to slip just enough to get the maximum grip. For more information on typical roads friction coefficient curves, refer to Kiencke and Nielsen [2000].

In summary, by using the previous equations, we found the values for: the friction co-efficient; the side, longitudinal and resulting slip; and the side slip angle. These values are then used as inputs to formulas described in Kiencke and Nielsen [2000], which apply an attenuation factor relating to the type of tire, in order to determine the longitudinal and side friction by transforming the wheel contact force. The later force is based on the gravitational force and the vehicle's weight, which act on both the rear and front drive shafts, depending on the vehicle's acceleration. Note that in our model we do not simulate the vehicle's suspension. Once we followed the method of calculation of the friction force of the two sets of wheels', we simply have to transform the forces on the front axle using the wheel turn angle δ_W, to get the traction at the front: F_{XF} and F_{YF}. For a better understanding the readers may refer again to Figure 3.6, showing the friction force acting on the front and rear wheels, identified as F_{WS} for the side force and F_{WL} for the longitudinal force.

Vehicle Model

Once the traction force being applied on each wheel of the vehicle model has been determined, other resistance forces can be added to the vehicle model. Thus, the wind resistance (drag), the rolling resistance and the gravitational force are added to the traction forces to get the final resulting forces on the vehicle body. Following from this summation, we finally transform the forces from the CoG or undercarriage coordinate system to the vehicle's inertial coordinate system, by using the transformation matrix $\underline{T}_{UIn}$:

$$m_{CoG} \cdot \begin{bmatrix} \ddot{x}_{In} \\ \ddot{y}_{In} \\ \ddot{z}_{In} \end{bmatrix} = \underline{T}_{UIn} \cdot \begin{bmatrix} F_{XFL} + F_{XRL} + F_{XFR} + F_{XRR} + F_{windX} + F_{GX} + F_R \\ F_{YFL} + F_{YRL} + F_{YFR} + F_{YRR} + F_{windY} + F_{GY} \\ F_{ZCFL} + F_{ZCRL} + F_{ZCFR} + F_{ZCRR} + F_{windZ} + F_{GZ} \end{bmatrix}$$

In this transformation, $\underline{T}_{UIn}$ represents a transformation matrix for rotating the forces vector from the undercarriage to the inertial co-ordinate system. In our application, we only apply a rotation along the z axis, since we only consider the yaw rate and we do not calculate any force creating pitch and roll angles on the vehicle. This is because the roads we simulate in the current scenarios do not include any elevation and the suspension is not modeled. Nevertheless, the gravitational force on the z axis is still used to calculate the rolling resistance and the wheels' friction force, as mentioned before. In the previously transformed matrix, the rolling resistance, which refers to the friction between the tire and road surface, is represented by F_R and it is calculated from the equation:

$$F_R = -RRR_0 \cdot F_Z - \frac{RRR_1 \cdot F_Z \cdot v_{CoGX}}{TRR_0} \tag{3.6}$$

In equation 3.6, RRR represents a constant of resistance assigned to the current road and TRR is a resistance considering the type of tire being used. Thus the rolling resistance will increase with the weight of the car and its velocity. In addition to the rolling resistance, the wind (drag) resistance is also added to the undercarriage system:

$$\begin{bmatrix} F_{windX} \\ F_{windY} \\ F_{windZ} \end{bmatrix} = \begin{bmatrix} -C_{aerX} \cdot (v_{CoGX} - v_{windX} \cdot \cos\psi - v_{windY} \cdot \sin\psi)^2 \\ -C_{aerY} \cdot (v_{CoGY} - v_{windX} \cdot \sin\psi - v_{windY} \cdot \cos\psi)^2 \\ 0 \end{bmatrix}$$

Even though the current scenarios do not specify the environment's wind velocity v_{wind}, the wind resistance comes from the co-efficient of aerodynamic drag C_{aer}, referring to the vehicle's body shape, which increases with the vehicle's velocity. Once all the forces of the vehicle's inertial system have been calculated, we simply transform them into the vehicle's acceleration using $F = m \cdot a$, which enables us to get the vehicle's new velocity and move it to its new position in the environment.

The final movement that is applied on the 3D vehicle object is a rotation on the z axis. The angle of rotation is given by the angular acceleration around the z axis, which is the quotient of the torque around this axis, divided by the vehicle's inertia. This torque refers to the yaw rate, defined in equation 3.7, for which l_F and l_R are described in Figure 3.6:

$$J_{Z\ddot{\psi}} = (F_{YFR} + F_{YFL}) \cdot l_F - (F_{YFR} + F_{YFL}) \cdot l_R \tag{3.7}$$

For more information or the details on the vehicle dynamics equations, an extensive representation is given in Kiencke and Nielsen [2000] and another overview of the longitudinal model is available in Huppé [2004].

3.3.2 Dynamics Software Engineering

The vehicle dynamics model is presented in Figure 3.3, where it uses the actuators' commands to calculate the vehicle's new dynamics data, stored in a vehicle data structure. To implement the vehicle dynamics models, we decided to use the same polymorphism that modeled the hierarchy of our simulator's objects. Within this model, the dynamics equations are separated inside the classes of objects to which they relate, as part of a hierarchy going from a simple moving object, to a motor vehicle and finally, to a car like a sedan. To speed up the process time required to simulate the dynamics of our vehicles, some of the previous equations have been partially modified since they are called very often by the simulator's engine. Indeed, the vehicle dynamics data must be updated at a very high rate, using a low value of delta time to demonstrate a good behavior and therefore, the update functions represent the first part to optimize in order to run more vehicles in real time.

The classes involved in dynamics simulation are presented in Figure 3.7, which also shows an overview of the hierarchy of objects running inside our simulator. First, the hierarchy of objects starts with SimObject3D, which derives from a Java 3D class that enables our objects to run in the 3D environment. This class is then extended by AnimatedObject3D, representing an object that can be moved at run-time (not a road or a post light). AnimatedObject3D and its further descendants use two objects as attributes to represent their dynamics information: the AnimatedObjectDynamic class representing the basic object dynamics and the AnimatedObject3DData class, representing the basic object dynamics data. Other objects deriving from AnimatedObject3D, like RoadVehicleCore and CarCore, have access to the same internal objects, but they have to cast them in more advanced versions, referring to their specific needs. For example, a RoadVehicleCore is using the AnimatedObject3DData object as a RoadVehicleData and the AnimatedObjectDynamic object as a RoadVehicleDynamic. Thus, vehicle dynamics data relating specifically to a car are included inside CarData and their dynamics simulation functions in CarCoreDynamic. For instance, the equation relating to any type of vehicle like the engine, transmission, brakes, suspension (not simulated here) are defined inside the RoadVehicleDynamic class, while the wheel model and part of the previous vehicle model is inside the CarCoreDynamic.

Finally, our software engineering model is very beneficial for the creation of new types of vehicles inside the simulator, which is more likely to happen. For example, a motor bike, a van or any type of truck could extend the RoadVehicleCore class for their specific properties and only implement specific dynamics models for their wheels and chassis, which makes the development task a lot easier. Moreover, using our dynamics model, we can manage to minimize the amount of different classes loaded in the virtual

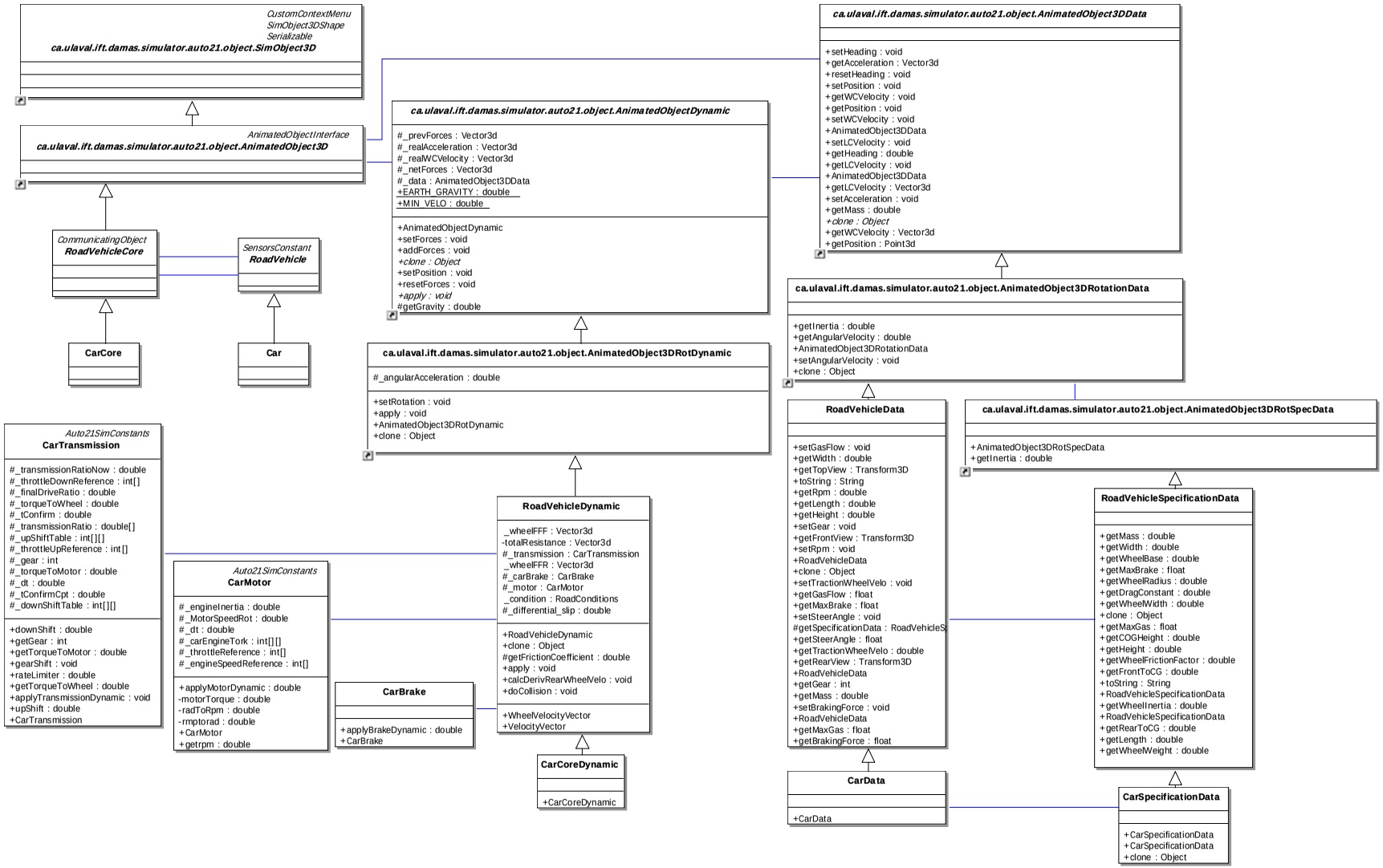

Figure 3.7: Class diagram based on the auto21.object.vehicle package, which represents the vehicle objects and their dynamics simulation classes.

machine and if the code that defines the dynamics must be optimized, this model allows us to make the modifications at one place only.

3.4 Sensory System

The sensory system is briefly presented, since the research on this subject is not a major concern at the moment and more details on the recommended types of sensor will be given in Section 4.4.1. Besides, research at the university of Calgary is in charge of determining the type of sensors that should be used inside our vehicle and only following from that, we can start implementing a detailed simulation model of the proposed sensors. For this reason, the following section only presents the specifications of two types of external sensors and ends by presenting the model of implementation of both the external and internal vehicle sensors in the simulator.

3.4.1 Sensors Specifications

The external sensors that are currently being simulated inside the HESTIA simulator are a front laser and a Global Positioning System (GPS). The laser is mounted on the vehicle's front bumper and it has a range of $100m$. This sensor outputs the distance with the front vehicle, with an accuracy of $\pm 5\%$ of the distance and the difference of velocity with the same vehicle with an accuracy of ± 0.1 km/h. Its update rate is 10 Hz and it uses a very narrow beam with an angular coverage of about $\pm 1^{\circ}$ on the azimuth and elevation plans. At the moment, we do not consider the weather conditions that greatly affect the behavior of this type of sensor (mostly for the laser) and our model needs to be enhanced to simulate a more realistic accuracy. Our simulated laser also outputs the delta acceleration value, by applying a differential on the delta velocity, already calculated by the laser.

The GPS we are simulating is somehow "unrealistic" for the currently available technology. Its update rate is 4 Hz and its accuracy is ± 1 m, which is much better than the update rates of 1 Hz and accuracy of ± 3 m of most GPS. To explain these results, we could consider that our GPS is in fact a Differential Global Positioning System (DGPS) [Wolfe et al., 2000], which compares the information of multiple GPS to increases its accuracy. However, this inconsistence should be fixed following the project's advances.

Finally, the internal sensors that are simulated inside the HESTIA simulator are only used to retrieve internal vehicle information, without adding noises of any kind. The internal sensors that are regrouped in each vehicle include: speedometer, accelerometer, yaw meter, gyroscopes for heading and pitch, sensors on the actuators (steering, gas, brake), etc. Therefore, with our internal sensor simulation model, the speedometer retrieves a perfect velocity value in real-time, and so do all the other internal sensors.

3.4.2 Sensors Software Engineering

The sensory system is presented in Figure 3.3 as part of our simulated vehicle model in Figure, where it uses both the vehicle's 3D shape information and the vehicle's internal data to provide internal sensors information to the driver. The sensory system also uses the 3D environment to provide the information required to simulate the external sensors.

The laser and GPS sensors are simulated inside our simulator by using tools provided by the Java 3D environment. Even though we are only using two types of sensors at the moment, the system was designed in such a way that other sensors, similar to the laser, can be easily added. Following from the recommendations on sensors, we should add other sensory units like: sonar, radar (Doppler, millimeter wave), camera, lidar, etc. The internal sensors we mentioned before are implemented by adding a filtering sensor interface over the available vehicle dynamics data.

Figure 3.8 shows the UML diagram of the sensory system developed inside the simulator. This diagram shows that the RoadVehicle object can include any sensor defining the Sensor class, which is then sub-classed by the internal and external sensors categories. Sensors are added to the vehicle by using the vehicle model definition file loaded at startup. To implement the model of laser that was defined in the section on specifications, we used the FrontLaserSensor class. This class uses JAVA 3D's picking tools to create a 3D laser beam that intersects with the closest vehicle it scans and returns the vehicle's reference and distance. Once it has the reference on the closest vehicle, it can retrieve its dynamics state and filter it according to the laser model. In contrast, other sensor instances (internal sensors), shown in Figure 3.8, simply use the data stored in the vehicle object, which makes them easy to program.

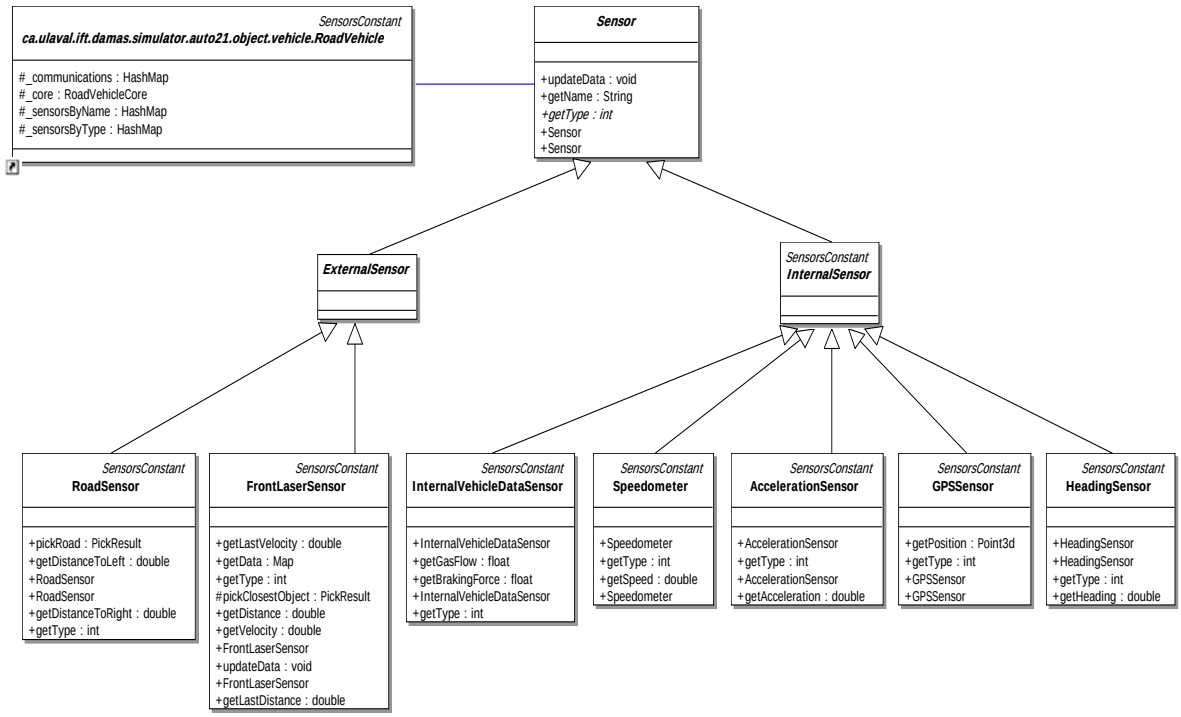

Figure 3.8: UML Class diagram of the simulator's sensors model.

3.5 Inter-Vehicle Communications

The communications modeled inside our simulators are vehicle-to-vehicle communications used to exchange information between moving vehicles. At the moment, the simulated media is a radio transmitter/receiver onboard each vehicle for two ways, point-to-point and multipoint communications. The general communication model includes adjustable delay and throughput which makes it easier to implement different radio frequencies or other communication medias. The presentation of the simulator's communication system begins by giving specifications on the radio simulation model in Section 3.5.1, and it is followed by the description of the general model of implementation inside the Auto21 simulator.

3.5.1 Communication System Specifications

The radio communications simulated at the moment is an Ad-Hoc network infrastructure in which terminals located on vehicles communicate with one another while the vehicles are in motion. The radio devices communicate at an unspecified frequency, at a maximum range of $100m$. The transmission rate used in the current simulations is

approximately 1000 bauds.

The time required to send data from one vehicle to another can be adjusted from the instant a request is made to the transceiver, to the instant another vehicle's receiver retrieves the entire data. This transmission time is calculated by function TT, which calculates the time required to reach the communication receiver of vehicle j, when a message m is sent from the communication transmitter of vehicle i:

$$TT_{ij}(m) = 2 \cdot (F1(\sigma_m) + F2(\sigma_m)) + F3(\delta_{ij}) \tag{3.8}$$

In equation 3.8, function $F1$, $F2$ and $F3$ calculate three different transmission delays that can be adjusted according to the simulated communication devices. The first function ($F1$), represents the time to modify the message's structure, in order to the meet the specifications of the communication protocol used in our system. The second function ($F2$) refers to the transmission time per bite required to send the packets of data. These two functions depend on the message's size in bites σ_m and they are added two times to consider both the encoding and decoding tasks executed by the transmitter and the receiver. The last function ($F3$) represents the transmission time of packets per meter, which depends on the distance between the transmitter and the receiver δ_{ij}.

At last, note that the communication model that we currently simulate could include more details in a future versions. But this model represents a preliminary implementation, since we have not decided the exact communication media that will be used in the Auto21 application. Consequently, the current communication model does not calculate the doppler effects due to the relative motion of vehicles and the communications between vehicles never fails as long as they are in the allowed radio transmission radius.

3.5.2 Communication System Software Engineering

As shown in Figure 3.3, the inter-vehicle communications of our simulator are used by the driving agents to provide the collaboration aspects required in a MAS. In this figure, the inter-vehicle communications use the information about other vehicles in the 3D environment to calculate the message transmission times. To develop the communication infrastructure inside the HESTIA simulator, we decided to use a client-server like model. This way, clients are communication modules that can receive and/or send messages. Such communication modules can be attached to any 3D objects loaded inside our highway environment, which means that a transceiver object can be either mounted on a light post or a moving car. In the same communication model, the server is a communication manager that administers every outgoing messages to dispatch them to the right receivers, at the right time. Hence, we could say that the manager (server)

represents the dynamics of the communication system. This is why the manager is the only class that has to be extended for specific communication technologies (like the radio communications), in order to model their specific dynamics.

Figure 3.9 shows the class diagram of the communication system and its main components. In this diagram, the only class relating to the radio system is the RadioCommunicationManager that extends CommunicationManager, around which all the communication process evolves. When a CommunicationModule implementing the CommunicationSender interface wants to send a message, it contacts the CommunicationManager, which has a list of all the CommunicationListener (receivers) in the environment. At that moment, the manager processes each admissible receiver using "technology specific" methods (in our case, the radio). Different instances of the sent message are generated for every receivers that are within reach and enqueued in a buffer, along with their arrival time. Generic methods in the RadioCommunicationManager class finally send these messages to the appropriate CommunicationListener, at the right time.

In the diagram of Figure 3.9, the SimCommMessage class represents the messages used at the simulator-level. When an agent wants to send a message, it uses the MessageContent class or one of its children and sends it using the VehicleCommunicationModule. Note that a vehicle may contain more than one VehicleCommunicationModule, which are defined in the vehicle model file and available through RoadVehicle, at runtime.

3.6 Driving System Interface

The driver interface is now briefly defined, since it does not represents a very interesting aspect on the vehicle simulation point of view. This interface mainly represents the link between the vehicle's driver, a driving agent in our case, and the simulated vehicle. The driver interface specifies the access rights that our drivers have in order to simulate a realistic environment in which the driver cannot have access or interfere directly with the vehicle dynamics. Therefore, the only assumption we make about the driver is that it has specific read and write rights. Our vehicles' drivers can be the driving agents presented in this thesis or any other entity as a user controlling the vehicle with the keyboard.

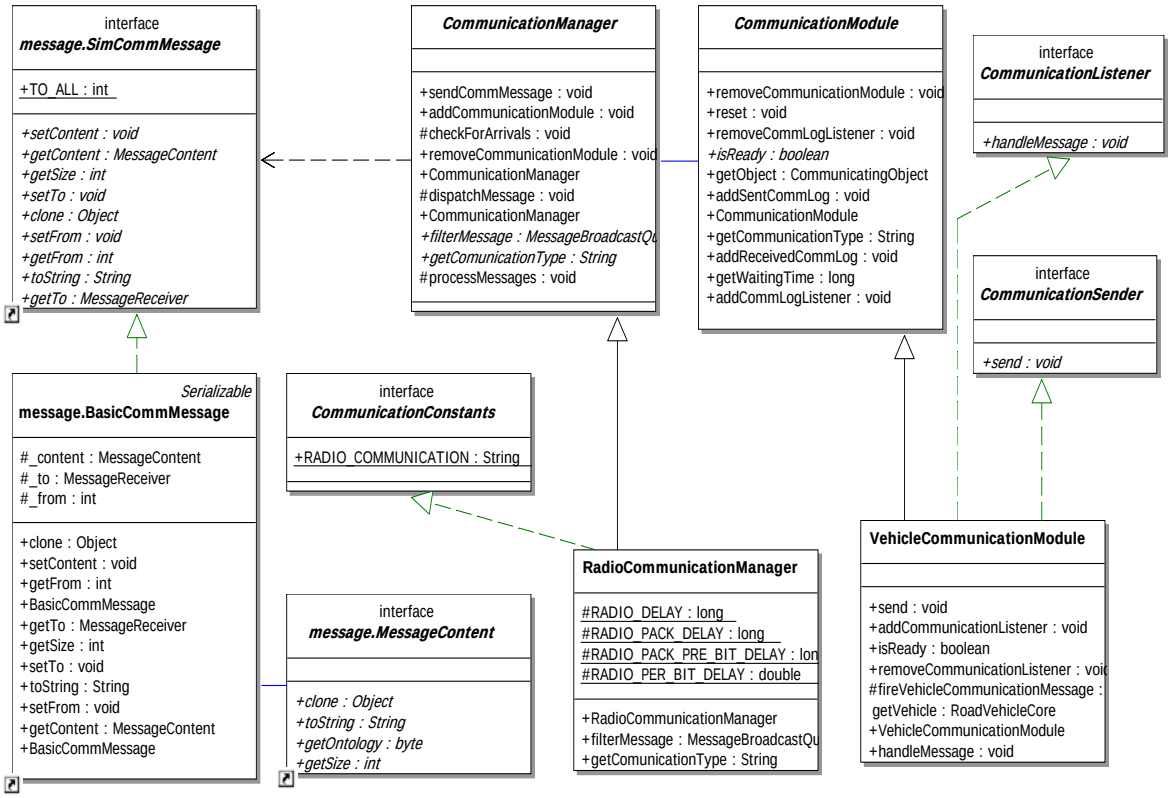

Figure 3.9: UML Class diagram of the simulator's communication model.

3.6.1 Driving System Specifications

The access rights given by the driving system concern read and write permissions about the three main classes it can use (as shown in Figure 3.3): (i)*Communication Module*; (ii)*Sensors*; (iii)*Controller*. The *Controller* can be modified by the driver in order to change its actions on: the gas pedal, the brake pedal, the steering wheel. The *Sensors* use the driver to notify it about the new data it senses, considering a constant refresh time and the driver can use the *Sensors* to get the latest sensed data. Finally, the *Communication Module* uses the driver to notify it of the arrival of a new message in its buffer, if this module is a receiver. If the module is a transceiver (or both), it can be used by the driver to transmit messages, the same way it has been presented in Section 3.5.

3.6.2 Driving System Software Engineering

Figure 3.3 represents the driving system interface as the group formed by the inter-vehicle communications, the sensory system and the vehicle controllers. This interface is in fact the only direct link with the vehicle object, which runs in the simulator's main loop. Therefore, it enables a driver, as a driving agent, to act or receive information from the vehicle in an asynchronous way, without having to wait for its turn in the simulator's running loop.

The driving system has been developed in accordance with the class diagram shown in Figure 3.10. The basic interface of Driver is implemented by different levels of driver, until it reaches drivers relating to specific types of agents, developed with the JACK agent language (at the bottom). Within this driver hierarchy, automated drivers based on JACK agents extend the JackAuto21Driver class, while automated drivers with non-JACK agent drivers extend Auto21Driver class and non-automated drivers, controlled by a joystick or the keyboard, can simply implement the Driver interface. The RoadVehicleCore class uses the Driver interface to get the changes on its actuator and modify the dynamics at every simulation time step. The access rights given to the drivers are controlled by the vehicle public interface, which is represented by the RoadVehicle class. The RoadVehicle class encompass the three major devices used by the driver, which were presented in Figure 3.3: (i) the *Communication Module*; (ii) the *Sensors*; (iii) the *Controller*.

Another benefit that should be mentioned about the driving system interface is that it controls the interactions between the agent thread and the simulator's engine. Since

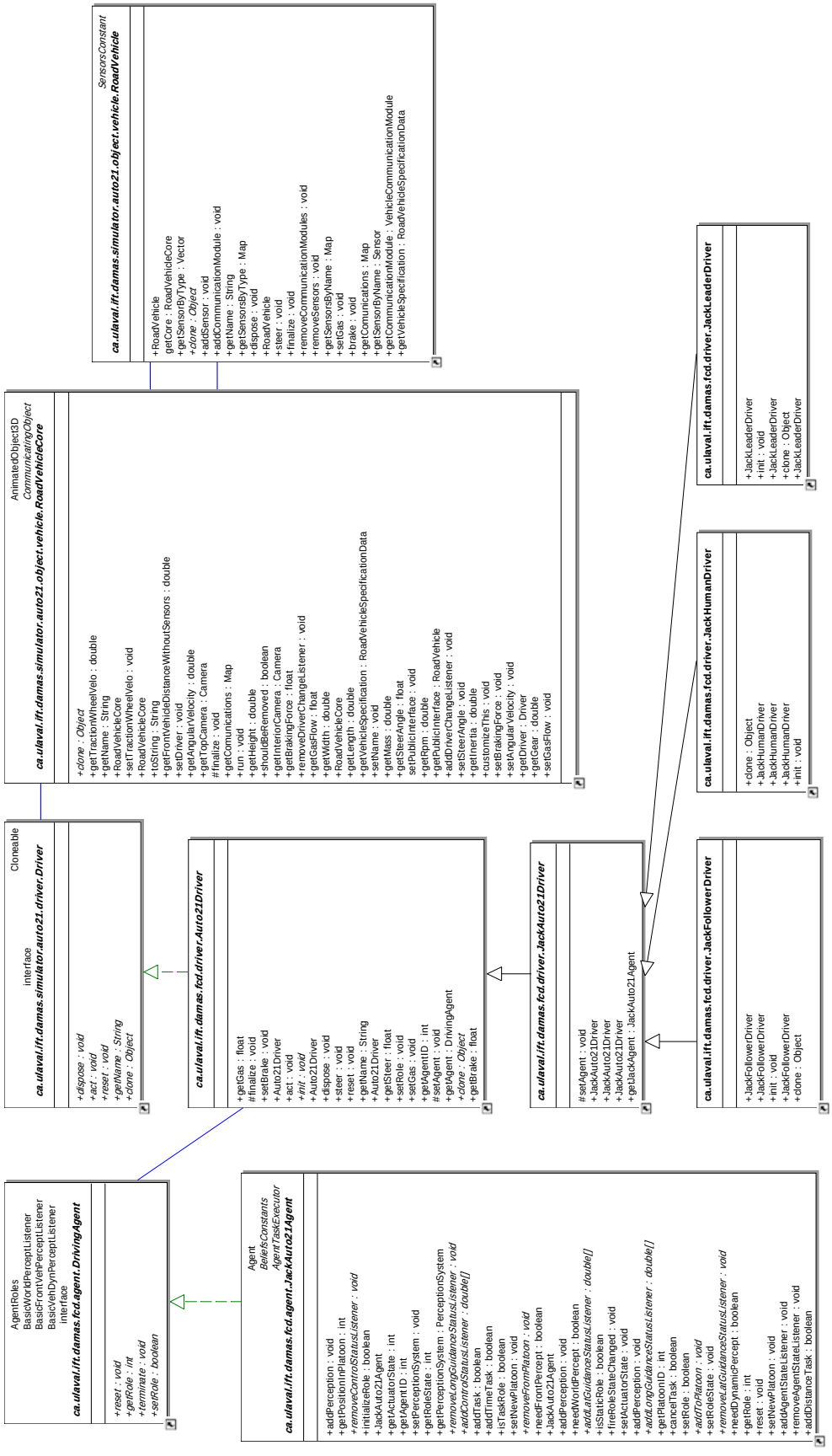

Figure 3.10: Class diagram of the Auto21 driver infrastructure for agents.

these two threads run in parallel, the driver interface allows the agents to modify data in an asynchronous way, even though this data is inside the simulator's running loop. This way, the driving agent can set a new value of gas throttle asynchronously and this new value will be queried by the simulator's engine at the next time step, without interfering with the agent's reasoning.

3.7 Collaborative Driving Scenarios

The main purpose of the HESTIA simulator is to test and improve the different models of Collaborative Driving System (CDS) that should be developed as part of the Auto21 project. To realize this task, the simulator must provide tools to automate the creation of test cases or scenarios and monitor the reaction of our system. Such tools have been developed as part of our simulator and they are described below, by referring to the *Scenario Manager* and the *Log Manager*. The *Scenario Manager* launches collaborative driving scenarios and manages each scenarios' tasks according to events specified in a scenario file. The *Log Manager* monitors the information about the state of the simulation through time and writes the details of each state to a log file.

3.7.1 Driving Scenarios Specifications

The different possible test cases we provide as part of our *Scenario Manager* are almost infinite. Currently, the *Scenario Manager* supports the automatic activation of the following agent tasks: (i) merge; (ii) split; (iii) follow. Moreover, the following driving tasks can also be automatically generated: (i) changing lane; (ii) driving at a specific velocity; (iii) driving at a specific acceleration; and (iv) maintaining a specific time or distance gap with the preceding vehicle. In order to automatically activate these tasks, the *Scenario Manager* offers to its users the possibility to specify the *pre-* and *post-conditions* that trigger the tasks. The current possible conditions (*pre* or *post*) are: a vehicle's specific velocity, acceleration, position (x or y), or distance with preceding vehicle; the scenario time; and a driving agent's role with its role's state.

Using the supported task definitions along with any combination of conditions, a vast variety of tests cases can be generated inside a scenario. However, the simulation results presented in this thesis mainly use the merge and split agent tasks, as it will be shown in Chapter 6.

3.7.2 Driving Scenarios Engineering

The two main components that compose the collaborative driving scenarios are shown inside the simulator software model in Figure 3.3. The *Scenario Manager* monitors the information on the state of the simulated vehicles and the 3D environment to trigger tasks that are executed by the driving agent. On the other hand, *Log Manager* monitors the *Scenario Manager*, the state of the simulated vehicles and the state of the vehicles' drivers to write this information to a log file.

The general model used to develop the scenarios and logs of our simulator is presented in Figure 3.11. This model shows the relationship between the XML scenario description files, creating a series of events in the simulation, and the simulation observers, recording logs to files throughout the simulation. A *Batch Scenario Manager* is used to automatically run multiple scenarios, which is very useful to support reinforcement learning functions for our Collaborative Driving System (CDS). Each scenario is managed by the *Scenario Manager*, which creates a series of threads called *Scenario Events*. These events represent our test cases and they monitor the simulation to trigger a task considering a specific pre-condition and end the same task on a specific post-condition. The task(s) that can be triggered as part of a *Scenario Event* may concern the driver (agent), the vehicle or the highway environment. At the moment, these tasks are mainly used to modify the driver's behavior, but it can also be used to simulate a breakdown of the car, its sensors or its communication system, or animate different objects or humans on the road.

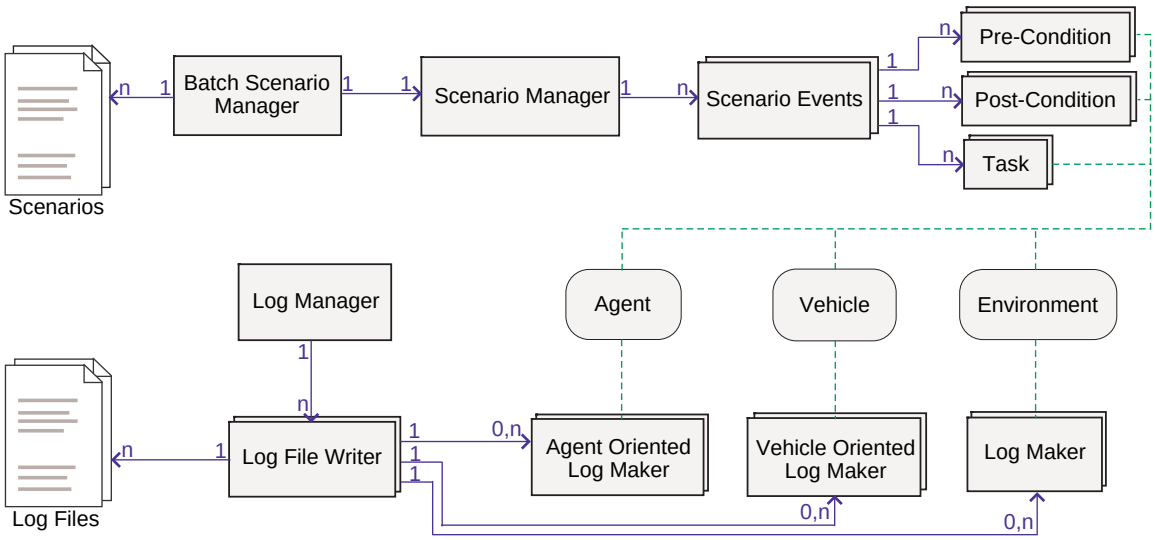

Figure 3.11: Abstracted model of the driving scenarios and log creation systems.

Every time a simulation runs in the HESTIA simulator, our *Log Manager* can record

the information about the drivers' actions, the vehicles dynamics data, communications, sensed data, or other events. The log system is loaded at startup, from XML files relating to the definition of different *Log File Writers*. Each *Log File Writer* includes one or many log makers, as shown in Figure 3.11, depicting log makers for the driver (agent), the vehicle and the environment. At their creation, log makers monitor a part of the simulation using events or a timer. The *Log File Writer*'s thread queries them when processing time is available and outputs the merged data from each log maker to a temporary file. All the temporary files are finally brought together inside an Excel workbook, by the *Log Manager*, at the end of the scenario.

The overall system enables us to run a series of simulations for a day, a week or more and gather the results to either analyze the reaction of our CDS to different events or learn from one scenario to another.

3.8 Summary

The simulator that has been described in this chapter regroups all the basic components required to simulate either a simple ACC or a complex Collaborative Driving System (CDS). The vehicles dynamics are detailed at a level that presents realistic behaviors and their code has been optimized to enable a large amount of vehicles to run in a same simulation in real-time. The sensors, communications and driving systems present all the requirements of a standard automated driving system. The scenario creation system enables the test the CDS in all sorts of situations, while the log system is very useful to analyze and learn from the behaviors of our drivers.

Obviously, this simulator can still benefit from further enhancement, depending on the priorities of the future phases of the Auto21 project. For instance, many aspects of the GUI could be improved, the 3D environment could be more representative of the targeted highways, the vehicles' technological components' simulation could also be improved, and useful tools could be added to the simulator. To this end, the different simulator enhancements are now listed as suggestions:

- The interface could be more user friendly. This way, our simulator could be used by people who are not working on the project, while facilitating our own interactions with the simulator.
- We could benefit a lot more from the Java 3D API, by adding different effects like vehicles' sounds, wheels that actually rotate and turn, and by supporting a

joystick for manual driving. In addition, the vehicles' 3D models could relate to a specific type of car and the highway could be detailed a little more.

- The current road is straight, but it could include different highway exit and entry points, and include changing road and weather conditions.
- Tools to support reinforcement learning could be developed or included as an infrastructure, part of our simulator. This would enable its users to develop learning functions on different aspects of our driving system.
- New sensors and communication system could be modeled, considering the recommendations of industrial partners in Auto21.
- The models of human drivers could be included inside the simulated scenarios, in order to increase traffic and study different aspects of collaborative driving.
- Different modules of the simulator could be distributed on many computers in order to increase the processing power.
- If we decide to include a greater amount of vehicles, in order to increase traffic, the dynamics simulation model of these vehicles could be calculated using discretized time, which would greatly improve their processing time.

Chapter 4

Auto21 Driving Agent Architecture

The development of complex real-time systems like the Collaborative Driving System (CDS) requires a robust architecture that supports problem specific requirements and possesses a series of attributes that most real-time control systems require. This chapter aims to detail the architecture defined for the Auto21 project and explain its development process and objectives. To this end, the chapter begins with Section 4.1, by presenting different automated driving systems currently used as part of similar projects. Then, Section 4.2 describes the architecture we developed (the Auto21 architecture [Auto21, 2004]), by presenting real-time architectures' requirements, and giving details on each divisions of this architecture. This is followed by the Section 4.3, presenting the software engineering of our architecture, while Section 4.4 concludes this chapter by presenting the techniques of implementation of such architecture.

4.1 Automated Driving Systems

Before describing the Auto21 driving architecture, we first specify the requirements of such an architecture in terms of "autonomous driving". First, it should be specified that our Collaborative Driving System (CDS) aims to realize "driving manoeuvres" to which we refer to, when describing the coordinated driving tasks including: (i) platoon following; (ii) platoon splitting (leaving); (iii) platoon merging (entering). Moreover, we also refer to the term "driving actions", when describing automated driving tasks like: (i) changing lane; (ii) maintaining a velocity; (iii) maintain an inter-vehicle distance; etc. The reference to driving manoeuvres and actions will be kept accordingly, throughout this chapter and the following ones.

The requirements of the Auto21 architecture can be detailed with the following list of functionalities. These functionalities often relate to other autonomous driving systems, which are described afterwards.

Surrounding objects (vehicles) sensing: One or many functions that use different sensors to determine the relative position of surrounding vehicles. These functions should be based on data fusion algorithms, which have the ability to give a precise representation of the state of neighbor vehicles.

Vehicle positioning: A function that determines the global position of the vehicle in real-time.

Inter-vehicle communication management: A set of functions using the vehicle's communication devices to send and receive messages according to a predetermined protocol.

Driving manoeuvres coordination: Different functions that use the communication system to coordinate driving actions taking part in a predefined plan structure. These functions handle all the manoeuvres that necessitate more than one vehicle.

Traffic management: Functions that use the vehicle's communication devices to receive recommendations on driving manoeuvres to execute, in order to improve the overall traffic.

Driving action planning: Functions that coordinate the actions of a single vehicle. As opposed to the "driving manoeuvres coordination", these functions are used for a manoeuvre or a higher-level driving action that only involves the vehicle being driven. In this case, a driving action is the action of modifying the vehicle's actuators: steering wheel, gas and brake pedals.

Adaptive cruise control: A function that maintains a safe distance with the preceding vehicle. This function uses a sensor or a vehicle state, communicated by the vehicle being followed.

Real-time emergency reactions: Functions that react to predefined problematic situations or dangerous states, often learned over time.

Note that the previous functionalities represent the complete requirements of a fully autonomous system, which is not an immediate objective for the upcoming years. However, our architecture should support all those functionalities, even though only a part of them are actually implemented within the first phase of the project, depicted in this thesis.

4.1.1 Communicative Control

Communicative control technologies, known as Cooperative Adaptive Cruise Control (CACC), are presented in this section because they implement some of the basic functionalities required for our architecture. In fact, most of the CACC, depending on their purpose and the sensors they use, provide the following functionalities: *Surrounding objects sensing*, *Vehicle positioning*, *Adaptive cruise control*, *Real-time emergency reactions*. A CACC is based on a simpler control technology called Adaptive Cruise Control (ACC) [Winner et al., 1996], which controls only the vehicle's gas and brake throttle, to maintain a safe distance with the front vehicle. The ACC operates together with the manually driven vehicle and has minimal sensors requirements: a simple laser or sonar. It has the benefit of moderating the vehicle's acceleration, to enhance comfort during traffic disturbance situations [Ioannou and Stefanovic, 2003].

The concept of the CACC is the same as ACC, except that in CACC, the vehicles use a communication system to share information with their neighbors. In the first experiments on CACC, the preceding vehicle continuously transmitted the acceleration and braking capacity information to the follower via point-to-point vehicle-vehicle communication [VanderWerf et al., 2001]. This simple model gave promising initial results, although the inter-vehicle communication policy had to be improved. Consequently, other models followed the simple model of CACC and some of them focused on the interaction between the human-driven vehicles and the controlled vehicles. Hedrick et al. [2003] simulated CACC equipped vehicles merging the lane of a highway formed of both manually driven and partially automated (ACC/CACC) vehicles and tested different acceleration/deceleration scenarios. Compared with the original implementation of the CACC, the model of Hedrick et al. [2003] used event-driven communication, in which a vehicle may communicate before changing lane, entering the highway or when it senses a high deceleration. This resulted in vehicles reacting with softer acceleration/deceleration when a vehicle was merging a lane, because of the longer reaction time provided by the vehicle-vehicle communications.

In a similar project, Rajamani and Zhu [2002] proposed a technology called Semi-Autonomous Adaptive Cruise Control (SAACC), which was presented as a platoon of vehicles that could be deployed in a near future. The priority for this application was to ensure string stability of the vehicles that are closely following each others, such that vehicle-to-vehicle spacing error does not grow toward the end of the platoon. The model of controller they proposed includes two hierarchical layers, where the upper-level determines the desired acceleration to ensure the stability of the string of vehicles, and the lower-level commands the throttle and brake required for this acceleration. The upper-level control function uses communication from the preceding vehicle to adjust its

acceleration, which eliminates the need for a centralized coordination station, required in most platoon models. The vehicles used in the SAACC model were ACC equipped vehicles with a radio receiver at the front and a transceiver at the back. This way, when the vehicle preceding the SAACC equipped vehicle can communicate, both vehicles can form a tight platoon. And when the preceding vehicle cannot communicate, the SAACC can still be used as a simple ACC. As their first results, Rajamani and Zhu [2002] simulated manoeuvres at string unstable frequency where the leader was disturbing following vehicles. They showed that this controller was able to reduce the disturbances and keep the string stable.

Other CACC systems have been proposed as control functions that relied on their communication system at the point they could eliminate the need for sonar, laser or other sensors [de Bruin et al., 2004]. In this case, vehicles are equipped with a Global Positioning System (GPS) and a communication system, used to share information on the vehicles' position with other vehicles inside a Wireless Local Area Network (WLAN). The longitudinal control function, used in this case, includes the estimated distance from the controlled vehicle's position to the front vehicle's communicated position. For their part, de Bruin et al. [2004] developed a CACC system that used a mailbox mechanism to manage the inter-vehicle communications. Their system includes: (i) the concept of a world model used by the mailbox to compare the information it receives; (ii) the verification of messages' consistency; (iii) the ability to forward messages when necessary. Due to the generic implementation of the world model, their system offers the advantage of being separated from the applications' logical relations, which makes this model of CACC easier to maintain.

4.1.2 Collaborative Driving Systems

A Collaborative Driving System (CDS) usually tries to form a platoon of vehicles, which is very close to the previous CACC, although following distances are shorter and the coordination techniques it uses are more elaborated. Car platoons aim to maintain very close following spaces between the vehicles to increase highway capacity, while in a standard ACC, the main objective is to maintain a safe distance to relieve the driver from spacing adjustments. The platoon's goal is achieved using an inter-vehicle coordination system, which is the core of the CDS, as this section shows it, with different examples.

The first objective of the CDS is to maintain the string of vehicles stable inside the platoon. This issue is even more important in the case a CDS than for simpler models like the SAACC, because of the following reasons:

1. The platoon communication infrastructure makes it easier to support the coordination techniques required to maintain this stability.

2. If the string does not maintain its stability, vehicles are not able to follow each others at closer distances (which is the purpose of the platoon formation).

3. If the string does not maintain its stability, we cannot ensure a global safety when vehicles enter and leave the platoon.

The second and third reasons summarize the tasks that the CDS has to perform, in order to keep the platoon stable, while the first reason highlights the fact that the two previous tasks are more easily achievable within the platoon communicative infrastructure. This infrastructure may be based on road-to-vehicle communications and/or vehicle-to-vehicle communications, similar to the WLAN. The projects described below, taken from three different countries, present different CDS projects making different use of their communication infrastructure.

The American CDS project

The PATH project, at the University of Berkeley in California represents a very well known implementation of a CDS on real vehicles [PATH, 2004]. In this project, platoons of vehicles use both lateral and longitudinal controllers to follow each others through a road equipped with magnetic nails. The automated vehicles developed by the PATH project are coordinated by both vehicle-to-vehicle and road-to-vehicle communications, using a road-side infrastructure for traffic management purpose. The general architecture proposed by the PATH project was initially used for the platoon of cars on Automated Highway System (AHS) [Hedrick et al., 1994], but it has further been reused for platoons of trucks, buses and even Mobile Offshore Base (MOB) (oil rigs) [Howell et al., 2004].

PATH's general architecture contains three hierarchical layers that are presented in Figure 4.1. The *regulation* layer is the lowest layer of control in which control laws are given as vehicle states or observation feedback policies to control the vehicle dynamics. The *coordination* layer contains the control and observation subsystems responsible for safe execution of the basic manoeuvres such as manual control, speed regulation or distance tracking. The *supervisory* layer contains the control and observation strategies that may trigger the coordination of manoeuvres through inter-vehicle protocols. Howell et al. [2004] proposed a decentralized coordination model to implement this layer, but the details on its protocols management was not given. Hence, the most accurate coordination model proposed inside the PATH project, and probably the most

interesting and complete one, is the model proposed by Bana [2001], which is presented in Section 4.4.5.

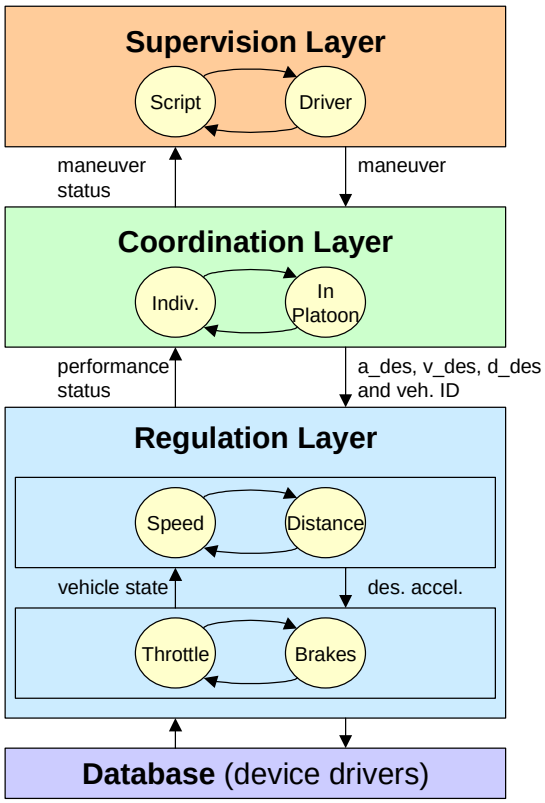

Figure 4.1: Architecture used for the PATH project in Howell et al. [2004].

The Japanese CDS project

For their part, Tsugawa et al. [2001] developed a platoon architecture similar to PATH's CDS, expect that they did not use a road side infrastructure to coordinate their platoons. Their vehicles use a GPS and a laser sensor to follow each others in the platoon. They developed a WLAN (communication network formed by neighboring vehicles) model based on a token-ring [Sakaguchi et al., 2000], which is used to communicate each vehicle's dynamic state inside a neighborhood.

The Japanese project takes advantage of their communication system to support the implementation of Differential Global Positioning System (DGPS) [Wolfe et al., 2000], which increases the GPS accuracy by comparing and merging all the vehicles' GPS data. Other vehicles' position information is also used to maintain relative distances and to coordinate manoeuvres like a vehicle splitting or merging from the platoon. In their coordination model, vehicles that request the execution of a manoeuvre must send

a message through the network and each vehicle has to accept the proposition for the manoeuvre to take place. This communication model ensures a certain degree of safety, but it does not try to decrease the amount of communicated data nor does it addresses the problem of large neighborhoods of vehicles.

The French CDS project

The French project ARCOS [Blosseville et al., 2003] aims to develop a communication infrastructure for vehicles equipped with ACC. Their inter-vehicle communication system enables vehicles to share their dynamic state and different sensed data. In the ARCOS project the collaboration among vehicles is done in order to provide the following functions: (i) notifying hazardous events to vehicles; (ii) regulating headways; (iii) anticipating collisions; (iv) preventing road departure. Another similar European project called CarTALK 2000 [Morsink et al., 2002] is focusing on driver assistance systems based on inter-vehicle communication. The main objectives of this project are the development of cooperative driver assistance systems and the development of a self-organizing ad-hoc radio network that will enhance the functions offered by current vehicles' ACC.

4.2 Hierarchical Representation

The architecture that has been specifically developed for the Auto21 project provides the functionalities supporting the CDS in similar way to the American and Japanese architectures. This architecture is based on previous real-time hierarchical agent-oriented architectures and it uses the CACC technologies, presented earlier, as its lower layer. During the modelling phase of our architecture, many factors have been considered by analyzing similar architectures, as presented in [Hallé et al., 2003]. In this thesis, we retained the following characteristics for the Auto21 architecture:

Flexibility: The architecture should have a flexible structure so that components can be easily changed or added during the development process or at a later time.

Modularity: The architecture should be based on a modular structure. A complex system, such as the one used for collaborative driving, should be divided into smaller sub-components which can be individually designed, implemented and tested.

Expendability: The architecture should be easily expandable. The system must be developed incrementally, and the additional components should not interfere with the previously tested components.

Real-Time Monitoring and Control: The architecture should be able to monitor events and react in real-time. Apart from the deliberative system, a reactive part of the architecture must generate control signals in deterministic time, under any circumstances. This ensures that the control system reacts at all time and during any situation.

Reliability and Robustness: The architecture must be able to execute planned sequences of actions under conditions of uncertainty. This guarantees that the system can rapidly responds to unexpected events such as component failures. It also provides a safe infrastructure to manage concurrent real-time activities.

The resulting architecture, is the one used to model the driving agents of the Auto21 project. Throughout this thesis we refer to the Auto21 architecture by using its global model, presented in Figure 4.2. This figure represents the hierarchical organization of layers, sub-layers and modules that support the CDS functionalities enumerated in Section 4.1, as well as the architecture characteristics that were just mentioned.

The decomposition of the different modules of our architecture, relating to specific functions, has been done considering the Auto21 Collaborative Driving System (CDS) project's decomposition. Such a decomposition provides the flexibility, modularity and expendability required for our architecture. This way, our partner from the University of Calgary, being specialized in geomatics, has been assigned the *Intelligent Sensing* sub-layer. Our second partner, the robotic laboratory at the University of Sherbrooke, is focusing on the *Vehicle Control* sub-layer. Sherbrooke also works closely with Calgary, on the *Intelligent Sensing*, to provide sensing requirements. Our partners at Sherbrooke have also been assigned the development of a general cooperation strategy, so they are collaborating with us, to develop the *Management* layer.

The technologies on adaptive controllers presented in Section 4.1.1 are similar to the base of our architecture, which includes the *Vehicle Internal Perception* module and *Vehicle Control* sub-layer. The *Vehicle Control* ensures real-time monitoring and control using a lower-level controller that is directly linked to the lower level perceptions. A higher-level of sensing and control allows the *Guidance* layer to provide modular guidance functionalities which makes the planning process a lot easier. Such plans are generated as part of the *Management* layer. Along with the *Vehicle Control*, the *Planning* sub-layer ensures reliability and robustness required for this CDS, at both the reactive and long term levels. Hence, plans are generated and used in a similar way to the CACC, which supports following, merging and splitting manoeuvres.

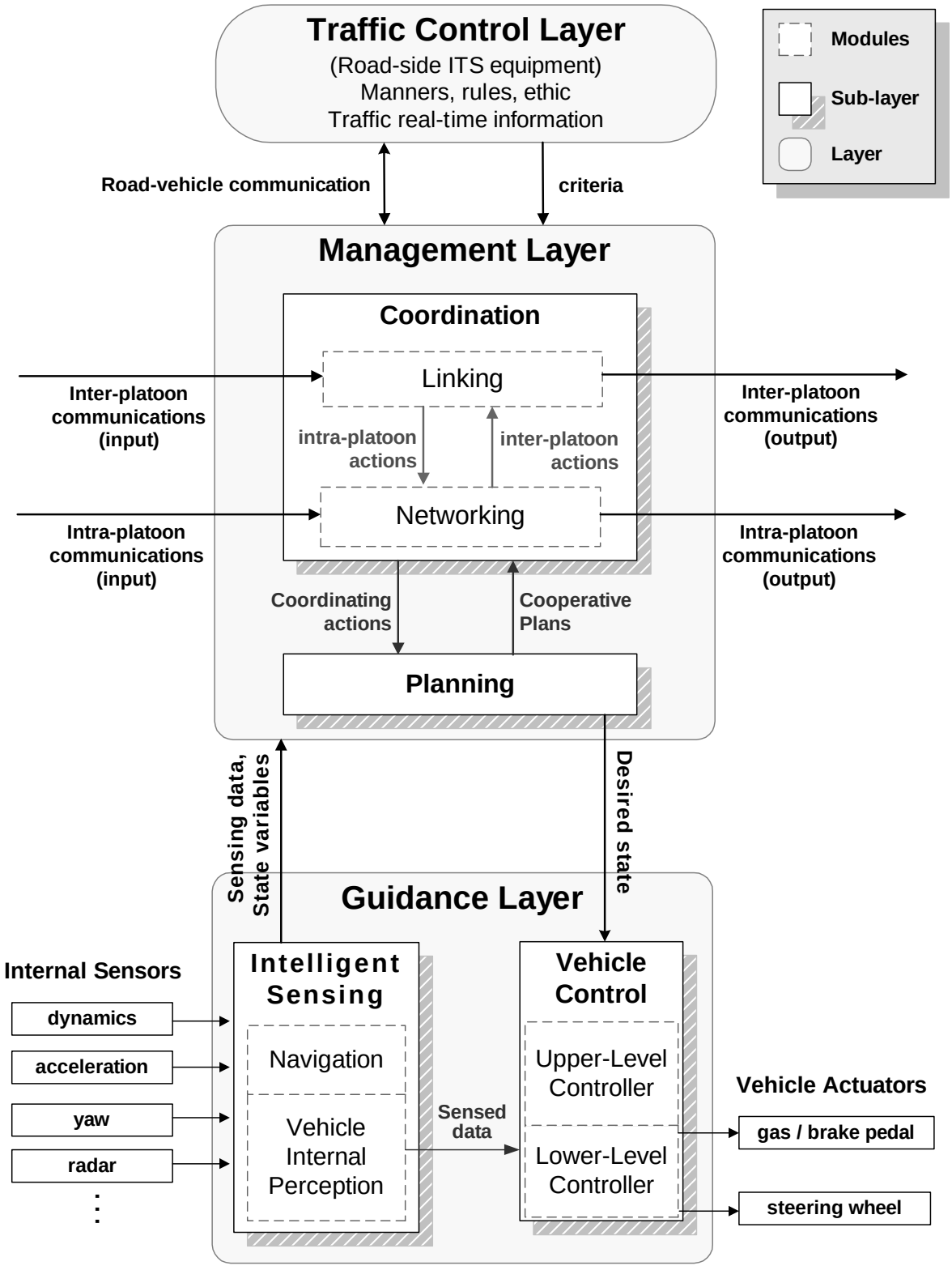

Figure 4.2: Auto21 hierarchical agent architecture.

The model we used for the *Guidance* layer is similar to the architecture presented by Tsugawa and the PATH project. Since the coordination level is an important aspect of our research project, it will be detailed in Chapter 5. However, aspects as the platoon string stability or traffic management are not elaborated in the description of our architecture. As part of the *Coordination* sub-layer, we included a *Linking* module, which collaborates with the *Networking* module to ensure that our architecture is expandable. This expendability supports the coordination of many platoons in collaboration with a *Traffic Control* layer, if this is necessary. For a better understanding, each layer is detailed below starting from the bottom *Guidance* layer in Section 4.2.1, following with the middle *Management* layer in Section 4.2.2 and ending with an overview of the *Traffic Control* in Section 4.2.3.

4.2.1 Guidance Layer

The functions of the *Guidance* layer consist of sensing changing states around the vehicle and activating the longitudinal actuators (gas/brake pedal) or lateral actuators (steering wheel). The *Guidance* layer's function within our architecture includes outputting sensed data and vehicles' state variables to the vehicle *Management* layer and receiving steering and vehicle velocity commands from the same *Management* layer. These considerations have led us to divide the guidance functionalities in *Intelligent Sensing* and *Vehicle Control* sub-layers, as depicted in Figure 4.2. Below, we propose different ways to implement the *Guidance* layer within its sub-components, although the methodology that will be used to respect the components' requirements can be changed through time.

Intelligent Sensing Sub-Layer

The *Intelligent Sensing* sub-layer acts as the entry point of our control system. It transmits raw information about the state inside and outside the vehicle for fast responses. It also sends filtered data to a deliberative system (*Management* layer), for a longer and more accurate reasoning. To achieve these tasks, our sensing system requires different sensors capable of working precisely in different types of environment, along with data fusion functions that create a global representation of the environment. In addition to the external sensors, we use internal vehicle dynamics sensors like: speedometer, accelerometer, yaw meter. This enables the *Intelligent Sensing* sub-layer to retrieve information about the driven vehicle's state. For a better understanding, the *Intelligent Sensing* sub-layer's internal organization is presented below, while ref-

erences to research advances concerning the sensing problem inside the Auto21 project are described in Section 4.4.1.

Inside our application, sensors are grouped in accordance with the percept types to which they relate. These groups are presented in Figure 4.3, which shows sensors on the left and the sensors' respective category to the right. Possible categories of sensed data include: (i) the *front target state*, (ii) the vehicle's *chassis dynamic state*, (iii) the *engine state*, etc. The *Navigation* module (part of the *Intelligent Sensing* sub-layer) is formed by data fusion and filtering algorithms. These algorithms generate higher-level information on the platoon and vehicle states. On the other hand, the *Vehicle Internal Perception* module produces lower-level data that can provide a quicker response time.

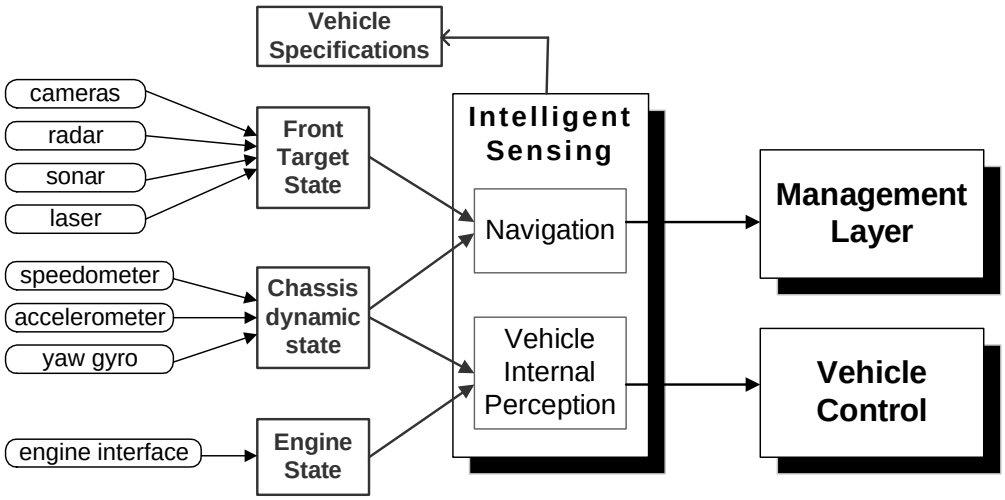

Figure 4.3: *Intelligent Sensing* sub-layer: detail.

At the moment, we consider that the *Intelligent Sensing* sub-layer should output the following information to the *Management* layer and/or the *Vehicle Control* with an unspecified precision:

- The vehicle's global position in three dimensions.
- The vehicle's front object's (vehicle) distance and difference of velocity at a range smaller than $100m$.
- The vehicle's velocity and acceleration.
- The vehicle's heading and yaw rate.
- The traction wheels' angular velocity.
- The engine's revolution and the transmission's gear number.

- The vehicle's current brake and gas percentage.
- The vehicle's current steering angle.

Vehicle Control Sub-Layer

The *Vehicle Control* sub-layer directly acts on the vehicle's actuators (pedals and steering wheel) and receives information from the *Intelligent Sensing* sub-layer and orders from the *Management* layer, as shown in Figure 4.2. To address the requests on desired vehicle states from the *Management* layer, the *Vehicle Control* sub-layer has been divided in two parts: *Upper-Level Controller* and *Lower-Level Controller*. This division is presented in Figure 4.4, which is a detailed version of the *Vehicle Control* sub-layer. The *Upper-Level Controller* responds to the state requests from the *Management* layer, while the *Lower-Level Controller* acts on the vehicle's actuators. In the case of the *Upper-Level Controller*, a state request refers to an order to bring the vehicle in a specific vehicle dynamics state, which can be described using the following vehicle requests: inter-vehicle spacing, vehicle velocity, global meeting point (described later), lane change, etc. Requests on the vehicle's state may also include a specification on the actuator reaction time, for smoother or sharper responses.

In the model of Figure 4.4, the *Upper-Level Controller* is shown as acting on two possible types of *Lower-Level Controller*: the *Lower-Level Longitudinal Controller* and the *Lower-Level Lateral Controller*. The *Lower-Level Longitudinal Controller* receives request on vehicle acceleration from longitudinal control algorithms (detailed in Section 4.4.3), situated inside the *Upper-Level Controller*. The acceleration requests are applied by the *Lower-Level Longitudinal Controller* by setting the necessary brake or gas throttle percentage. Similarly, the *Lower-Level Lateral Controller* applies steering command on the vehicle's actuator, considering the request on vehicle orientation given by the *Upper-Level Controller*. As Figure 4.4 shows it, the vehicle actuators that are required for control tasks are: (i) a gas throttle actuator; (ii) a brake actuator; and (iii) a steering wheel actuator.

4.2.2 Management Layer

The *Management* layer has the task to determine the movement of each vehicle under the cooperative driving constraints using data from: (a) the *Guidance* layer; (b) vehicles coordination constraints through the inter-vehicle communication; and (c) the *Traffic Control* layer through the road-vehicle communication (refer to Figure 4.2). To

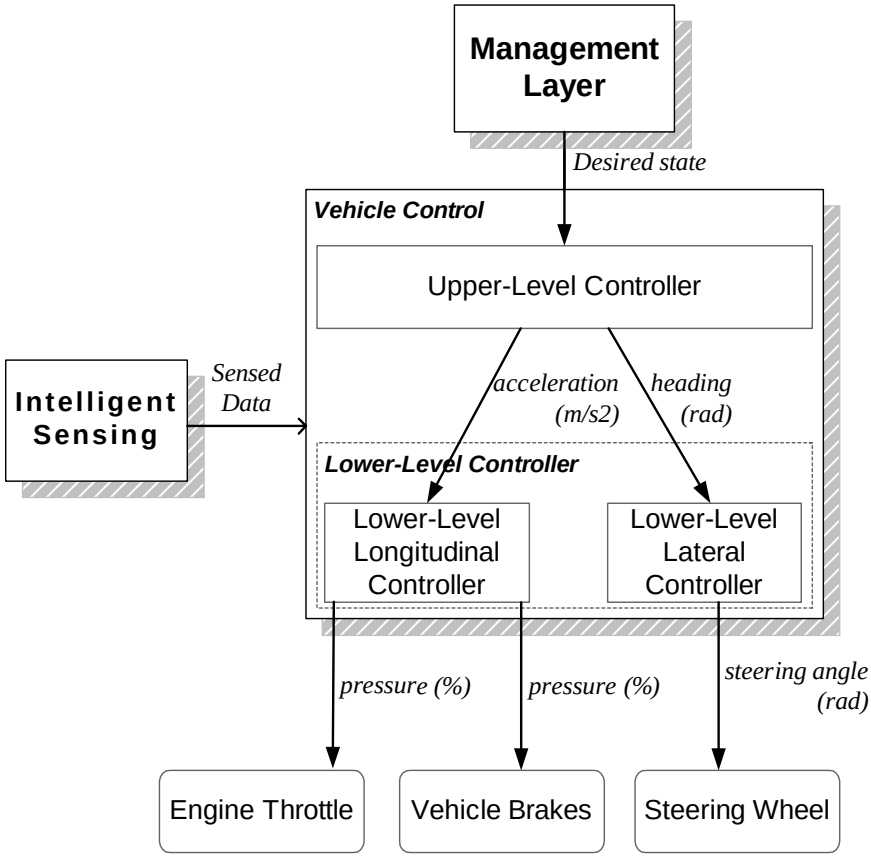

Figure 4.4: *Vehicle Control* sub-layer: detail.

determine the movement of each vehicle under the cooperative constraints, this layer needs to reason on the place of the vehicle in the platoon when this platoon remains unchanged (intra-platoon coordination), and its place in a new platoon when this platoon changes (inter-platoons coordination). The intra-platoon coordination is handled by the *Networking* module and the inter-platoon coordination, by the *Linking* module, together forming the *Coordination* sub-layer. Generally, the task of the *Linking* module consist of communicating with the *Traffic Control* layer to receive suggestions on lane change actions to perform. Once the *Linking* module has chosen an action to perform, the manoeuvres involved in this action (likely splitting or merging a platoon) are coordinated through intra-platoon policies. Each manoeuvre is defined by one or a series of plans that is managed in time using the *Planning* sub-layer.

Planning Sub-Layer

The *Planning* sub-layer has a strong link with the *Coordination* sub-layer since both are part of the same layer (*Management*) and the *Coordination* sub-layer uses the *Planning* sub-layer to create and maintain cooperative plans (refer to Figure 4.2). In a first time, the *Planning* sub-layer is responsible of maintaining a *platoon formation plan* with other vehicles, by using the *Networking* sub-layer. For the *Planning* sub-layer, a platoon formation plan is a set of driving actions in time or according to events, that should be taken to maintain the platoon formation. In a second time, the *Planning* sub-layer uses complex driving plans that request driving actions from the *Vehicle Control* sub-layer inside the *Guidance* layer. Those driving plans constitute the different steps to execute as part of a driving task (manoeuvre), delimited by both a pre- and post-condition.

Coordination Sub-Layer

As shown in Figure 4.2, the *Coordination* sub-layer is divided in intra-platoon needs handled by the *Networking* module, and inter-platoon needs handled by the *Linking* module. The *Linking* module is detailed in Section 5.1 where two models of inter-platoon coordination are presented: centralized and decentralized. Both models use a traffic management function based on vehicles' cost on the highway, which is detailed in Section 4.4.5. To provide the required traffic management functions, the model that implements the *Linking* module can use information from the *Traffic Control* layer, as shown in Figure 4.2. Other models of implementation of the *Linking* module, like the decentralized model, use a communication based on a mobile Ad-Hoc network to replace the *Traffic Control* layer. The mobile Ad-Hoc network allows vehicles to share

traffic information without requiring a road based infrastructure.

The main task of the *Linking* module evolves around the lane change actions. It must determine when a vehicle should do a lane change, by coordinating itself with neighboring vehicles and making the resulting manoeuvre safe to execute. Once the *Linking* module has determined that it can safely execute a lane change, the *Networking* module coordinates the driving actions involved in the manoeuvre.

The *Networking* module being the most important subject of this research, it will be detailed in Section 5.2. Since it handles the intra-platoon coordination, the *Networking* module must coordinate manoeuvres as: (i) following a vehicle; (ii) merging a platoon; and (iii) leaving a platoon. The two latter manoeuvres are directly related to the lane change driving action. In this context, three main coordination models have been defined for the intra-platoon actions: (i) a centralized networking based on the platoon's leader; (ii) a decentralized networking model based on social laws; (iii) a networking system based on teamwork for agents.

To clarify the two different types of coordination involved in the *Coordination* sub-layer, it should be recalled that for the inter-platoon coordination, a centralized model refers to the centralization on a road-side system. And for the intra-platoon coordination, the centralized model refers to the centralization on the platoon's leader

4.2.3 Traffic Control Layer

The *Traffic Control* layer presents the traffic regulation and management systems that optimize traffic by guiding vehicles and platoons. As Figure 4.2 shows it, the *Traffic Control* layer is based on a road-side infrastructure, which provides all the necessary information on the traffic-flow, in real-time. Hence, the *Traffic Control* layer works closely with the different vehicles' *Linking* module, which provides the information on these vehicles' state, gathered to map the traffic-flow. The vehicles' *Linking* module is also being used by the *Traffic Control* layer to suggest optimal driving actions to vehicles, using the observations on specific traffic neighborhoods.

It should be mentioned that our architecture presents the *Traffic Control* layer, only as a suggestion. Therefore, the model that will be used for the *Linking* module will determine whether or not this layer should be developed. This means that in a decentralized model of the *Linking* module, a road-side infrastructure would not be required, but could be replaced by a mobile network based on wireless communications, as the Advanced Intelligent Mobile Entertainment (AIME) proposed by Nortel Networks and

Redknee.com[1]. However, the function or heuristic behind the traffic management system can be defined immediately, as we do it in Section 4.4.6, but the way suggestions on driving actions are communicated to vehicles depends on the model of *Linking* module: centralized or decentralized.

For the moment, we only define the main functions of the *Traffic Control* layer and all other aspects are left aside for its future development. We must keep in mind that the details of implementation of the *Traffic Control* layer will be greatly influenced by the existing inter-vehicle communication infrastructure and possible road-side information systems developed by car builders or transport Canada. As a first draft, the Auto21 architecture, in Figure 4.2, presents this layer as a road-side infrastructure composed of sign boards, traffic signals and road-vehicle communications, as well as a logical part including: social laws, social rules, weather-manners and other ethics, etc.

4.3 Auto21 Architecture Software Engineering

Now that the Auto21 architecture has been detailed, the architecture's software engineering models can be presented. The software models provide a better understanding of our automated driving system's capabilities inside its testing environment (the simulator presented in Chapter 3).

Figure 4.5 represents a global view of the software models referring to the *Guidance* and *Management* layers of Figure 4.2. Each component in this figure refers to the previous architecture, as indicated by the dotted lines surrounding the right component(s). The rounded boxes represent the most important components of our engineering model and names written in parenthesis, at the bottom of these boxes refer to the Java class used to implement this component. For instance, the *Upper-Level Controller* refers to a class called BasicVehicleGuidance in our application, which uses a set of *Guidance Functions* answer to the *Jack Auto21 Agent* class. The latter class represents an *Auto21 Agent*, which is equal to the *Management* layer of our architecture. The *Intelligent Sensing* sub-layer is represented in our application by the PerceptionSystem class, which is provides data on the environment to the two previous components and the *Lower-Level Controller*. This controller is represented in our application by the BasicVehicleControl class, which acts directly on the *Auto21 Driver* interface. This interface represents the link with our simulator's actuators (refer to Section 3.6) for the moment, but it could

[1]More information on the AIME is available as a news release at http://www.nortelnetworks.com/corporate/news/newsreleases/2000b/07_06_0000377_aime_redknee.html

also refer to the vehicle's CAN-Bus[2] in the case of a real vehicle.

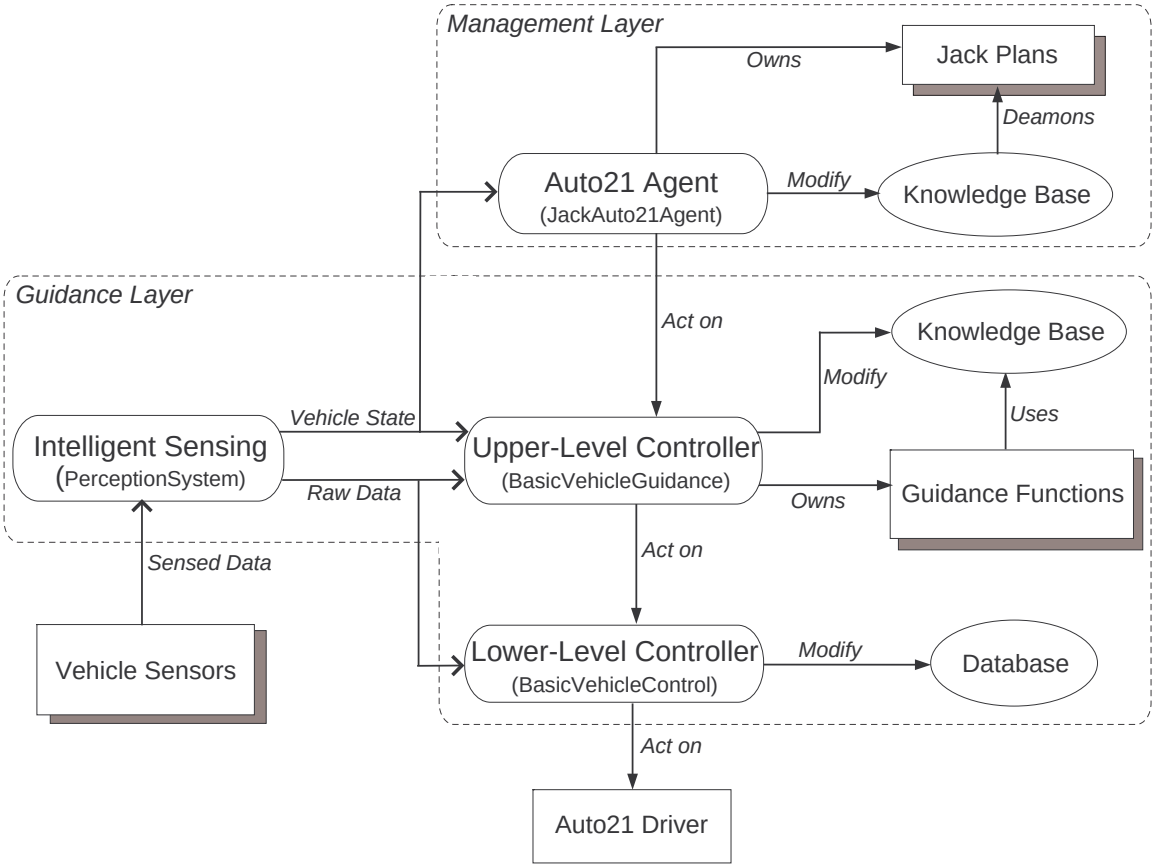

Figure 4.5: A global view at the Auto21 architecture's design model.

For a better understanding of each hierarchical divisions presented in Figure 4.5, a detailed presentation referring to the actual classes developed in our application is given in the next sections. Note that the emphasis is on the *Guidance* Layer, which is detailed within software models of its two sub-layers, since the Management is already detailed in Chapter 6 from an agent-oriented point of view. Therefore, Section 4.3.1 details the PerceptionSystem that refers to the *Intelligent Sensing* sub-layer, while Section 4.3.2 describes the *Vehicle Control* sub-layer. Instead the agent-oriented theory behind the *Management* layer, the final section (Section 4.3.3) describes the general model of implementation of *JACK Plans*, which are the direct users of the *Guidance* layer. Section 4.3.3 gives a better understanding of the relation between the *Guidance* layer's controllers and the *Auto21 Agent*, while the same relation from the "agent's point of view" is detailed in Chapter 6.

[2]The Controller Area Network (CAN) Bus is is a serial communication system used on many motor vehicles to connect individual systems and sensors, as an alternative to conventional multi-wire looms.

4.3.1 Intelligent Sensing Sub-Layer Engineering

Our architecture's *Intelligent Sensing* sub-layer (detailed in Figure 4.3) has been implemented in a main class called *Perception System*. This class receives a direct feed from the vehicle's sensors and dispatches the sensed information to registered listeners in the form of raw data or high-level vehicle states. Figure 4.6 represents this class' functions as part of the links going from the sensors to higher level data structures and finally to different categories of perception listeners. These listeners are the different levels of control defined in our initial architecture, which have specific perceptual needs. Furthermore, the perception listeners' databases (the *Database* and *Knowledge Base* in Figure 4.5) are directly related to the categories of sensed data defined in Figure 4.6. To create such structured sets of filtered data, the perception system is using data fusion algorithms. For the moment, the data fusion algorithms have been implemented in an abstracted way, as part of this shell, and should be refined following from the conclusions of the researches realized by our partners at the University of Calgary.

4.3.2 Vehicle Control Sub-Layer Engineering

The vehicle control classes have previously been presented in Figure 4.5, but they are described in this section by detailing the set of control functions they use. As an elaborate representation, we refer to the actual classes in our application, so the previous *Lower-Level Controller* and *Upper-Level Controller* classes will be called BasicVehicleControl and BasicVehicleGuidance, as it was mentioned at the bottom of their box, in Figure 4.5. To remind the definition of those two concepts, BasicVehicleControl refers to a *Lower-Level Controller*, while BasicVehicleGuidance refers to an *Upper-Level Controller*. As it will be explained, these two classes are abstract classes and they are extended by specialized versions that add methods to respond to specific needs.

For instance, the BasicVehicleGuidance class is a guidance system that is further specialized by two other sub-implementations, as shown in Figure 4.7. Each guidance system uses a set of *Guidance Functions* that represent longitudinal or lateral control algorithms and that must derive from the GuidanceFunction class. The BasicVehicleGuidance can be used for a standard vehicle with limited automation, thus requiring limited sensors and *Guidance Functions*. This is represented by the fact that it only uses two driving functions (velocity and acceleration) and only requires the data from the vehicle dynamics Knowledge Base (VehDynKB). The same reasoning applies to the GlobalVehicleGuidance, which is a guidance system for a vehicle equipped with a GPS that has steering actuation capabilities. The VehicleGuidance class further derives GlobalVehi-

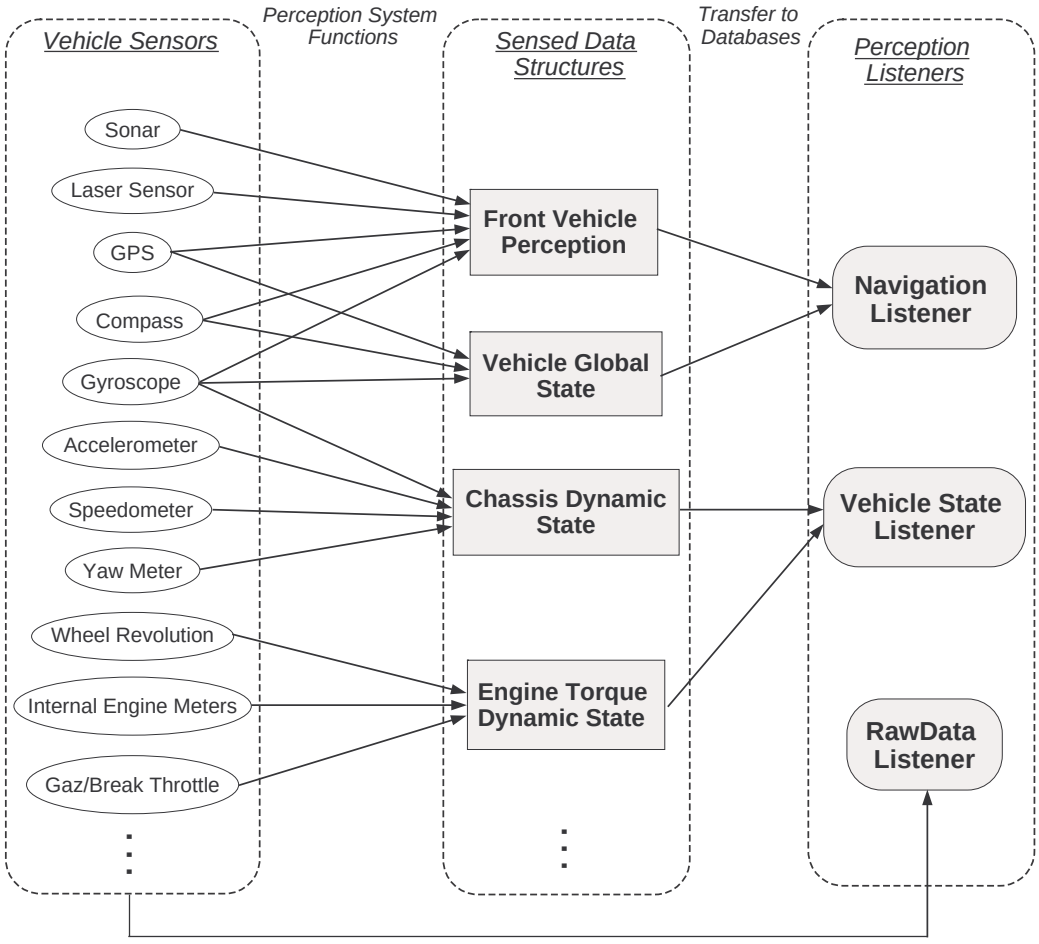

Figure 4.6: Overview of the data structure and listener types in the Auto21 *Intelligent Sensing* sub-layer.

cleGuidance to implement the ACC functions and requires a laser or any type of sensors that provides the data structure required to update the FrontVehKB.

The *Lower-Level Controller* is implemented in a first time by the BasicVehicleControl class. This class is used by the BasicVehicleGuidance to transform orders for vehicle acceleration in brake or gas throttle percentage. The BasicVehicleControl class is further derived by VehicleControl, which is used for vehicles automated with ACC. Since the ACC requires a front vehicle sensing device to provide collision avoidance in situations of emergency braking, we defined a specific class for its purpose (VehicleControl), which makes the BasicVehicleControl more flexible.

A final observation that could be made about the *Upper-Level Controller* is that it is roughly easy to implement new lateral or longitudinal control algorithms in our CDS architecture. These algorithms refer to the group of functions extending the GuidanceFunction class, which are identified as the *Longitudinal Control Algorithms* in Figure 4.7. The GuidanceFunction class already includes all the necessary methods to integrate the *Guidance Function* with the rest of the application, so a new algorithm can be added by writing its code inside a simple method. Therefore, only four simple steps are required to develop a new *Guidance Function*: (i) implement the control algorithm in the guide() method; (ii) give a unique identifier to this function; (iii) specify its percepts needs; (iv) add this new *Guidance Function* class to a class that extends BasicVehicleGuidance. For a better understanding of the control algorithms presented in Figure 4.7, Section 4.4.2 presents some examples of the algorithms used for the Auto21 CDS.

4.3.3 Management Layer Engineering

Inside the Management layer, we previously a Planning sub-layer that generates plans to drive the vehicle during different manoeuvres and a Coordination sub-layer that communicates with other vehicles based on pre-defined coordination protocols. Both of these functions have been implemented in our application by using *JACK Plans*, which are sophisticated action recipes that can be used by a JACK agent (more details in Section 6.2). Since an action can be either to use act on a controller of the *Guidance* layer, call another plan, or communicate with another agent, we managed to develop our architecture's *Management* layer's hierarchy in a *JACK Plan* model.

The general model of Figure 4.8 presents the Auto21 architecture's components on the right and the toolkits we used to implement each component, on the left. The *Networking* module and the *Planning* sub-layer are implemented in our application, by

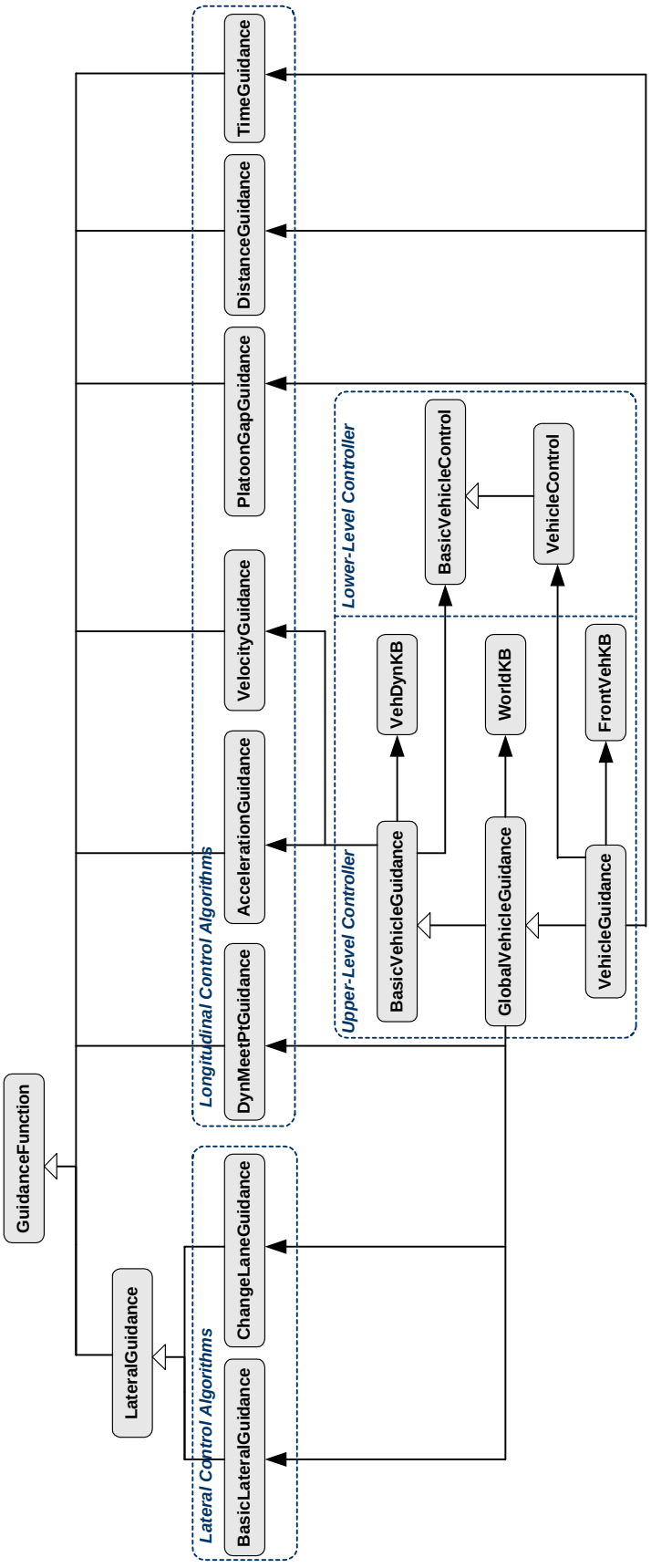

Figure 4.7: Overview of the relation between the different components of the *Vehicle Control* sub-layer.

using a library of JACK plans. The *Upper-Level Controller* is implemented using the GuidanceFunctions that were presented in Section 4.3.2, while the *Lower-Level Controller* is implemented inside the BasicVehicleControl, as it was shown earlier, in Figure 4.5. The implementation toolkits that are shown in Figure 4.8 are all used in a hierarchical way. Indeed, JACK plans relating to coordination protocols use plans from the *Planning* sub-layer to execute actions involved in a manoeuvre. Then, the plans relating to driving manoeuvres use the *Guidance Functions* to execute specific driving actions and the *Guidance Functions* use the BasicVehicleControl to apply their actions on the vehicle's actuators.

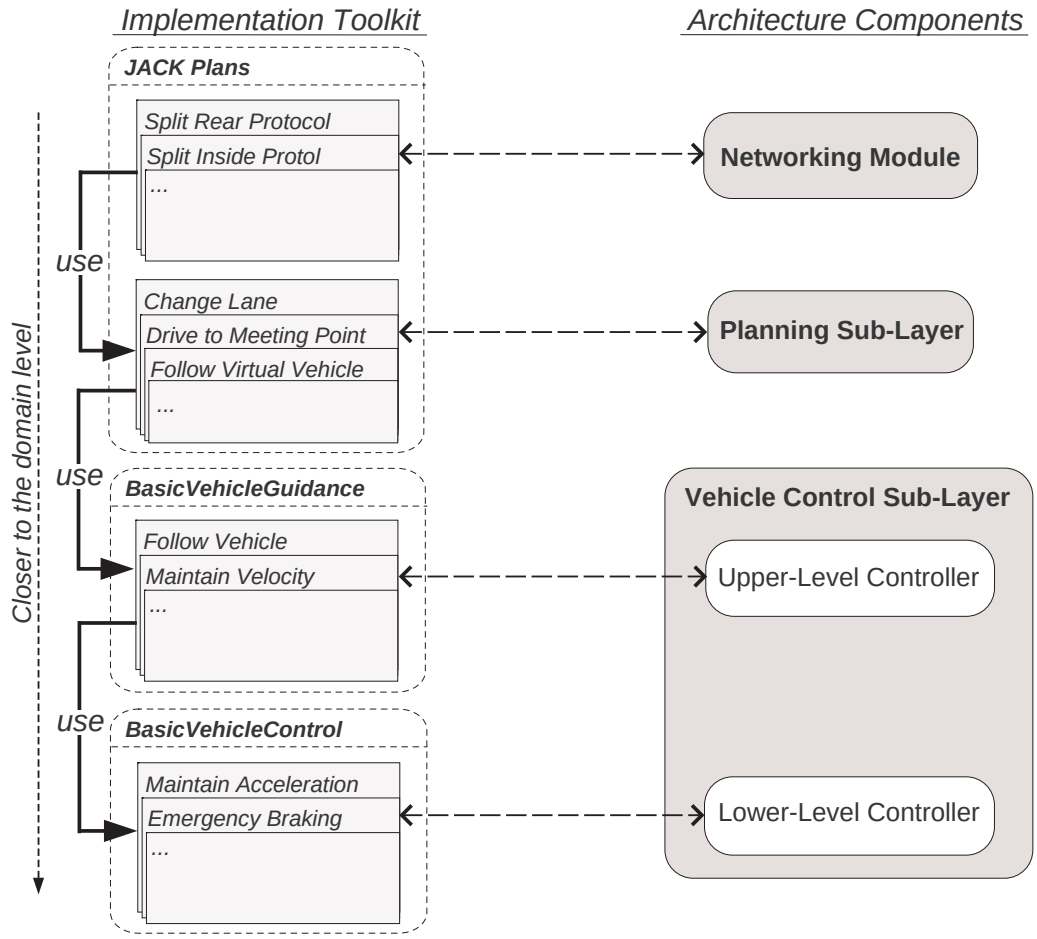

Figure 4.8: The components relating to the *Planning* sub-layer, inside the architecture's hierarchy.

The implementation toolkit's hierarchy presented in Figure 4.8 also points out the fact that the higher a plan or function is in the hierarchy, the farther it is from the domain level (automated driving). Thus, the coordination plans can be developed in an abstracted way that is not concerned by any modifications relating to the vehicle. This makes our Collaborative Driving System easier to maintain and enables us to test

different types of vehicle controllers.

4.4 Auto21 Architecture Integration Schemes

The architecture of the Auto21 CDS has been presented along with its model of software design, so we can now focus on the schemes behind the integration of each component of the architecture inside its execution environment (the simulator). These schemes refer to a vast range of research domains that have been studied by our group and our partners in Auto21. Thus, the schemes used to integrate the *Intelligent Sensing* sub-layer of our architecture are presented using the work of our colleagues at the University of Calgary, in Section 4.4.1. For the *Vehicle Control* sub-layer, we present in Section 4.4.2, the research done at the University of Sherbrooke along with our own research in Section 4.4.3. Finally, the schemes that we used to integrate the *Planning* and *Coordination* sub-layers and the *Traffic Control* layer are presented in Sections 4.4.4, 4.4.5, and 4.4.6.

4.4.1 Sensing Scheme

The schemes that have been used to integrate the *Intelligent Sensing* sub-layer inside our CDS are presented here, by referring to our partners' researches, at the University of Calgary. Note that these schemes have not been developed in the CDS we are currently using, since our partners did not publish their final conclusions on the required sensing devices.

Before describing the concept proposed by our partners to analyze the data from our vehicles' sensors, we now describe the types of sensors we plan to use. The initial sensing devices that have been considered as part of our colleagues' research are: GPS receivers, inertial navigation systems, millimiter wave radar, video imaging, laser ranging and sonar. The research programs on autonomous driving vehicles that were presented in Section 4.1.2 have proposed the following sensors, which should also be considered in our future investigations: different types of laser, different long and short range radars and lidars.

Using the previous list of sensors, our partners proposed different models to analyze and improve the data received from the sensors. They first tested the sensors on manually driven vehicles. Then, they focused on the positioning systems and the problem

of using GPS in a urban environment and by doing so, results on perception within a platoon of vehicles are now appearing. They also managed to improve the position received from GPS in a urban environment that interferes with the GPS's signals. In a first step, they used Doppler information and GPS augmentation with a rate gyro to filter the data [Mezentsev et al., 2002]. Then, they developed algorithms for map matching using methods such as Kalman filtering [Basnayake and Lachapelle, 2003] and fuzzy logic [Syed and Cannon, 2004], which resulted in an improved and reliable position output using noisy inputs. Finally, our colleagues also tested different sensors for vehicles positioning in a platoon of four vehicles driven by humans executing such manoeuvres as splitting and merging [Cannon et al., 2003]. Within this platoon context, they focused on an approach using a GPS with a moving base station and received centimeter level position results that were available for and about every car within the platoon.

4.4.2 Lower-Level Controller Scheme

The schemes used to integrate the *Lower-Level Controller* in our CDS, part of the *Vehicle Control* sub-layer of Figure 4.4, are now defined. It should be recalled that the *Lower-Level Controller* receives vehicle acceleration commands from the *Upper-Level Controller* and modifies the vehicle's brake and gas throttle actuators to attain its target. Although Figure 4.4 showed both lateral and longitudinal *Lower-Level Controllers*, only the longitudinal one has been defined. The work on the *Lower-Level Controller*, which is now presented, was provided by our partners at the University of Sherbrooke.

The initial function proposed by our partners to implement the *Lower-Level Longitudinal Controller* transforms the desired acceleration into its respective traction force and torque, as defined in the vehicle's wheel model (described for our simulator in Section 3.3.1). The wheel torque can then be transformed into the right engine torque, which finally results in the required gas throttle, when referring to a map of throttle versus engine revolutions [Huppe et al., 2003].

In a second approach that our partners proposed for the *Lower-Level Longitudinal Controller*, they focused on the analysis of the platoon string stability. In this context, Sherbrooke's team tried to minimize the oscillations considering the inter-vehicle spacing inside the platoon. To ensure this stability, our partners guided their lower level longitudinal controller using an algorithm based on pentic polynomial in time, which limits the control efforts (i.e. high accelerations). Results in the robotic environment showed the stability of the string in different situations, as well as smoother reactions to vehicle entrance in the platoon [Huppe et al., 2003].

4.4.3 Upper-Level Controller Scheme

The *Upper-Level Controller* has been defined from the architecture point of view in Figure 4.4, but it is now presented by showing the schemes that were used to integrate with the rest of our application. The *Upper-Level Controller* is implemented by "multi-purpose" control algorithms that can be used by the *Planning* sub-layer to give driving action orders. All the control algorithms of the *Upper-Level Controller* have one thing in common, they use the driving action orders in input and they output a command of vehicle acceleration, applied by the *Lower-Level Controller*. To recall the place that the control algorithms take in our architecture, the reader can refer to Figure 4.7 where "practical" algorithm can be integrated to our CDS as part of Java classes deriving from the GuidanceFunction class. Since theory and practice are often very different one from another, the theoretical control algorithms is first presented. Afterwards, the specific algorithms, which have been programmed and adapted to our application for specific purposes, are described using some examples.

Theoretical Control Algorithms

Theoretical models of the *Upper-Level Controller*'s algorithms have been proposed by both our team and our partners at the University of Sherbrooke. The algorithms from Sherbrooke are first presented, followed by ours.

Our partners proposed an inter-vehicle distance control function to implement the *Upper-Level Controller*, used as part of a platoon of vehicles. This function takes in consideration a controlled vehicle i's position (e_i), velocity ($\dot{e}_i$) and acceleration ($\ddot{e}_i$) relative to the vehicle it follows and their respective errors. The same control function always verifies the vehicle stability inside the platoon by considering the different control gains that must be respected [Huppe et al., 2003]. To ensure the stability, the control function calculates the optimal gains using a Linear Quadratic Regulator (LQR), which gives a representation of the gains in function of previous parameters ($e_i, \dot{e}_i, \ddot{e}_i$) and a given control effort (the output of this controller). Another approach, using a Kalman filter, has been considered by our partners, to estimate the parameter on the relative acceleration ($\ddot{e}_i$) and its error. Instead of having to explicitly calculate the parameters, this method uses additional sensors or simply receives the information via communications.

At last and in parallel with Sherbrooke's work, we developed algorithms for both the lateral and longitudinal control issues inside the *Upper-Level Controller*. Our al-

gorithms also include two categories of inter-vehicle distances maintenance functions, where one uses a gap in time (in seconds) and the other uses a distance (in meters). As an initial implementation, two different longitudinal control algorithms were developed inside the *Upper-Level Controller*: an algorithm based on an equation using front distance and relative velocity and an algorithm using a neural network, learning the required velocity for its vehicle, at a specific range of velocity and inter-vehicle distances [Hallé et al., 2003]. The latter controller suffered from an inexhaustive learning phase due to the simulator status at that time, even though it managed to provide smoother reactions at the price of poor reaction time to critical scenarios. Similarly, the first controller also calculated a velocity to command to the *Lower-Level Controller*, but it was in accordance with the relative velocity with the preceding vehicle and their inter-vehicle distance. Furthermore, two different longitudinal control algorithms, outputting requests on acceleration, have recently been used in our CDS [Hallé et al., 2004]. These algorithms both use a different gap type: a distance in meters similar to the controller proposed by Tsugawa et al. [2001] and a distance in seconds, similar to the controller of Daviet and Parent [1996]. At the moment, we use an implementation of both of these algorithms inside our *Upper-Level Controller*, which enables the request of either a gap in time or in distance, depending on the actions required by the driving manoeuvre being executed.

Finally, we also proposed a lateral control algorithm to perform lane changes driving actions. This controller is part of the *Upper-Level Controller* which acts on the *Lower-Level Lateral Controller*, but does not form a complete lateral controller. Indeed, it only performs one lateral behavior (lane changing), without being able to follow vehicles moving laterally or follow curves on the road. The lane change behavior that has been used is the one from Hatipoglu et al. [2003], where the vehicle has to follow the path defined by a positioning function, in order to change lane. This function is a simple sigmoid function $p(t) = 1/1 + e^{(-\alpha t)}$. and it is being used to control the wheel angle according to d^2p/dt. In these functions, α is used to control the duration of the driving action and d represents the distance with the target lane. Again, we assume that the road curative does not change during the lane change driving action.

Practical Control Algorithms

When the control algorithms have been implemented in the driving system currently running in our simulator, they had to be slightly modified and adapted considering their behavior throughout the different scenarios. Thus, the details on the exact algorithms being used inside the simulated CDS's *Upper-Level Controller* are now presented. First, the longitudinal control algorithm that maintains a specified distance in time is detailed

to demonstrate how to use it inside our application. Then, another longitudinal control algorithm example will be presented to demonstrate how the *Upper-Level Controller* supports the merge manoeuvre.

The longitudinal control algorithm, which is now described, has recently been used in Hallé et al. [2004] to maintain inter-vehicle distances inside our platoon formations. It is based on Daviet and Parent [1996]'s time gap function represented as:

$$a_i = \delta a + \frac{1}{h}(\delta v + k(\delta x - (gap \cdot v_{i-1}))) \tag{4.1}$$

where a_i is the acceleration of the i^{th} vehicle in the platoon, δa is the difference of acceleration, δv is the difference of velocity, δx is the inter-vehicle distance and gap is the desired time between vehicles.

Our implementation of control function 4.1, as used in our application, is represented by Algorithm 2. The GUIDE-TIME-GAP function represents the abstract version of the actual implemented controller, where new ratio that have been tuned up during the simulation process were added. The additions we have made enables the function to react to more or less critical situations and moderate the reaction of the *Upper-Level Controller* when closing on the front vehicle. GUIDE-TIME-GAP uses as inputs a list of variables that are taken from the up to date knowledge base kb, and the desired time gap $interVehicleTime$.

The knowledge kb base refers to a data structure that was presented earlier in Figure 4.5, which represents an historic of sensed data. When new information is sensed by the *Intelligent Sensing* sub-layer, it is added to this knowledge base. The addition of new information to the knowledge base is also used to trigger the update of our *Upper-Level Controller*, which only recalculates its acceleration command when the environment changes. Thus, a daemon monitors the knowledge base and if the data received by the sensor has a high value of difference with the current knowledge, it calls an update on the current control algorithm, in the *Upper-Level Controller*.

Algorithm 2 uses its inputs to first verify if the current environment is considered as an emergency state (IS-EMERGENCY-STATE()). Then, it calculates a brake value ($brakeFilter$) to add if the controlled vehicle is too close to the preceding vehicle. It finally adds the brake value to the control function of Daviet and Parent [1996], and lowers the returned acceleration, if this one is not comfortable (considering *ComfBrake* and *ComfAccel*).

The other longitudinal control algorithm that is presented as part of our examples of implementation of the *Upper-Level Controller*, is a "meeting point" control function.

Algorithm 2 function GUIDE-TIME-GAP(kb, $interVehicleTime$)

returns *accel*, the commanded acceleration to the low-level controller
{All the vehicle's state data are implicitly taken from the input kb}
inputs: kb, the up to date knowledge base of the driven vehicle
$velocity$, the vehicle's last velocity (from kb)
δt, time gap with the front vehicle (from kb)
δa, acceleration difference with the front vehicle (from kb)
δv, velocity difference with the front vehicle (from kb)
δx, distance gap with the front vehicle (from kb)
$interVehicleTime$, the commanded inter-vehicle gap in time
local variables: $safeDistance$, the safe distance in meters for this state
rt, the refresh time of the sensors' data
$brakeFilter$, a negative acceleration value filtering the result
accel, the acceleration to command
$k1$, the scalar k from control equation [Daviet and Parent, 1996]
$k2$, a damping ratio, as the vehicle is close to $safeDistance$
constants: *MinDistance*, qualifies an important difference in states values
ComfAccel, the comfortable acceleration value
ComfBrake, the comfortable negative acceleration value
C, an adaptive weight for the difference of acceleration
if IS-EMERGENCY-STATE(kb) **then**
return *highbrakingvalue*
$safeDistance \leftarrow min(interVehicleTime \times velocity, MinDistance)$
$rt \leftarrow$ get refresh time from kb
$k1 \leftarrow min(1/rt, 3/velocity)$
$k2 \leftarrow |interVehicleTime - \delta t|$ / rt
if $\delta t < interVehicleTime$ **then**
$brakeFilter \leftarrow (interVehicleTime - \delta t)$ / $interVehicleTime$
$accel \leftarrow \delta a \times C + k2/(rt \times C) \times (\delta v + k1 \times (\delta x - safeDistance))$ - $brakeFilter$
{Ensures a comfortable acceleration}
if $accel > ComfAccel$ **then**
return $ComfAccel \times \sqrt{accel/ComfAccel}$
else if $accel < ComfBrake$ **then**
return $ComfBrake \times \sqrt{accel/ComfBrake}$
else
return *accel*

We developed this function from scratch, to provide a support to the driving plan used when a vehicle is merging a platoon. This algorithm, which could be called a Meeting Point Cruise Control (MPCC), is used when a vehicle is not currently driving inside a specific platoon formation, but needs to coordinate with this platoon, at a specific position on the highway. Currently, the MPCC only acts on longitudinal actuators with a meeting point in one dimension, supporting only a straight road.

Algorithm 3 presents this control algorithm along with two input variables that are the up to date knowledge base and the desired "dynamic" meeting point's information. The meeting point is described by a constant velocity, an initial longitudinal position and an initial time. Briefly, the GUIDE-MPCC function commands a velocity to reach the meeting point, by considering that the point was initially at $mpPos$ meters, at $mpTime$ seconds and that the point is moving at a velocity of $mpVelo\ m/s$. GUIDE-MPCC chooses a velocity to follow the curve shown in Figure 4.9. Thus, it uses a much higher velocity at the beginning, to reach the meeting point and slowly meets the constant velocity when it gets closer to the meeting (time n on the graphic). Since, the velocity and acceleration control functions used by the MPCC function react with a certain delay to meet the desired dynamic state, the MPCC function has to take the delay factors in consideration. Therefore, the MPCC function orders velocities considering the delays and smoothly meets the meeting point. Finally, the MPCC returns the acceleration given by the GUIDE-VELO function in order to meet the desired velocity.

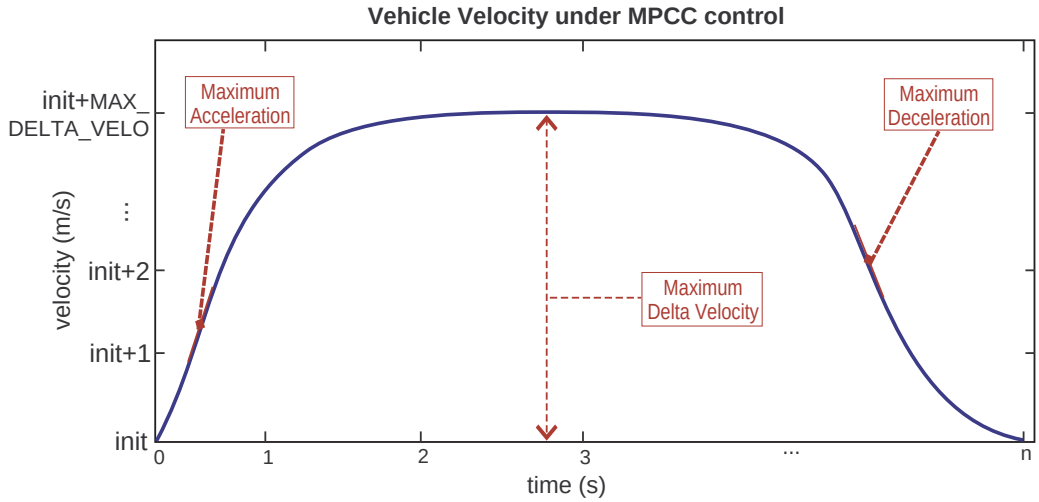

Figure 4.9: Model of the vehicle's desired velocity using a MPCC controller.

Algorithm 3 function GUIDE-MPCC(*kb*, $mpVelo$, $mpPos$, $mpTime$)

returns *accel*, the commanded acceleration to the low-level controller
{All the vehicle's state data are implicitly taken from the input *kb*}
inputs: *kb*, the up to date knowledge base of the driven vehicle
velocity, the vehicle's last velocity (from *kb*)
position, the vehicle's last position (from *kb*)
time, the last sensed data's time (from *kb*)
acceleration, the vehicle's last acceleration (from *kb*)
$mpVelo$, the meeting point constant velocity
$mpPos$, the meeting point initial position
$mpTime$, the meeting point initial time

local variables: $deltaTime$, time elapsed from the initial meeting settings
$deltaVelo$, difference of velocity with the meeting velocity
$timeToVelo$, time in sec. to reach the meeting velocity
$mpDistance$, distance from the initial meeting position
$distanceToVelo$, distance covered to reach meeting velocity
$commandVelo$, velocity to order to the velocity controller

constants: *HalfVehLength*, half of a standard vehicle length
MaxDeltaVelo, maximum velocity difference with $mpVelo$
MaxDeltaAccel, maximum acceleration to maintain $mpVelo$

if IS-EMERGENCY-STATE(*kb*) **then**
  return *highbrakingvalue*
$deltaTime \leftarrow time - mpTime$
$deltaVelo \leftarrow velocity - mpVelo$
$mpDistance \leftarrow deltaTime \times mpVelo/1000 + mpPos - (position + HalfVehLength)$
$timeToVelo \leftarrow$ GET-TIME-REACH-VELOCITY(*velocity*, $mpVelo$) / 1000
$distanceToVelo \leftarrow timeToVelo \times deltaVelo$
if $distanceToVelo \times$ sign of $mpDistance \geq mpDistance \times$ sign of $mpDistance$ **then**
  return GUIDE-VELO(*kb*, $mpVelo$)
else if $|mpDistance| < 1 \wedge |deltaVelo| < 0.5$ **then**
  return GUIDE-VELO(*kb*, $mpVelo + mpDistance/timeToVelo + deltaVelo$)
else
  $commandVelo \leftarrow mpVelo + MaxDeltaVelo \times$ sign of $mpDistance$
  if $velocity \simeq commandVelo \wedge |accceleration| < MaxDeltaAccel$ **then**
    return *velocity*
  else
    return GUIDE-VELO(*kb*, $commandVelo$)

4.4.4 Agent Oriented Planning Scheme

In Section 4.3.3, we presented how the *Planning* sub-layer was being used in our application, in relation to other components of our architecture. One of these components is the *Networking* module, which includes communication protocols that require the plans relating to driving manoeuvres, available inside the *Planning* sub-layer, as shown in Figure 4.8. In order to explain how the *Planning* sub-layer performs the tasks required by other components, this section describes the schemes behind the integration of the *Planning* sub-layer, according to the flow of activities that generate driving plans inside our CDS. This flow is based on a model presented in Section 2.1.3, called the Belief Desire Intention (BDI) agent model Rao and Georgeff [1995], which is explained below.

A global view of the activities generated inside the *Planning* sub-layer by the BDI model is presented in Figure 4.10. Plans that are executed inside the *Planning* sub-layer refer to intentions that could be either a *Joint Intention* or a *Local Intention*. A *Local Intention* is triggered when the agent's current beliefs relate to a desire the agent already had and make this desire, a possible intention. This "match" between the beliefs and desires raises the *Local Intention* and/or a possible *Joint Intention* that has to be coordinated with other agents (i.e., an intention to merge a platoon). A *Joint Intention* coming from another agent, like agent 'j' in the example of Figure 4.10, also relates to a plan inside the *Planning* sub-layer.

Considering the interactions with the *Networking* module, the plans are being used in a sort of Partial Shared Plans (PSP), as depicted in Section 2.2. Using this model of *Planning* sub-layer, individual plans resolve coordination issues by negotiating with vehicles involved in the collaborative task and by creating a mutual PSP that relate to the *Joint Intention*.

To support the flow of activity presented in Figure 4.10, our plans include contextual issues according to the agent's beliefs. These plans specify the details of their execution, as well as pre- and post-conditions relating to the agent's intentions. Thus, the kind of planner we developed for the *Planning* sub-layer relates to the Procedural Reasoning System (PRS) architecture presented in Section 2.1.3. The choice of "recipe" type of plans should be both a flexible and time efficient solution for our agents. Even though the plans execution is not completely predictable, as it considers evolving contextual issues, the overall CDS benefits from such a planning system that helps keeping track of the agents' adopted strategies and current intentions. Hence, plans' structures can be kept in a database and relate with certain driving categories, which helps agents to share their intentions. It should finally be mentioned that considering the type

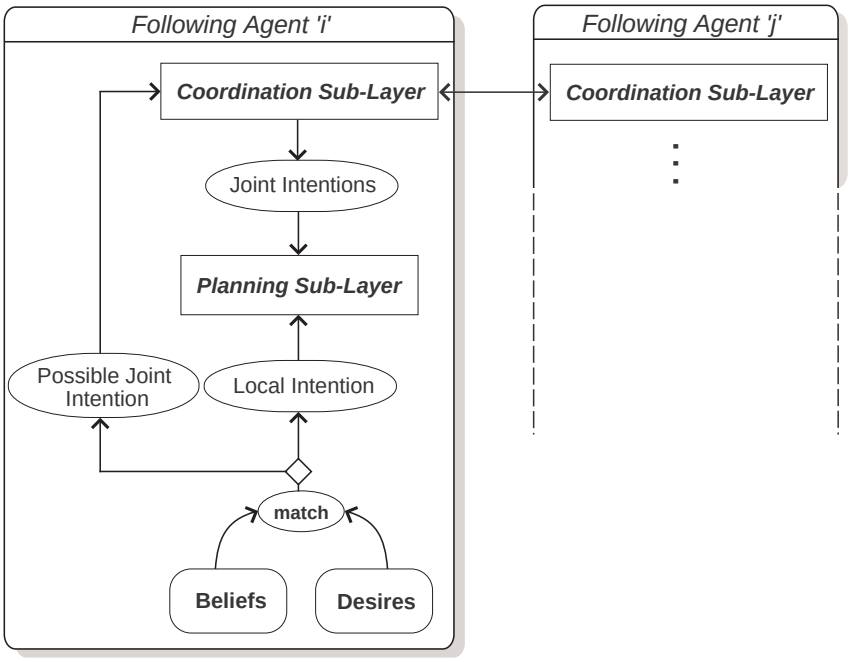

Figure 4.10: Model of the BDI agent oriented *Planning* sub-layer.

of coordination used in the *Networking* module, the *Joint Intentions* may only be the leader's intention in the case of a centralized coordination, as described in Section 5.1.1.

4.4.5 Inter-Vehicle Coordination Scheme

The following section presents an overview of the methodology we decided to use to coordinate multiple vehicles evolving in a same neighborhood. As it is presented below, we use a methodology proposed by Bana [2001], which is more appropriate for the coordination of automated vehicles than other traffic management schemes [Davidsson et al., 2004]. The methodology of Bana [2001] allows vehicles to coordinate actions like lane changes by giving priorities to the actions they desire, based on their importance. This coordination methodology is implemented inside the *Linking* module, in charge of inter-platoon coordination, as it was explained in Section 4.2.2. The *Linking* module works closely with the *Traffic Control* layer in the case of a centralized linking coordination. It also gives driving manoeuvres orders to the *Networking* module, following from its coordination process, which is explained below.

The coordination methodology of Bana [2001] is presented below by starting with the description of its symbols, while two examples referring to this methodology are given in Section 5.1. The symbols used in the inter-vehicle coordination functions of

Bana [2001] are described in the following list:

- u^i: the action of a single vehicle i.
- $\overline{u}$: the vector of actions of vehicle 1 to n $(u^1, ..., u^n)$.
- $\overline{u}_i$: = the vector including the coordinated action of vehicle i and the actions of its co-manoeuvring vehicles.
- C^i: the cost of the current vehicle i (according to its position and state)
- $C^i_{\overline{u}_i}$: the cost of the state of vehicle i after the execution of a safe set of actions $\overline{u}_i = (u^1, ..., u^n)$.
- N^i: the neighborhood of vehicle i.

In the definition of Bana [2001]'s methodology, a cost is associated with vehicles according to their state on the highway (their current lane number and destination), which must be minimized by planning actions according to this function:

$$C^i_{\overline{u}^*_i} = min_{\overline{u}_i}(C^i_{\overline{u}_i}) \tag{4.2}$$

In order to coordinate an action $\overline{u}_i$ that affects other vehicles, vehicle i's *Planning* sub-layer generates a set of possible actions relating to its local intentions. The *Linking* module then chooses an action $\overline{u}^*_i$ that minimizes its cost $C^i_{\overline{u}^*_i}$ (the cost reducing action), according to a domain-level heuristic presented in Section 4.4.6. To choose the minimum cost action, the *Linking* module assigns a cost to the different states its vehicle can be in, by considering the current traffic information of its neighborhood. The cost reducing action is then communicated by agent i and by other agents in a same neighborhood, in order to share each others' action propositions. Finally, the *Traffic Control* determines the vehicle with the highest cost (highest priority) in a specific neighborhood and accepts the action it proposed.

Once a priority action has been chosen (let say it was agent i's action), neighbor vehicles involved in the action or manoeuvre of agent i are given *safety permissions*. These *safety permissions* command neighbor vehicles to take i's co-manoeuvring actions instead of taking their own actions. For other neighbor vehicles, which have not been given *safety permissions*, the same priority process is used until every agent has been assigned an action. If agent j was next in line, the *Linking* module verifies if its action does not conflict with i's action, by considering if the required co-manoeuvring vehicles are available and if certain safety factors are respected. Consequently, each possible vehicle j, which is in i's neighbors ($j \epsilon N^i$), chooses an action to execute according to its impact on i's coordinated action ($\overline{u}_i$). Thus, the choice of j's action is done

in accordance with a function that minimizes the maximum cost of vehicles in the neighborhood N^i:

$$max_{j \epsilon N^i} C^j_{\overline{u}^*_i} = min_{\overline{u}_i}(max_{j \epsilon N^i} C^j_{\overline{u}_i}) \tag{4.3}$$

Where $C^j_{\overline{u}_i}$ is the cost of vehicle j's state, after the coordinated action $\overline{u}_i$ has been executed. Using this cost function, subsequent vehicles j are forced to choose an action which brings them in a state that does not interfere with i's cost. Therefore, the coordination between vehicles in the neighborhood N^i is ensured, since all the actions accepted by the *Traffic Control* layer are considered as safe. Detailed steps, to define how this coordinated action is applied in collaboration with the neighborhood, are presented in PATH's report [Bana, 2001], but is not presented here.

4.4.6 Traffic Management Techniques

The task of managing traffic flow on our automated highway is realized in relation with the inter-vehicle coordination methodology, implemented inside the *Linking* module and presented in Section 4.4.5. As mentioned in Section 4.2.3, the traffic management methods are implemented inside the *Traffic Control* layer of our architecture (refer to Figure 4.4), who's function is to regulate the traffic flow. The *Traffic Control* layer may be developed as part of the a road-side infrastructure or a wireless mobile network, depending on the chosen inter-platoon coordination model (refer to Section 5.1).

The *Traffic Control* layer is implemented by an algorithm that receives real-time information about vehicle density and velocity profile on the road. This algorithm was inspired by a strategy proposed by Bana [2001], which works in accordance with its inter-vehicle coordination methodology, described in Section 4.4.5. Within the model proposed by Bana [2001], the *Traffic Control* layer determines the entry flow rate for a specific neighborhood in a link L, which refers to a highway segment controlled by a set of sensors. Then, it guides the link's cruise velocity v_L, as well as the maximum platoon size (length) P_L, by communicating with the vehicles' *Linking* module. In addition, the *Traffic Control* layer determines the required actions by vehicles at every lane, in the form of lane change suggestions. To achieve this, it uses a representation of a vehicle's status, defined by the position of the vehicle in a platoon: *Alone, Head, Tale, Follower*, its lane number, and its type: entering or leaving the highway. The vehicle's status then determines its *cost* and its priority for actions within its neighborhood, in accordance with equations 4.2 and 4.3.

4.5 Conclusion

In this chapter we presented a hierarchical architecture, which is being used to resolve the problems of control and coordination of our real-time Collaborative Driving System (CDS). Section 4.3 presented the analysis and resulting models that enabled us to develop the drivers' architecture. The software engineering process of our architecture was based on design patterns and object oriented concepts, that enabled us to have a flexible structure that can be modified with a certain ease. Given these facts, the components of our architecture and the developed classes and code can be more easily exported, as we plan to do with other simulators or vehicles. In addition, this flexible architecture helped us to test the different integration schemes surrounding the ones we presented in Section 4.4.

Chapter 5

Driving Agents Coordination

The coordination of automated vehicles is one of the major problems that must be resolved in order to build a Collaborative Driving System (CDS). To resolve this problem, an heuristic based on vehicles' cost on the highway, similar to the one presented in Section 4.4.5, has been used to implement the *Coordination* sub-layer of our architecture (refer to Figure 4.2). This sub-layer is divided in two modules that represent the two coordination aspects depicted in this chapter: inter-platoon coordination supported by the *Linking* module and the intra-platoon coordination supported by the *Networking* module.

This chapter describes different methods of inter and intra-platoon coordination, by starting with the inter-platoon coordination problem. Thus, Section 5.1 presents two coordination models that can be used to implement the *Linking* module of our architecture: centralized inter-platoon coordination model, decentralized inter-platoon coordination model. Note that the inter-platoon coordination models only constitute a solution we propose to resolve this problem and have not been developed in our application, as opposed to the intra-platoon coordination models.

Following the description of two inter-platoon coordination models, three models of intra-platoon coordination are presented in Section 5.2: centralized intra-platoon coordination, decentralized intra-platoon coordination, and teamwork intra-platoon coordination. In that section, the coordination of a single driving manoeuvre (split or merge) inside a single platoon is presented in much more details, by describing how each intra-platoon coordination model resolves the task of coordinating neighboring vehicles' actions.

5.1 Inter-Platoon Coordination Model

The inter-platoon coordination is the highest level of coordination inside our architecture, presented in Figure 4.2 and it is supported by the *Linking* module that collaborates with the *Networking* module for the intra-platoon manoeuvres. The *Linking* module focuses on two important aspects of autonomous cooperative driving: the vehicles' safety and traffic optimization. In order to optimize the traffic flow, suggestions are given to vehicles, in the form of lane changes orders, which result in platoon merge and split manoeuvres. To realize its tasks, the *Linking* module uses a coordination heuristic, which was presented in Section 4.4.5, and deploys it through two possible coordination models that are presented below. Before describing the inter-platoon coordination models, the reader should understand that a vehicle is involved in the coordination process when it merges or creates its first platoon and it ends when the vehicle has left a platoon to switch to manual driving.

5.1.1 Centralized Inter-Platoon Coordination

The centralized inter-platoon coordination is a model which considers the existence of a road-side *Traffic Control* system, defined in Section 4.2.3, and inspired by the road-side system used in the PATH project [Bana, 2001]. The centralized model simplifies the coordination task a lot more than the decentralized model, presented in Section 5.1.2, since it implements the inter-vehicle coordination heuristic (Section 4.4.5) inside one "master entity".

Figure 5.1 represents the example of two platoons $P1$ and $P2$ following each others in lane 2 and two vehicles in the neighboring lane (lane 1). In this figure, the initial state is defined by vehicles $L3$ and $L4$, which just entered on the highway and need to change to lane 2 where platoon $P1$ and $P2$ already reside. Simultaneously, vehicles $F25$ and $F13$, already part of platoons $P1$ and $P2$, both try to coordinate manoeuvres to split from their respective platoon, because they need to join the right lane to exit the highway. In this situation, different information is exchanged between the *Traffic Control* layer and the vehicles' *Linking* module, in order to coordinate the lane changes of those vehicles. At the final step of this coordination, the *Traffic Control* layer communicates to neighboring vehicles, the manoeuvres that can be safely executed. These vehicles finally execute the safe manoeuvres and coordinate this execution by using one of the intra-platoon coordination presented in Section 5.2.

In more details, the inter-platoon coordination process begins when the *Planning*

sub-layer of vehicles $L3$, $L4$, $F25$ and $F13$ orders the *Coordination* sub-layer to change lane in order to minimize its vehicle's local cost. At that moment, the four vehicles propose inter-platoon manoeuvres in the form of actions to coordinate, described by: $\overline{u}^{L3}$, $\overline{u}^{L4}$, $\overline{u}^{F25}$, $\overline{u}^{F13}$ (shown in Figure 5.1). Recall that the description of symbols relating to the coordination of agents' i actions $\overline{u}_i$ was presented in Section 4.4.5. The proposition of actions $\overline{u}_i$ is realized by the *Linking* module of each vehicle, which communicates with the highway's *Traffic Control* layer.

Following the proposition of actions by vehicles $L3$, $L4$, $F25$ and $F13$, the *Traffic Control* determines that vehicles $L3$ and $L4$ have the priority, because it is more important to empty the right lane as fast as possible. Thus, the *Traffic Control* informs concerned vehicles, that a merging manoeuvre for both vehicle $L3$ and $L4$ will be executed. The proposed actions $\overline{u}^{F25}$ and $\overline{u}^{F13}$ are therefore rejected and actions $\overline{u}^{*}_{L3}$ and $\overline{u}^{*}_{L4}$ are executed. The actions that are executed can represented as two sets of actions:

- $\overline{u}^{*}_{L3} := \{u^{L3}\} \cup \{u^{p} : p \in N^{L3}(u^{L3})\}$
- $\overline{u}^{*}_{L4} := \{u^{L4}\} \cup \{u^{q} : q \in N^{L4}(u^{L4})\}$

This represents the mergers' actions (u^{L3} and u^{L4}), along with the co-manoeuvring actions from vehicles p and q in the neighborhoods N^{L3} and N^{L4}. In the example of Figure 5.1, vehicles p are represented by the vehicles of platoon $P2$, while vehicles q are represented by platoon $P1$ which is split in two, to create a second platoon $P3$. The resulting co-manoeuvring actions are u^{P1}, u^{P2} and u^{P3}, which are coordinated with platoons $P1$, $P2$ and the newly formed platoon $P3$. This means that considering the type of intra-platoon coordination (centralize, decentralized, teamwork), an interface inside the *Linking* module specifies which vehicle(s) act(s) as the platoon x's representative(s) and receive(s) the co-manoeuvring action u^{Px}.

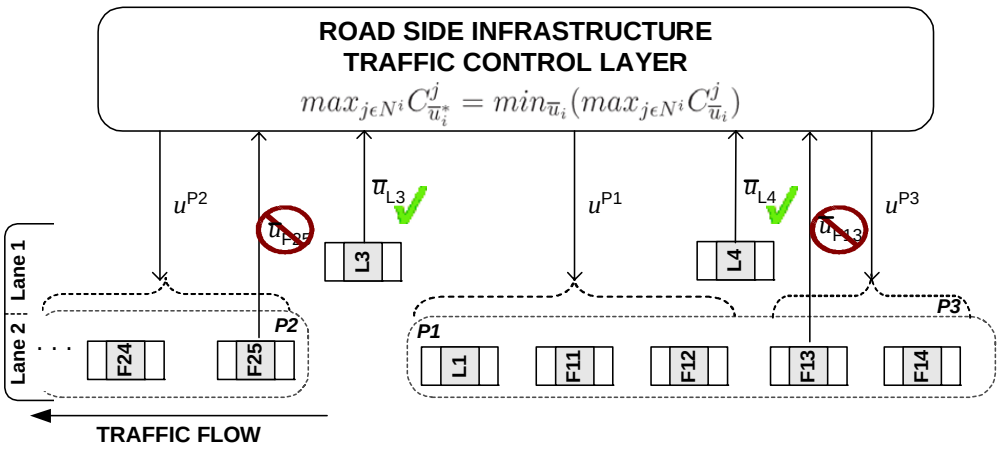

Figure 5.1: Centralized decision making using a *Traffic Control* layer.

As it has been shown, the implementation of the *Traffic Control* functions into only one big system, centralizing all of the traffic knowledge, makes the administration and coordination tasks simpler, but necessitates a large amount of communications from and to each vehicles on the highway. In addition, it requires sensors at different segments on the road to monitor traffic, so it relies on the existence of such an infrastructure.

5.1.2 Decentralized Inter-Platoon Coordination

Instead of using road-based sensors, vehicles' sensors along with their inter-vehicle communication system, could be used to realize the inter-platoon coordination, inside the *Linking* module. Such a coordination is defined as decentralized and it has the benefit of not requiring the addition of costly Intelligent Transportation Systems (ITS) equipment. The decentralized inter-platoon coordination model creates a merging interface through each vehicle's *Linking* module and optimizes the communications by minimizing the round trips usually necessary to come to mutual intentions between platoons. On the other hand, the decentralized coordination is more complex, so to handle it, we decided to use an approach including mobile agents which can resolve both the coordination and communication problems. Within this approach, a mobile agent acting on the behalf of a platoon coordinates itself with others by using less bandwidth, since its coordination is done locally, as it is described in this section.

In our decentralized model, mobile agents can move across the different platoon representative vehicles, which include an onboard system that contains a Knowledge Base (KB) of the neighborhood traffic. The mobile agent coordination model we are considering is a *blackboard* model for indirect coordination [Cabri et al., 1997]. To apply the mobile agent model to collaborative driving, the neighborhoods organization that was presented in Section 4.4.5 is also being used in this model. Each neighborhood has its own blackboard which resides on the biggest, previously elected, platoon. Within this model, every platoon representatives in the neighborhood can locally coordinate themselves on the blackboard in an asynchronous way, which lowers the latency times.

The blackboard model is presented in Figure 5.2 where mobile agents representing neighbor platoons are sent to the blackboard situated on platoon 1. From this blackboard, they coordinate themselves with platoon 1's representative ($M.A.1$) using the same vehicle states cost assignment function that the centralized model, which was introduced in Section 4.4.5. Mobile agents can then share the knowledge they acquired through the different blackboard they visited so far. They can also gain new knowledge stored in the blackboard, which comes from other mobile agents that visited it. This way, a mobile agent can retrieve information about the driving agents' intentions in the

platoon it represents. It can also manage to set the vehicle's cost (relating to Section 4.4.5) and inform platoon members about their actions' priority.

In the example represented by Figure 5.2, we considered two platoons $P1$ and $P2$ following each others in lane 2 and two other platoons $P3$ and $P4$ following each others in lane 1. We suppose that both platoon $P1$ and $P2$ want to remove a vehicle from their platoon (split), but they need to ensure that this manoeuvre can be safely executed. Therefore, the representative of each platoon Pi in a same neighborhood sends a mobile agent $M.A.i$ with the information about its platoon's intentions. All the mobile agents gather on a blackboard and share their intentions. In our example, a split manoeuvre is proposed by $M.A.1$ and $M.A.2$, but safety rules only allow one platoon to execute a manoeuvre that modifies its length (like the split) at the same time, when these platoons are following each others. If we suppose that platoon $P2$ has the highest cost, $M.A.1$'s proposition is rejected and the split inside $P2$ is accepted. Consequently, $M.A.2$ has to coordinate co-manoeuvring actions with $M.A.4$ and perhaps $M.A.1$ and $M.A.3$, if it is required to ensure safety. For instance, $P4$'s co-manoeuvring action is to accelerate or create a gap to enable the vehicle splitting from $P2$ to change lane, while $P1$'s co-manoeuvring action may be accelerating to allow $P2$ to create a gap to safely execute the split.

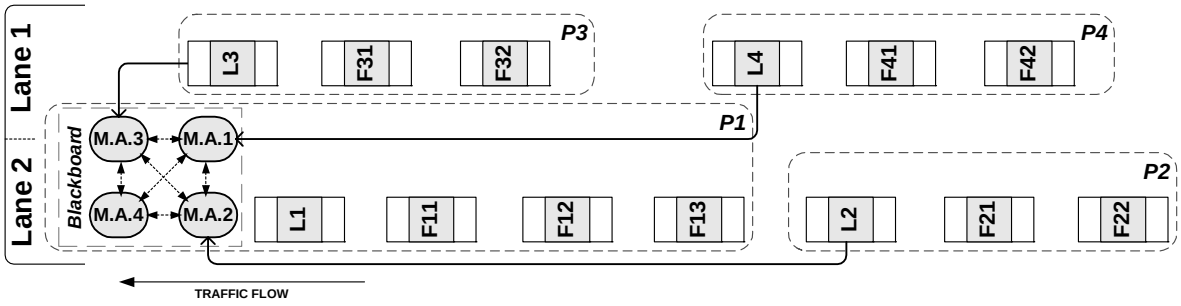

Figure 5.2: Decentralized decision making using mobile agents.

Our model based on mobile agents is beneficial on communicative aspects, since mobile agents can coordinate themselves with available agents and leave information for upcoming ones on the blackboard. Nevertheless, the mobile agent model could still consider using a road-based *Traffic Control* layer if such an infrastructure exists. Mobile agents could then be sent from this layer and from the *Linking* module to provide more information on traffic segments. Eventually, to address the problem of "inter-neighborhood" coordination, the result of the coordination realized on the blackboard of a specific neighborhood could be further coordinated in a second level of hierarchy, which would use the mobile agent with the highest priority to represent this neighborhood. This second level of hierarchy would gather mobile agents from a larger neighborhood, similar to road segments, and would resolve the problem of coordination

for interconnecting neighborhoods, thus ensuring their safety.

In the decentralized inter-platoon coordination model that has been presented, we are using one mobile agent per platoon, which is a solution that could lower communications by diminishing the amount agents travelling on the network. The mobile agent model also contributes to the growth of available bandwidth and the diminution of network latency [Lange and Oshima, 1999]. Finally, note that many details of implementation have not been settled at this time, since this is an initial proposition and the inter-platoon coordination problem is not the main focus of our research at the present time.

5.2 Intra-Platoon Coordination Model

The intra-platoon coordination is handled by the *Networking* module described in Figure 4.2. This module receives orders in the form of manoeuvres to execute, from our architecture's *Linking* module. To realize the task of coordinating neighboring vehicles' manoeuvres, the *Networking* module can be developed based on three possible models described below: (i) centralized intra-platoon coordination; (ii) decentralized intra-platoon coordination; and (iii) teamwork intra-platoon coordination. Briefly, each intra-platoon coordination model has to support the state transition shown in Figure 5.3. According to this figure, a coordination model has to enable a vehicle, driven manually or inside a platoon, to change lane, thus leaving its previous state, and safely reach the other lane, thus acquiring a new state: *Manual Driving* or *In-Platoon Driving*. Note that at the moment, the lane changes are considered as automated driving modes, but they could also represent the simulation of a human driver's lane change, if we estimate that this task is too dangerous to be automated.

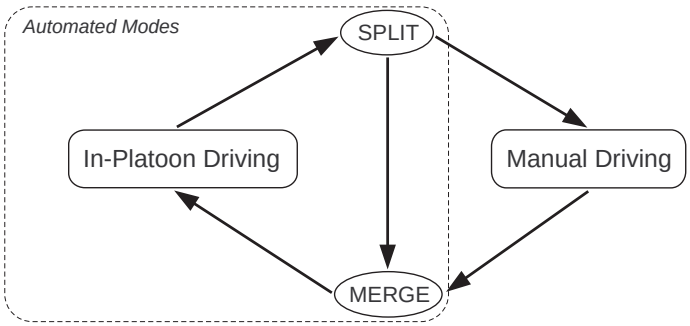

Figure 5.3: Vehicle state transitions handled by the intra-platoon coordination.

Because the coordinated driving actions involved in a merge or split manoeuvre are

directly linked to our architecture's distributed *Planning* sub-layer, we defined a comparison of possible intra-platoon coordination approaches from Durfee's representation of distributed planning [Durfee, 1999], presented in Section 2.2. As a brief recapitulation, Durfee defined planning models for Multiagent systems as either: *centralized planning for distributed plans*, *distributed planning for centralized plans*, *distributed planning for distributed plans*. The first, *centralized planning* model can be compared to coordination models used so far for platoons architecture centralized on the leader, as it is the case in the PATH project [Varaiya, 1993]. Within the *centralized planning* model, the distributed plans can include synchronization actions, leaving more flexibility to the plan executors, as it was done in a recent version of PATH's architecture [Bana, 2001]. On the other hand, the fully decentralized *distributed planning for distributed plans* can be implemented using a novel approach to inter-vehicle coordination in CDS: teamwork for driving agents [Hallé and Chaib-draa, 2004]. For a better understanding, four models reaching from *centralized planning for distributed plans* to *distributed planning for distributed plans* are presented in Figure 5.4, where the inter-vehicle communication involved in each model is highlighted. This figure briefly represents the fact that the decentralized model uses a minimal number of agents and the teamwork model uses most agents that communicate depending on the situation. Note that for each of these approaches, the guidance and control systems are decentralized for every vehicle involved [Huppe et al., 2003].

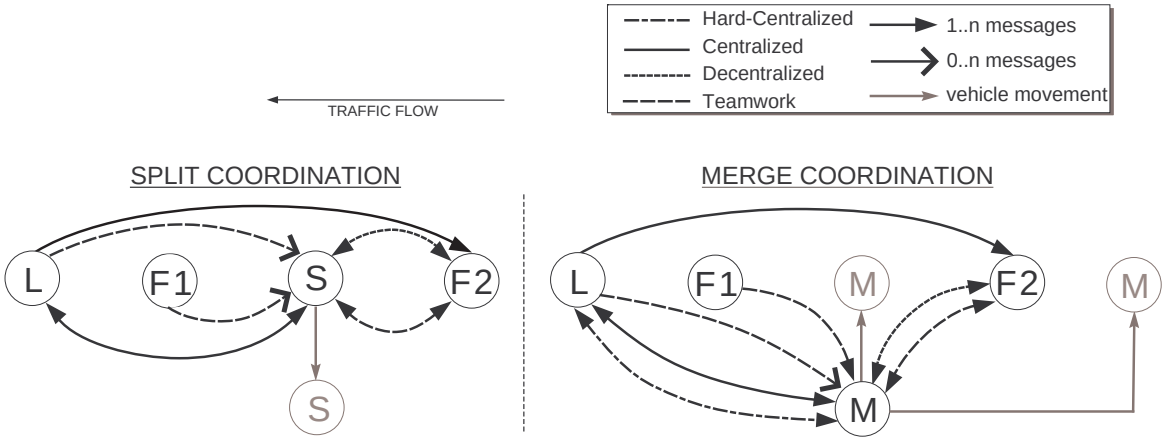

Figure 5.4: Four coordination models of the merge and split manoeuvres.

To highlight the differences between the intra-platoon coordination models presented in Figure 5.4, each model is formally represented by describing its communication policy, based the agent's belief state. Such policy represents the rules that force an agent to communicate in reaction to a specific belief about its platoon's dynamic state or another agent's intention to execute a manoeuvre. In order to present these communication policies, the following notation relating to an agent's possible states and communication protocols has been defined:

- An agent's platoon's identification p: $P = \mathbb{N}$
- An agent's intra-platoon manoeuvre υ, that is executed in platoon p (where S=Split, F=Follow, M=Merge): $\Upsilon^p = \{S, F, M\}$
- An agent's position ω inside a relative platoon p: $\Omega^p = \{2, 3, ..., MaxPlatoonSize\}$
- An agent's role γ in a team θ (for the teamwork model): $\Gamma^\theta = \{Splitter, Observer, Gap\_Creator, ...\}$
- An intra-platoon coordination model λ (where C=Centralized, D=Decentralized, and T=Teamwork): $\Lambda = \{C, D, T\}$
- A communication protocol $\sigma_{\lambda_\upsilon}$ that can be executed to coordinate a manoeuvre υ using a model λ: $\Sigma^\lambda = \Sigma^C \times \Sigma^D \times \Sigma^T = \{\sigma_{C_S}, ...\} \times \{\sigma_{D_S}, ...\} \times \{\sigma_{T_S}, ...\}$

According to this model, the *Networking* module's reasoning about other vehicles' intentions can be represented by using a simple communication policy as: $\pi^\lambda_{i\Sigma} : B^\lambda_i \mapsto \Sigma^\lambda_i$. Where B^λ_i represents the agent i's belief state, which is defined according to each coordination model λ's specific considerations. It must finally be mentioned that the position defined by Ω^p considers the vehicle's global longitudinal position. Such a representation was required to determine what should be the vehicle's position in the platoon; this way, the vehicle does not need to be in the same lane as the platoon, to determine its relative position (in the case of a merge).

5.2.1 Centralized Intra-Platoon Coordination

A centralized intra-platoon coordination means that the task of communication executed to coordinate the vehicle formation is centered on one vehicle: the leader. In this case, the leader is the head vehicle of the platoon, and as mentioned earlier, it is driven by a human (simulated) in our first phase of development. To maintain the platoon formation, the leader is the only entity that can give orders, in which case the followers only apply requested changes. During a "centralized" split manoeuvre, three vehicles are involved: the leader, the splitter, the vehicle following the splitter (if it exists). During a merge, the same configuration of vehicles is involved: the leader, the merger, the vehicle which will follow the merging vehicle (if it exists). For both of these manoeuvres, the merger or splitter communicates its need to do a manoeuvre, and then the leader gives requests for inter-vehicle distance, change of lane, meeting point or velocity to respect, to vehicles involved in the manoeuvre.

For the merge manoeuvre, we have defined two sub-models of the initial centralized intra-platoon coordination. The first one simplifies the task and involves only two vehicles, by requesting the merging vehicle to always merge at the end of the platoon. We refer to this type of coordination as the *Hard-Centralized* model. In a second model, simply called the *Centralized* model, the leader specifies the optimal in-platoon merging position, considering the merging vehicle's position (parallel to the platoon). This way, the *Centralized* model involves three vehicles, if the merging vehicle's position is in front or farther than the platoon's tail vehicle.

According to the previous formalism, the leader i uses the following communication policy to collaborate with a merging or splitting agent j:

$$\pi^{C}_{i\Sigma} : P_i \times \Upsilon^{p}_{j} \times \Omega^{p}_{j} \mapsto \Sigma^{C}_{i} \tag{5.1}$$

To represent the application of policy 5.1, we show as a representative example, the leader's beliefs state, resulting in the execution of a communication protocol that coordinates the merge manoeuvre. In this case, an agent $L1$, leader of platoon 4 reacts to a merge request by using the communication protocol σ_{C_M} (C_M: Centralized Merge) with a manoeuvring agent ma, if its belief state becomes:

- $p_{L1} = 4$
- $\tau^{4}_{ma} = M$
- $\omega^{4}_{ma} \in [1, MaxPlatoonSize]$

It should finally be mentioned that the followers are restricted to apply requests from the leader in the centralized model. Therefore, their communication policy is a simple request-confirm protocol that does not need to be detailed.

5.2.2 Decentralized Intra-Platoon Coordination

In the concept of a decentralized platoon, the leader is still the platoon representative, but this is only for inter-platoon coordination. Thus, every platoon member has a knowledge of the platoon formation and is able to react autonomously, by communicating directly with others. An agent's common knowledge is initialized when it enters the platoon and is updated using the broadcasted information about new vehicles' merge or split (done at the end of such manoeuvres). This is similar to the platoon coordination of Sakaguchi et al. [2000] presented in Section 4.1.2, except that our model does not involve all the platoon members in the execution of a manoeuvre. Indeed, the agents in

our decentralized model are only informed at the end and start of a manoeuvre, whereas the model of Sakaguchi et al. [2000] informs each agent at every step of the manoeuvre through its token ring infrastructure. Our decentralized model is described below, by presenting the theory behind the decisions on communication of this model: social laws. The concept of social laws in the merge and split manoeuvres is briefly presented and followed by its theoretical representation, which highlights the differences between the beliefs used for the communications inside this model as opposed to the centralized and teamwork models.

In the decentralized intra-platoon coordination model, the leader is only in charge of maintaining the manoeuvres' safety by notifying others of any emergencies, similarly to the centralized approach. For the split manoeuvre only two vehicles are involved: the splitter and the vehicle following the splitter (if it exists). For the merge, once the merging vehicle has chosen a platoon, only two vehicles are involved as well: the merger, the vehicle which will follow the merged vehicle after the merge (if it exists). For these two manoeuvres, we eliminate the intermediate that was the leader, because every platoon members have the knowledge of their platoon configuration. Using social laws, the action of creating a safe gap for the split and merge manoeuvres is handled by a platoon member (a follower), which uses its own belief, without communicating with the leader. For instance, the intention of creating a safe gap will only be allowed (through social laws) for the vehicle at the right distance from the merging vehicle, while the other platoon members determine (through social laws) that it is not their task. Furthermore, the same reasoning applies to the vehicle behind the splitter, which, according to social laws, decelerates to let the vehicle leave the platoon, following from its intention to split.

In our model of decentralized intra-platoon coordination, we refer to strategies on social laws that restrict or dictate the agent's behaviors [Shoham and Tennenholtz, 1995] and its relative plans. These laws can then determine an agent's possible communicative action(s) when the agent's belief about another agent's intention is modified. The decision on communications, based on social laws, is represented using the previous policies' formalism:

$$\pi^{D}_{i\Sigma} : P_i \times \Upsilon^{p}_{j} \times \Omega^{p}_{i} \times \Omega^{p}_{j} \mapsto \Sigma^{D}_{i} \tag{5.2}$$

In this case, and as opposed to policy 5.1, the agent i can be any vehicle that uses the beliefs about its own state and the state of the agent initiating the manoeuvre (agent j). By relating to the example of the merge manoeuvre, an agent ag's communications can be determined by agent ag's state and the merging agent ma's state, in accordance with the policy of the decentralized model $\pi^{D}_{i\Sigma}$. Hence, agent ag being in platoon 4, will communicate with the decentralized merge protocol σ_{D_M}, if its belief state becomes:

- $p_{ag} = 4$

- $\tau^4_{ma} = M$
- $\omega^4_{ag} = \omega^4_{ma} - 1$
- $\omega^4_{ma} \in [1, MaxPlatoonSize]$

For a better understanding of the mechanics behind the social laws and how they relate to the communication policy 5.2, a description according to the formalism of Shoham and Tennenholtz [1995] is finally presented. As Section 2.2 described it, a social law's action $a \in A$ is defined as a communicative action or protocol that agent i uses with agent j. $\varphi \in \mathcal{L}$ is the social law's set of general sentences, which in our application, can represent a sentence about the position of agent i's vehicle, relative to agent j's vehicle. Finally, $s \in S$ represents in our application, the states of the driven vehicle and neighboring vehicles and $sl \in SL$ are different social laws restricting vehicles that should collaborate in this manoeuvre. Again, in order to relate to the social laws' representation of sentences, states and actions, refer to Section 2.2.

If we use the social law formalism to represent the decision to communicate σ_{D_M} in our previous example, the steps leading to a decision could be seen as follows. First, the opposite of the initial state of agent ag ($\omega^4_{ag} \neq \omega^4_{ma} - 1$ instead), let say state s_1, would satisfy a more general sentence φ_1 in $\mathcal{L}$. This sentence could have the form of *Manoeuvring Vehicle Is, or Will Be, in Front of my Vehicle*, which would relate to different belief states, including s_1. In this case, $s_1 \models \varphi_1$ would hold, and a social law sl_1 describing the previous policy 5.2 would accept $(\sigma_{D_M}, \varphi_1) \in sl_1$, which regulates the decentralized communication protocol for the merge. Finally, the transition $T(s_1, \sigma_{D_M}, sl_1) = \emptyset$ would lead non-collaborative platoon members to "no reactions" on the arrival of a message notifying them of ma's intention to merge. On the other hand, the agent that "should" collaborate in the merge manoeuvre (let say agent ca) would apply another transition T, referring to the policy defined by $\pi^D_{i\Sigma}$. This transition would lead the agent ca to the execution of σ_{D_M}.

In conclusion, the policies based on social laws make the intra-platoon coordination simpler, as agents are largely restricted in their messaging possibilities and very few agents are included in the coordination process. The restrictions could be relaxed by detailing the social laws, thus enabling more agents to intervene in the coordination process, which could increase safety. On the other hand, as this is an initial proposition for the decentralized model, it will be kept as is, to show the difference of using a low communication model. Using a policy relating to social laws, the decentralized model is not very flexible and the addition of new laws may reach its limit, get too complex or raise conflict issues.

5.2.3 Teamwork Oriented Intra-Platoon Coordination

The decentralized model, leads us to the development of a more organized decentralized concept, which is the model of teamwork, gaining in popularity in the field of Multiagent System. In order to develop teams of agents for the Auto21 CDS, we had to provide a Team Oriented Programming (TOP) [Pynadath et al., 1999] infrastructure that enables agents to be assigned roles, as part of predefined team structures. Consequently, the STEAM architecture was picked to model our own infrastructure, because it provides a hierarchy of roles and team operators that can be monitored to ensure safety during tasks execution Tambe and Zhang [2000]. Moreover, the STEAM architecture provides domain-independent directives that enable our driving agents to support responsibilities and commitments to the platoon (team) manoeuvres.

This section first presents a theoretical representation of the teamwork model, which should provide a better understanding of the decision on communication aspects made in the teamwork model, in comparison with the centralized and decentralized models. Following the presentation of the theoretical teamwork policy, the team formations used for tasks involved in our CDS are described, to end with the presentation of some examples relating to the use of STEAM's architecture in the Auto21 project.

Teamwork Theoretical Representation

The theoretical communication policy of the teamwork coordination model can be represented using the formalism introduced earlier. However, the policy of the teamwork oriented coordination is different from the previous models (centralized and decentralized) because the belief state of an agent results in an intention for a role in the team and not directly in a protocol of communication Σ, like the previous policies. Hence, agent i's intention for a role in a team Θ, initiated by agent an j in platoon p, is defined by:

$$\pi_{i\Theta}^{T} : P_i \times \Gamma_j^{\theta} \times \Upsilon_j^{p} \times \Omega_i^{p} \times \Omega_j^{p} \mapsto Intention(\Gamma_i^{\theta}) \tag{5.3}$$

Compared with policies 5.1 and 5.2, policy 5.3 requires more beliefs about other neighboring agents (team members) to take a decision relating to communications. Therefore, the teamwork model involves more communications at the beginning and the end of a manoeuvre to ensure that the beliefs required by the teamwork communication policy are shared by all team members.

Following from an intention towards a role, agent i requests this role to other team members, using a simple request and confirm protocol. If this role assignment is ac-

cepted, agent i starts communicating with other team members, according to its team oriented communication framework. The STEAM architecture defines a generic communication policy, which is apart from the domain level (CDS) and is further detailed in this section. Therefore, the teamwork coordination model includes two policies on communication that can be defined as $\pi_{i\Sigma}^{TR}$ for the role request policy and $\pi_{i\Sigma}^{TS}$ for STEAM's team communication policy:

$$\pi_{i\Sigma}^{TR} : Intention(\Gamma_i^{\Theta}) \mapsto \Sigma_i^T \tag{5.4}$$

$$\pi_{i\Sigma}^{TS} : \Gamma_i^{\Theta} \times B_{\Theta}^T \mapsto \Sigma_i^T \tag{5.5}$$

The policy 5.4 represents the role request protocol, which, in our case, is a request and confirm protocol. This policy depends on the TOP framework and is only used to assign roles during the team formation. The second policy (5.5) is triggered once the agent i is part of the team Θ and it is used to select communications based on the team's belief state B_{Θ}^T. The selection of messages is realized by using STEAM's Selective Communication (SC) actions (Section 2.2.4), which are detailed in a CDS perspective, in this section. Note that a third policy could have been defined for the team initiator (i.e. merger, splitter), to represent its intention to form the team. It would basically be the same as $\pi_{i\Theta}^T$, except that it would only use local beliefs. But since the intention to form a team emerges from the *Planning* sub-layer (as shown in Section 4.4.4), it is not concerned by the networking policies described here.

Auto21 Team Formations

For the Auto21 project, we have defined three major teams: the "platoon formation", the "split task" team, the "merge task" team. The first team, is a persistent team, using persistent roles, for long-term assignments as it is the case for the platoon formation. The two latter are task-teams using task-specific roles, for shorter-term assignments, as those teams do not exist after the task completion. Figure 5.5 illustrates the formation used for a "split task" team, where the leaf nodes represent roles and the internal node represents a sub-team of observers, which is detailed later. Moreover, Figure 5.6 depicts the operators used by these formations in a tree, similar to Soar's plan hierarchy [Rosenbloom et al., 1991]. In the tree's hierarchy, team operators are surrounded by brackets (i.e.[]), while the other operators are standard individual operators.

The operator tree is finally used to manage team tasks execution by constraining each roles to specific operators inside this hierarchical tree. In this context, agents evolving in the team infrastructure are able to execute two possible types of operator:

domain level, as described in the tree of Figure 5.6, and architecture-level (STEAM operators), as defined in Section 2.2.4.

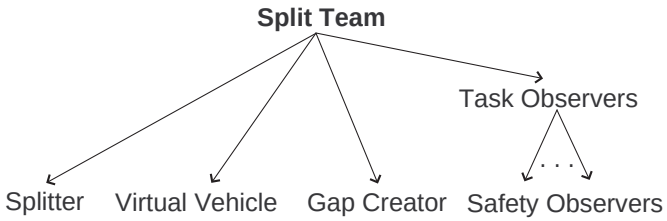

Figure 5.5: Split task team's role organization.

The "platoon formation" team is the simplest team, where every agent holds the intention of maintaining a stable and safe platoon formation, which regulates their communications on critical or unsafe situations. This team is the simplest one since, at the moment, we consider that each platoon member has the same task, which consists of following the front vehicle in a safe manner. Hence, this formation only requires two persistent (long-term assignments) roles:

- *A leader* which is filled by the head vehicle that mainly communicates with others using SC actions (refer to Section 2.2.4). Since the goal here, is to maintain a stable platoon formation, an unsafe deceleration can be seen as a percept that could endanger the goal achievement, therefore influencing the leader to inform others of this fact using SC actions. The probability of such a communicative act is discussed later.
- *Follower* is a role filled by all the platoon members that are not at the head. At the moment, each follower's goal is to maintain the safe inter-vehicle distance with the preceding vehicle. Since we managed to keep the platoon stable during platoon driving scenarios that did not include any vehicle entrance or exit, an agent in this role does not communicate to others. Thus, the task of maintaining a safe distance is realized by using the vehicle's front sensor and possible information from the leader.

The "merge task" team being similar to the "split task" team, we only depict the merge. This team is centered around the [Insert vehicle] team operator, shown in Figure 5.6, which is executed either as an insertion at the end or an insertion within the platoon. The insertion at the tale being simpler, we did not detail its branch in the operators' hierarchy tree. As shown in Figure 5.5, there are four different roles and one sub-team involved in the split (similar to merge), which are detailed below. In order to identify the agents relating to vehicles involved in the merge and split manoeuvres, Figure 5.7 is used to describe the different agent roles used inside the "merge task" team.

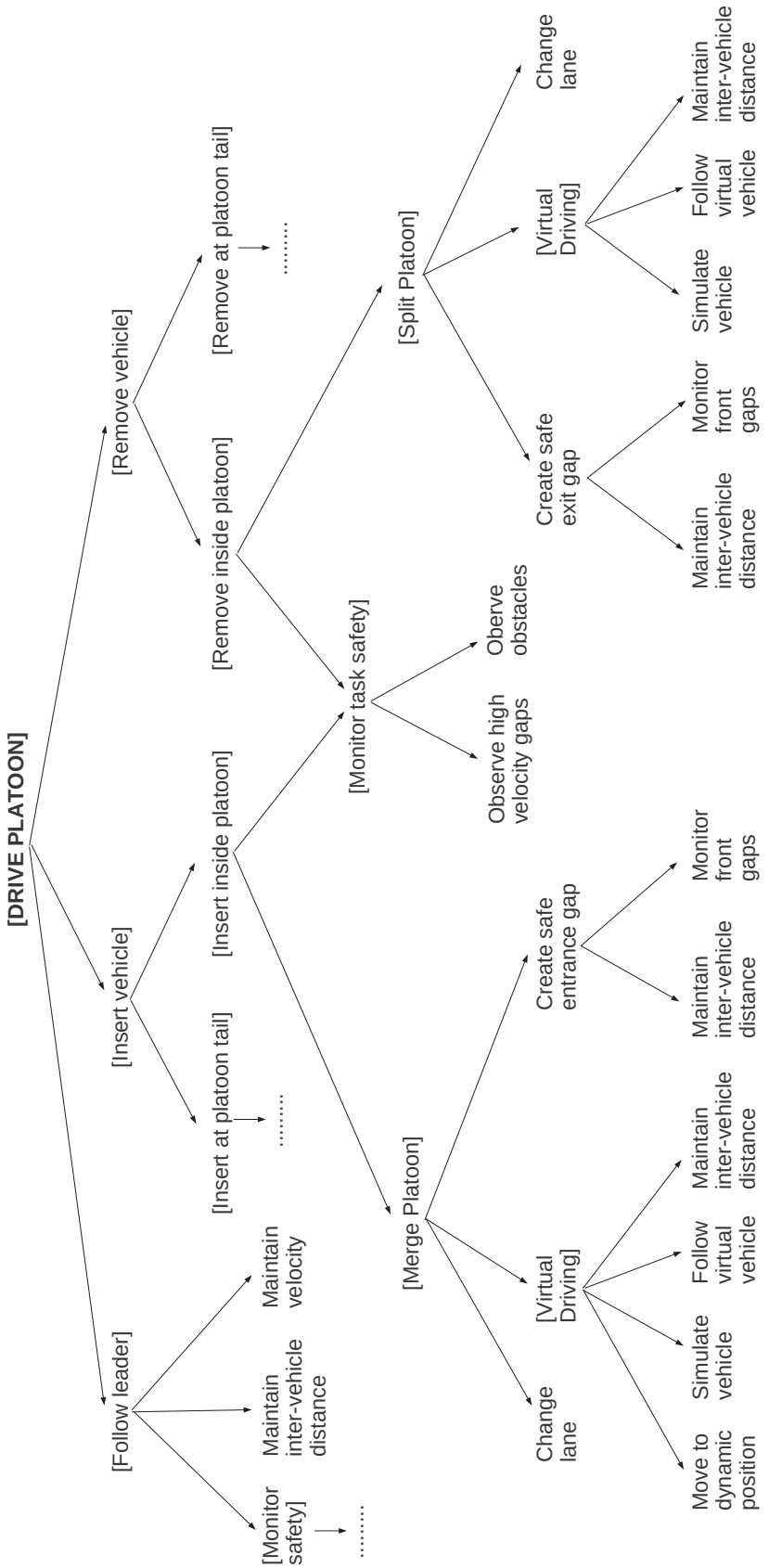

Figure 5.6: Platoon team operators tree.

- *A Merger* is the role filled by the agent which initiates the "merge task" team by broadcasting its will to merge a platoon (vehicle *L2* in Figure 5.7). The operators restricting the merger's actions are the ones within the [Merge Platoon] team operator. The Move to dynamic position operator is used by the merger when the task team has the belief about the entry position for the merging vehicle. The *Merger* role also uses the Follow virtual vehicle operator, which is a virtual representation of *L2*'s future preceding vehicle (*F1*). This virtual vehicle is followed by *L2* before it actually senses the real vehicle with its laser. Finally, the Change Lane operator is used here, to switch to the platoon's lane and complete the merge. When the merger is stable in the platoon, a Coherence Preserving (CP) action is broadcasted in order to manifest the achievement of the team's goal. Considering those operators, the *Merger* role has an "AND" logical relation with the rest of the team, as defined for teams in STEAM (Section 2.2.4), since its performance is crucial to this manoeuvre.

- *Gap Creator* is a role taken by the agent driving the vehicle behind the merging position, in the platoon (vehicle *F2* in Figure 5.7). Within this role, an agent defines the entry position for the merger, since its vehicle will be behind the merger after the lane change. The *Gap Creator* role requires its filler to execute the Maintain inter-vehicle distance operator, which maintains a distance large enough to safely fit a vehicle. Then it has to execute the Monitor front gaps operator when the merger is changing lane. A high gap between the last front vehicle percept reading indicates the arrival of the merging vehicle in the platoon. This is followed by a new inter-vehicle distance goal. The *Gap Creator*'s operators are included in the [Merge Platoon] team operator, since the aforementioned individual operators are directly linked with the merger's operators execution. Considering the specifications on role relations for our STEAM oriented team, this role is also in an AND relation.

- *Virtual Vehicle* is a role that was introduced to ensure a stable task execution. This role helps the task executor, when it is in a different lane, to follow the vehicle that was or will be in front of it. In the split and merge manoeuvres, this role is taken by vehicle number *F1* from Figure 5.7. Within the [Split Platoon] team operator, the *Virtual Vehicle* role applies the Simulate Vehicle operator that results in the communication of information about its velocity, if it is modified after the splitting vehicle has changed lane and before the split manoeuvre is over. Within the [Merge Platoon] team operator, the same operator is applied after vehicle *F1* has transmitted an initial representation of itself to the merging vehicle. This role thus ensures a safe entrance of the merger and eliminates the need to create a virtual representation of the merger to help its future preceding vehicle (*F2*). Indeed, since vehicle *F2* is filling the *Gap Creator* role, which forces him to respect a safe gap with *F1*, and the merging vehicle is also keeping distance with the virtual representation of *F1*, both have the same reference. Thus eliminating the need to create a virtual

vehicle for the *Gap Creator* (*F2*). As its predecessor, the *Virtual Vehicle* role is in an AND relation for the [Insert platoon] operator.

- *Safety Observers* is a role taken by one or more agents. The constraint on the role fillers, is that they must be in a position ahead from the task executor, so they can monitor dangers in advance. Using the communication selectivity presented next, agents in the *Safety Observers* role communicate their belief about dangers or unsafe deceleration to others, by taking in account the dangers of sudden movements during a task execution. Agents filling in this role conjointly execute the [Monitor task safety] safety team operator, therefore executing observation plans individually. The *Safety Observers* role can be filled in by multiple agents, which have an OR relation. The combination of the *Safety Observers*' operators with the rest of its team is done using a ($\Longrightarrow$) role dependency. This means, that the execution of the three precedent roles is crucial to achieve the goal and maintain this role, but the execution of the *Safety Observer* role is not critical.

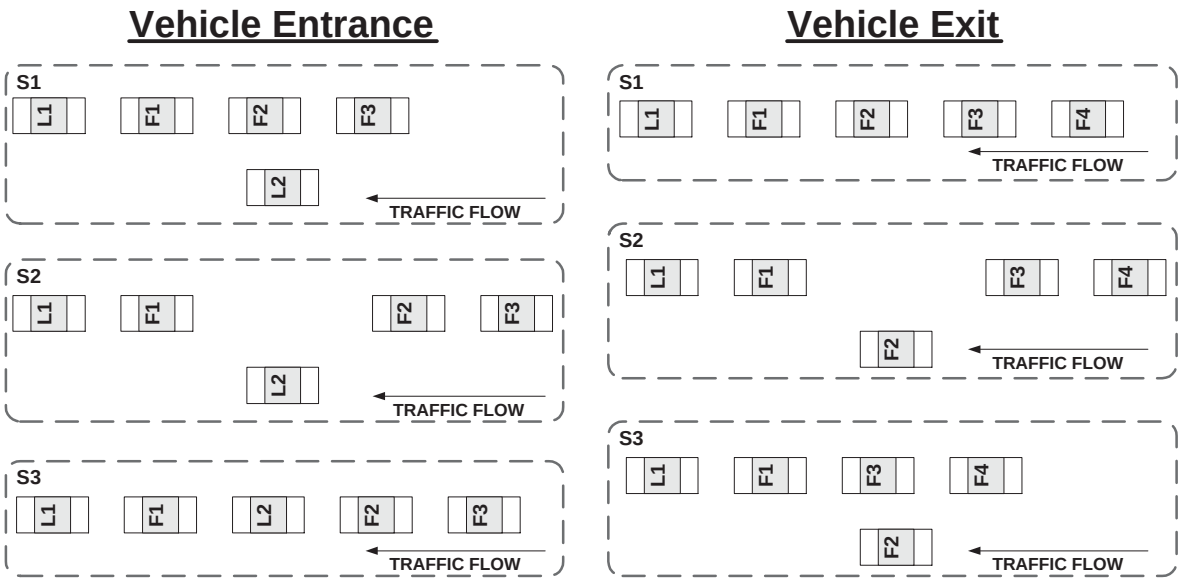

Figure 5.7: The three steps of the removal (split) and insertion (merge) of a vehicle in the platoon.

Intra-Team Communications

The Selective Communication (SC) actions taken from STEAM are used to synchronize mutual beliefs within the execution of team operators. For the precedent "merge task" team, the agent that fills in the *Virtual Vehicle* role communicates changes to its virtual representation (a new velocity for example) if it believes that the merging

vehicle does not pick him up on its sensor and that this velocity change is important enough. To illustrate this situation, Section 2.2.4 already described a decision tree referring to probabilities about these beliefs, in Figure 2.6, which represented the Selective Communication (SC) of the STEAM framework [Tambe and Zhang, 2000]. As opposed to the general application of SC actions proposed in STEAM, we do not use them for task team formation and dissolution. This means that SC actions are not used for the CP actions, which notify of the [Insert vehicle] and [Remove vehicle] operators' achievement or creation. Thus, the purpose of SC in our model is to apply a selection over the communication sent to maintain mutual beliefs during the execution of a team operator. By relating to the previous example, the agent filling the *Virtual Vehicle* role uses the tree defined in Figure 2.6 to make a decision on whether it will communicate an update on its position, thus synchronizing the team's belief on its virtual representation. This is reflected by the function defined in Section 2.2.4 which takes in consideration the expected utility of communicating and not communicating:

$$EU(C) > EU(NC) = \rho * \sigma * C_{mt} > C_c + (1 - \sigma) * C_n$$

In the merge example, the cost for communication C_c is higher than normal, since more communications are involved during a manoeuvre and we do not want to saturate the network. Then, if the vehicle *F1* has to modify its velocity during the merge, the probability ρ that this new information on *F1*'s velocity is commonly known, mainly depends on the probability P(*L2*,*F1*) that the merging vehicle *L2* has *F1* in its sensor's range (if it is in the platoon). Furthermore, probability σ that this information opposes a threat to the merge manoeuvre depends on the difference between *F1*'s knowledge about its velocity and the team's belief. If the team is highly out of synchronization, the agent communicates at a higher probability. As a final note, it should be mentioned that the C_{mt} and C_n costs are set to an average-low value for this type of task.

Since the decision-theoretic communication selector is available through the SC actions, part of the TOP framework, we are using it within all of the teams and roles presented earlier. Team knowledge relating to sub-operators' pre- and post-conditions is awarded a great value for σ and C_{mt} to insure the communication of this type of information, even though agents may have doubts on the team's belief about it. Moreover, these probabilities should be adapted through testing, like it has been done for domains like the RoboCup soccer challenge [Nair et al., 2004], where data traces of team behaviors were used to learn their probabilities' distribution. This approach proposes an offline learning approach on patterns of communication within the team that can be applied to specify the probabilities of the SC operators.

5.2.4 Discussion

The three to four networking models previously presented for intra-platoon coordination are the main topic of our research on automated vehicles communication systems, so they should be enhanced in the following phases of this project. As presented in Chapter 4, the intra-platoon coordination takes place inside our architecture's *Management* layer, which makes it dependent on the sensing, control and planning systems. Considering that we only rely on the fact that the *Guidance* layer provides the required environment state information and driving functions to the *Management* layer, our architecture gives us much flexibility on the choice of model that will finally be used to develop our *Networking* module.

As part of the proposed coordination models, the centralized model is the easiest one to implement, because its coordination functions can be developed for the leader only, while other agents just have to provide the functions to execute the leader's orders. As it was shown in policy 5.1, the centralized communications only require one agent (the leader) to have beliefs about its platoon members in order to coordinate split and merge manoeuvres through communications. However, the centralized system does not provide much flexibility if the followers require more autonomy in their driving actions. The decentralized model is more complex considering that the agents have more autonomy and there must not be any conflict in their intentions, since the execution of their driving task is done in real time. While the goal of the decentralized approach is partly to lower the communications between vehicles (by eliminating the leader), a safe driving behavior also needs to be guaranteed, although this may raise the number of exchanged messages in problematic situations. As policy 5.2 highlights it, each platoon members must keep beliefs about their own state and other platoon members' states in order to support the decentralized communication protocols Σ^D. The teamwork approach, on the other hand, is the most complex model, but it can be developed easily if the frame of the TOP infrastructure is available. This approach makes the overall system highly flexible, as agents already have a high value of autonomy, but again, this may be a problem considering different role allocation or possible reallocation. At last, note that this model and the decentralized model promote a communication system with the ability to broadcast easily in the vehicle's neighborhood, since the group or team surrounding the vehicle requires an up to date belief state about their neighbors at all time in order to support the communication policies 5.2 and 5.3.

Chapter 6

Driving Agents Engineering

Software engineering is a very important aspect in building highly flexible agents that can be modified to test and improve our driving system. This type of engineering is realized through an analysis phase from which emerges models and diagrams representing different development phases in a standardized notation like the Unified Modeling Language (UML). UML is an appropriate choice to describe Java classes issued from the object-oriented paradigm, but in order to describe agent-oriented concepts, another notation is required. In this chapter, we first introduce an analysis of the Multiagent System (MAS) we developed, by using the newly adopted standard of Agent UML (AUML)[1]. This introduction provides a good understanding of our driving agents' general structures and their flow of activities when they coordinate with other agents. Following from the AUML analysis, details are provided on the agent-oriented software engineering, by relating to our driving agents' implementation in the JACK language. This should provide a good understanding of the tools and infrastructures that have been developed to support the architecture requirements presented in Chapter 4. Finally, this chapter details the infrastructure that was developed in JACK to support the teamwork coordination model presented in Section 5.2.3.

The presentation of the agent-oriented engineering in the Auto21 project begins with the AUML models in Section 6.1, followed by the JACK-oriented models in Section 6.2, and the teamwork-oriented models in Section 6.3. To end this chapter, simulation results on the driving agents' behaviors in platoon formations is presented in Section 6.4 by focusing on the centralized and teamwork models of intra-platoon coordination.

[1]For more information on AUML, visit http://www.auml.org

6.1 Multiagent System Modeling

The analysis of the *Management* layer of our architecture (refer to Section 4.2.2) is now presented using a new standard brought by Foundation for Intelligent Physical Agents (FIPA), which is called Agent UML (AUML). AUML extends a series of diagram types that most developers are familiar with, from working with UML. Three levels of abstraction have been defined as part of the Agent Interaction Protocols (AIP) and are presented in their application for our Collaborative Driving System (CDS)'s agents. The first level includes the *agent diagrams* similar to *class diagrams*, the second level includes *state* or *statechart diagrams*, *collaboration diagrams*, *protocol diagrams*, *activity diagrams* for the whole Multiagent System (MAS), and the third level includes different *state diagrams* depicting a single agent's possible states transitions. AUML's suite of diagrams has been used to model and engineer our architecture's *Coordination* sublayer (mostly for the *Networking* module), which was introduced in Chapter 4, and the resulting diagrams are presented in this section. In the following examples, the focus will be on the centralized coordination, since it presents a more understandable view of the behavior of the intra-platoon coordination.

6.1.1 Agent UML Level 1: Agent Model

The first level of AUML modeling defines the classes of agents available in our system and the functionalities they provide on the communicative or actuating actions level. Figure 6.1 represents a UML diagram that details the functions of the different agents on a "programming" point of view along with the classes hierarchy that highlights the polymorphism used for the implementation of these agents. The DrivingAgent interface, on top of the diagram, defines the basic functionalities an agent must implement to be used as part of our simulated automated vehicles, which are basically vehicle sensing and actuating requirements. This interface is further implemented by the abstract JackAuto21DrivingAgent class, which provides the actual implementation of generic driving agents functions as well as functionalities relating to the use of the JACK agents framework, as presented in Section 6.2. JackAuto21DrivingAgent also extends the Agent class from JACK, which enables our agents to be taken in charge by JACK framework's task manager. At last, different classes of agents that include specific actuating and communicating functions are shown at the bottom of this diagram.

To provide a representation which is more appropriate than Figure 6.1's UML diagram, we detailed our agents' functionalities and behaviors using an AUML *agent diagram*, which has been proposed by Bauer [2001]. As an example, Figure 6.2 represents

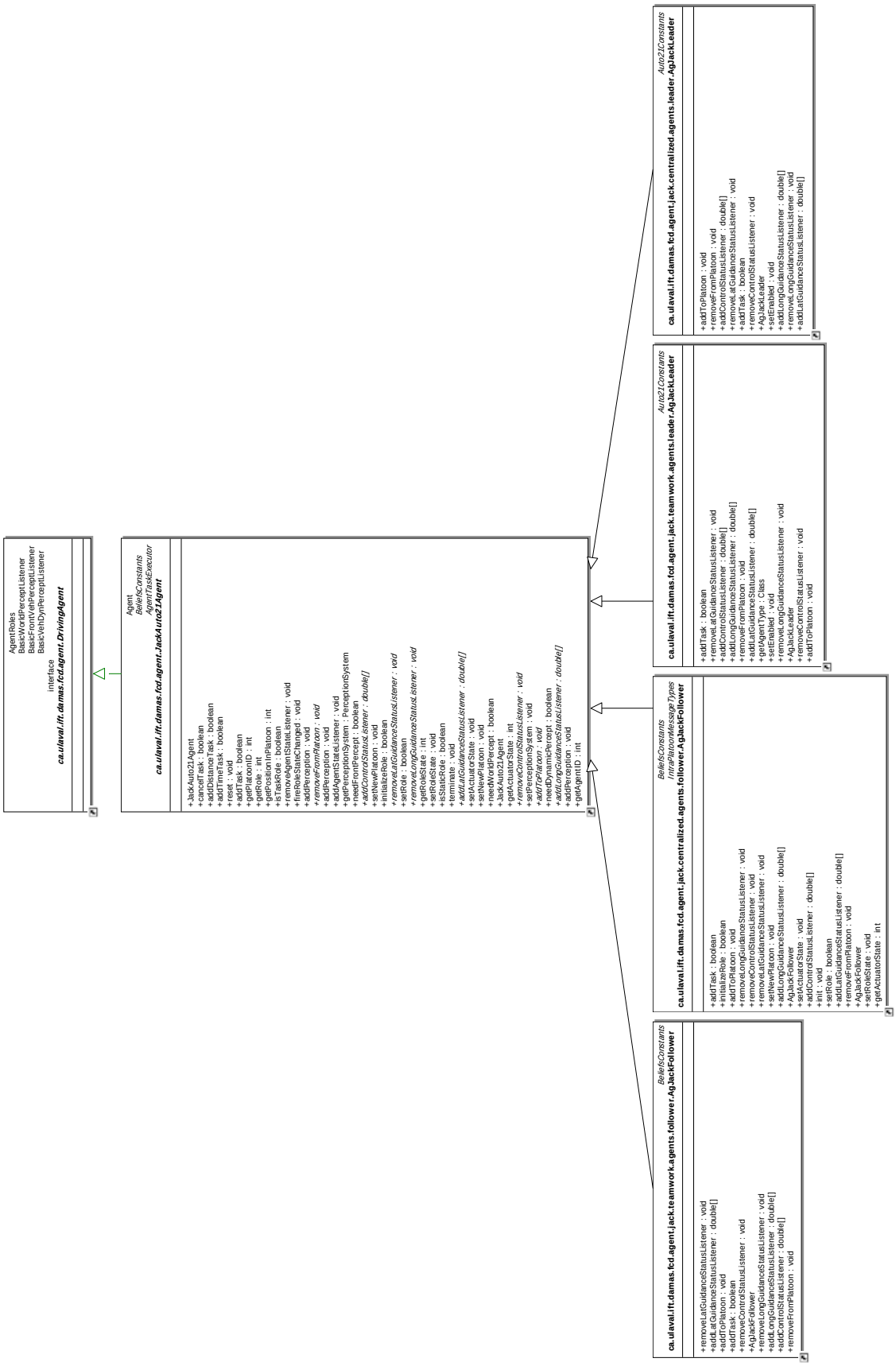

Figure 6.1: Class diagram of the possible JACK Agent deriving from a common abstract agent skeleton.

the *agent diagram* of a AgJackFollower agent (follower agent) which references same class, from the centralized package, presented in Figure 6.1. For a better understanding, Figure 6.2's diagram includes the links with other classes, as the SimulationRoadVehicle, which includes the agent's actuators. As an improvement to agent-oriented modeling, Figure 6.2's diagram defines the agent's possible roles at the top of the agent box: Follower, Splitter, Merger. Besides that, the agent attributes presented on this diagram are only the ones relating to the agent's state (belief state) and the functions beneath are actions that the agent can execute on its environment, while the other functions listed at the bottom are standard Java methods. The bottom of the agent structure represents the "library" of communicative acts the agent can execute as part of specific protocols. In this example three protocols from the centralized coordination model are presented with the messages used by "follower" agents.

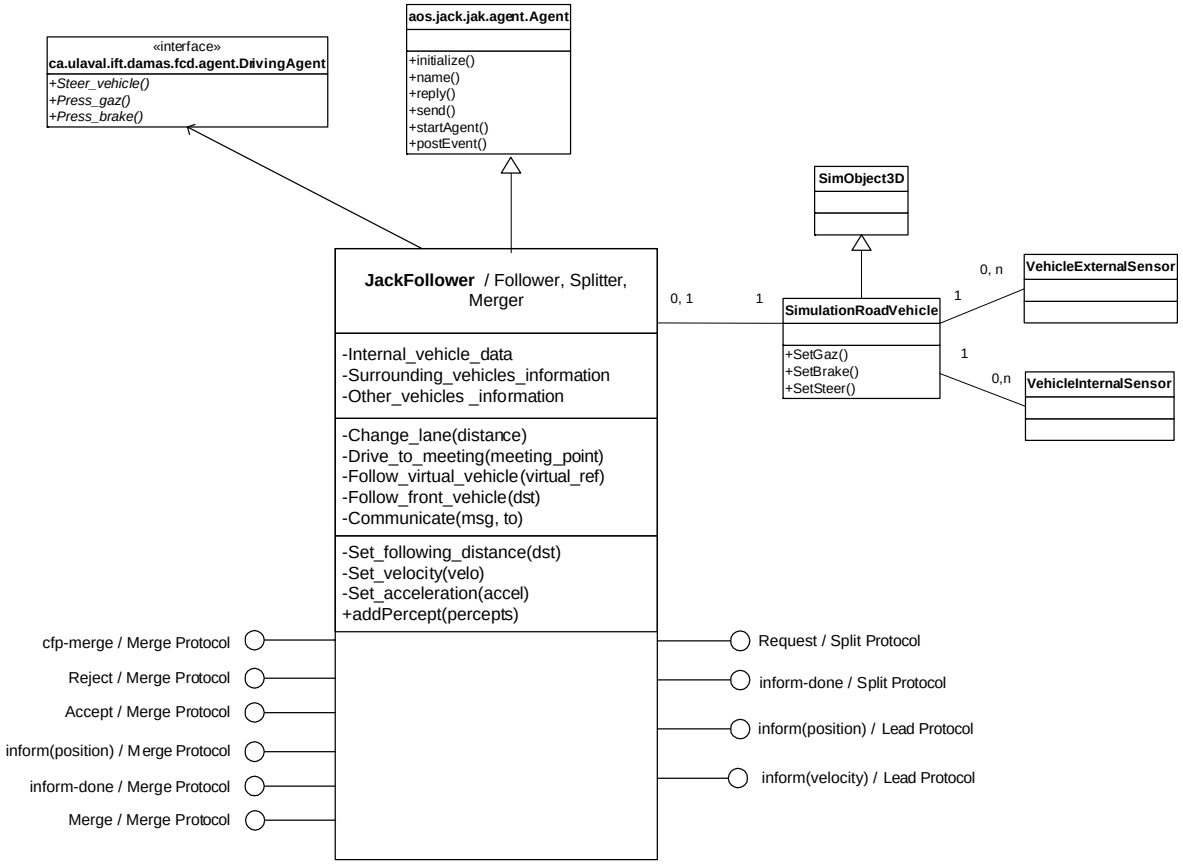

Figure 6.2: AUML agent diagram of the JACK follower agent (AgJackFollower).

6.1.2 Agent UML Level 2: Coordination Protocols Model

As the first level of AUML briefly defined the agents' structure, hierarchy and possible actions, the second level, proposed by Odell et al. [2000], is useful to highlight the

agents' collaboration process in reference to the communicative acts presented at the bottom of the *agent diagram*. The protocol that will be shown as an example is the merge protocol, which is richer in interactions and can relate to the split protocol, as the agents involved in the manoeuvre are basically the same. Only the level 2 *state diagram* and *protocol diagram* are presented here, since they provide all the information required to understand and model the coordination process, and the other level 2 diagrams only present different points of view about the same information. These two diagrams also provide a better understanding of the pre- and post-conditions surrounding the communicative actions of our driving agents. In this context, they ensure coherence between the state transitions of the intra-platoon manoeuvres and ensure their completion. For a better understanding of the following coordination examples, Figure 6.3 shows the vehicles and respective agents involved in the merge manoeuvre, where agent 1.x is the vehicle which creates the gap for the merging vehicle 2.1.

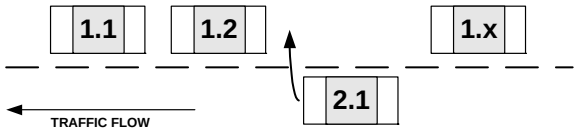

Figure 6.3: Agents' identifications for the merge example.

The *state diagram* presented in Figure 6.4 represents the conversation used to manage the merge manoeuvre, between platoon 1's leader and the merging vehicle 2.1 (refer to Figure 6.3). This diagram clarifies the behavior of our agents' *Networking* module by specifying the messages that must be sent in reaction to the belief state of the agents involved in the merge manoeuvre. The states of the merge manoeuvre are represented by the rounded shaped boxes and the arrows represent the possible messages, with the ID of its sender. These states dictate the agents' possible messages, while the state transitions occur following from such messages. In this example, vehicle 2.1 is *Free* until it sends a Call For Proposal (*CFP-Merge*) to a nearby platoon. At that moment, the conversation is in an *Open* state until the manoeuvre has been realized or someone stops it.

To model the flow of messages between the possible roles our agents can take, the *protocol diagram* is a convenient solution inspired from UML's *sequence diagram*, which has always been used by FIPA to model protocols. In our case, the *protocol diagram* has been used to model the three categories of intra-platoon coordination protocols presented in Section 5.2 and ensure the coherence of the inter-agent communications during the simulation scenarios. Figure 6.5 shows such a diagram, which represents the messages exchanged between the three agents involved in the merge manoeuvre, within their respective roles. It should be mentioned that our protocol extends the Contract Net template of protocol for Multiagent System [FIPA, 2002] and provides a practical

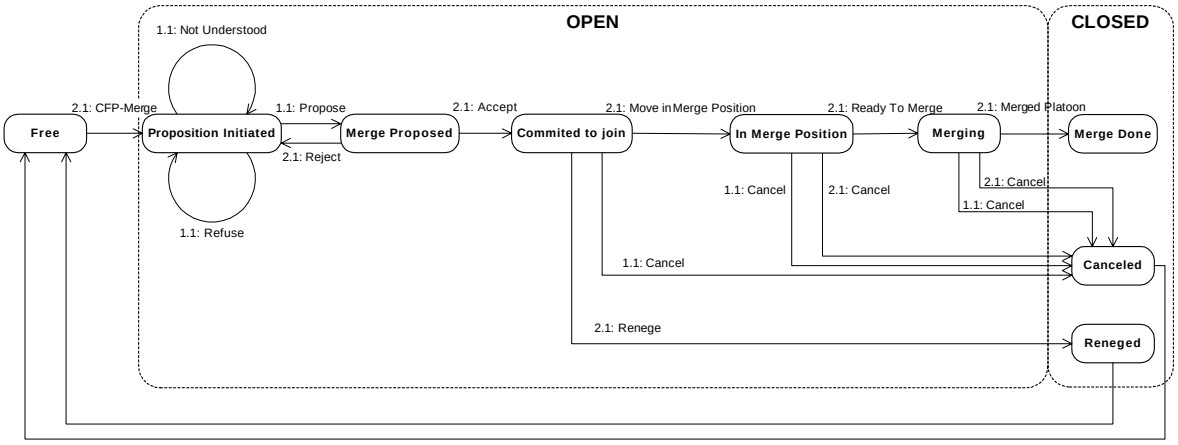

Figure 6.4: AUML Level 2 *state diagram* of the merge protocol.

way for agents to negotiate the merge manoeuvre participants, at the beginning of the collaboration. The different *lifelines* (vertical lines) represent the agent's roles and the *activation bars* (vertical rectangles) on the same lines refer to the agent's deliberative process. In this example, once vehicle 2.1 has accepted platoon 1, a specific role change occur for him and the leader agent. Furthermore, this role change also represents a switch between the *Linking* module of our architecture, which relates to the *Free* and *Platoon representative* roles, and the *Networking* module, which relates to the *Merger* and *Merge Organizer* roles. A final observation about Figure 6.5's *protocol diagram* is that it represents the bridge between the design of the *Networking* module and its implementation in JACK, which is detailed in Section 6.2. Indeed, the diamond boxes appearing on some transitions represent logical conditions part of an agent's plans, while the *activation bars* represent JACK's plans.

6.1.3 Agent UML Level 3: Agents' State Transition

The AUML level 2 diagrams that we just presented showed the coordination process of the complete MAS of our CDS, but did not detail each agent's "local" actions within the merge manoeuvre. The level 3 of AUML on the other hand, can provide a better understanding of the deliberative process that must be implemented inside a single agent's *Coordination* sub-layer, in relation with the *Planning* sub-layer. Apart from detailing the states an agent goes through, the level 3 diagrams also point out the preconditions leading to an agent's actions and state changes. Therefore, this type of diagram represents the guidelines that should be followed to provide reliable and robust transitions during the coordination and the driving process, supported by the agent coordination system presented in Section 6.2.6 and the agent driving system presented in Section 6.2.7.

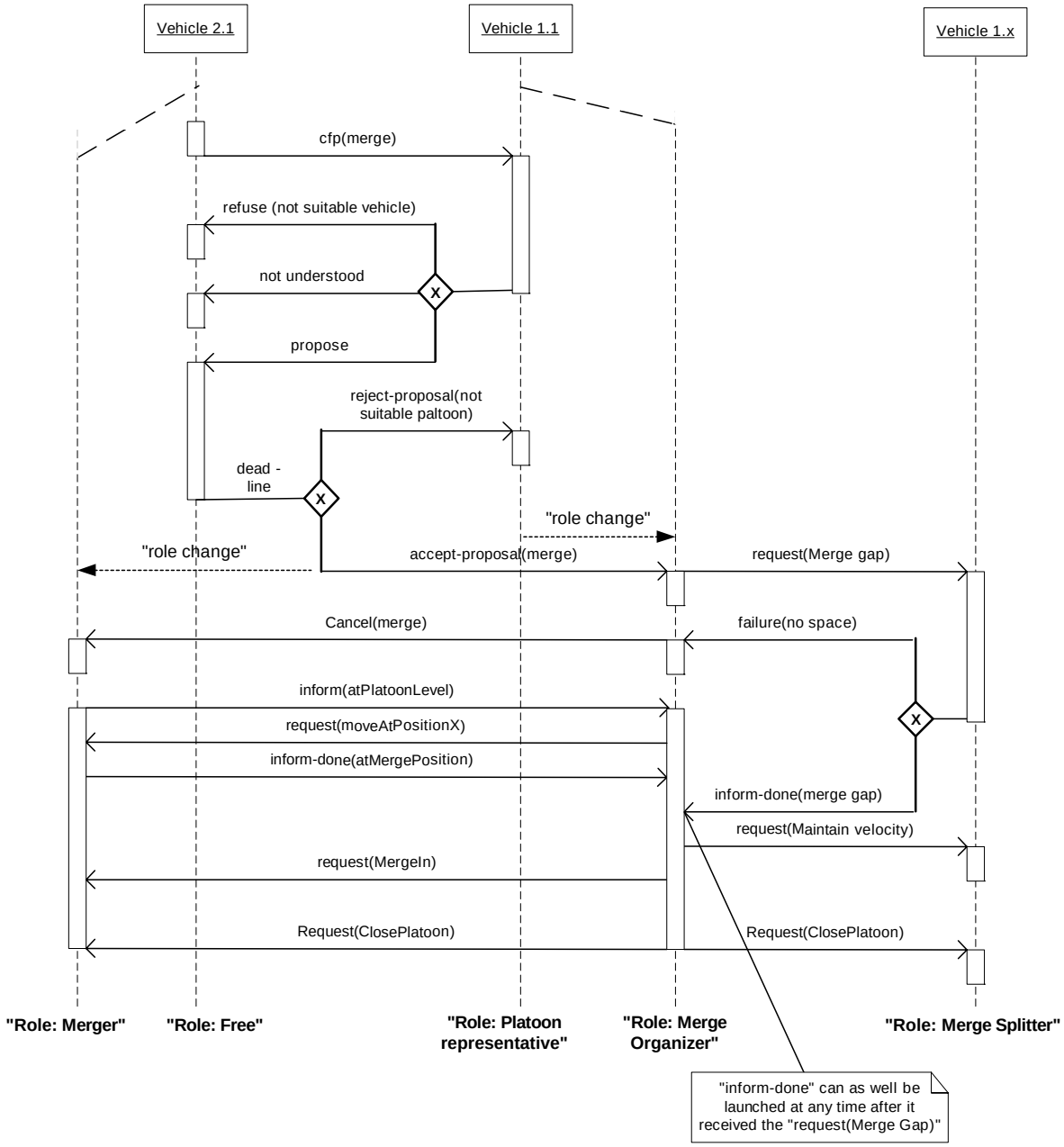

Figure 6.5: AUML Level 2 *protocol diagram* of the merge protocol.

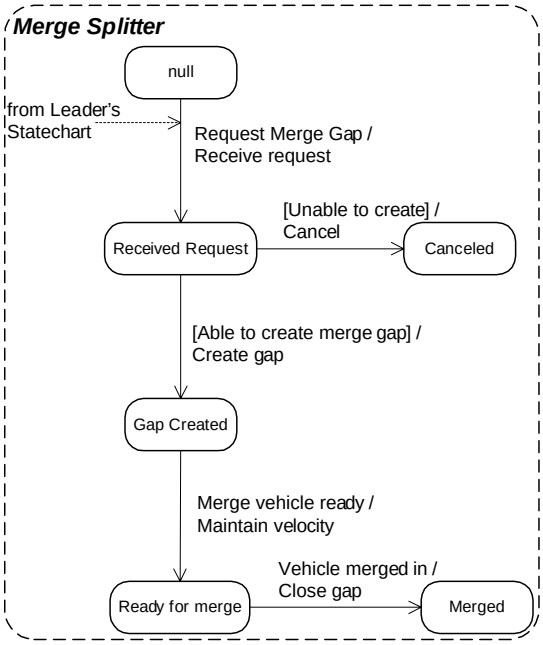

Figure 6.6: AUML Level 3 *state diagram* of the merge protocol focusing on the merger's follower agent (Gap Creator role).

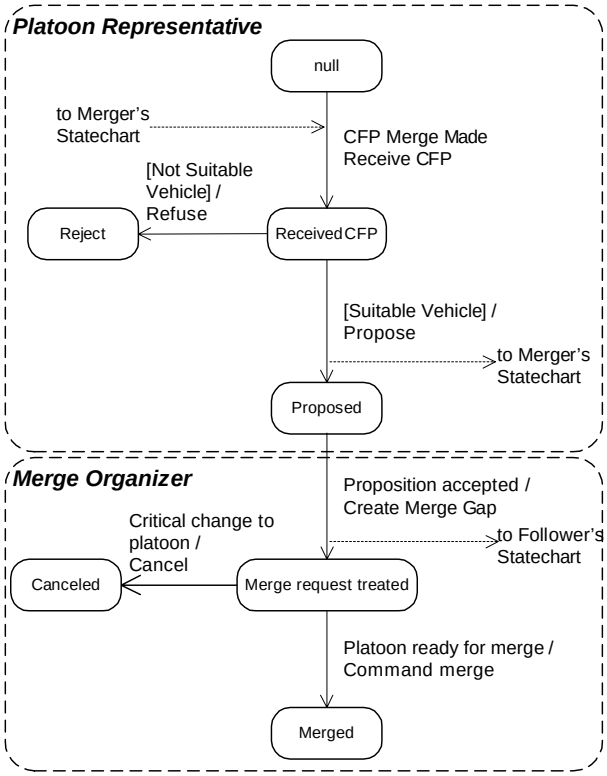

Figure 6.7: AUML Level 3 *state diagram* of the merge protocol focusing on the leader agent.

The three agents involved in the centralized coordination of the merge manoeuvre are detailed within their state transition, using level 3 state diagrams in Figures 6.6, 6.7 and 6.8. Details are provided on each state transitions, by giving a short description of the event, condition and action involved in the transition, on top of the arrows going from one state box to another. In this short description, the event that triggered the transition is written first, followed by the transition condition, which is surrounded by "[ ]", and the actual transition action, written after the "/" sign. Note that interactions between each agent's state diagram is possible using the dotted arrows, which usually outline the inter-agent communications, part of the coordination protocol.

In more details, Figure 6.6 represents the vehicle following the merging vehicle throughout the progression of the manoeuvre, by defining its collaboration with the leader. Figure 6.7 shows how the leader manages this manoeuvre by coordinating both the merger and its follower in a transition similar to the protocol presented earlier in the second level of AUML. Finally, the statechart of the merger, referencing the previous leader's protocol, is presented in Figure 6.8.

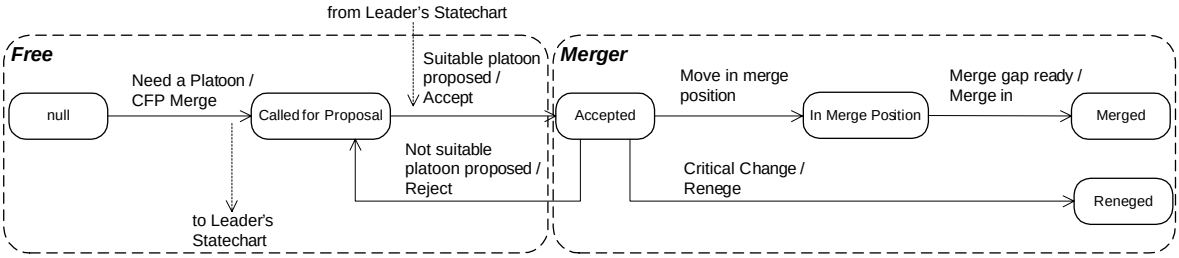

Figure 6.8: AUML Level 3 *state diagram* of the merge protocol focusing on the merger agent.

6.2 JACK Agent-Oriented Modeling

In the previous section, we presented the communicative behaviors that the driving agents should reflect in our CDS. Now, we detail how such a behavior was attained, by detailing the programming tools we used to develop the Auto21 agents. This description is based on JACK's agent programming framework and it should provide a better understanding on how the agents reason in the test scenarios we simulate.

As mentioned earlier, the JACK oriented agent model used to develop the Auto21 agents is based on the Procedural Reasoning System (PRS) architecture, proposed by Georgeff and Ingrand [1990] and detailed in Section 2.1.3. Therefore, this section is separated in sub-sections relating to the main components of the PRS architecture, as well as the components we added to create a complete agent infrastructure supporting our Collaborative Driving System (CDS). This presentation focuses on Java and JACK classes and the software infrastructure that was created to implement the functionalities of the *Planning* sub-layer and the *Networking* module. More details on the additions to the *Networking* module's software infrastructure, in order to support the teamwork model, are given in Section 6.3.

The current section begins with an overview of the main components of JACK's programming language. Then, Section 6.2.2's description of JACK's "Capability" should give an overview of the organization of our agents' behaviors into capabilities. This description is followed by the presentation of three components that are essential to provide our agents with the ability to reason, act and collaborate: (i) planning system; (ii) communication system; (iii) knowledge bases. Each component is presented in Figure 6.9 along with its relation with the agent's driving and coordination systems.

The "Jack Plans" of Figure 6.9 are included inside JACK capabilities and they are presented in Section 6.2.3, which details the steps of their execution inside JACK's framework. The "Auto21 Knowledge Base" represents an historic of sensed data that

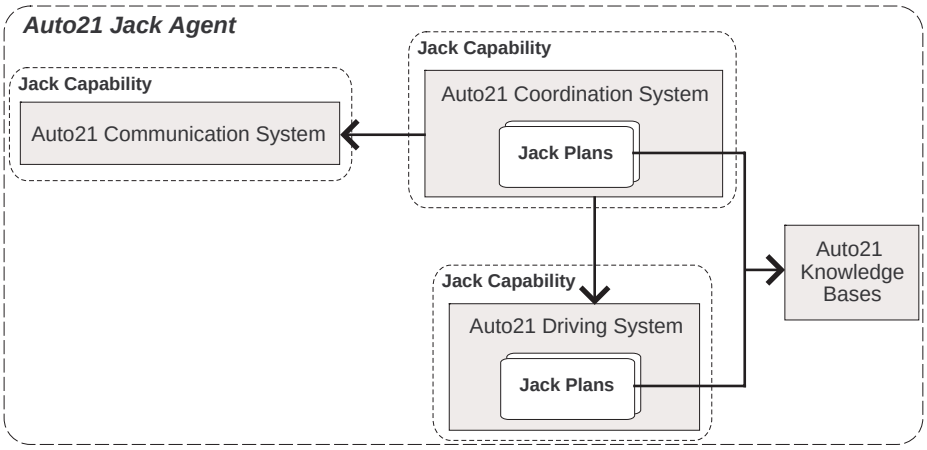

Figure 6.9: JACK components' relationships inside an Auto21 driving agent.

enables plans to reason about their environment, and it is presented in Section 6.2.4. The "Auto21 Communication System" represents the agent's communication protocol manager and it is described in Section 6.2.5. Following the description of these three components, the agent's coordination and driving systems are presented in Sections 6.2.6 and 6.2.7. This should provide a better understanding of the way the *Management* layer of our architecture (refer to Figure 4.2) was modeled inside JACK agents. The coordination system represents the *Networking* module of our architecture (the *Linking* module has not been implemented), while the driving system represents the *Planning* sub-layer. Both of them have been developed with JACK plans included in separate JACK capabilities. As mentioned in Section 6.1.2, our coordination protocols, designed with Figure 6.5, were developed by implementing the activation bars of this diagram with JACK plans. The same applies to the transition arrows of the state diagrams presented in Section 6.1.3, which represent the model of state transition that has been developed with JACK plans, as part of the coordination and driving systems.

6.2.1 JACK Programming Language

Being inspired by the PRS architecture, JACK is also a software representation of the Belief Desire Intention (BDI) agent model of Rao and Georgeff [1995] described in Section 2.1.3. The concept of the BDI agent is applied in JACK by using four dynamic structures along with a queue of internal and external events. Basically, the event queue generates a list of options in accordance with the BDI agent's desires. These options use a belief database to match the current situation with a plan in a library that represent the agent's possible intentions. Once an intention is chosen by the agent, the relating plan is executed and monitored by a task manager, as a Finite State Machine (FSM).

More specifically, JACK framework implements the previous concepts by using five main classes that can be extended for specific usage:

Plans: Plans are pre-compiled procedures that depend on a set of conditions to be applicable. They answer to internal or external events using *relevance* and *context* criterions defined in JACK language's documentation [AOS, 2004]. Moreover, plans are closely related to event types (like BDI events) and settings, since they occur when an event arise.

Events: Events are messages that are handled by plans and can be sent from the agent itself or from an object that has a reference on this agent. JACK offers a number of event models for different needs, represented as: *internal stimuli, external stimuli* and *motivations*, as well as a another category for inter-agent message events.

Beliefsets: A *beliefset* provides a data structure that enables the agents to collect, query and infer on a database of knowledge. The *beliefset*'s queries are very useful to determine an agent's environmental context, which is a criterion to determine the plans to execute on a specific event.

Capabilities: A capability is used to describe an agent's behavior or role. An agent can have zero to n capabilities. Each capability regroups a set of agent internal components that define: (i) the internal or external events the agent can send or respond to; (ii) which beliefs database (*beliefset*) it instantiates or refer to; (iii) and which plans it can use to act or reason.

Task Managers: Task managers govern how an agent handles the concurrent execution of plans, when they are committed to more than one task execution.

6.2.2 JACK Agents' Capabilities in Auto21

JACK's capability is the first structure to be analyzed in this section since it embraces most of JACK's other structures in what is the agent-oriented programs' version of classes' inheritance. Capabilities enabled us to define a set of behaviors relating to the agent components, defined in Sections 6.2.4, 6.2.5, 6.2.6 and 6.2.7, which we reused for other agents. Figure 6.10 shows an informal model of the capabilities used by both the following vehicles' agent and the leader vehicle's agent. Each capability is briefly described within its box, by mentioning the major Knowledge Base (KB) and JACK plans it uses to provide the required behaviors.

The capability model of Figure 6.10 shows that the leader and follower can share some capabilities, which makes the implementation of new agents easier. CapEnterPaltoon and CapLeavePlatoon include plans to coordinate the split and merge manoeuvres

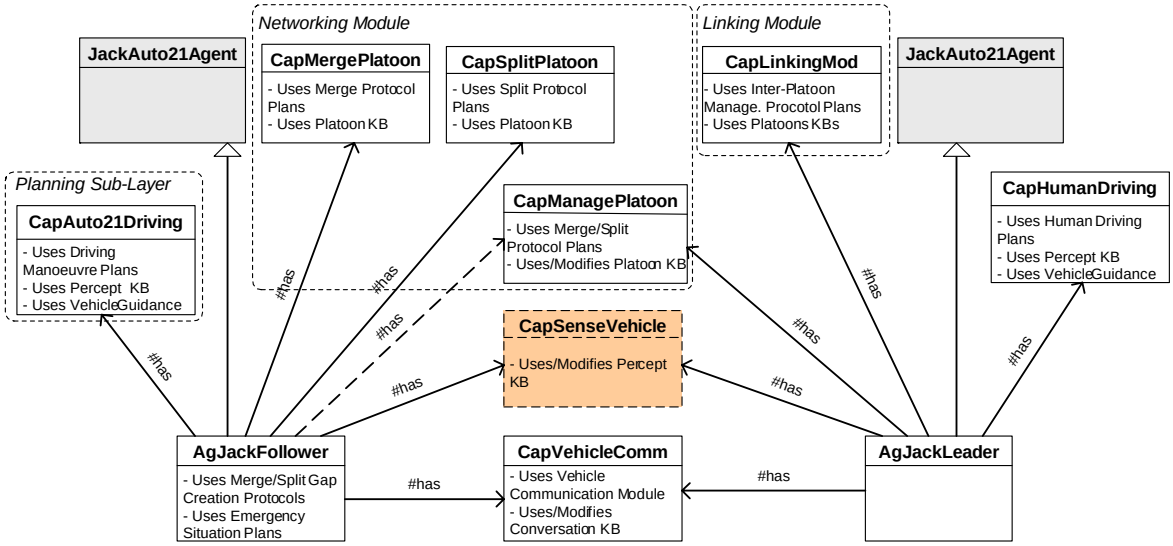

Figure 6.10: JACK capabilities usage by both the follower and leader agents.

and refer to the protocol required for a vehicle that wants to leave or enter a platoon. CapManagePlatoon includes the plans used to reply to a request from a vehicle entering and leaving the platoon, meaning that these plans are used as answers to the plans of CapEnterPaltoon and CapLeavePlatoon, inside the same protocols. As Figure 6.10 highlights it, the previous capabilities represent the implementation of the *Networking* module of our architecture, while the CapLinkingMod of the agent AgJackLeader represents the architecture's *Linking* module (refer to our architecture model in Figure 4.2). CapManagePlatoon is linked to both the AgJackLeader and the AgJackFollower, but the follower only has this capability in the case of a decentralized or teamwork intra-platoon coordination model (refer to Section 5.2).

6.2.3 JACK Agents' Plans Execution Framework in Auto21

The JACK plans that each capability "owns", according with its behavior, is now detailed by describing the way a plan is launched and executed in JACK. In our case, JACK's plans execution framework is being used for two major purposes: first for coordination issues and second as a recipe for a driving manoeuvre. If we relate to the Auto21 architecture of Figure 4.2, this means that JACK's plans execution framework is being used to execute driving plans for the *Planning* sub-layer, as well as plans referring to coordination protocols for the *Coordination* sub-layer. As it will be shown later in this section, the plans' execution is triggered by events that are launched by an agent, a capability, a *beliefset*, an automatic event or another plan. In Figure 6.11, the different event senders, inside the Auto21 agents, are presented along with their relation with

groups of capabilities. These groups outline capabilities having plans responding to similar types of events.

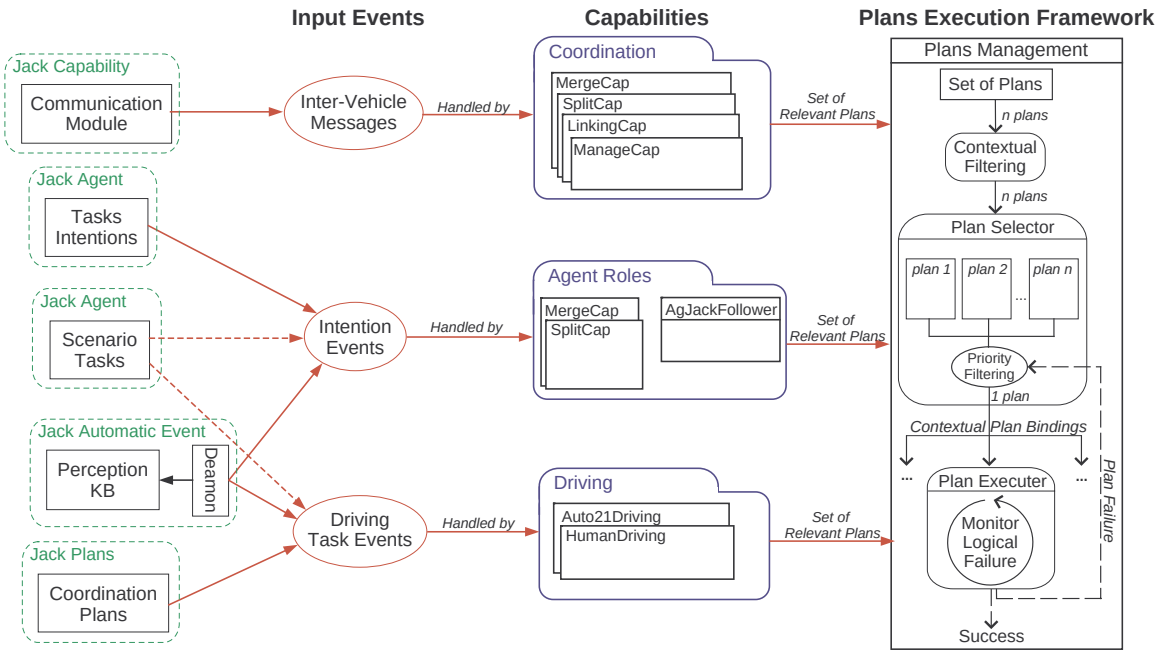

Figure 6.11: Running loop of the JACK planning system.

Plans relating to the same type of event are selected considering their relevance to this specific event and they are sent to the *Plan Management* system, shown at the right of Figure 6.11. Relevant plans are further filtered considering the current context, which is defined by the *beliefsets*. If more than one plan goes through all these filtering steps, one of them is selected upon a specified priority ("Plan Selector" from Figure 6.11) and the others are left in the queue.

Once the highest priority plan has been instantiated, it is executed one or many times considering the different possible contextual bindings, as shown inside the "Plan Selector". A contextual binding is specified as a query to the *beliefset*, in a method called "context", and all the possible matches with the *beliefset* query result in running a different plan binding. For example, contextual bindings can be used to send a message to each members of a specific platoon in parallel, by developing only one plan, who's task is to send a message to the agent defined in the context. In this case, the contextual method would be "get agents in platoon #2", which would result in plan instances being created for each platoon member and messages being sent to each member as part of each plan instance.

At the instant a JACK plan begins its execution ("Plan Executer" in Figure 6.11), the success of logical statements inside the plan is always monitored to determine if the

plan has failed or not. Logical statements can be defined in the form of different queries to the *beliefsets* or by using logical statements, as part of predefined JACK statements, executed in the form of a Finite State Machine (FSM) [AOS, 2004]. A logical statement can be seen as a statement that launched an action inside a plan with the possibility of specifying a logical condition that has to be maintained while the action is executed. A logical condition refers to the state of the environment, defined by the agent's KB, which is described in Section 6.2.4 with an example of logical statements.

In brief, JACK's plans execution framework offers the possibility of executing plans according to specific events, by considering the state of agent's environment. By adding the notion of logical statements, JACK's framework can verify if specific conditions have been achieved or maintained during the execution of the plan. This enables us to determine if a problem occurred (logical condition not respected) and resolve it, if possible.

6.2.4 Auto21 Agents' Knowledge Base

The agents developed for the Auto21 project use a structure called Knowledge Base (KB) that provides an historic of the agent's states in time. Such a structure is used by every agent, as shown in Figure 6.2 that presents an AUML agent diagram, which includes KBs presented as different sources of information and data (i.e. internal_vehicle_data) at the top of the agent box. A KB enables JACK plans to reason about the current state of their vehicle and environment, by using the logical statements that were introduced in the previous section. In this section we introduce JACK's KB class, called a *beliefset*, and the KB classes that we developed in Java. This introduction is completed with the presentation of a KB monitoring class provided by JACK's API called *Cursor*, and an example of the use of KBs through *Cursors* inside our JACK plans.

As an example of the usage of KBs within our driving agents, Figure 6.12 presents the class hierarchy of some *beliefsets* and Java KBs with the respective JACK *Cursors* we extended for the Auto21 driving agents. The KB classes that we used for the Auto21 agent extend either the *ClosedWorld* class (for JACK *beliefsets*) or the AbstractKnowledgeBase class (for the Java KBs we developed). To provide support for the logical statements of our JACK plans, we have implemented a series of *Cursor* acting as observers on the Java KBs. Some of these *Cursors* are presented on top of the plan structure of Figure 6.12, as an example. BelVehiclesVirtuality.__Cursor represents a daemon (database observer that triggers an event) on the *beliefset* of virtual vehicles (BelVehiclesVirtuality), VelocityCursor is a velocity daemon on the vehicle's dy-

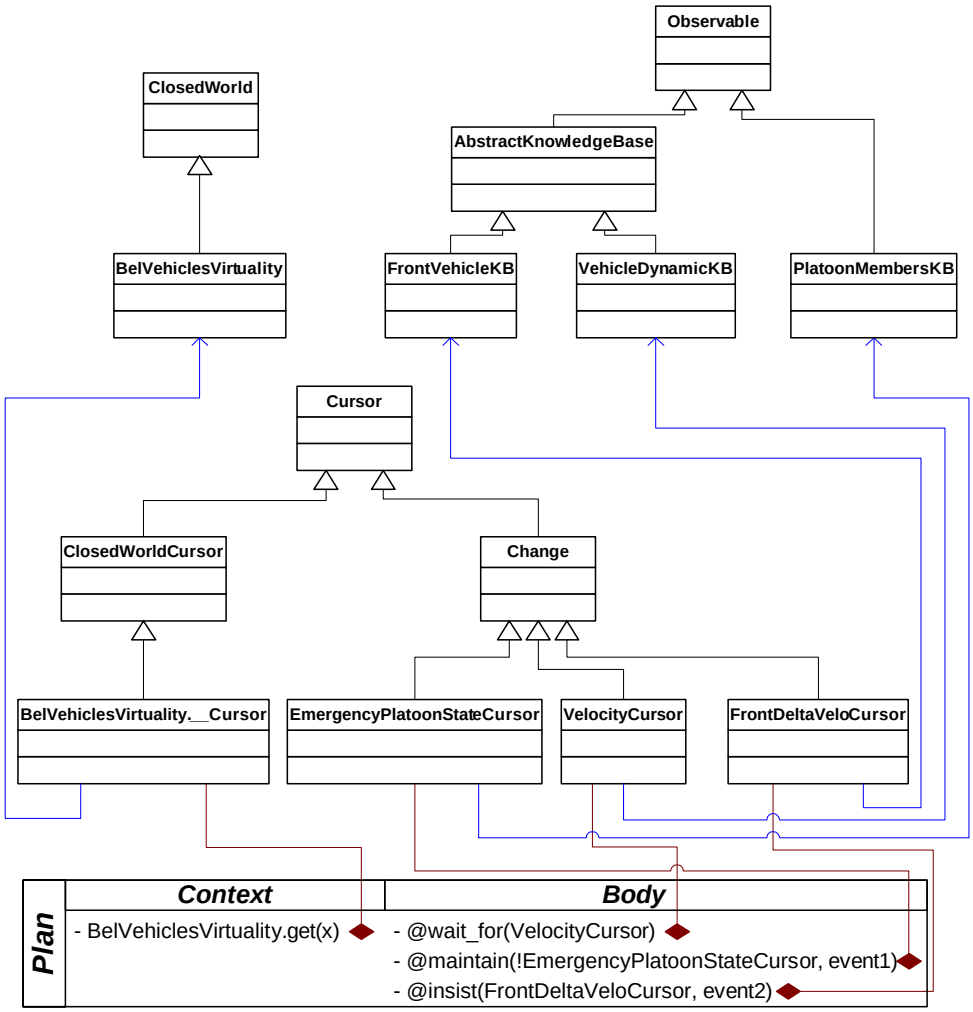

Figure 6.12: JACK oriented beliefs structures and respective *Cursors* for planning usage.

namics KB (VehicleDynamicKB), FrontDeltaVeloCursor is a velocity daemon on the front vehicle KB, and EmergencyPlatoonStateCursor is an emergency situation daemon on the platoon members KB (PlatoonMembersKB).

In order to understand how *beliefsets*, KBs and *Cursors* are used, the bottom of Figure 6.12 shows a box representing a JACK plan using *Cursors* in two possible ways. First, a *Cursor* can be used as part of the plan's *context* method, with the purpose of selecting and instantiating one or many plans, as it was shown in Figure 6.11. Second, *Cursors* can be used inside the plan's body to monitor the achievement of specific logical conditions.

In the example presented in Figure 6.12, logical statements provided by JACK's framework are used to ensure that the steps of the plan are applied according with the state of the environment, described by the agent's KBs. Thus, the BelVehiclesVirtuality.get(x) method is used in the plan's context to get all the virtual vehicles inside BelVehiclesVirtuality and instantiate a plan for each one. The @wait_for(VelocityCursor) statement allows the plan's body to wait for its vehicle to reach a specific velocity value in VehicleDynamicKB before executing further actions. The @maintain(!EmergencyPlatoonStateCursor, event) statement monitors the changes to PlatoonMembersKB in order to ensure that the condition of "no emergency" is maintained during the execution of the action relating to "event1". The last statement, @insist(FrontDeltaVeloCursor, Event), monitors FrontVehicleKB to ensure that a specific inter-vehicle velocity condition will be reached, by executing the action relating to "event2" until the velocity condition is reached.

6.2.5 Auto21 Agents' Communication System

To provide our agents with the ability to send and receive messages according to a particular standard, we developed an agent-oriented communication system. Our communication system was developed as part of a JACK capability (CapVehicleComm) and its tasks are to filter the messages that the agent sends and receives, according to basic conversation protocol management functions. Within this section, the presentation of the Auto21 communication system begins with a description of the process involved in receiving and sending inter-agent messages. This is followed by the a detailed representation of the protocol management class (ConversationManager), which ensures that our agents communicate according to the rules of our communication protocols.

Before going further, it should be mentioned that our communication system uses the inter-vehicle communication devices presented in Section 3.5, even though JACK al-

ready provides an inter-agent messaging infrastructure. JACK's communication framework is based on a communication layer called the DCI network and uses the UDP transport protocol [AOS, 2004], which is more useful for web-oriented agents than "simulation-oriented" agents like ours. Indeed, our inter-agent communications must simulate the restrictions and latency times of a radio network, which would have made the use of JACK's communication infrastructure problematic.

Consequently, we cannot use JACK's communication infrastructure, and this is why we developed our own communication sending/receiving system inside the CapVehicleComm capability. A general view of the classes involved in this system is provided by the UML class diagram of Figure 6.13. This capability includes both the vehicle's communication devices (like a radio) and the agent conversation manager, which is presented later. The communication devices are used to send the agent's own messages, which are generated by the coordination system (Section 6.2.6), and to receive messages sent by other vehicles. When CapVehicleComm receives a message from another vehicle, it filters the message using the information provided by the message's class (VehicleMessage). For instance, CapVehicleComm verifies if the message is addressed to the agent it represents, by considering its ontology: intra-platoon, inter-platoon, intra-team. If the message goes through this filter, it is sent to the agent coordination system's plans (described in Section 6.2.6) using either a EvRcvInterPltnMsg (event of a received inter-platoon message) or a EvRcvIntraPltnMsg (event of a received intra-platoon message). If no plan can handle a message that was addressed to this agent, because it is not part of any protocol, a default plan called PlUnhandledIntraMsg will respond with a default "Not Understood" message.

On the other hand when CapVehicleComm is used to send messages to others, the process is simpler than receiving a message. In this case, plans from the coordination system raise a EvSndVehMsg event which is handled by a plan from the CapVehicleComm capability. Since EvSndVehMsg includes attributes on the conversation this message relates to, the most appropriate plan from CapVehicleComm is triggered to finally send the message using the inter-vehicle communication devices.

To ensure that messages are being sent and received according to our communication protocols, each entering and leaving message must be added to the ConversationManager class, presented in Figure 6.13. This class only acts as a conversation KB which returns true or false on its different possible "add message" methods to specify if the manager is able to add a message to the current conversation. Therefore, when an Auto21 agent receives a message, it is handled by a plan from the coordination system (detailed in Section 6.2.6), which calls the "add message" method of ConversationManager to make sure that this message was received in accordance with its protocol. For example, if

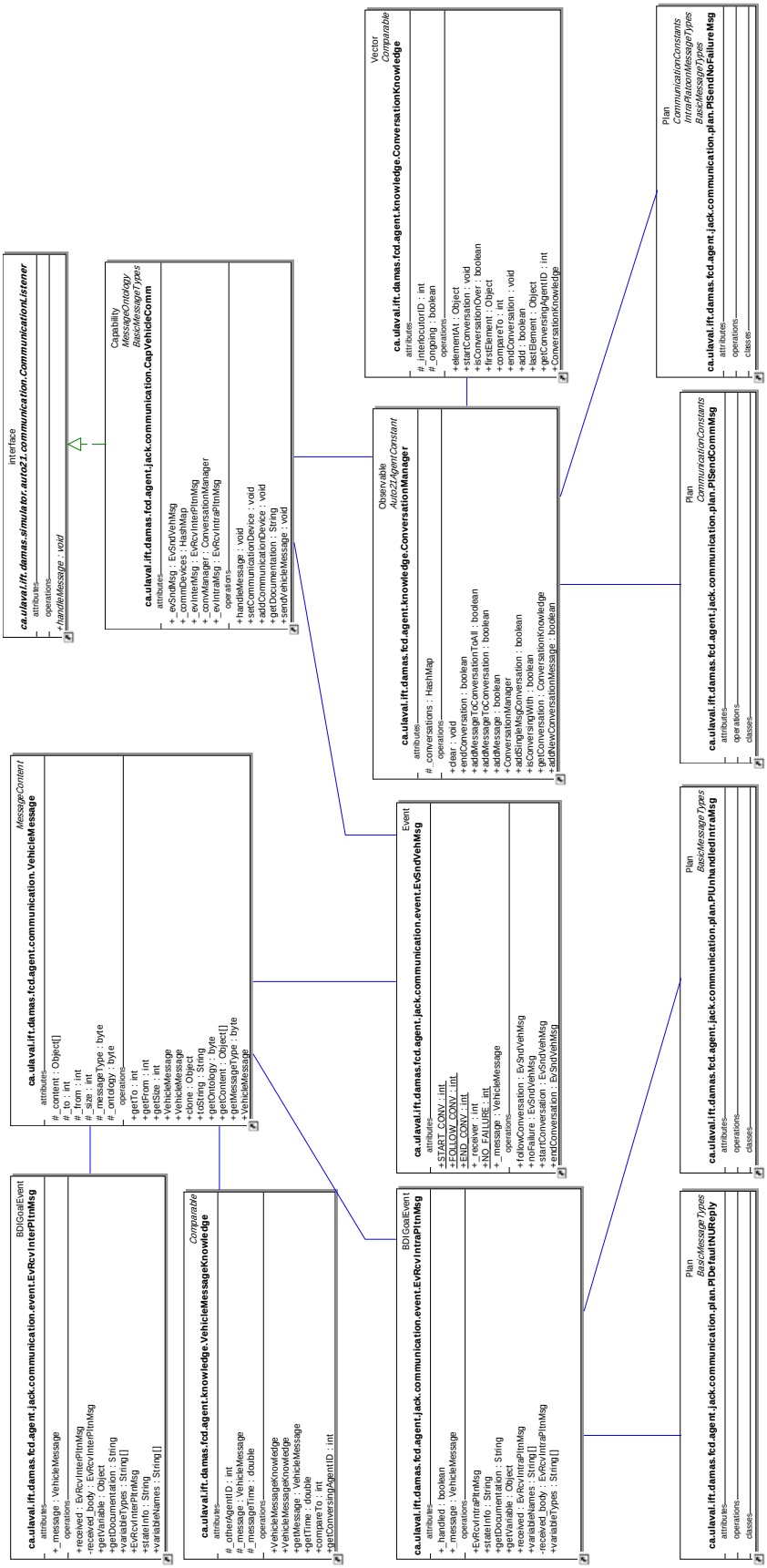

Figure 6.13: Class diagram of the main Java and JACK classes related to inter-vehicle communications.

a plan is created to receive the answer about a vehicle's position, the "add message" method of ConversationManager will be invoked as part of the plan's context (refer to Section 6.2.3). If there is no conversation taking place at that moment (the answer is not a valid reply), the "add message" method fails, and this plan is not chosen (context fails). The agent then returns a "Not Understood" message to the sender, since it never asked for information about this vehicle's position.

Finally, the ConversationManager class can also be used as an inter-agent messages historic, since conversations are indexed inside this class' KB. This way, plans that are part of our coordination system can query the ConversationManager to retrieve a conversation with a specific agent and verify which messages have been exchanged with him. The knowledge on a conversation is returned in the form of a ConversationKnowledge object which contains 1 to n VehicleMessageKnowledge objects, both defined in Figure 6.13.

6.2.6 Auto21 Agents' Coordination System

The previous sections presented the agent engineering components that we developed to provide our agents with a plans execution framework, a communication system and knowledge bases. The following sections describe how these components are used inside the agent-oriented design of our architecture's *Management* layer, starting with the coordination system. This system refers to our architecture's *Coordination* sub-layer, but since we only developed the *Networking* module so far, it only instantiates the *Networking* functions through a series of plans (coordination plans) part of the Merge, Split and Manage capabilities (refer to Figure 6.10). These coordination plans have been defined in the AUML agent diagram of Figure 6.2, which shows at the bottom of the agent box, the parts (circles representing plans) of different coordination protocols handled by this agent. The coherence of these plans is ensure by JACK's plans execution framework, described in Section 6.2.3, which support the steps of the coordination models defined in Section 5.2. The coordination plans also use the Knowledge Bases detailed in Section 6.2.4 to trigger a driving manoeuvre and its relating coordination plans or to execute and reason about the manoeuvres currently being coordinated. Finally, the coordination plans use the communication system that was presented in Section 6.2.5 to support the input/output of the coordination process. This section first presents the state transitions involved in the coordination process and then, it shows how the coordination system synchronizes the plans associated with these transitions.

For a better understanding of the tasks of the coordination system, you may refer to the protocol diagram that was presented earlier in Figure 6.5. In this figure, the

transitions (horizontal lines) represent the inter-agent messages sent and received as part of the coordination process, while the life lines (vertical rectangles) represent the coordination plans. As another similarity with Figure 6.5's diagram, the coordination system assigns a *role* to agents, according to the task they executed in the coordination protocol. This role is then placed in different states (*in progress*, *success*, *reneged*, *cancel*, *reject*) to ensure the coordination between an agent's own intentions. The AUML level 3 diagrams presented in Section 6.1.3 describe those states in more details for the merge manoeuvre. For each agent, *reneged*, *reject* and *cancel* states refer the failure of the manoeuvre for this role. Furthermore, the "Merged" state represents the *success* state of the role and all the other states following "null" represent *in progress* states. Note that the transition between an agents' roles is considered as setting the previous role to the *success* state and the new role to *in progress*.

By presenting the transitions between an agent's role states, Figure 6.5 outlines the fact that an agent can only execute one task at a time. Our agents' coordination system is therefore responsible for synchronizing an agent's roles execution in accordance with the transitions of Figure 6.5, as well as the state transitions of Figures 6.6, 6.7, and 6.8. Our agents' possible roles include: *Split, Merge, Follow, Lead, Free, SplitForMerge, SplitForSplit, SafetyObserver, VirtualVehicle, GapCreator*. These roles are separated in categories and the roles considered as "task roles" are synchronized by the coordination system which allows the agent to fill only one on these roles at a time. The "task roles" include: *Split, Merge, SplitForMerge, SplitForSplit, GapCreator*. To support this role restriction, the JackAuto21Agent agent class informs our coordination plans about the "task roles" they have the right to execute, by referring to a priority list of categories of "task roles". Basically, JackAuto21Agent gives permissions for certain "task roles" and informs a coordination plan if its role can no longer be executed, as shown below.

When a task-oriented coordination plan is initialized, it first verifies if it has the permission to fill its task role, by calling the JackAuto21Agent class, before executing any communicative action. If the permission is given, the plans monitor the validity of this role throughout the protocol's execution, in order to stop the coordination if the role becomes invalid (*canceled*, *reneged* or *reject*). For example, when a coordination plan relating to the *Split* role begins its execution, it first raises a "@maintain(role state, role type)" logical statement (refer to Section 6.2.4). If the condition on the role's "state" is not maintained, the plan fails and the role is cancelled. Other failures may come from plan execution problems, but they have the same result, where the role is placed in a *cancel* state to inform other agents of this failure. Once a role has been cancelled, the agent immediately switches from the task role to its "default role", which in most cases is a *Follow* role (when the vehicle is in a platoon) or a *Free* role. Therefore, our coordination system based on plans' synchronization ensures a certain degree of safety

when executing our agents' tasks and it guarantees that the agent always executes its default role if a failure occurs.

6.2.7 Auto21 Agents' Driving System

The Auto21 agent-oriented driving system represents our architecture's *Planning* sub-layer (refer to Figure 4.2) and the driving manoeuvres (and actions) it can execute. The driving actions presented by the driving system are described in the AUML agent diagram of Figure 6.2, which shows, in the middle of the agent box, a list of some of those actions. The main tasks of the driving system are to synchronize the different driving manoeuvres that the agent may want to execute at the same time and to synchronize the agent's access to the guidance functions presented in Section 4.3.3.

To resolve the manoeuvres synchronization problem, we defined a set of driving modes that the driving system could be in, to provide a solution similar to the task synchronization resolved by the previous agent coordination system. This way, a driving plan initiating a specific manoeuvre (*higher-level driving plan*) has to set and maintain a specific driving mode in the JackAuto21Agent agent class, as it is shown in this section. In a different way, plans that act directly on the guidance functions from Section 4.3.3, which we call *lower-level driving plans*, are synchronized using a plan locking system. Thus, when a *lower-level driving plan* is initialized, it is locked and is the only one that can act on the guidance functions, as shown below.

Higher-Level Driving Plans Synchronization

A representation of the *higher-level driving plan* synchronization system, based on driving mode, is given by the *state diagram* of Figure 6.14. This diagram shows that the driving system can be in four possible modes: *Emergency, Automated, In Platoon, Manual*, which have predefined transitions between each others. To support these transitions, we associate a specific driving mode which each *higher-level driving plan* to dictate which driving mode must be initialized before executing such plan.

For instance, if the agent is executing the *Human Behaviors* plan in the *Manual Driving* mode (refer to Figure 6.14) and it suddenly wants to execute the *Cruise Control* plan from the *Automated Driving* mode, the agent must switch to the *Automated Driving* mode. In this case, the *Manual Driving* mode is cancelled and the *Human Behaviors* plan no longer has the authorization to run, thus enabling the *Cruise Control*

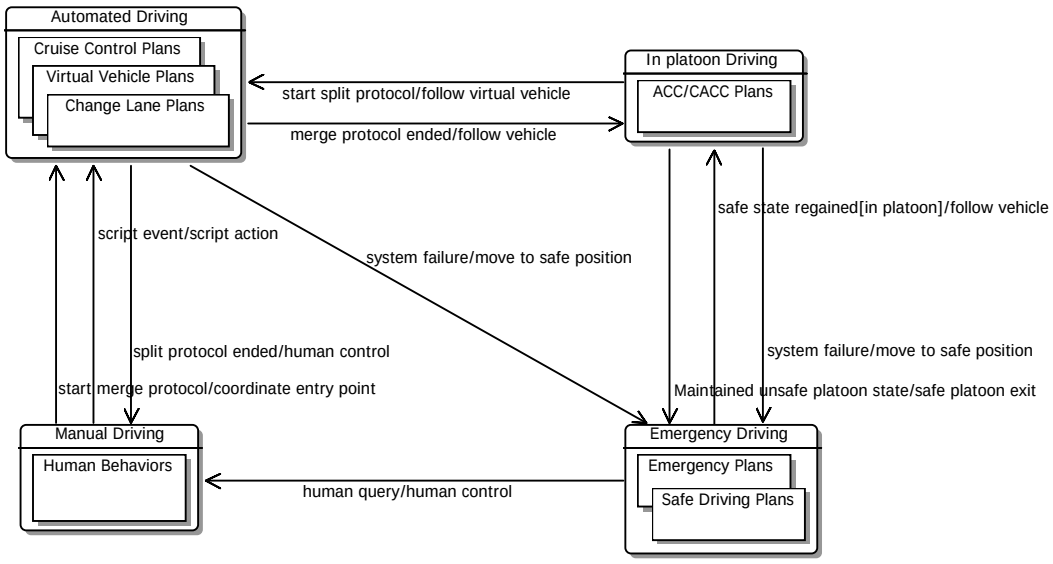

Figure 6.14: Statechart diagram for the agents' possible driving modes.

plan to be executed. To support the synchronization on driving modes, the plans are only authorized to execute their driving tasks while the driving mode is maintained, by using the "@maintain(driving mode)" logical statement. This solution is similar to the solution presented for the coordination system (Section 6.2.6) and it enables the agent driving system to respect the states transitions of Figure 6.14.

Lower-Level Driving Plans Synchronization

Once the driving system has acquired a driving mode, the different plans relating to this mode can execute *lower-level driving plans* that act directly on the guidance functions. For a better understanding, the relation between the *lower-level driving plans* and the guidance functions is also represented in our architecture, in Figure 4.2, where the "Desired state" arrow going from the *Planning* sub-layer to the *Vehicle Control* sub-layer refers to this relation.

In order to coordinate the different *lower-level driving plans* that can be executed at the same time, each plan has been assigned a level of priority and a plan locking system has been developed. Hence, a *lower-level driving plan* that uses a guidance function has to be in a "locked" status to run. If the plan fails to be locked because another plan is already locked or a higher priority plan unlocks it, the rest of the JACK procedure that called it also fails.

For example, if the driver is in the *follow* role, a plan from the coordination system

calls an Adaptive Cruise Control (ACC) driving plan, thus initializing the *In Platoon Driving* mode and locking the ACC plan. While acquiring the right inter-vehicle distance, if the sensors fail and a higher priority driving plan is launched to resolve the situation, the ACC plan fails. This situation is represented by the "system failure" transition going from the *In Platoon Driving* mode to the *Emergency Driving* mode, in Figure 6.14. In this example, the failure to keep the driving plan locked results in a chain of events, detailed below, that makes the *In Platoon Driving* mode fails and finally, the *follow* role fails. Therefore, the plan locking system resolves conflicts among *lower-level driving plans* and supports the coordination of driving actions through JACK's plans execution framework.

To provide a better understanding of the way the driving system handles emergencies, Figure 6.15 represents a UML activity diagram that depicts the classes and actions involved in an emergency event. The emergency event is first launched using a *Cursor* on emergency states and it is handled by PreEmergencyPlan, which makes sure this is a confirmed emergency situation before setting the driving mode to *Emergency* and cancelling other driving plans. This leads to the possible execution of EmergencyPlan, since the diamond boxes represent a decision activity. Subsequently, the SafeDrivingPlan will be locked and a chain of possible alternative activities will ultimately set the driving mode to "idle" and meet the ending condition of this activity. Of course, in a brighter set of events, it is possible to reach the *In Platoon Driving* mode and cancel the emergency plans after the emergency event was raised, but this diagram only depicts the activities related to the complete execution of the emergency scheme.

Virtual Vehicle Driving

When our driving agents enter or leave a platoon, they must go through the *Automated Driving* mode, which includes the change lane driving actions occurring in "semi-blind" situations. This situation is caused by the incapacity of our vehicles' front laser to receive a good perception of the front vehicles that are not in the same lane. To resolve this perception problem, we developed a virtual vehicle driving system, which is detailed below. First, the perception problem is described by referring to Figure 6.16, which shows that a merging vehicle *L2* must go from state *S2* to *S3* and keep a specific distance with vehicle *F1* that is not always in its sensor range (red cone). In the case of a split, vehicle *F2* has the same problem when going from state *S1* to *S2*, so it must rely on another source of perception than its sensors.

The virtual vehicle driving model that we developed to resolve the previous perception problem modifies the information provided by our sensors (when they cannot

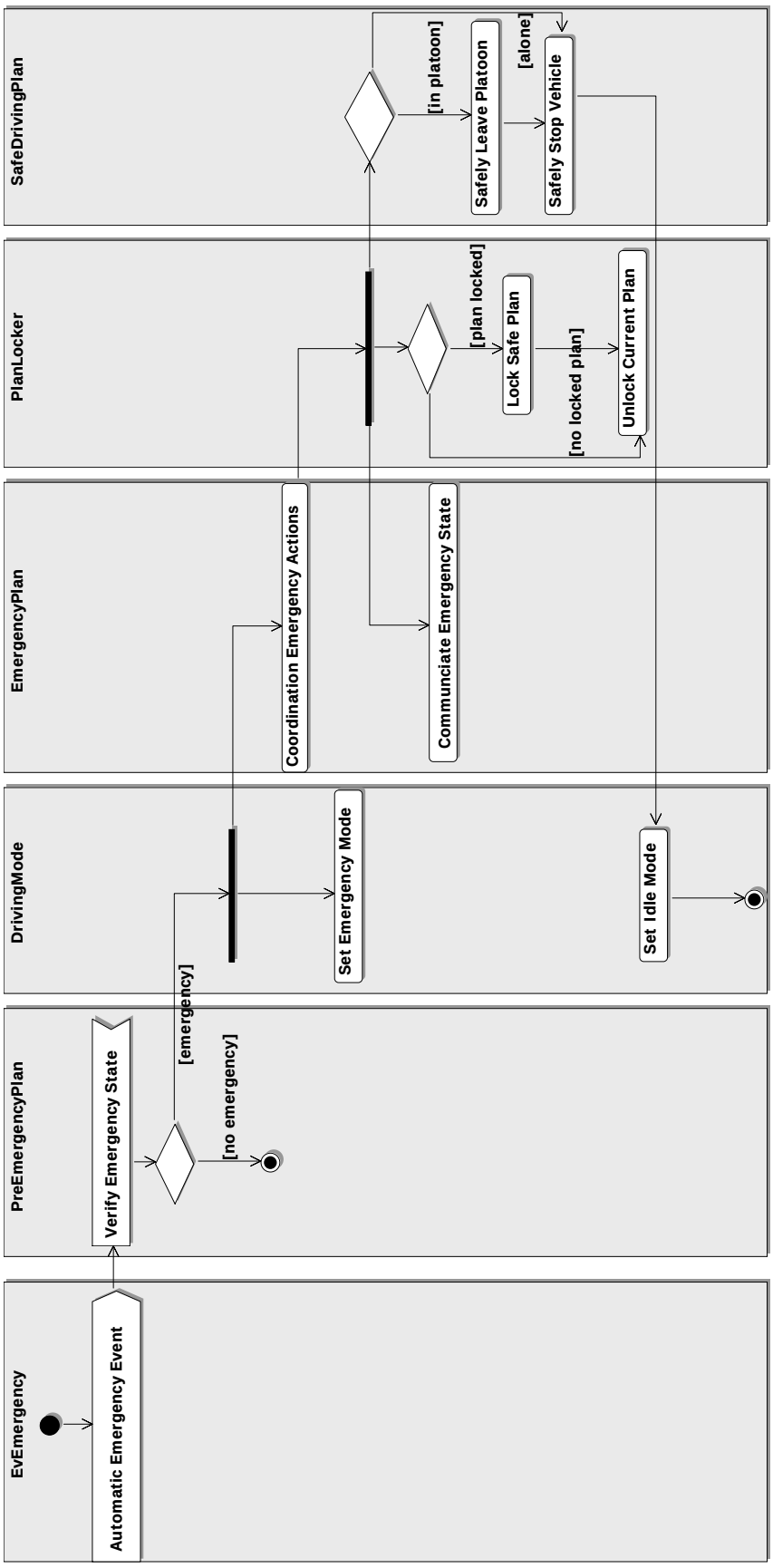

Figure 6.15: Activity diagram representing the transition occurring during an emergency event.

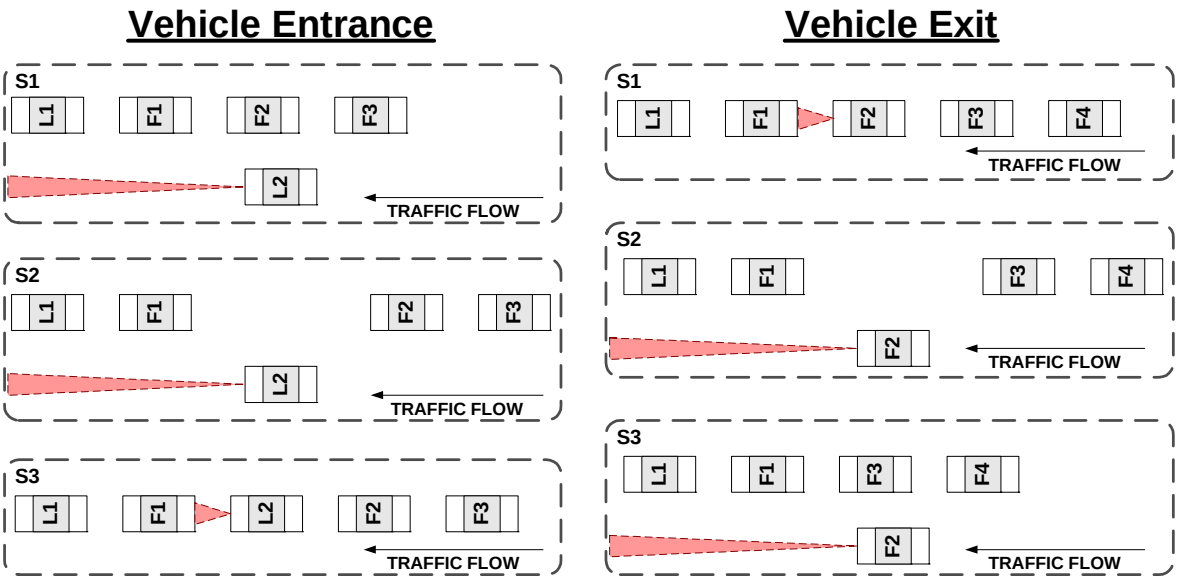

Figure 6.16: The three steps of the removal (split) and insertion (merge) of a vehicle in the platoon.

sense the preceding vehicle) and gives this new information to our guidance functions. With this model, the guidance functions, like the ACC functions we used to drive in the platoon, can be reused during the lane change. This smoothes the control effort changes caused by the modification of the state in front, without any necessary changes to the *Guidance* layer. Thus, we simply modify the information inside the Knowledge Base (KB) of the guidance functions to give these functions a virtual belief of the vehicle to follow.

Figure 6.17 shows the FrontVehFilter class, used by *higher-level driving plans* to filter the guidance KB's knowledge. Other classes involved in the virtual vehicles creation are shown in this diagram along with a brief summary of their activities inside their box. The two main virtual vehicle creation plans are PlCreateVirtualVehicle and PlCreateVirtualCommVehicle. The PlCreateVirtualVehicle plan is used by vehicle *F2* in the merge example of Figure 6.16 and vehicle *F3* in the split example. This plan allows them to modify the FrontVehFilter when they sense that a vehicle has entered or left the platoon, in order to keep their distance until the manoeuvre is completed. On the other hand, the PlCreateVirtualCommVehicle plan is only used in the teamwork coordination model (refer to Section 5.2.3) by the merging or splitting vehicle to create a virtual vehicle from communicated information. This plan uses a Java class called VirtualCommVehicleMaker, which directly acts on the FrontVehFilter by considering its own vehicle dynamic percepts and the dynamic state that *F1* communicated about its vehicle.

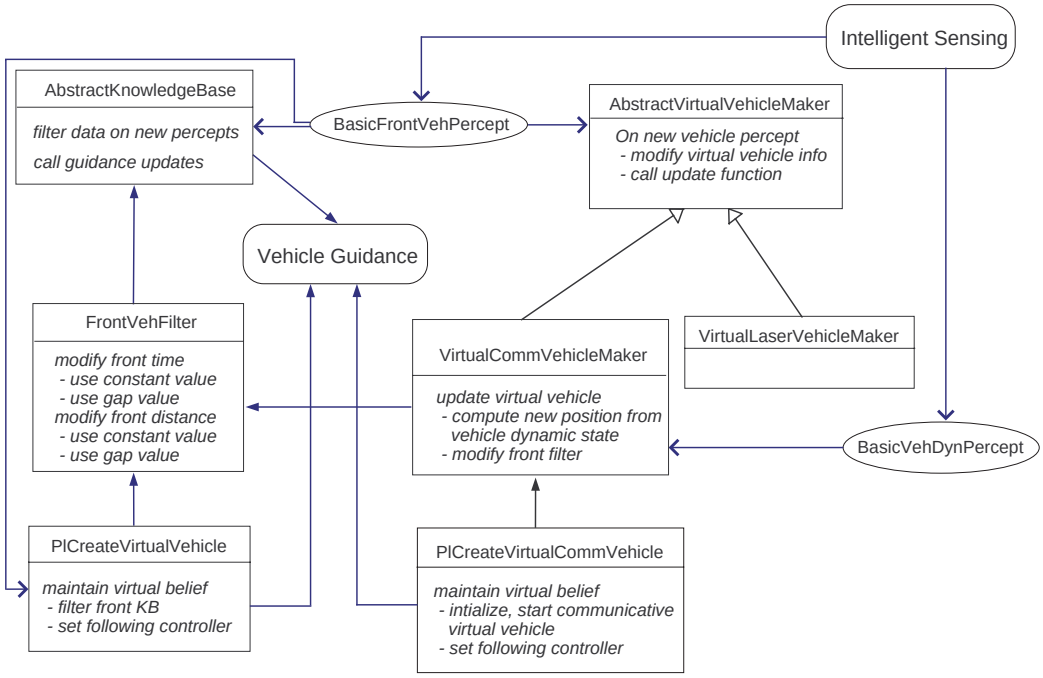

Figure 6.17: Classes and tasks involved in the creation of virtual vehicles.

6.2.8 Discussion

JACK's agent-oriented programming environment proved itself very useful in supporting coordination and driving issues in our real-time environment. The additions we have made to the plans execution framework allowed the Auto21 agents to synchronize their tasks at different levels of hierarchy including the coordinated manoeuvres and the different possible driving plans. The overall agent infrastructure is both reliable and robust, which allows us to safely perform the tasks required to drive inside a platoon. Even though agent-oriented softwares development usually represents a more fastidious task than object oriented development, the generic plans execution framework that was extended along with JACK's capability structures makes further agents' improvement tasks easier to perform.

6.3 Teamwork Oriented Modeling

Section 6.2 already presented many aspects of the agent-oriented architecture that are being used by agents in each of the intra-platoon coordination models that were described in Section 5.2. To complete the description of the agent-oriented infrastructure

that we developed for the Auto21 project, we now focus on agent engineering aspects that are specific to the teamwork model. Our team oriented infrastructure has been developed over the agent infrastructure presented in Section 6.2 and includes the previous agents' coordination system, as well as the agents' driving systems. For a better understanding of the teamwork architecture we developed, as well as the teams involved in this intra-platoon coordination model, refer to Section 5.2.3, which gives an introduction necessary to the understanding of the teamwork software models presented below.

The team-oriented infrastructure presented in this section uses and extends Java and JACK components in order to relate to the STEAM [Tambe and Zhang, 2000] teamwork architecture and our specific needs. At the moment, JACK is also developing an extension to support Team Oriented Programming (TOP), called JACK Teams^{TM}that supports the vision of shared plans and beliefs, and team formation using roles. Although this extension to JACK looks promising, it is not complete at the moment and lacks a lot in documentation to realize dynamic task-team formation or to be used with our custom simulated communication devices. Therefore, we have developed our own TOP extension to JACK, being inspired by JACK Teams^{TM}, although it will be possible in a near future to merge to AOS' new extension.

Figure 6.18 gives an overview of the different components that have been added to the Auto21 agent-oriented model (presented in Section 6.2) to provide a TOP infrastructure. JACK's components are presented in boxes with shadows, while Java classes are presented in normal boxes. The agents that evolve in our team formations are TeamAgent agents and they all have the teamwork capability named CapTeamwork, which includes all the necessary data structures and generic plans based on STEAM, allowing a basic support for the agent's roles management. The other capability presented in this figure is CapRoleX, which is an example of capability relating to any specific role X (X = Split, Merge, Virtual Vehicle, etc.). CapRoleX allows the agent to perform the role X's actions by using all the necessary plans to support this role's coordination protocols and manoeuvres. Therefore, an agent can have as many CapRoleX capabilities as it can perform roles. As a final note, Java Knowledge Base (KB) structures and JACK *beliefsets* have been added to the KBs that were presented in Section 6.2.4 to provide a set of local and team beliefs observing each others' changes, as explained below.

6.3.1 Teams Shared Beliefs

In order to support the teamwork coordination model, we developed additional Knowledge Base (KB) for different purposes and one of these KBs is a team *beliefset* that

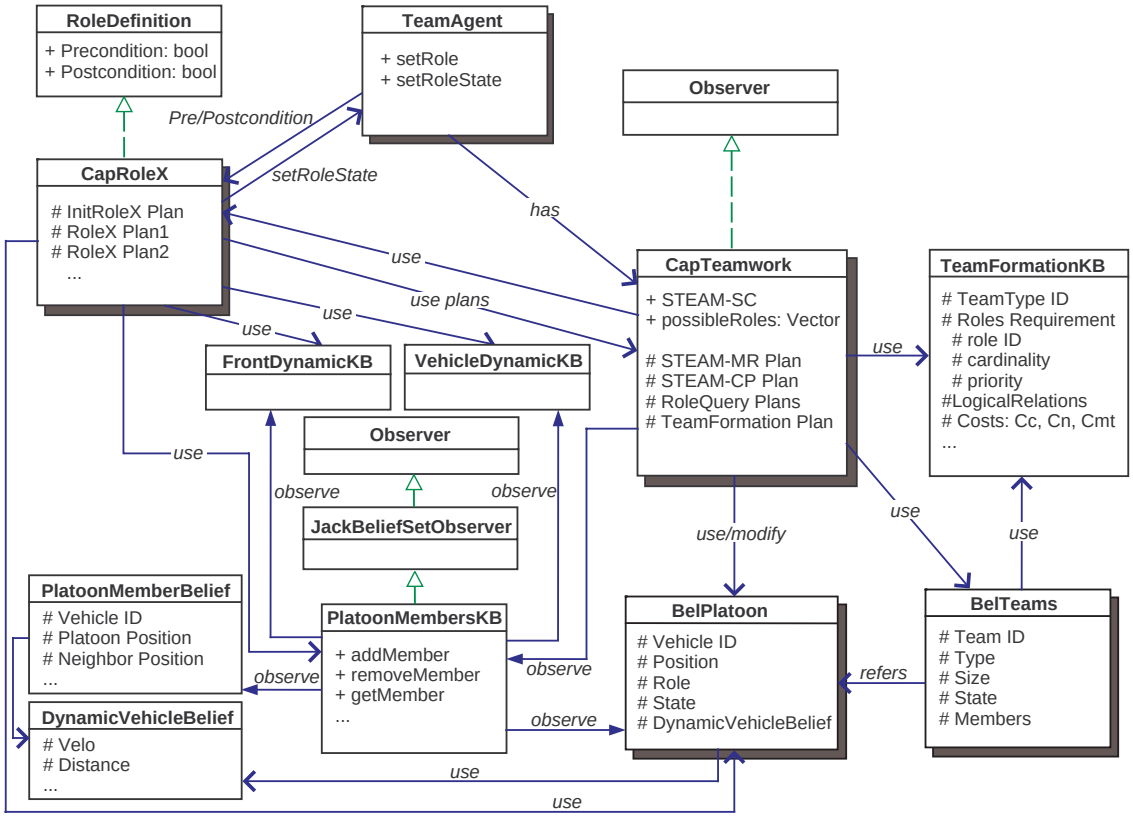

Figure 6.18: Model of the main classes involved in the team-oriented infrastructure.

stores the team's shared beliefs. This *beliefset*, called BelPlatoon (refer to Figure 6.18), represents the shared KB of the team formation and it is linked with the agent's local KB, called PlatoonMembersKB, that was presented in Section 6.2.4. Therefore BelPlatoon includes the beliefs about each members dynamic states (velocity, position at a specific time, role, acceleration, etc.) and other useful information like their platoon ID, their position in this platoon and their position according to the local agent's surroundings (front, rear, left, right). PlatoonMembersKB and BelPlatoon take in consideration each others' changes, so whenever a new belief is communicated by a team member, the update on BelPlatoon triggers a similar update on the agent's local KB: PlatoonKB. In the same way, the addition of a new knowledge in PlatoonKB may trigger a similar addition in BelPlatoon. This happens if the agent presumes that this new knowledge is already known by other team members, or if the agent decides to communicate (share) this new knowledge using the Selective Communication (SC) operator. Only the latter reason has been implemented in our application, as shown later in this section, since the first reason brings a lot of uncertainty. In summary, the relation between the agent's two main KBs is $BelPlatoon \subseteq PlatoonMembersKB$.

Apart from the team's shared *beliefset*, the teamwork capability CapTeamwork also uses a BelTeams *beliefset* that represents all the team formations currently known by this agent, as shown in Figure 6.18. In addition to this *beliefset*, the different possible types of teams are defined inside a static KB called TeamFormationKB. This KB is used by CapTeamwork since it contains all the information required to form and use a team, as well as the teams' logical roles relations.

6.3.2 Team Operators

To create the Team Oriented Programming (TOP) infrastructure required to implement the teamwork intra-platoon coordination model, all of the STEAM related operators and beliefs have been developed inside the CapTeamwork capability. This capability includes all the STEAM operators that were described in Section 2.2.4: (i) Monitor and Repair (MR); (ii) Coherence Preserving (CP); and (iii) Selective Communication (SC).

To support MR operators, "MR plans" have been developed to communicate MR oriented messages whenever a conflicting role state arises. For example, the plans presented inside the CapRoleX capability, in Figure 6.18, relate to a role X and they will ultimately set this role to an ending state, like *success* or *failed*, by calling the setRoleState method from TeamAgent. This method verifies if the role's post-condition has really been met, in the case the state was set to *success*. By calling the Postcondition method defined in the RoleDefinition interface, the role capability (CapRoleX) returns

a boolean expression relating to whether or not the role has realized its goal. In the latter case, a "MR plans" is raised from CapTeamwork.

CP operators have not been developed in the current TOP infrastructure, but they should be implemented inside the CapTeamwork capability to react to the same type of event as the previous MR actions. This task should be realized by using the logical role relations defined for each team formation, inside the TeamFormationKB.

Finally STEAM's SC operators are available in CapTeamwork and they are used to determine if a new local knowledge should be communicated to other team members. Figure 6.19 presents an example of the use of SC operators by depicting all the classes involved in the execution of such an operator. The flow of events is described by each rounded box representing activities and by arrows that represent information exchanged in the form of events and method calls. Figure 6.19 first shows that a new DynamicVehicleBelief (new distance) can represent a modification to the PlatoonMembersKB that is important enough to notify its observers. CapTeamwork being one of those observers, the SC operator is triggered to verify if this new belief should be communicated to the team. By verifying the status of its agent's beliefs, the SC operator may communicate the new belief (new distance) and update its team belief, considering the value returned by the SC function presented in Section 5.2.3.

6.3.3 Formation of Dynamic Teams

When an agent is part of a platoon formation where it only follows the preceding vehicle, it is considered as having the *follower* role in the "platoon formation" team, presented in Section 5.2.3. This team is considered as a static (persistent) team, which is the default team the agent is in when it leaves a task team. Therefore, an agent can leave its static team, only to form a task team and it automatically retrieves its status in the static team (if it has not left the platoon) when the task team is over, without necessitating any communications. The task teams include the "merge task" team and the "split task" team, which are the most complex teams because they are formed dynamically using the framework presented below.

The formation of dynamics teams (task teams) is supported by CapTeamwork, which includes two categories of team formation plans: (1) plans that initialize and form a team; and (2) plans that answer to other agents' queries to fill a role in their team. The first category of plan is triggered by the plan described as InitRoleX inside CapRoleX, in Figure 6.18. This plan launches the TeamFormation plan inside CapTeamwork, which represents a generic method to form any team. The TeamFormation plan uses TeamFor-

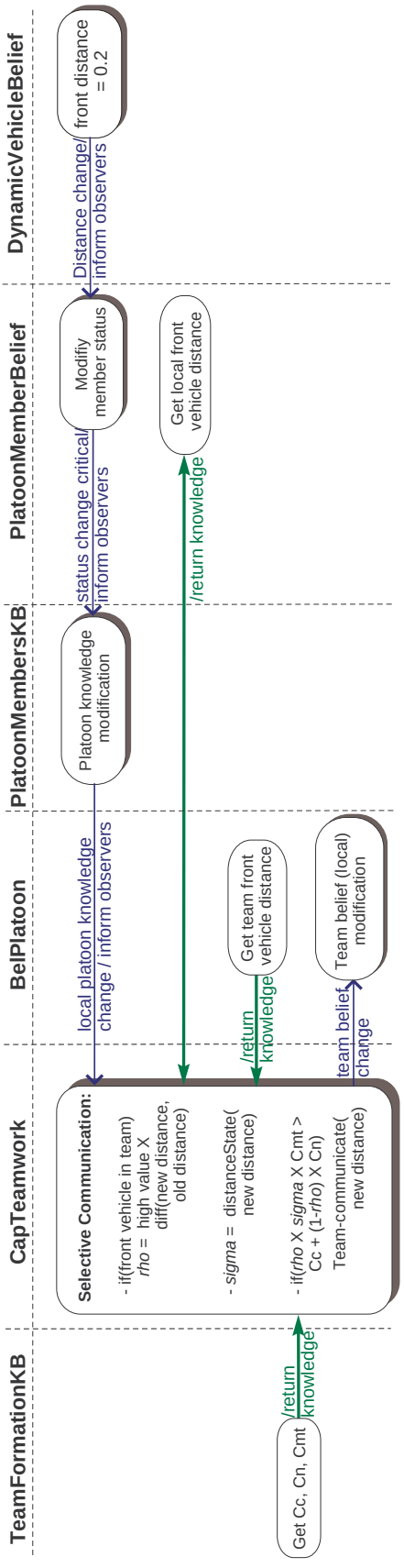

Figure 6.19: Diagram describing activities relating to each belief structure class in the scenario of shared belief states.

mationKB to get the team definition and send a team formation query to every possible team members.

On the other hand, the second category of plan is triggered by the arrival of a message sent by another agent that wants to form a team. To answer this team formation query, the RoleQuery plan from CapTeamwork uses its possibleRoles method to return a set of possible roles that the agent can fill in the team. For instance, an agent in a position preceding the entrance gap of a merging vehicle has the *Virtual Vehicle* role at the highest priority in the set of roles returned by possibleRoles. At last, the RoleQuery plan replies to the initial team formation query by specifying the role it wants to fill in that team, which is the one having the highest priority.

6.3.4 Discussion

Even though not all the TOP components have been implemented in our application, everything was initially defined to support them. Further possible additions may include the use of "unachievable conditions" inside the roles' definitions of the TeamFormationKB. These conditions could be implemented inside automatic events (supported by JACK), which would trigger "MR plans" at any moment, in the event a role would become unachievable. Furthermore, as it was mentioned before, logical relations between roles operators could be added in the definition of team structures in TeamFormationKB. CP operators could then be implemented as plans inside CapTeamwork and they would be triggered by observers monitoring team members' logical role relations inside BelPlatoon.

6.4 Driving Agent Coordination Experiments

The intra-platoon coordination models presented in Chapter 5 have been implemented using the models presented in this chapter and finally tested using the simulator presented in Chapter 3. The test that we ran in our simulator allowed us to get a first glance at the behaviors of our CDS and compare different intra-platoon coordination approaches. First, the centralized coordination model from Section 5.2.1, developed in accordance with the agent model of Section 6.2, was tested with platoon merge and split scenarios inside the simulator. Then, the same scenarios were used to test the teamwork coordination model from Section 5.2.3, developed in accordance with the agent model of Section 6.2 and 6.3. Finally, the decentralized model presented in Section 5.2.2 was

partly implemented and therefore only preliminary results referring to this model are presented here.

Before presenting the results based on software simulation scenarios, the limitations of our three coordination models are presented in Section 6.4.1. Afterwards, Section 6.4.2 describes the evaluation model we used for our preliminary test scenarios. Section 6.4.3 then presents the simulation results, followed by the analysis of each coordination model in Section 6.4.4. To conclude, Section 6.4.5 discusses the results and analysis of our intra-platoon coordination models.

6.4.1 Coordination Models Limitations

A Collaborative Driving System being a very complex system, many issues have been left aside within the first phase of development, related in this thesis. In order to focus on coordination aspects, control and realistic sensing issues have not been excessively detailed, since they were not considered as a priority. Therefore, the impact of a better control and sensing system should be considered during the analysis of our results on platoon coordination.

For instance, the simulated vehicle model presented in Section 3.3 can bias the longitudinal acceleration value when a high gas or brake throttle is applied. Likewise, when a vehicle is changing lane during a split or merge manoeuvre, the lateral and angular accelerations may also differ from a real driving environment. The lower-level controller presented in Section 4.4.2, which applies the right throttle or brake values considering an order on acceleration, is also a factor influencing our results. Since lower-level control issues were not mandated in our sub-project, the lower-level control was only developed to provide the basic functions required to support the coordinated driving tasks. Therefore, the improvement of this controller could also improve the results we present later in this section.

The sensing issues should also be considered as an important factor, influencing our results. First, the simulation model of our vehicles' sensors, presented in Section 3.4, provides in most cases a perfect information on the environment. Thus, our sensors do not modify the sensed data, in order to simulate the right error factor of the sensor units. Second, the current implementation of the *Intelligent Sensing* sub-layer of our architecture, presented in Section 4.4.1, does not filter the sensors' data to improve the values and give a better representation of the environment. Consequently, the sensed data provided by our CDS are not perfectly simulated and the two previous issues should be considered when analyzing our results.

6.4.2 Evaluation Model

The evaluation model used to analyze the results of the teamwork and centralized coordination models are based on platoon splitting and merging scenarios. Each scenario is simulated in the software environment presented in Chapter 3, on a straight road, with the same dynamics, sensing and communication models. Scenarios as the platoon split and merge are executed at a velocity of $20m/s$, when the platoon is stable (each member can maintain the right inter-vehicle distance and their velocity is stable) and they end when the platoon becomes stable again (the platoon is formed plus or minus one vehicle in the case of a merge or split). The inter-vehicle distance maintained by the vehicles, when they follow each others inside a platoon is a gap of 0.2 sec., while the gap created to allow lane changes (in front and behind the lane changing vehicle) is 0.5 sec. For each scenario, the *Guidance* layer presented in Section 4.2.1 is used by our driving agent and the control algorithm 2 is used to maintain inter-vehicle distances. Thus, only the way our agents communicate changes inside the results presented in Section 6.4.3. This section uses graphics on the vehicles' dynamic values, which highlight the differences between our coordination models. These results and additional information are then analyzed in Section 6.4.4, which points out the key differences and advantages/disadvantage of each coordination model.

The metrics used to analyze the teamwork and centralized models include:

1. the platoon members' acceleration.
2. the platoon members' velocity.
3. the followers' inter-vehicle distances.
4. the difference between each follower's inter-vehicle distance and the safe inter-vehicle distance.

These metrics are first presented inside a specific scenario: a noisy platoon merge scenario. The states or steps the platoon goes through when executing such a scenario are illustrated in Figure 6.20. The scenario starts with state *S1*, when vehicle 3 requests to merge the platoon lead by vehicle 1. Following this request, at state *S2*, a gap is created between vehicle 2 and 4, while vehicle 3 waits for stability before changing lane. From state *S2* and until state *S3*, vehicle 1 decelerates of -0.5m/s^2 to reach a velocity of about 17m/s. This deceleration results in the deceleration of vehicle 2 (following vehicle 1) in state *S3*. At state *S3*, vehicle 1 stops decelerating to accelerate of 0.5m/s^2 to reach back a velocity of 20m/s in state *S5*. At state *S4*, vehicle 3 starts changing lane, while vehicle 1 and 2 accelerate to reach back the velocity of 20m/s. During state *S4*, the noise created by the previous deceleration of vehicle 1 and 2 makes the merge

manoeuvre more complicated, as vehicle 3 must respect a safe distance (in time) of $0.5s$ with vehicle 2 when it enters the platoon. At state *S5*, vehicle 3 has merged the platoon and vehicle 2 is still accelerating to reach the right distance with vehicle 1 ($0.2s$). The distance between vehicle 2 and 3, in state *S5* depends on the coordination model and differs from one model to another, as shown in the graphics below. Finally, at state *S6*, the platoon is completely stable and all vehicles in the platoon have a distance of $0.2s$ between each others.

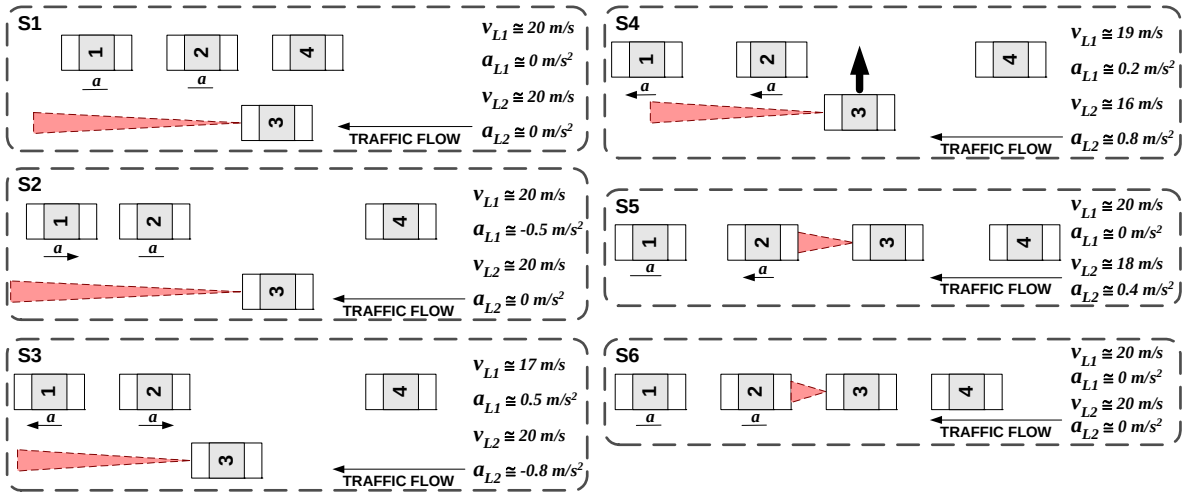

Figure 6.20: Noisy merge test scenario through the six main platoon states.

Following the presentation of results based on the previous four metrics with the noisy merge scenario, metric number 4 is used to analyze two possible instances of the noisy merge scenario for each coordination model (centralized and teamwork). The same metric is finally used to analyze results on the splitting scenario, when this scenario is executed without any noise (instability), using the centralized model with noise, and using the teamwork model with noise. Note that the noise in the split scenario is added in the exact same way as the merge scenario in Figure 6.20, where the leader decelerates about 2 seconds before the splitter changes lane. Finally, the last evaluation model, detailed in Section 6.4.4, analyzes the average amount of exchanged messages and the agent framework of each coordination model: centralized, hard-centralized, decentralized, teamwork.

As mentioned in Section 6.4.1, our control and sensing models are limited considering their realism, and therefore our evaluation does not use metrics as the gas and brake throttle values. Moreover, information relating to detailed analysis of the platoon stability, as the oscillation between vehicles, is not used to analyze our coordination models, since our lower-level controller may represent itself a source of instability. Results based on "crash" scenarios have also been omitted in our comparison of coordination models, since they only diverged on the communication protocol basis and not on the vehicles'

behaviors.

Finally, further test scenarios should include a variety of uncertain events as components failures (sensors, communication system, vehicle's components, etc.) or different driving conditions (traffic, weather, road, etc.) that could be arranged in different ways in order to generate multiple test scenarios. This would allow us to compare the ability of each coordination model to respond to uncertainty, which is a very important aspect that will nevertheless be mentioned in Section 6.4.4.

6.4.3 Simulation Results

As it was previously mentioned, preliminary simulation results have been collected according to four metrics, through different test scenarios. These initial test scenarios only evaluate the teamwork and centralized coordination model, while the other two coordination models presented in Section 5.2 (hard-centralized and decentralized) are only analyzed in Section 6.4.4. The hard-centralized model presents the same results as the centralized model in the case of a split manoeuvre, while in the case of the merge manoeuvre, it only differs in the time it takes to place the merging vehicle at the end of the platoon. Therefore, this difference is only suitable to compare the impact of the hard-centralized model on the global traffic, but not to compare the impact on the platoon members according to our metrics. On the other hand, the decentralized model cannot be compared using the same simulation results, since at the moment, it does not fully supports the simulation scenarios presented in this section.

To begin with, the simulation results of a noisy (unstable) merge scenario are presented using our previous four metrics. The same scenario was executed with both the centralized and teamwork coordination models and the graphics presented below show the behavior of each vehicle involved in this manoeuvre. This scenario is first described, then follows an analysis of the results shown in each graphic.

The noisy merge scenario begins in state *S1* of Figure 6.20, when vehicle 3 wants to merge the platoon lead by vehicle 1 and formed by two followers: vehicle 2 and 4. At that moment, all the vehicles try to maintain a velocity of $20m/s$, as shown in Figures 6.21 and 6.22. At time $15s$ in the centralized model and $16.5s$ in the teamwork, the merging vehicle starts decelerating to meet the entrance point of the platoon, as shown in Figures 6.23 and 6.24 respectively. Around the same time, vehicle 4 also decelerates to create the gap between this vehicle and vehicle 2 (gap of $0.5s$). Following these events, the merging vehicle (vehicle 3) manages to meet the platoon's entry point around time $35.5s$ for the centralized model and $36.5s$ for the teamwork model (state *S2* in Figure

6.20). At that time, vehicle 3 is ready to change lane and merge the platoon. However, around the same time, the "noise" or instability is added to the merge scenario. This noise comes from a sudden deceleration from the leader (vehicle 1) as shown in Figures 6.23 and 6.24. This deceleration is kept for 5 seconds and causes the platoon string to become unstable, which makes the lane change of vehicle 3 more difficult. During this moment of instability, vehicle 3 steers to change lane and enters between vehicle 4 and 2 (state *S4* in Figure 6.20). When vehicle 4 senses vehicle 3 with its front laser sensor and vehicle 3 senses vehicle 2, both vehicle have to adjust their velocity to maintain an inter-vehicle distance of $0.5s$ This fact is shown in Figure 6.25 and 6.26, after time $35s$, when vehicle 3 finally senses a vehicle in front, and the distance sensed by vehicle 4 becomes much lower. Recall that the front distances are calculated in seconds and they represent the inter-vehicle distance in meters divided by the vehicle's velocity. Finally, around time $44s$ vehicle 3 and 4, in both the centralized and teamwork models, have to close the platoon by accelerating to maintain an inter-vehicle distance of $0.2s$ instead of the previous $0.5s$.

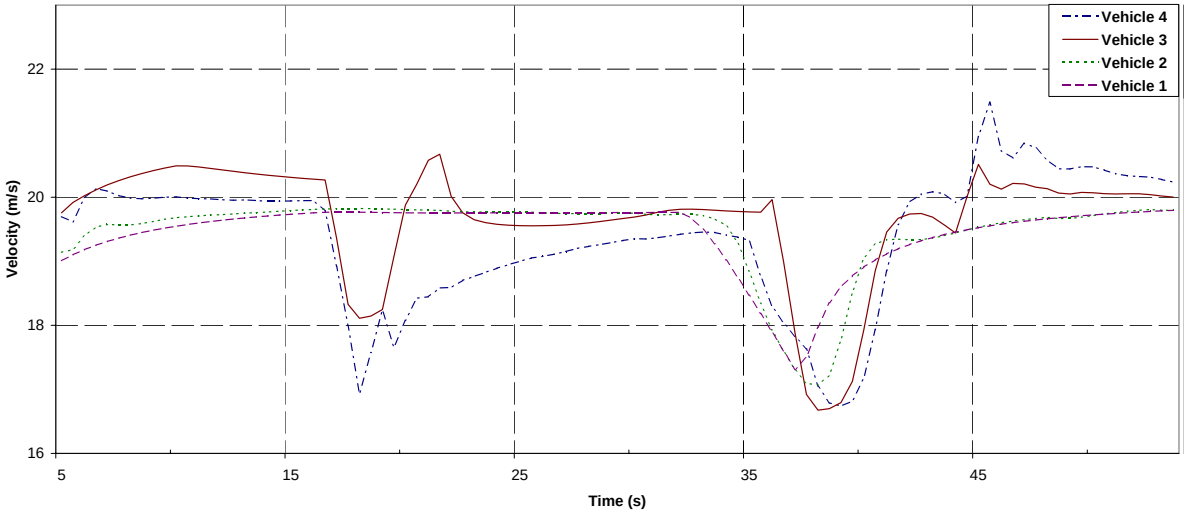

Figure 6.21: Vehicles' velocity in a noisy merge scenario using the centralized model.

The noisy merge scenario enables us to highlight the differences between the teamwork and centralized coordination models, when the platoon becomes unstable during the merge of vehicle 3 (the lane change). By comparing the curves in Figures 6.21, 6.22, 6.23 and 6.24, the acceleration and velocity values of vehicle 1 and 2 are practically the same since they are not affected by the merge occurring behind. However, vehicle 3 and 4 reach lower velocities (higher delta velocity) after time $35s$ (state *S4* in Figure 6.20) in the centralized model and their acceleration fluctuates more than the same vehicles in the teamwork model. Moreover, the instability added to this scenario creates a major problem for the centralized model to keep safe inter-vehicle distances between vehicle 4, 3 and 2, as shown in Figure 6.27 and 6.28. These figures show the difference between the inter-vehicle distances kept in front of vehicles 2, 3 and 4, and the front distance

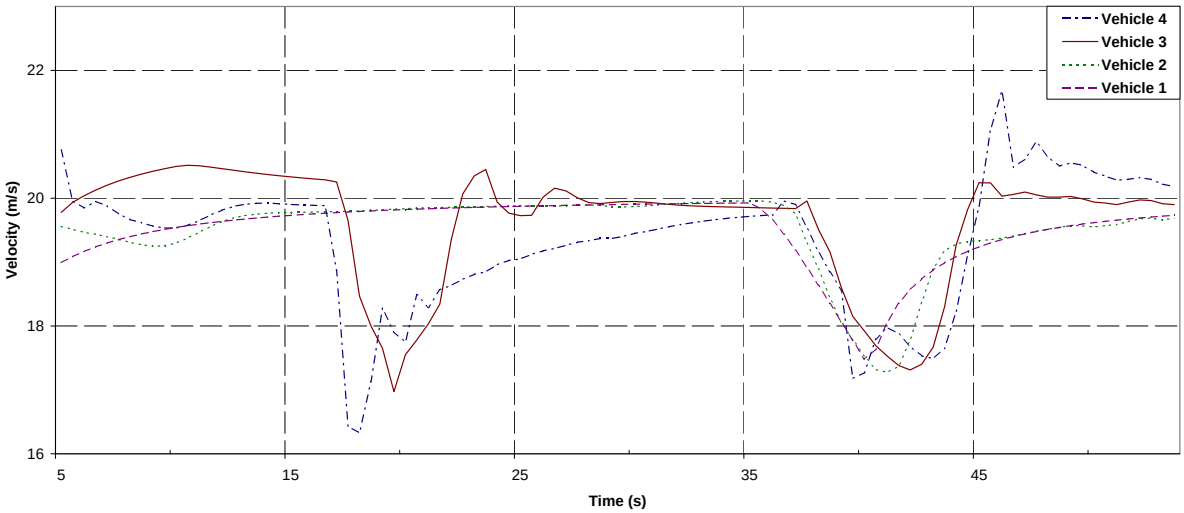

Figure 6.22: Vehicles' velocity in a noisy merge scenario using the teamwork model.

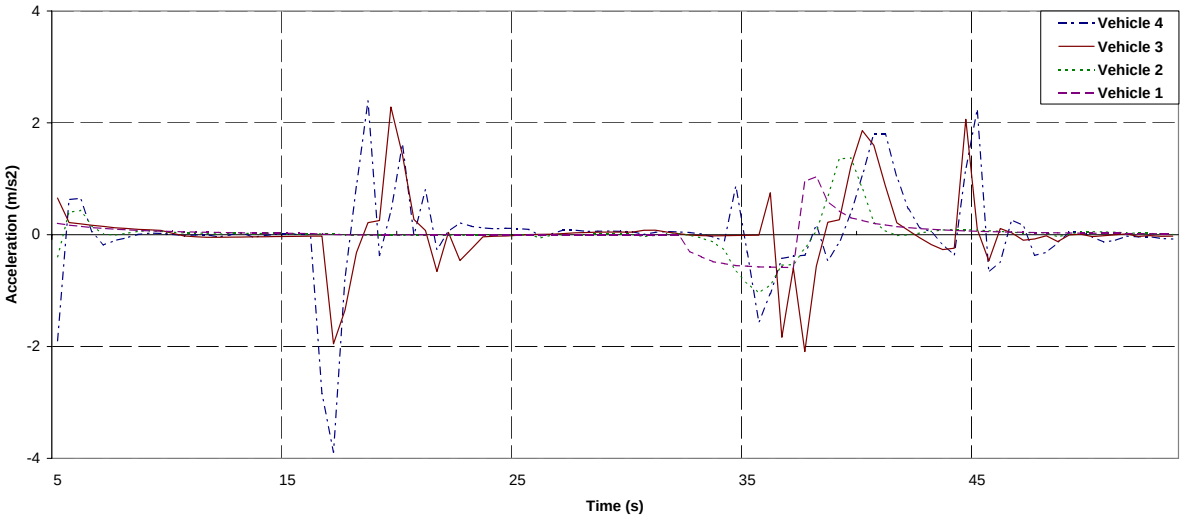

Figure 6.23: Vehicles' acceleration in a noisy merge scenario using the centralized model.

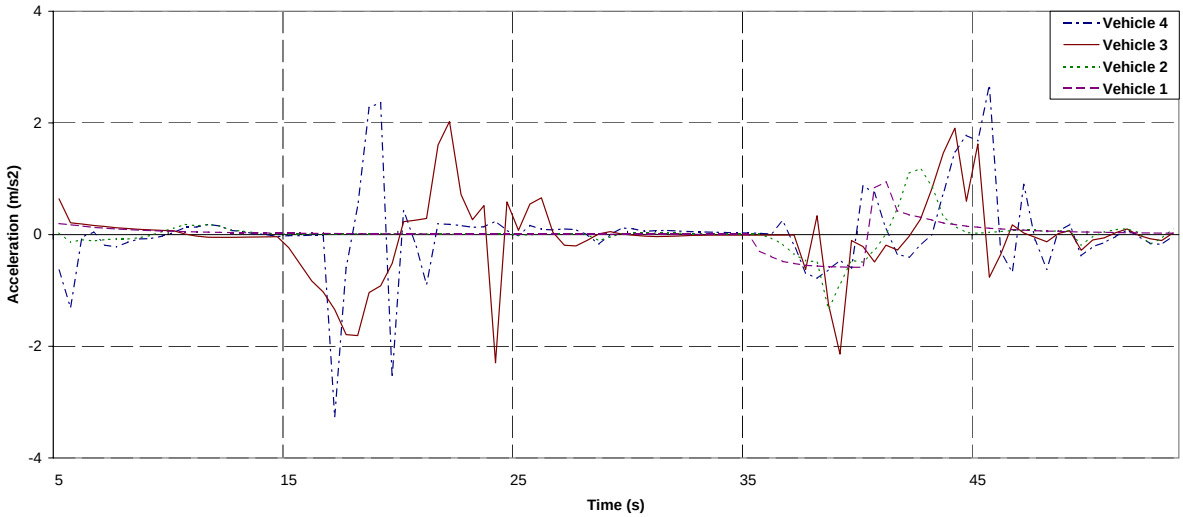

Figure 6.24: Vehicles' acceleration in a noisy merge scenario using the teamwork model.

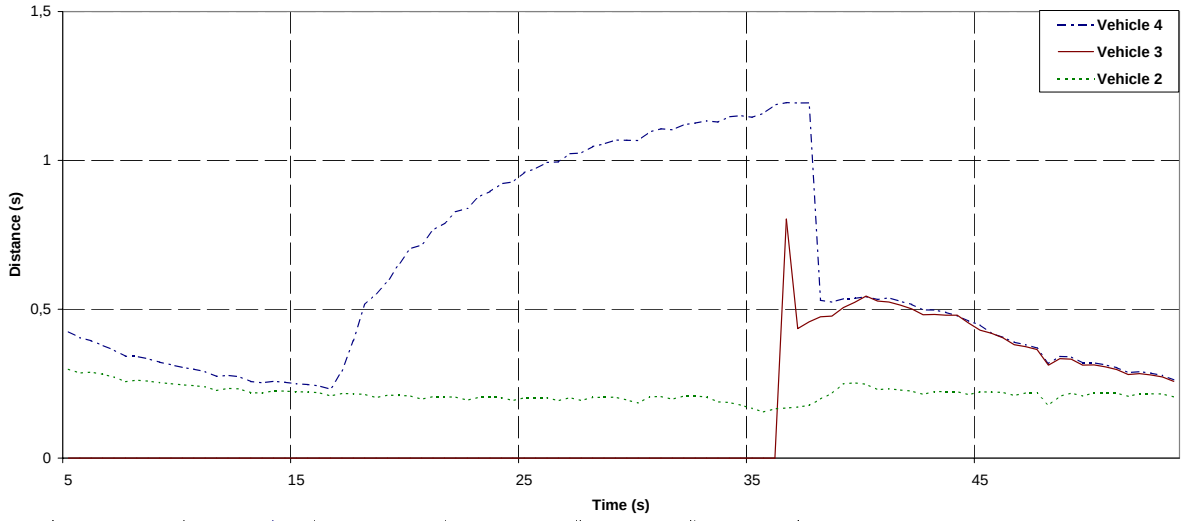

Figure 6.25: Inter-vehicle time distances in a noisy merge scenario using the centralized model.

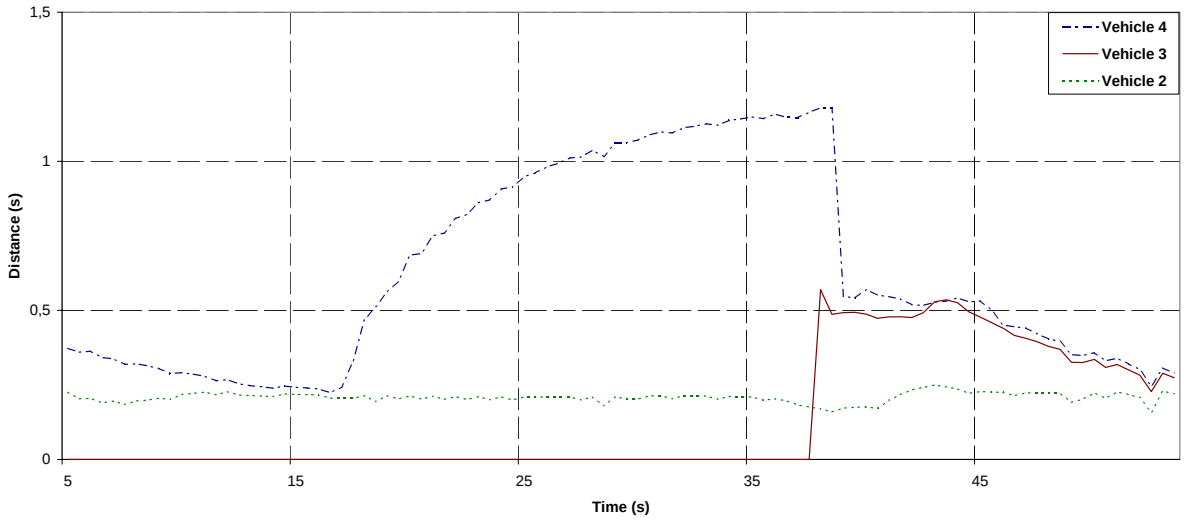

Figure 6.26: Inter-vehicle time distances in a noisy merge scenario using the teamwork model.

considered as "safe". In other words, these graphics show the ability of each vehicle to keep a safe distance with the preceding vehicle, which refers to the "error factor" that should correspond to a value of zero. In these two graphics (Figure 6.27 and 6.28), starting from time 16s, the front vehicle of vehicle 4 is considered as vehicle 3 and the safe distance is 0.5s, which explains a peak in vehicle 4's distance. The interesting difference between Figure 6.27 and 6.28 occurs at time 35s, when vehicle 3 changes lane to merge the platoon. In Figure 6.27, the vehicle 3 using the centralized coordination has difficulties keeping the distance at zero and the "error factor" reaches -0.08s and 0.04s Apart from being unsafe, the interval of 1.2s between these two distances values was created in only 4 seconds, which represents a great instability. On the other hand, the vehicle 3 using the teamwork coordination model (Figure 6.28) receives information on vehicle 2's position and velocity through the communication dictated by the "virtual vehicle" role, presented in Section 5.2.3. Vehicle 2 communicates this information in response to its sudden deceleration, resulting from the deceleration of vehicle 1. This allows vehicle 3 to have the knowledge of vehicle 2's new state and adjust to its velocity before or while changing lane. As a result, the "error factor" on vehicle 3's inter-vehicle distance only reaches -0.025s and 0.025s. Therefore, the teamwork model manages to keep safer distances and minimize the instability, as shown by its lower accelerations and changes in velocity.

In a different analysis, we decided to show two possible "turn of events" for the centralized coordination model used in a noisy merge scenario. In the previous noisy merge scenario, the centralized model managed to realize the merge manoeuvre in a time comparable to the teamwork model, but the results were not as safe. In contrast, if the noise appears a slight second before the merger changes lane, the leader senses the

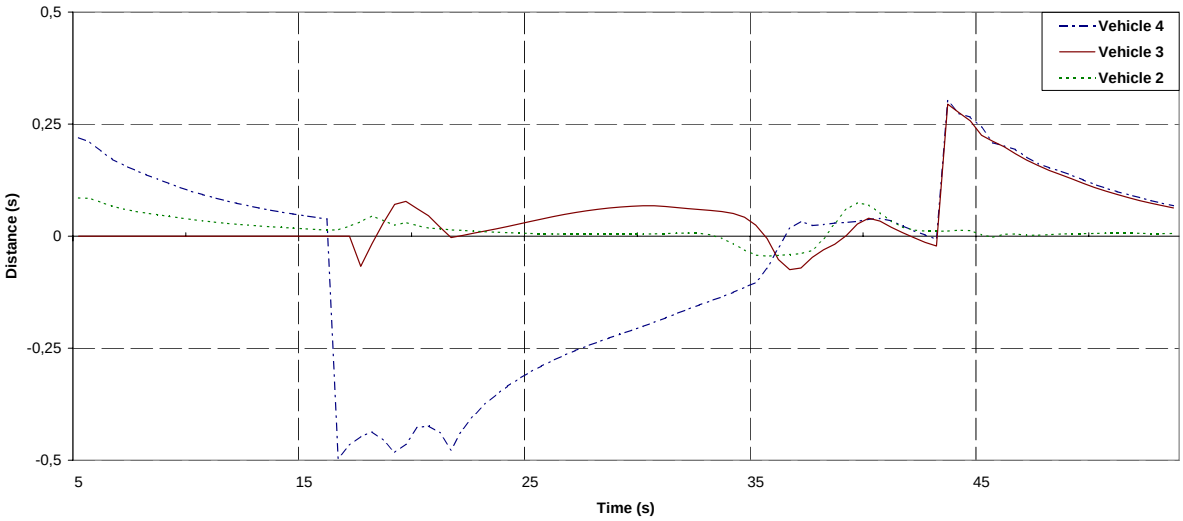

Figure 6.27: Difference with the inter-vehicle time distances and the safe distance, in a noisy merge scenario using the centralized model.

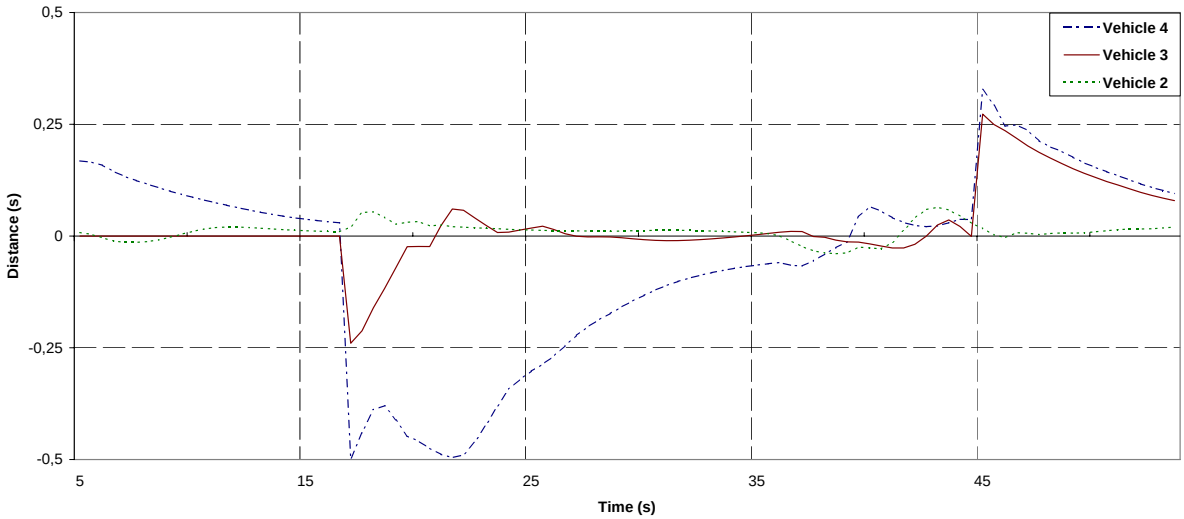

Figure 6.28: Difference with the inter-vehicle time distances and the safe distance, in a noisy merge scenario using the teamwork model.

noise in time and waits to gain stability before commanding the merger to change lane. In this case, vehicle 3 changes lane later, but the inter-vehicle distances are kept to safer values as shown in Figure 6.29. In this figure the "unsafe", but faster merge ends at time 43.5s (when the safe distance rises because of the new safe inter-vehicle distance of 0.2s) for vehicle 3, while the slower but safer merge ends at 54.5s. In conclusion, the results based on the centralized model can be as safe or safer than the results we showed from the teamwork model, at the cost of a slower execution time.

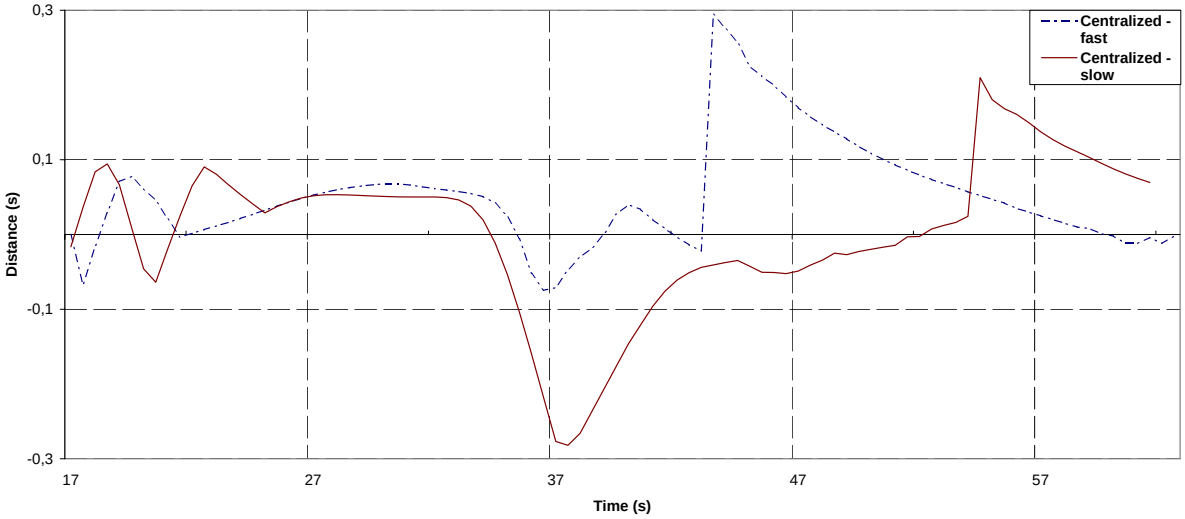

Figure 6.29: Difference with the inter-vehicle time distances and the safe distance of the merging vehicle, in two merge scenarios using the centralized model.

The safest results in a noisy merge scenario can be achieved by using the teamwork model with more communications from the "virtual vehicle". This fact is shown in Figure 6.30, which presents the different safe distances kept by vehicle 3 in the teamwork model. This figure compares a scenario where the "virtual vehicle" communicates 3 messages about its position and velocity, with a "virtual vehicle" communicating only 1 message. By adjusting the variables of the SC team operator, presented in Section 5.2.3, we managed to lower or raise the interval of messages sent by the "virtual vehicle". This resulted in a safer and faster execution of the merge manoeuvre in the case of a "virtual vehicle" sending more messages. Consequently, the teamwork model allows us to adjust the SC operator in order to spare the communication bandwidth or use all the bandwidth to enable a better and safer execution of the platoon manoeuvres.

By comparing the results of a normal merge scenario with noisy merge scenarios based on both the centralized and teamwork models like we did in Figure 6.31, it is easier to draw conclusions on our coordination models. This figure shows the three major possible different results on the safe distance kept by vehicle 3 during its merge. During a normal merge scenario, without any noise, the vehicle 3's error on the front distance

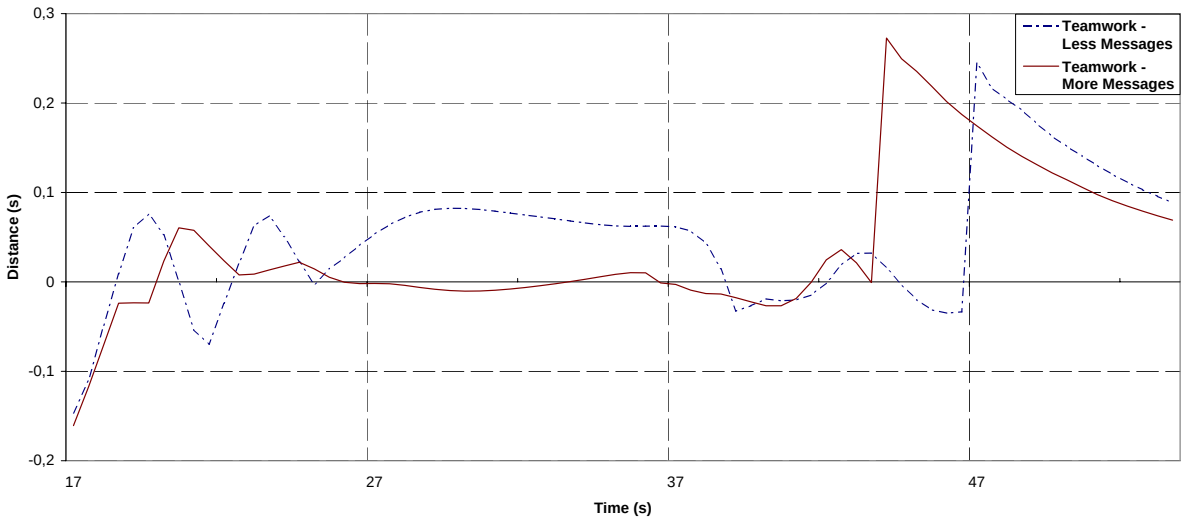

Figure 6.30: Difference with the inter-vehicle time distances and the safe distance of the merging vehicle, in two merge scenarios using the teamwork model.

only reaches -0.015s and 0.015s, which results in a merge manoeuvre being executed in a very safe way. On the other hand, a noisy merge scenario coordinated with the teamwork model makes it more difficult for vehicle 3 to keep a safe front distance during the lane change occurring around time 37s. However, the "error factor" on the distance kept using the teamwork model is less than half the error of the centralized model in the same situation: -0.08s and 0.04s for the centralized model; -0.025s and 0.025s for the teamwork model. Another interesting difference between these two coordination models is the time they require to execute the noisy merge scenario. Indeed, the centralized model manages to execute the lane change task faster than the teamwork model, since the teamwork model required more time at the very beginning of the merge manoeuvre to form the merge team. On the other hand, the teamwork model completes the merge manoeuvre faster than the centralized model since it does not require as much time to stabilize the platoon after the entrance of vehicle 3 in the platoon.

In the scenario of a vehicle splitting from a platoon, similar conclusions can be drawn. In Figure 6.32, the results on the error of vehicle 3's (the splitting vehicle) front distance are compared when using a scenario without noise (normal), the teamwork coordination model with noise and the centralized model with noise. In this example, the execution of vehicle 3's lane change happens at time 21.5s on the three curves and the following results differ greatly. When the lane change is execute, it should be recalled that the splitting vehicle must maintain a safe distance with its preceding vehicle in the platoon it just left, until the splitting vehicle completely and safely reaches the other lane. Therefore, even though the splitting vehicle does not sense its preceding vehicle in the platoon, it must continue keeping a safe distance with it.

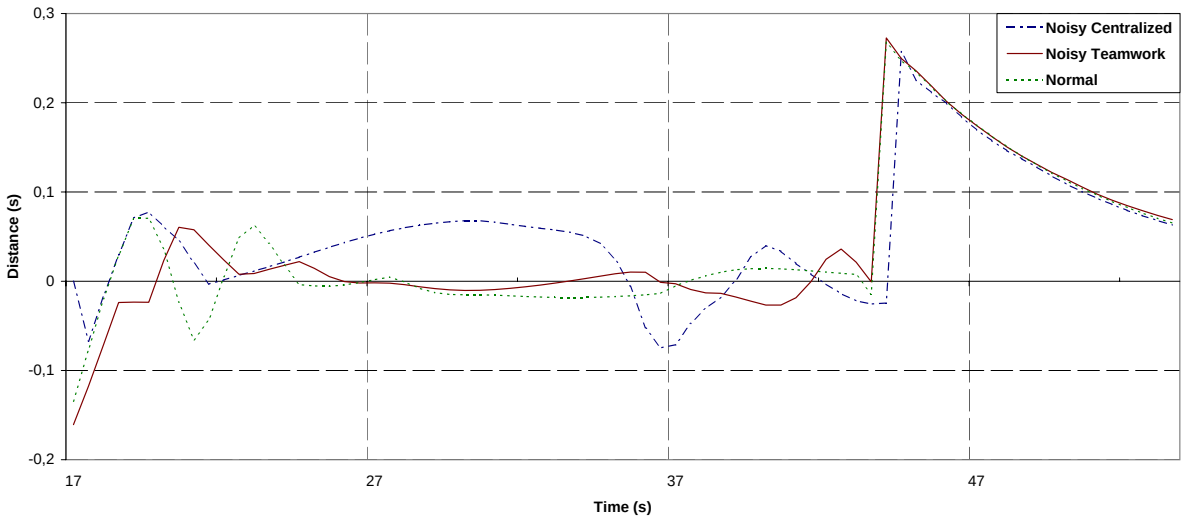

Figure 6.31: Difference with the inter-vehicle time distances and the safe distance of the merging vehicle, in three merge scenarios using different coordination models.

In the case of the normal scenario shown in Figure 6.32, the safe distance is kept quite easily. The teamwork has a little more difficulties to keep the safe distance since its results are based on a noisy split scenario where the leader decelerates during the lane change. But the information communicated by the "virtual vehicle" (vehicle 2) around time $27s$ allows vehicle 3 to accelerate and keep a safe distance even though it does not sense vehicle 2 on its sensor anymore. With the centralized model executed in the same noisy split scenario, vehicle 3 has much more difficulties to keep a safe distance with vehicle 2 after it has changed lane (time $21.5s$). After the lane change and before the split has been complete, the safe distance reaches a value of -0.7s because the leader decelerated and vehicle 3 was not aware. This means that vehicle 3 is beside vehicle 2 around time $30s$ (when the split manoeuvre ends), which may causes both vehicles to touch each others.

By summarizing the previous results, our test scenarios demonstrated that the teamwork coordination model was more appropriate when the platoon becomes unstable or when uncertain events arise. However, the centralized model ensures a limited amount of messages, as shown in the next section, even in unstable situations where the teamwork requires more messages to ensure safety. On the other hand, the execution time in unstable situations with the centralized model can be a lot slower. Apart from being safer than the centralized model, the teamwork model diminishes platoon instability, which create waves inside the overall highway traffic. Indeed, the deceleration of vehicle 1 and 2 in our noisy merge scenario creates a wave, which is stopped quickly in the teamwork model. This fact can be illustrated by the variations in acceleration we showed earlier, where the teamwork model resulted in smaller variations than the centralized model, for the same scenario.

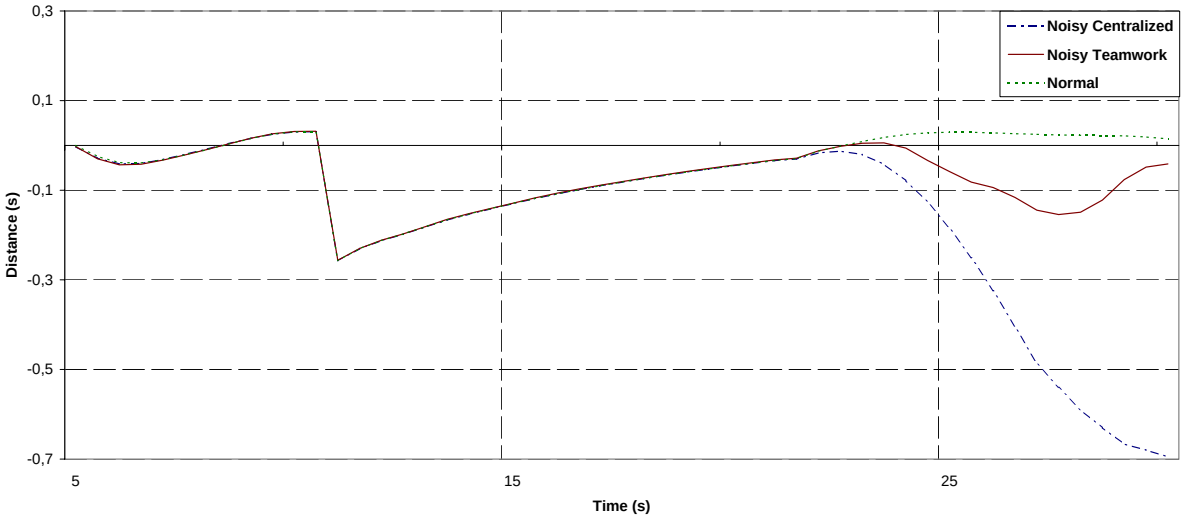

Figure 6.32: Difference with the inter-vehicle time distances and the safe distance of the splitting vehicle, in three split scenarios using different coordination models.

6.4.4 Models Analysis

In the previous section, graphics on the dynamic states of the vehicles involved in the split and merge manoeuvres using both a centralized and teamwork coordination model were presented. Now, all four intra-platoon coordination models: centralized, hard-centralized, decentralized, teamwork, presented in Section 5.2, are analyzed based on different aspects, but mainly communication and flexibility issues.

To begin this analysis, the amount and size of broadcasted messages during the split and merge manoeuvres in the average scenario, for each of the four coordination models, are presented in Table 6.1. "Nb Messages" represents the number of messages, of any size, that have been broadcasted to one or many vehicles during this manoeuvre. "Messages Size" is the total amount of bits that have been transmitted through messages during this manoeuvre. Table 6.1 also presents the amount of "JACK plans" required to support each coordination model, showing the flexibility of their respective framework. These JACK plans refer to the coordination plans presented in Section 6.2.6, making it possible to support a specific manoeuvre, with a specific communication protocol. Note that strategies on decentralized approaches are still under development, so the results on the decentralized model presented in this table are only initial results that may vary depending on the extension that should be added in the future.

By using the results presented in Table 6.1, along with the results presented in Section 6.4.3, each coordination model were analyzed and their respective advantage and disadvantage are summarized as follows:

Table 6.1: Total of messages and plans used by coordination model.

	Nb Messages		Messages Size		JACK Plans	
Coordination	Merge	Split	Merge	Split	Merge	Split
Centralized	9	6	506	112	18	14
Hard-Centralized	7	-	486	-	16	-
Decentralized	10	7	674	290	20	13
Teamwork	11.25	8.25	882	454	10	8

1. **The hard-centralized model:**

   Advantage: It exchanges the lowest amount of messages for the merge manoeuvre since it forces the merging vehicle to insert itself at the end of the platoon (refer to Table 6.1). This model sometimes has a faster execution time than the centralized model, since it does not wait for the platoon to create a merging gap before changing lane.

   Disadvantage: Depending on its position and the current traffic on the highway, this model may require more time for the merging vehicle to place itself at the right position. To prove this fact, test scenarios in dense traffic situations will have to be realized, so at the moment, this is only an hypothetical analysis. Another disadvantage that should be proven through further test scenarios is the traffic disturbance that a merge manoeuvre, coordinated with the hard-centralized model, should create. As the merger must reach the platoon's tail by either accelerating or decelerating (considering its position), it creates traffic waves, which diminish the highway's capacity.

2. **The centralized model:**

   Advantage: It requires a low amount of messages to coordinate the split and merge tasks, as shown in Table 6.1. This can be explained by the fact that the leader coordinates every aspect of the manoeuvres and following agents do not have to request and confirm the role of each vehicle in the manoeuvre, so the leader can only request tasks from other vehicles. In addition, in other models like the decentralized and teamwork models, each platoon leader must have a knowledge of their platoon formation to respond to split and merge requests, while the centralized model only requires one agent to have this knowledge: the leader. A final reason explaining this low amount of messages compared with the teamwork model is the fact that the centralized model does not send messages to update the state of the splitter or merger's front vehicle when this vehicle changes lane.

Disadvantage: The latter reason explaining the lower amount of messages sent in a centralized model results in a disadvantage mentioned in Section 6.4.3. During our "noisy" merge and split scenarios, the merging and splitting vehicles had more problems keeping safe distances when they changed lane. This resulted in dangerous situations or in a manoeuvre that required a lot more time to be executed, as shown in Figure 6.29. Another disadvantage of the centralized model is the fact that in average, more than three quarters of the messages were sent or received by the leader, creating a bottleneck for this vehicle. As for the previous model, the centralized coordination uses static coordination protocols supported by the leader, which has the disadvantage of not allowing much flexibility on the coordination of unexpected situations. Indeed, during a merge manoeuvre, if the platoon velocity changes after the meeting point had been communicated to the merging vehicle, the leader will either cancel the task or wait until it retrieves its previous state. In the case of the "slow centralized" example presented in Figure 6.29, the platoon was able to retrieve its previous state after 10 seconds, but in the case of dense traffic, the leader would have to cancel the task, as opposed to the teamwork model which can communicate the new platoon state through the "virtual vehicle". Finally, a model centralized on one unit is less flexible and robust since the whole platoon relies on only one vehicle. For instance, if the leader has problem with its communication devices, it could cause all of its followers to crash.

3. **The decentralized model:**

Advantage: This model has more flexibility than the centralized model since the leader and its followers have a similar degree of autonomy. The decentralized model has the advantage of involving only two vehicles, while the rest of the platoon only has to update its knowledge at the beginning and end of a manoeuvre. Therefore, the leader is not "flooded" by all the messages, as opposed to the centralized model, and fewer vehicles have to communicate, as opposed to the teamwork model.

Disadvantage: Since the decentralized model is not based on a common architecture (like STEAM) and it only relates to social laws, its communication protocols are developed in a "static", less generic way. This makes the extension of the decentralized model more complicated since specific laws have to be developed for every situations, which explains the high amount of plans required to support this model (refer to Table 6.1). Like we mentioned it for the centralized model, the decentralized model is not as safe as the teamwork model, which can update the platoon state through the "virtual vehicle". In

fact, this model is probably the most unreliable coordination model in dangerous situations, since it only involves two vehicles close to each others and does not have any percepts about the situation ahead of the platoon. However, this fact should be proven through test scenarios involving multiple uncertain events. Finally, this model also needs to communicate to initialize and maintain common knowledge within the platoon, which explains a higher amount of messages than the centralized model.

4. **The teamwork model:**

   Advantage: By using this coordination model, more vehicles are involved in a manoeuvre, but this has the advantage of dividing equally the communication load involved in the coordination. As mentioned previously, and as proved in Section 6.4.3, the teamwork model is the coordination model that can handle uncertain situations or other types of instability in the platoon. Indeed, vehicles executing a manoeuvre can adapt themselves to dynamic changes, as other models (centralized, decentralized) either have to wait for stability, cancel the task or execute it in an "unsafe" manner. Another advantage of the teamwork model is its TOP framework, which extends the STEAM architecture, as presented in Section 6.3. This framework enabled us to lower the amount of coordination plans required for teamwork, since only plans specific to the CDS had to be created to support the teamwork coordination in Auto21. Table 6.1 showed that almost half of the total of plans of the other coordination models were required to develop the teamwork model. Therefore, this generic framework has the advantage of making extensions to the teamwork model easier to develop.

   Disadvantage: The most important disadvantage of the teamwork may be the amount of communications it requires. In order to assign roles and maintain a common belief inside the teams, vehicles in the teamwork model exchange more messages than the centralized model, as shown in Table 6.1. Moreover, the teamwork model also exchanges more messages than the decentralized model, but this is explained by the fact that other vehicles (mainly the "virtual vehicle") sometimes have to communicate updates on the platoon state during unstable situations. Another communication aspect that may be problematic is the fact that vehicles executing roles as the "safety observer" and "virtual vehicle" (presented in Section 5.2.3) use Selective Communication (SC) operators that communicate depending on variable probabilities. Thus, the amount of communication may escalate, during unstable situations, but the SC operators should set their variable in relation with the available bandwidth, and this should resolve the problem.

6.4.5 Discussion

Our simulation results gave us a great perspective at the behaviour of our coordination models, which enables us to lead further research considering aspects of CDS as inter-vehicle communications, execution time and vehicles' safety. The hard-centralized or centralized models present a good option if the most important issue is to minimize communications and lower the autonomy of the followers, in order to prevent 'unwanted" behaviors. The decentralized and teamwork models on the other hand, can be benefit on the execution time and the autonomy they give to followers often results in increasing safety, because followers decide which action to execute instead of receiving ordered actions by the leader. Nevertheless, the greater autonomy proposed by those two models can raise problems in future test scenarios, since the task of coordinating vehicles in a decentralized way is more complex than coordinating them using a master entity like the leader. A final note that should be mentioned about the decentralized and teamwork models is that they rely on a broadcasting communication system (point-to-multipoint) to exchange messages among neighboring (platoon members) vehicles. Therefore, the amount of communication they require would greatly increase if we would use point-to-point communications.

Finally, by "theoretically" comparing our coordination models with similar models like Sakaguchi et al. [2000]'s platoon architecture based on a token-ring (presented in Section 4.1.2), our models, and mainly the teamwork model, manage to minimize communications since they do not communicate on a constant interval. However, in order to provide a demonstration version of our CDS on real vehicles, multiple test scenarios involving uncertain events will have to be executed in our simulator. This should enable the teamwork model to learn and adapt the variables of its SC operators considering different contexts for each manoeuvre.

Chapter 7

Conclusions

This thesis began with the description of the Auto21 project and different problems relating to Intelligent Transportation Systems (ITS). Then, a wide overview of the motivations behind the development of a Collaborative Driving System (CDS) was presented and details were given on the motivation behind our research project. Then we defined the objectives of the CDS project in Auto21 and more specifically the objectives of the project relative to DAMAS laboratory, to end with the objectives of this thesis.

The following chapter presented an introduction to agent and Multiagent systems, followed by details on the agent coordination models that related to the models we tested for the coordination of our platoons. Then, the simulator we developed at DAMAS (the HESTIA simulator) was described by focusing on aspects such as: (i) the 3D environment; (ii) the vehicle dynamics; (iii) the sensory system; (iv) the inter-vehicle communications; (v) the driving system interface; and (vi) the collaborative driving scenarios. After the presentation of our simulated driving environment, the Auto21 driving agent architecture was described. This description began with a brief overview of other Collaborative Driving System (CDS), followed by the description of our hierarchical decomposition, the architecture's software engineering, and the techniques and schemes behind the integration of our architecture. Following this description, the architecture's *Coordination* sub-layer was detailed by presenting different models of inter- and intra-platoon coordination. The integration of the architecture inside an agent-oriented model was finally presented in the last chapter. This presentation began with an overview of the Agent UML (AUML) methodology and was followed by the description of the agent oriented models we used to develop our Multiagent System (MAS). These models provided a more applicative view at our autonomous driving system and focused on the communication aspects, handled by the agent framework. To end this chapter, results on simulation scenarios focusing on the coordination aspects of our

agents were presented and analyzed.

7.1 Contributions

This thesis brought many contributions to both the fields of MAS and CDS. These contributions can first be described by relating to the objectives presented in Section 1.4, which have been achieved in this thesis.

- The architectures and techniques that should be used to build automated vehicles have been studied and presented. A detailed report on these architectures was produced written in Hallé et al. [2003] to complete this thesis.
- A flexible and reusable real-time control architecture was developed considering the specific needs of our CDS project.
- Different coordination techniques, to be used inside the previous architecture, were analyzed and the most promising ones were developed and presented in this thesis.
- The previous coordination techniques were adapted to the problem of coordination of a platoon of vehicles. They were incorporated inside our driving agent model considering our driving environment's specifications.
- Different vehicle simulators were analyzed and a CDS simulator was designed. This simulator was designed considering the specific needs of our CDS project and considering the languages and frameworks used to develop our driving agents.
- The previous design of simulator was developed and adapted to the needs of our project, throughout its progress.
- The coordination and communication models we proposed for our CDS were tested under different platoon scenarios and an analysis presented the advantage and inconvenient of each model.

In addition to the contributions relating to our initial objectives, this thesis also contributed to advances in different areas:

- In the area of simulation of autonomous driving systems, our simulator provided a first implementation of such simulator in Java. Moreover, our simulator is probably one of the only simulator of its category that can run on a normal computer, which is a major advantage.

- In the area of simulation at the DAMAS laboratory, this new knowledge in simulation environment is very benefic to other members. In addition, our simulator is reusable for other similar projects.
- In the area of agent engineering, the use of the new methodology of AUML reinforced the popularity of agent-oriented engineering concepts, which are not very well known at the moment. More specifically, the models relating to the implementation of an agent framework based on JACK presented one of the rare high-level framework using JACK.
- In the area of real-time control, the integration of our architecture inside an agent-oriented model presented a practical application of agent models like BDI, as a "wrapping" entity for vehicle controllers and sensors technologies.
- In the area of agent teamwork, our implementation of the STEAM architecture with an extension specific to the domain of CDS provided a new working application of this theory and helped increasing the popularity of teamwork models.
- In the area of communication for automated driving systems, we developed a novel approach based on the teamwork model, which presented promising results. Moreover, the models of decentralized intra- and inter-platoon coordination also presented new concepts for this domain, but their results are still to come.

7.2 Concluding Remarks

To conclude this thesis, it should be recalled that the use agent teamwork models look promising for the CDS domain and these models should be pushed forward a lot more. Our research, based on Multiagent System (MAS) is different from previous research project on automated platoon of vehicles, since we propose a decentralized communication approach and the theory of teamwork is the key to it.

It should also be mentioned that compared to similar CDS projects as the PATH project, our project only employed a very small team, but we managed to develop a working demonstration of an automated platoon of vehicles. Of course, our advances were not the same, but as a first step in the development of a CDS in Canada, we can be proud of what we have accomplished.

Finally, the Collaborative Driving System presented in this thesis could be used in a fully autonomous system, using vehicles equipped with longitudinal and lateral guidance system. But within the presented scenarios, we did not specify if the lateral

control was automated or a simulation of a human driver, thus we are still opened to both avenues. This collaboration system could then be used in a navigation system inside a car as an Adaptive Cruise Control (ACC), which would more easily acquire the public's favor.

7.3 Future Work

The research presented in this thesis related to a vast field of research going from vehicle simulators, to vehicle automated controls and agent coordination methodologies. Since this was a first, in the development of CDS at DAMAS and inside the Auto21 project, we had to focus on all these fields at once, but now that the bases of our project are set, we can move on and extend our coordination techniques. The key to success, if we want to compete with projects as PATH, will be to centralize our efforts on specific and innovative aspects of inter-vehicle communications for automated vehicles.

As this project's future works, the longitudinal guidance system should be improved, which could enable us to lower the communication probabilities in the selective communication decisions of the Team Oriented Programming (TOP) infrastructure. The different coordination strategies should be further extended and many more scenarios involving uncertainty should be taken into account using the simulator. These additional simulation scenarios should include at some point "simulated human driver" to show the behavior of our system in mixed traffic. Finally, the teamwork model should also be extended using RMTDP role re-allocation strategies [Nair et al., 2003] and reinforcement learning applied to the decisions on communication, to improve the results involved in those new simulation scenarios.

Bibliography

AHSRA (2004). Evolving ahs. [Online]. http://www.wrsn.net/aboutus.htm, (accessed the 24th of August 2004).

AOS (2004). JACK Intelligent Agents^{TM} 4.1. Software Agents Development Framework.

Auto21 (2004). [Online]. http://www.auto21.ca/, (accessed the 30th of April 2004).

Balmer, M., Cetin, N., Nagel, K., and Raney, B. K. (2004). Towards truly agent-based traffic and mobility simulations. In *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 1, pages 60–67, New York, USA.

Bana, S. V. (2001). Coordinating automated vehicles via communication. Ucb-its-prr-2001-20, University of California, Berkeley.

Barth, M. J. (1997). *Automated Highway Systems*, chapter The Effect of AHS on the Environment, pages 265–292. Plenum Press, New York.

Basnayake, C. and Lachapelle, G. (2003). Accuracy and reliability improvement of standalone high sensitivity gps using map matching techniques. In *Proceedings of Annual Meeting, U.S. Institute of Navigation*, pages 209–216, Albuquerque, N.M.

Bauer, B. (2001). Uml class diagrams revisited in the context of agent-based systems. In Ciancarini, P. and Weiss, G., editors, *Proceedings of Agent-Oriented Software Engineering (AOSE 01)*, number 2222 in LNCS, pages 1–8, Montreal, Canada. Springer-Verlag.

Beirness, D., Simpson, H., and Desmond, K. (2002). The road safety monitor 2002: Risky driving. Technical Report 02H, Traffic Injury Research Foundation.

Blosseville, J. M., Hoc, J. M., Riat, J. C., Wautier, D., d. l. Bourdonnaye, A., Artur, R., Tournié, E., Narduzzi, C., and Gerbenne, E. (2003). French contribution to the functional analysis of 4 key active safety functions (arcos project). In *Proceedings of the 10 th ITS World Congress*, Madrid, Spain.

Bose, A. and Ioannou, P. A. (2001). Evaluation of the environmental effects of intelligent cruise control vehicles. *Journal of the Transportation Research Board*, 1774:90–97.

Burckhardt, M. (1993). *Fahrwerktechnik: Radschlupf-Regelsysteme*. Vogel-Verlag, Würzburg.

Cabri, G., Leonardi, L., and Zambonelli, F. (1997). Coordination in mobile agent applications. Technical report no. dsi-97-24, Universita di Modena.

Cannon, M., Basnayake, C., Crawford, S., Syed, S., and Lachapelle, G. (2003). Precise gps sensor subsystem for vehicle platoon control. In *Proceedings of ION GPS/GNSS-2003*, pages 213–224, Portland, OR.

Conde, C., Ángel Serrano, Rodríguez-Aragón, L. J., Pérez, J., and Cabello, E. (2004). An experimental approach to a real-time controlled traffic light multi-agent application. In *Proceedings of AAMAS-04 Workshop on Agents in Traffic and Transportation*, pages 8–13, New York, USA.

DAMAS-Auto21 (2004). [Online]. http://www.damas.ift.ulaval.ca/projets/auto21/, (accessed the 30th of April 2004).

Darbha, S. and Rajagopal, K. (1998). Intelligent cruise control systems and traffic flow stability. California PATH Research Report UCB-ITS-PRR-98-36, Texas A&M University.

Davidsson, P., Henesey, L., Ramstedt, L., Törnquist, J., and Wernstedt, F. (2004). Agent-based approaches to transport logistics. In *Proceedings of AAMAS-04 Workshop on Agents in Traffic and Transportation*, pages 14–24, New York, USA.

Daviet, P. and Parent, M. (1996). Longitudinal and lateral servoing of vehicles in a platoon. In *Proceedings IEEE of the Intelligent Vehicles Symposium, 1996*, pages 41–46.

de Bruin, D., Kroon, J., van Klaveren, R., and Nelisse, M. (2004). Design and test of a cooperative adaptive cruise control system. In *Proceedings of the 2004 IEEE Intelligent Vehicles Symposium*, pages 392– 396. Delft University of Technology.

Dresner, K. and Stone, P. (2004). Multiagent traffic management: A reservation-based intersection control mechanism. In *Proceedings of 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 2, pages 530–537, New York, USA.

Durfee, E. (1999). Distributed problem solving and planning. In Weiss, G., editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 121–164. MIT Press, Cambridge, MA.

Durfee, E. and Lesser, V. (1987). Using Partial Global Plans to Coordinate Distributed Problem Solvers. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 875–883.

Elliott, B. J. (1999). Road rage: media hype or serious road safety issue? In *Proceedings of the Third National Conference on Injury Prevention and Control*, Australia.

FIPA (2002). Contract net interaction protocol specification. Technical Report SC00061G, Foundation for Intelligent Physical Agents. http://www.fipa.org/specs/fipa00061/SC00061G.html, (accessed in April 2004).

Georgeff, M. and Ingrand, F. (1990). Real-time reasoning: The monitoring and control of spacecraft systems. In Press, C. S., editor, *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications(CAIA 90)*, volume 1, pages 198–205, Los Alamitos, Calif.

Georgeff, M. and Lansky, A. (1987). Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-87)*, pages 677–682.

Giampapa, J. A. and Sycara, K. (2002). Team-oriented agent coordination in the retsina multi-agent system. Technical Report CMU-RI-TR-02-34, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA. Presented at AAMAS 2002 Workshop on Teamwork and Coalition Formation.

Gillen, D., Chang, E., and Johnson, D. (2000). Productivity benefits and cost efficiencies from its applications to public transit: The evaluation of avl. California PATH Working Paper UCB-ITS-PWP-2000-16, University of California, Berkeley.

Girard, A. R., de Sousa, J. B., and Hedrick, J. K. (2001). An overview of emerging results in networked, multi-vehicle systems. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 2, pages 1485–1490, Orlando, Florida.

Grosz, B. and Kraus, S. (1999). *Foundations and Theories of Rational Agencies*, volume 14, chapter The Evolution of SharedPlans, pages 227–262. Kluwer Academic.

Grosz, B. J. and Kraus, S. (1996). Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357.

Gutoskie, P. (2001). Canada's road safety targets to 2010. Technical Report TP 13736 E, Transport Canada.

Hallé, S. and Chaib-draa, B. (2004). Collaborative driving system using teamwork for platoon formations. In *Proceedings of AAMAS-04 Workshop on Agents in Traffic and Transportation*, pages 35–46, New York, USA.

Hallé, S., Chaib-draa, B., and Laumonier, J. (2003). Car platoons simulated as a multiagent system. In Muller, J.-P. and Seidel, M.-M., editors, *Proceedings of Agent Based Simulation 4 (ABS4)*, pages 57–63.

Hallé, S., Laumonier, J., and Chaib-draa, B. (2004). A decentralized approach to collaborative driving coordination. In *Proceedings of the 7th IEEE International Conference on Intelligent Transportation Systems (ITSC'2004)*, Washington, D.C., USA.

Hallé, S., Gilbert, F., Laumonier, J., and Chaib-draa, B. (2003). Architectures for collaborative driving vehicles: From a review to a proposal. Rapport de recherche DIUL-RR-0303, Université Laval, Ste-Foy, Québec.

Hatipoglu, C., Ozguner, U., and Redmill, K. A. (2003). Automated lane change controller design. *IEEE Transaction on Intelligent Transportation Systems*, 4(1).

Hedrick, J., Sengupta, R., Xu, Q., Kang, Y., and Lee, C. (2003). Enhanced ahs safety through the integration of vehicle control and communication. California PATH Research Report UCB-ITS-PRR-2003-27, University of California, Berkeley.

Hedrick, J. K., Tomizuka, M., and Varaiya, P. (1994). Control issues in automated highway systems. *IEEE Control Systems Magazine*, 14(2):21–32.

Holfelder, W. (2003). Special report on intervehicle real-time communication. Berkeley Wireless Research Center Seminar. DaimlerChrysler Research and Technology North America.

Howell, A. S., Girard, A. R., Hedrick, J. K., and Varaiya, P. P. (2004). A real-time hierarchical software architecture for coordinated vehicle control. In *Automotive Software Architecture Workshop 2004*, San Diego, CA. To be published in LNCS.

Huhns, M. N. and Stephens, L. M. (1999). Multiagent systems and societies of agents. In Weiss, G., editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. The MIT Press, Cambridge, MA, USA.

Huppe, X., de Lafontaine, J., Beauregard, M., and Michaud, F. (2003). Guidance and control of a platoon of vehicles adapted to changing environment conditions. In *Proceedings of IEEE International Conference on Systems Man and Cybernetics, 2003.*, volume 4, pages 3091–3096.

Huppé, X. (2004). Guidance et commande longitudinale d'un train de voitures adaptés aux conditions routières et climatiques canadiennes. Master's thesis, Université de Sherbrooke, Sherbrooke, Canada.

Inman, V., Sanchez, R., Bernstein, L., and Porter, C. (1996). Travtek evaluation orlando test network study. Final Report FHWA-RD-95-162, Federal Highway Administration.

Ioannou, P. and Stefanovic, M. (2003). Evaluation of the acc vehicles in mixed traffic: Lane change effects and sensitivity analysis. PATH Research Report UCB-ITS-PRR-2003-03, University of Southern California.

ITS, T. C. (1999). An intelligent transportation systems plan for canada: En route to intelligent mobility. Executive Summary TP 13501 E, Transport Canada.

Jaques, A. (2003). Canada's greenhouse gas inventory. Technical report, Environment Canada, Ottawa, Canada.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial Intelligence*, 177(2):277–296.

Jennings, N. R., Corera, J., and Laresgoiti, I. (1995). Developing industrial multi-agent systems (invited paper). In *Proceedings of 1st Int. Conf. on Multi-Agent Systems (ICMAS '95)*, pages 423–430, San Francisco, USA.

Jennings, N. R., Faratin, P., Johnson, M. J., Norman, T. J., O'Brien, P., and Wiegand, M. E. (1996). Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2&3):105–130.

Kiencke, U. and Nielsen, L. (2000). *Automotive Control Systems: For Engine, Driveline and Vehicle*. Springer Verlag.

Knublauch, H. (2002). Extreme programming of multi-agent systems. In *Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, pages 704–711, Bologna, Italy. ACM Press.

Kourjanski, M., Gollu, A., and Hertschuh, F. (1998). Implementation of the smartahs using shift simulation environment. In *Proceedings of The SPIE Conference on Intelligent Systems and Advanced Manufacturing*, volume 3207, pages 3207–23.

Lange, D. B. and Oshima, M. (1999). Seven good reasons for mobile agents. *Communications of the ACM*, 42(3):88–89.

Levesque, H. J., Cohen, P. R., and Nunes, J. H. T. (1990). On acting together. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90)*, pages 94–99, Los Altos, CA.

Liang, C.-Y. and Peng, H. (2000). String stability analysis of adaptive cruise controlled vehicles. *JSME International Journal Series C*, 43(3):671–677.

Lin, W. and Leung, H. (2002). Comparison of vehicle detectors used in intelligent transportation systems. Technical report, NCE, Auto'21.

Ljungberg, M. and Lucas, A. (1992). The OASIS air-traffic management system. In *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence (PRICAI '92)*, Seoul, Korea.

Mercedes-Benz (2004). Distronic adaptive cruise control. [Online]. http://mbusa.com/brand/models/tech_demos/distronic.html, (accessed the 24th of August 2004).

Mezentsev, O., Lu, Y., Lachapelle, G., and Klukas, R. (2002). Vehicular navigation in urban canyons using a high sensitivity receiver augmented with a low cost sensor. In *Proceedings of ION GPS*, pages 363–369, Portland, OR.

Morissette, J.-F., Chaib-draa, B., and Plamondon, P. (2004). Resource allocation in time-constrained environments: The case of frigate positioning in anti-air warfare. In *Proceedings of Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO)*, Metz, France.

Morsink, P., Cseh, C., Gietelink, O., and Miglietta, M. (2002). Design of an application for communication based longitudinal control in the cartalk2000 project. In *Proceedings of IT Solutions for Safety and Security in Intelligent Transport (e-Safety)*, Lyon.

Moss, S. and Davidsson, P., editors (2001). *Multi Agent Based Simulation*, volume 1979 of *LNAI*. Springer Verlag.

Nair, R., Tambe, M., and Marsella, S. (2003). Role allocation and reallocation in multiagent teams: Towards a practical analysis. In *Proceedings of the second International Joint conference on agents and multiagent systems (AAMAS)*, pages 552–559.

Nair, R., Tambe, M., Marsella, S., and Raines, T. (2004). Automated assistants for analyzing team behaviors. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 8(1):69–111.

NEREUS (2004). [Online]. http://damas.ift.ulaval.ca/projets/TeamWork, (accessed the 30th of September 2004).

Network, W. R. S. (2004). Committed to safer roads across the world. [Online]. http://www.wrsn.net/aboutus.htm, (accessed the 3rd of August 2004).

Odell, J., Parunak, H., and Bauer, B. (2000). Extending UML for agents. In *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*.

Paquet, S., Bernier, N., and Chaib-draa, B. (2004). Comparison of different coordination strategies for the robocuprescue simulation. In *Proceedings of The 17th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, LNAI 3029, pages 987–996, Ottawa, Canada. Springer-Verlag.

PATH, C. (2004). [Online]. http://www.path.berkeley.edu/PATH/Publications/Media, (accessed the 9th of August 2004).

Proper, A. T. (1999). Its benefits: 1999 update. Technical Report FHWA-OP-99-012, Mitretek Systems Inc.; U.S. Department of Transportation.

Pynadath, D., Tambe, M., Chauvat, N., and Cavedon, L. (1999). Toward team-oriented programming. In *Proceedings of the 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL)*, volume 1757, pages 233–247.

Pynadath, D. V. and Tambe, M. (2002). The communicative multiagent team decision problem: Analyzing teamwork theories and models. *Journal of AI research*, 16:389–423.

Rajamani, R., Tan, H.-S., Law, B. K., and Zhang, W.-B. (2000). Demonstration of integrated longitudinal and lateral control for the operation of automated vehicles in platoons. *IEEE Transactions on Control Systems Technology*, 8(4):695–708.

Rajamani, R. and Zhu, C. (2002). Semi-autonomous adaptive cruise control systems. *IEEE Transactions on Vehicular Technology*, 51(5):1186–1192.

Randall, L., Allen, J., and Salido, R. A. (2000). North american transportation in figures. Technical Report BTS00-05, U.S. Department of Transportation; Statistics Canada et al., Washington, DC.

Rao, A. S. and Georgeff, M. P. (1995). Bdi agents: from theory to practice. In *Proceedings of the First International Conference on Multiagent Systems*, pages 312–319, San Francisco. The MIT Press.

Rosenbloom, P. S., Laird, J. E., Newell, A., and McCarl, R. (1991). A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: Modern Approach*. Prentice Hall Series in AI, Upper Saddle River, NJ, 2nd edition.

Sakaguchi, T., Uno, A., Kato, S., and Tsugawa, S. (2000). Cooperative driving of automated vehicles with inter-vehicle communications. In *Proceedings of the IEEE Intelligent Vehicles Symposium 2000*, pages 516–521, Dearbsorn, MI, USA.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: Off-line design. *Artificial Intelligence*, 73(1-2):231–252.

Smiley, A. and Brookhuis, K. (1987). Alcohol, drugs and traffic safety. *Road users and traffic safety*, pages 83–105. J.A. Rothengatter & R.A. de Bruin (Eds.), Assen: Van Gorcum.

Smith, I. and Cohen, P. (1996). Towards semantics for an agent communication language based on speech acts. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Smith, M. (1993). Environmental implications of the automobile. SOE Fact Sheet 93-1, Environment Canada, Ottawa, Canada.

Stanton, N. A. and Young, M. S. (1998). Vehicle automation and driving performance. *Ergonomics*, 41(7):1014–1028.

Stevens, W. (1995). Summary and assessment of findings from the precursor analysis of automated highway system. Technical Report WN95W0000124, The MITRE Corporation.

Stone, P. and Veloso, M. (1999). Task decomposition and dynamic role assignment for real-time strategic teamwork. In Muller, J. P., Singh, M. P., and Rao, A. S., editors, *Intelligent Agents V — Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555 of *Lecture Notes in Artificial Intelligence*, pages 293–308. Springer-Verlag, Heidelberg.

Stough, R. R. (2001). *Intelligent Transportation Systems: Cases and Policies*. Edward Elgar Pub. LTD.

Sukthankar, R., Hancock, J., and Thorpe, C. (1998). Tactical-level simulation for intelligent transportation systems. *Journal on Mathematical and Computer Modeling*, 27(9-11).

Swaroop, D., Hedrick, J. K., Chien, C., and Ioannou, P. (1994). A comparison of spacing andheadway control laws for automatically controlled vehicles. *Vehicle System DynamicsJournal*, 23(8):597–625.

Syed, S. and Cannon, M. (2004). Fuzzy logic based-map matching algorithm for vehicle navigation system in urban canyons. In *Proceedings of ION National Technical Meeting*, San Diego, CA.

Tambe, M. (1997). Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124.

Tambe, M. and Zhang, W. (2000). Towards flexible teamwork in persistent teams: extended report. *Journal of Autonomous Agents and Multi-agent Systems, special issue on Best of ICMAS 98*, 3:159–183.

Touran, A., Brackstone, M., and McDonald, M. (1999). A collision model for safety evaluation of autonomous intelligent cruise control. *Accident Analysis & Prevention*, 31(5):567–578.

Tsugawa, S., Kato, S., Tokuda, K., Matsui, T., and Fujii, H. (2001). A cooperative driving system with automated vehicles and inter-vehicle communications in demo 2000. In *Proceedings of the 2001 IEEE Intelligent Transportation Systems Conference*, pages 918–923.

VanderWerf, J., Kourjanskaia, N., Shladover, S., Krishnan, H., and Miller., M. (2001). Modeling the effects of driver control assistance systems on traffic. Technical Report Paper No. 01-3475, U.S. National Research Council Transportation Research Board 80th Annual Meeting, Washington D.C.

Varaiya, P. (1993). Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, 32.

Wada, M., Mao, X., Hashimoto, H., Mizutani, M., and Saito, M. (2004). ican: pursuing technology for near-future its. *IEEE Intelligent Systems*, 19(1):18–23.

Winner, H., Witte, S., Uhler, W., and Lichtenberg, B. (1996). Adaptive cruise control system aspects and development trends. SAE technical paper no. 961010, Special Publication SP-1143, Detroit, USA.

Wolfe, D. B., Judy, C. L., Haukkala, E. J., and Godfiey, D. J. (2000). Engineering the world's largest dgps network. In *Proceedings of OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 1, pages 79–87.

Woll, J. D. (1997). Radar-based adaptive cruise control for truck applications. SAE Technical Paper 973184, Eaton Vorad Technologies.

Wooldridge, M. (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, Cambridge, MA.

Wooldridge, M. (2002). *An Introduction to MultiAgent Systems*. John Wiley and Sons Ltd.

Xu, Q., Hedrick, K., Sengupta, R., and VanderWerf, J. (2002). Effects of vehicle-vehicle / roadside-vehicle communication on adaptive cruise controlled highway systems. In *Proceedings of IEEE Vehicular Technology Conference (VTC)*, pages 1249–1253, Vancouver, Canada.

Zhang, Y., Volz, R. A., Ioerger, T. R., and Yen, J. (2004). A decision-theoretic approach for designing proactive communication in multi-agent teamwork. In Haddad, H., Omicini, A., Wainwright, R. L., and Liebrock, L. M., editors, *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 64–71, Nicosia, Cyprus. ACM.