CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF TRANSPORT SCIENCES

JAN MACEK

AGENT-BASED MODELING OF ITS SYSTEMS IN VEHICLE SIMULATOR

DIPLOMA THESIS

2022

CZECH TECHNICAL UNIVERSITY IN PRAGUE
Faculty of Transportation Sciences
Dean's office
Konviktská 20, 110 00 Prague 1, Czech Republic

K616 ..... **Department of Vehicle Technology**

MASTER'S THESIS ASSIGNMENT

(PROJECT, WORK OF ART)

Student's name and surname (including degrees):

Bc. Jan Macek

Study programme (field/specialization) of the student:

master's degree – IS – Intelligent Transport Systems

Theme title (in Czech): **Modeloání ITS systémů na bázi agentů ve vozidlových simulátorech**

Theme title (in English): **Agent-based modeling of ITS system in vehicle simulator**

Guidelines for elaboration

During the elaboration of the master's thesis follow the outline below:

- Agent based modeling and multi-agent systems introduction and application examples.
- Research of possible application of MAS or ABM for simulation of ITS systems.
- Research of available modules and libraries for implementation of MAS in different programming languages.
- Propose and design the methodology and algorithms for implementation for the simulation of an ITS system into an existing IVS using ABM.
- Evaluate advantages and disadvantages of ABM in IVS for ITS simulation.

Graphical work range: Determined by the supervisor

Accompanying report length: 55 pages of text (including figures, plots and tables) at minimum

Bibliography: RAILSBACK, Steven F. aand Volker GRIMM. Agentbased and individual-based modeling: a practical introduction. Second edition. Princeton: Princeton University Press, 2019. ISBN 978-069-1190-839

Master's thesis supervisor: **Ing. Dmitry Rozhdestvenskiy, Ph.D**

Clas Rydergren, PhD

Date of master's thesis assignment: **June 30, 2021**
(date of the first assignment of this work, that has be minimum of 10 months before the deadline of the theses submission based on the standard duration of the study)

Date of master's thesis submission: **November 30, 2022**
a) date of first anticipated submission of the thesis based on the standard study duration and the recommended study time schedule
b) in case of postponing the submission of the thesis, next submission date results from the recommended time schedule

L. S.

doc. Ing. Petr Bouchner, Ph.D.
head of the Department of Vehicle Technology

prof. Ing. Ondřej Přibyl, Ph.D.
dean of the faculty

I confirm assumption of master's thesis assignment.

Bc. Jan Macek
Student's name and signature

Prague .....June 30, 2021

Abstract

TODO

Contents

1 Introduction

In a world where vehicle transport plays an inseparable role in the society, with an ever increasing demand, it is important to analyze and study driver behaviour and inherently interactions and relationships between drivers and their surroundings. This is supported by the fact that, even though substantial advancements in autonomous driving are being made, for the near-future, humans will still have to control vehicles themselves and therefore be exposed to a substantial risk of danger. Data shows that about 95 % of traffic accidents are a result of human error [1]. Each traffic accident has got a tremendous effect on socio-economic growth. A study be the European Union states that accident-related expenses (including cost of fatality) cost 1,8 % of EU GDP [2]. A rather non-cynical point of view is that each life lost is a failure in society itself and an effort should be made to diminish fatal accidents.

A method that has proven to be effective at studying driver behaviour and traffic safety is by using interactive vehicle simulators (IVS), which allow to undertake experiments in a safe, controlled and reproducible way. Because driving simulator is basically a digital twin of a real vehicle, it is naturally reasonable to make the interaction between the driver and IVS as close to reality as possible, which inherently improves data quality of the simulation and potentially also the range of IVS application. An IVS has got a broad spectrum of employment. It is not only used as a tool to research driver behaviour, but also used in development and testing of advanced driver-assistance systems, extending the simulator to a hardware-in-the-loop or a vehicle-in-the-loop system, which enables to test real hardware and simulate full road testing.

Simulating the traffic environment is a complex problem, mainly because of its highly dynamic characteristics. All vehicles need to interact with each other and act upon other drivers' actions. Because agent-based simulations have proven to model complex behaviour well, this modelling technique seems like a suitable solution for achieving a realistic traffic environment for IVS.

The goal of this thesis is to investigate multi-agent systems (MAS), their principles and evaluate usages of these systems related to ITS research, where their shared characteristics of distributed interoperability could prove to make agent-based modelling a strong tool for simulating ITS solutions. The output of the experimental/practical part of the thesis should be A MAS-based simulation framework for facilitating and streamlining the process of ITS solutions simulation.

In the first two chapters, a review of the state-of-art research about Intelligent Transport Systems and Multi-agent Systems is presented, with an emphasis on the respective field's system model point of view. In the third chapter, requirements for the proposed framework are specified, based on the preceding research. In the fourth section, a self-proposed

framework architecture is proposed, in line with MAS paradigms and adapted for ITS solutions simulation. In the fifth section, the implementation requirements are specified, mainly discussing the simulator software integration and additional development platforms and libraries that could potentially facilitate the framework design process. In the sixth section, a technical implementation of the framework is presented, showing the implementation details while also serving as the software's documentation. In section number seven, a validation of the proposed system is conducted by implementing a distributed cooperative ITS system into an IVS, using the proposed framework and assessing its performance and overall results.

2 Intelligent Transport Systems

Intelligent transport systems describe an initiative to utilize modern technology in order to optimize and increase efficiency of processes common to the domain of transportation. This usually involves enabling different actors in the transportation system to communicate with each other and share available information (much like multi-agent systems). With the availability of shared information, complex, data-driven systems can be built utilizing state-of-the-art tech concepts like machine learning, computer vision and IoT, while integrating humans, roads and automobiles, improved street protection, efficiency and stability. İt also address environmental issues via controlling traffic to prevent congestion or lessen their adverse effects.

When looked at these solution from a systemic point of view, they can be, in less or more ways, resemble multi-agent systems. This section will be focused more on the *inductive* part of research, examining various existing ITS solutions and trying to recognize their agent-based characteristics and principles, to determine possible application of MAS for simulation of ITS systems, whether suitable application for agent-based modelling of ITS is feasible. , The insights gained by reviewing current state of ITS will serve as a base for further sections, where considerations regarding model framework design and its validation will be discussed.

ITS can provide itself most useful in the following aspects of transportation: *Mobility*, *Safety*, *Environment*. To provide an example, regarding mobility, one of the most common systems is routing and navigation for road vehicles. Such systems can factor in time, distance and even emissions when providing route guidance. This leads to increased performance of road networks. Regarding safety, one can mention emergency management systems like E-Call, which provides automated post-crash assistance by contacting emergency services as soon as an incident happens. In terms of the environment, ITS allows for demand management through *electronic fee collection*, which makes flexible charging for road usage possible, based on vehicle type and emissions category [4]. Another, more general example of emissions reduction of ITS is through reducing congestions, as road vehicle emissions

Figure 1: Illustration of an ITS topology [3]

have proven to be a significant environmental factor not only in emission production per se, but also one that humans are most exposed to. In a study conducted by the Harvard School of Public Health, air pollution from traffic congestion in 83 of the USA's largest urban areas contributes to more than 2,200 premature deaths annually, costing the health care system at least $18 billion [5].

2.0.1 IVS-based research

Using virtual simulation tools to research and evaluate impacts of new and existing ITS is a great tool, as it can greatly reduce the overall cost of development, achieved by faster, and flexible development in virtual space allowing for frequent, iterative evaluation of features.

It is important to say that ITS solutions vary in terms of *driver engagement*. Because the primary use-case of interactive vehicle simulator is to research driver behaviour, the scope of the current state of ITS will be focused on ITS solutions that integrate automotive transport and have some form of driver engagement, favouring solutions that have a large degree of interaction with the driver.

As per (2), there are ITS use-cases with high degree of driver interaction (determined by trivial logic). The observed use-cases with high driver engagement were determined as such:

- Geo-info service

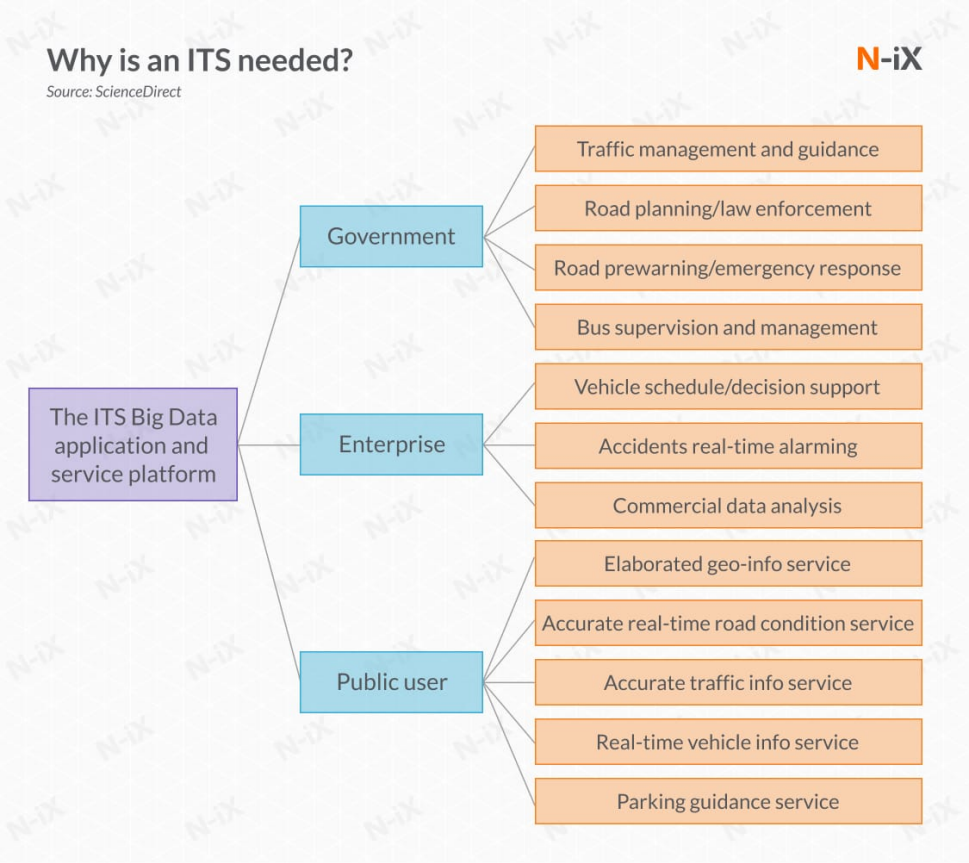

Figure 2: Reasons for ITS usage

- Real-time road condition service
- Accurate traffic-info service
- Real-time vehicle info service
- Parking guidance service

These use-cases should be considered when ITS simulation framework design decisions will be made in the practical part of the thesis.

2.0.2 MAS-compatibility

Secondly, it is important to acknowledge whether there is an ITS that could be simulated using the MAS theory, so that the information gathered in the first section of this thesis could be successfully applied.

Intelligent transport systems are, by nature, systems combining several actors together to achieve a common goal. Would this mean that every ITS can be, in fact, modelled as multi-agent system? The answer is that it does *not* apply to all of them.

Historically speaking, ITSs that were developed and proved helpful for optimizing traffic were *centralized*, meaning there is a single core in charge of logic and decision-making, organizing other units/actors within the system, while interacting with the drivers in

the traffic as external actors. The main advantage of these systems is that they are less complex, therefore it is easier to develop a resilient and safe product. The main drawback of this method is that they are heavily affected by the size of instances, increasing the complexity and often result in exceedingly large computation time, making the solution sub-optimal [6].

As with the *distributed* systems, the main challenge making individual agents solving sub-problem, cooperating well to achieve good and predictable results, as has been illustrated in section (3). Distributed systems can be used to solve traffic optimization problems on a larger scale. These systems have seen larger usage especially in the recent times, as newly produced vehicles are equipped with powerful computers that can be used to transfer the computational burden from the core computer in centralized systems. Also state-of-art communication protocols like DSRC and ITS 5G, which enable low-latency direct communication between vehicles.

The one example of an ITS where implementations have been carried out in both centralized and decentralized principles numerous times is a *Network traffic control*. The goal of this system is to use knowledge about the traffic on the network to control traffic lights and other active traffic control elements to optimize traffic flow and decrease travel time. In [7], centralized and distributed network traffic control systems were compared. The conclusion was that although the centralized system was able to achieve better performance higher global efficiency, the decentralized solution required significantly less computation time (40 % in their experiment case).

Other decentralized, MAS-based systems that are more exposed to the user (driver) include the *Adaptive Cruise Control* (ACC), for example. ACC extends the usual cruise control systems that maintain vehicle speed by adapting to speed of the vehicle in front, if there is any. In fact, recent development has lead to a further improvement, making ACC a cooperative system (CACC) that enables vehicles to adopt a *driving strategy* by communicating with the infrastructure (V2I communication).

2.0.3 Current research

Secondly, it is important to choose a system that is time-relevant and would bring benefit to current as well as future IVS and HMI research activities. Therefore, the third quality for choosing an ITS to implement would be one that is still subject to research and current trends in the ITS industry and explore state-of-the-art ITS. In order to determine which ITSs are relevant, it is important to review ongoing project of governmental bodies and their strategies and also current trends in automotive ITS research activities.

In Europe, there are initiatives to centralize decision making and ITS deployment on the scale of the European continent. The reason behind this initiative is quite clear - to enable Europe-wide interoperability between deployed ITS and ITS-enabled vehicles. It is

without a question that such organization would closely work with the industry, helping to create conditions for novel ITS technologies to be deployed.

ERTICO One such European organization is the *European Road Transport Telematics Implementation Coordination (ERTICO)* - a public-private partnership organization with close to 120 members, connecting 8 different sectors in the ITS Community, including service providers, suppliers, traffic and transport industry, research institutions and universities, public authorities, user organizations, connectivity industry as well as vehicle manufacturers [8].

As of now, ERTICO's activities focus on the following areas:

Connected Cooperative & Automated Mobility - As the computational power of newly-produced vehicles is increasing dramatically with every new generation, as well as the number of sensor data, ERTICO states their focus is on utilizing the large amounts of real-life data to deepen the machine learning models, as well as building an infrastructure that will allow handling this data. C-ITS (Cooperative Intelligent Transport Systems) is also mentioned, whose principles are being put into practice by several projects, namely European Truck Platooning (ETPC), Advanced map-enhanced driver assistance systems (ADASIS) and more. ERTICO's main contribution is to facilitate creation of ITS ecosystem by following a multidisciplinary approach involving all relevant transport stakeholder sectors.

Clean & Eco- Mobility - As has been already mentioned, smart mobility innovations make a major contribution towards reducing the impact of transport-induced pollution, which has got a non-negligible contribution to global greenhouse gas emissions production [9]. Below are the four main objectives in the are of Clean & Eco-Mobility.

- Develop a common approach to the evaluation of ITS deployment as a tool for emissions reduction
- Contribute to smart mobility solutions being recognized as a tool for reducing emissions
- Achieve interoperability of electro-mobility
- Contribute to creating an ICT network with seamless and interoperable electro-mobility services

Urban Mobility - Another focus of ERTICO is Urban Mobility, where the main goal is to provide "Mobility as a Service" (MaaS), which is a system that could decrease congestions and provide low-carbon and -emission multi-modal transport solutions.

Transport & Logistics - ERTICO states that the current European world of transport and logistics is too fragmented, so an effort to develop solution for connecting logistics information system would optimize cargo flows and facilitate supply chain management.

In conclusion, the ERTICO organization helps to reduce time to market of innovative, state-of-the-art technologies, increasing inter-operability between individual ITS by promoting an open framework for integration and deployment of intelligent transport services.

As described above, there are potential systems that are yet to be fully implemented and deployed for consumer use.

Cooperative ITS Cooperative Intelligent Transport Systems (C-ITS) refers to transport systems, where ITS subsystems (personal, vehicle, roadside and central) cooperate. This enables and provides an ITS service that offers better quality and an enhanced service level, compared to the same ITS service provided by only one of the ITS sub-systems [10]. An example of the concept of the technology can be seen in the figure (3) below.

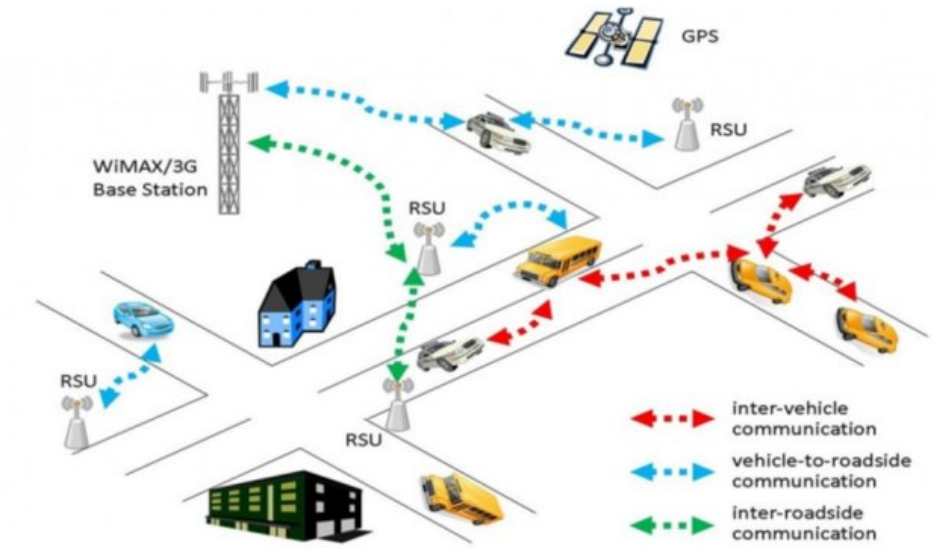

Figure 3: Example schematic of C-ITS scenario

The concept of C-ITS was developed by the European Commission, representatives of industry and authorities in the European Union. In 2016, it was agreed on a coordinated establishment of intelligent transport systems in Europe. It is considered to be one of the various tools to facilitate achieving the *vision zero*, which is a project that aims to mitigate all fatalities involving road transport.

The European Commission outlined its plan for the coordinated deployment of C-ITS in Europe in its communication 'A European strategy on Cooperative Intelligent Transport Systems', in which it also states that the full-scale deployment of C-ITS services and C-ITS enabled vehicles is expected to start in 2019.

The main feature of C-ITS is its distributed intelligence across vehicles and the infrastructure, which is a novel concept in the ITS world. Consequently, it is easy to see similarities to how MAS are described. Regarding the topic of this thesis, this could pose as an argument to implement one of C-ITS systems in the practical part of this thesis.

Vehicles and infrastructure equipped with C-ITS can, for example, communicate a warning to each other, after which the drivers are informed about the upcoming traffic situation in time for them to take the necessary actions in order to avoid potential harm. Other potential benefits of the use of C-ITS include reduced congestion and improved driver comfort. In short, vehicles share data directly between each other (V2V) and with the infrastructure (V2I) using ad-hoc short range telecommunication. The two types of communications are sometimes together referred to as Vehicle-to-everything (V2X) communication.

This technology aims to benefit both manually-driven vehicles as well as autonomous self-driving vehicles. The main use-cases, which were developed as standalone services using C-ITS technology can be seen in the figure (4) below.

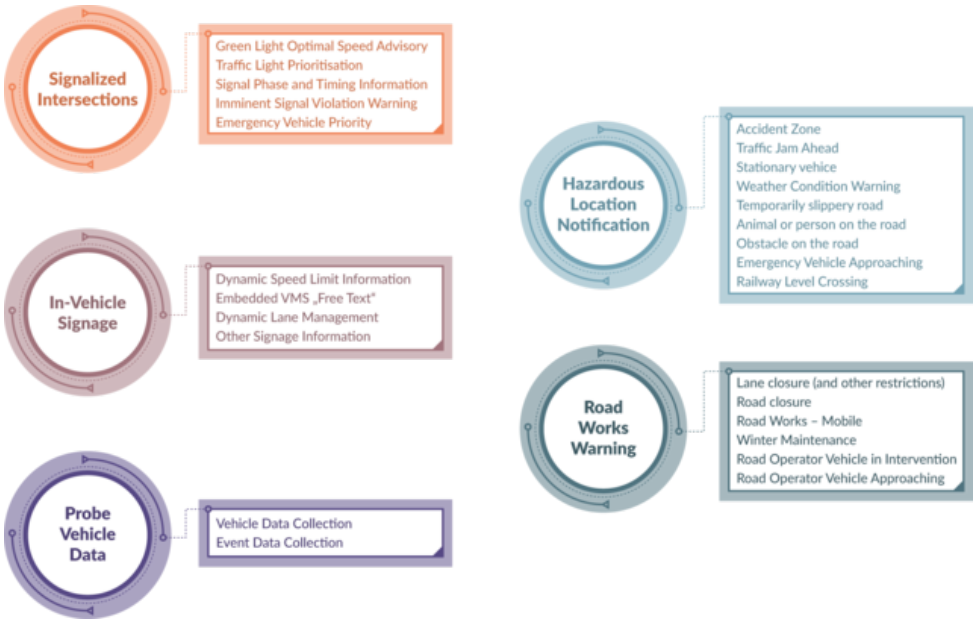

Figure 4: C-ITS use cases [10]

In general, the traffic safety and traffic flow improvements can be grouped based on which operational tasks they serve [11]:

- Provide *information* to road users to improve road safety and comfort.
- Display *regulatory boundaries* to inform road users of specific obligations, restrictions or prohibitions
- Provide *warnings* to road users about incidents ahead in their exact nature.

To provide more insight to the design and applications of C-ITS systems, two systems are introduced - In-vehicle Information System and Cooperative Adaptive Cruise Control. These systems were selected because their operation is conditional to user engagement and are considered state-of-the-art application, utilizing the C-ITS technology specification. For those reasons, they are also relevant for the IVS-based research.

2.1 In-Vehicle Information System

The In-Vehicle Information System, shortly IVIS, is a term that comprises a vast number of vehicle technology solutions that assist the driver either by providing information about the vehicle and the surrounding environment or serving as an interface to control vehicle systems.

The integral part is the *Infotainment system* (fig. 5), which is a video/audio interface, providing control through elements such as touch screen button panel or voice commands. Moreover, it integrates other vehicle technology elements, such as the CAN interface, connectivity modules (e.g. Wi-Fi, GPS), sensors etc [12]. A screen panel is an excellent medium to provide additional safety information to the driver, especially when the gauge clusters have been replaced by an additional screen which improves the HMI aspect by reducing the disruptive effect of checking the screen and making the displayed information more noticeable.

The conventional, more basic capabilities of IVIS are HVAC control, multimedia controls, navigation support and parking assistance (i.e. parking camera view). These systems on their own, however, aren't valuable with regard to the thesis topic. The important feature of IVIS is the integration with the emerging C-ITS technologies, which greatly extend the capabilities of IVIS, i.e. displaying warning and awareness messages from the V2X interface. This fact makes IVIS a good candidate to implement to the IVS in the practical part, because the V2X C-ITS are a great subject for research as of now and the distributed nature of the systems corresponds to the agent-based requirement of the thesis topic. On top of that, the important HMI aspect of IVIS could provide great value for future utilization in research using IVS.

The general, a high-level solution for IVIS simulation is to provide C-ITS awareness and warning information system by implementing the C-ITS messaging service standard, which provides interface for broadcasting such information. simulating message-based communication between road users & infrastructure and provide interface to display the information on the infotainment screen.

2.2 Cooperative Adaptive Cruise Control

The Cooperative Adaptive Cruise Control (CACC) is an extension to an already well proven ITS - Adaptive Cruise Control. As has been described above, this system extends the base (adaptive) cruise control system by utilizing the V2V information broadcasted by other road users. The system has proved to reduce the number of shock-waves by reducing oscillations that would otherwise happen without speed information sharing and increasing capacity of the traffic network, albeit only with higher penetration rates ($\geq 40\%$) [13]. Though, regarding the introduced thesis topic requirements, the system in itself doesn't provide any extra engagement from the driver side.

Figure 5: Illustrative example of IVIS interface [12]

However, another system that built upon the concept of CACC by utilizing information from the infrastructure, namely the traffic signals, has been also recently developed this system can be extended by information fs about signal phasing and consequently adopting a speed that would eliminate the need to stop before a red light, avoiding idle time. The system is called *Green Light Optimization Speed Advisory* (GLOSA). According to [14], the results of simulating deployment of this system demonstrated that even using a simple control algorithm with the aim to avoid the stop&go a reduction of fuel consumption and emissions in the region of 5 to 12 % has been observed. The study in [15] even concluded that with enough penetration, the idle (stop) time could be decreased by more than 70% (see fig. 6 below).

Because from the ongoing reserach and possibilities of such system, the proposed simulation architecture should be capable of CACC implementation - facilitating building a GLOSA traffic controller simulation and respective interface for transmitting such information to vehicles, which will react to a received message conveying GLOSA timing information accordingly.

Project C-Roads In order to study the effects and refine the future deployment process of C-ITS, a project organized by the EU member states and road operators, called *C-Roads* was established. The project's main objective is to harmonize C-ITS deployment activities across Europe. Within the C-Roads project, there have been established 5 work groups (WG), each focusing on a specific topic/scope regarding C-ITS objectives and priorities for research, testing and pre- deployment of C-ITS [16].

- WG1: Develop an EU agenda for testing

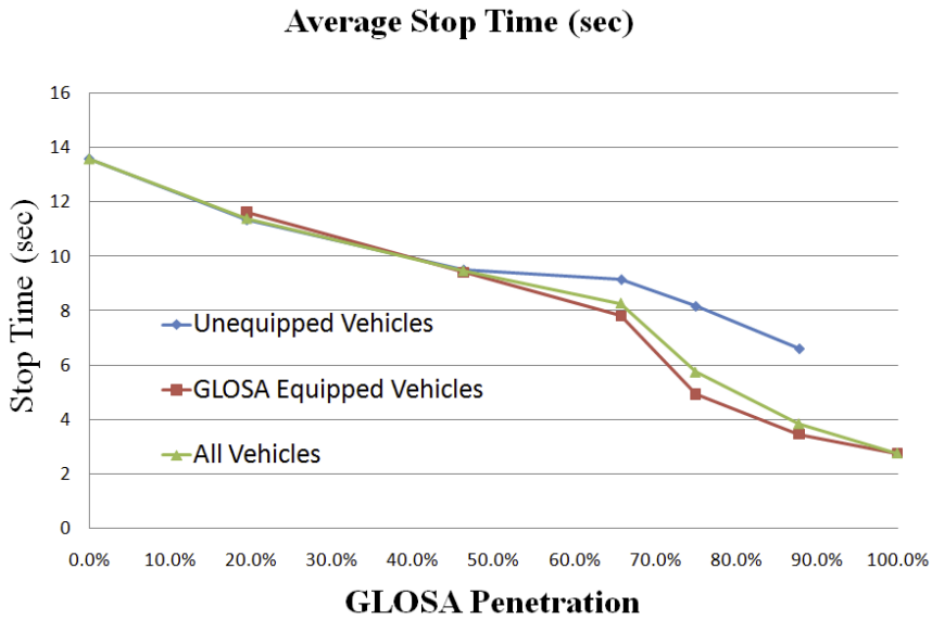

Figure 6: Idle time reduction based on GLOSA penetration rate [15]

- WG2: Coordination and cooperation of R&I activities
- WG3: Physical and digital road infrastructure
- WG4: Road Safety
- WG5: Access and exchange of data & cyber-security
- WG6: Connectivity and digital infrastructure

Regarding the scope of this thesis, *WG3* is the most informative segment. The focus of this group was, in the first place, regarding the infrastructure support for automated vehicles (SAE L4). One of the objectives was linking relevant physical and digital infrastructure, assessing relevance between them and automated vehicles, and mapping infrastructure elements to use-cases as *generic driving tasks* (GDT):

- Sensing & Perception
  - Ego localization
  - Environmental awareness (object classification and incident detection)
  - Enhanced perception (for limited visibility scenarios)
- Planning
  - (dynamic) information and regulations
  - Safe and appropriate navigation plans
  - Cooperative planning
- Actuation

- Motion Control
- Minimum Risk Manoeuvre

It is therefore advisable to keep these General Driving Tasks in mind when thinking about the proposed framework features. To maximize the number of potential use-cases of the designed MAS simulation framework, it shall offer basic components to simulate all aforementioned GDTs, as because previous R&I activities have shown that the tasks above will be a fundamental part of the future C-ITS deployment.

To put these GDT use-cases into a better perspective, a mapping was created (table (1))that links the aforementioned GDTs to existing, relevant ITS solutions. This should clear up and simplify the process of defining requirements to the proposed system, as the new framework requirements could be easily adopted from existing system documentation of the particular mapped ITS.

Table 1: GDT to ITS mapping

GDT		ITS Mapping	
Group	Name	Type	Description
Sensing & Perception	Ego-localization	HD Maps	Geo-fencing
		AD	Self-driving algorithms
	Environmental awareness	IVIS	Intersection Collision war.
			Emergency Vehicle war.
			Stationary Vehicle war.
			Traffic Jam war.
			Traffic accident war.
	Enhanced perception	IVIS	Overtaking war.
			Intersection Collision war.
			VRU warning
Planning	Information and regulations	CACC	Green Light Optimal Speed Advisory
	Safe navigation plans	Navigation	Dynamic Vehicle Routing
	Cooperative planning	AD	Platooning
Actuation	Motion Control	AD	Self-driving algorithms
	Minimum Risk Manoeuvre	AD	Self-driving algorithms

Legend

Abbreviation	Meaning
HD	High-definition
AD	Autonomous driving
IVIS	In-Vehicle Information Systems
VRU	Vurneable Road User
CACC	Cooperative Adaptive Cruise Control

2.3 Conclusion

In this section, the Intelligent Transport Systems have been introduced. This field is an important part of traffic engineering, utilizing traffic data and mathematical modelling to reduce congestion, improve traffic safety and reduce emissions. The main point of this section was to discover important features of intelligent transport systems that are related to simulation and integration into vehicle simulators. find a suitable ITS to implement as a means for the MAS architecture validation proposed in this thesis. Three indicators for quantitative analysis have been determined (Driver engagement, MAS-compatibility, research relevance) and various ITS projects have been investigated, including the R&I efforts on the EU level. The results suggested that C-ITS systems are a current trend in research and numerous projects (e.g. C-Roads) have been established, paving the way for the future of interconnected vehicle mobility. Afterwards, a methodology for qualitative analysis was introduced and used to determine the best ITS candidates for implementation into an IVS. The resulting best fitted systems were both from the C-ITS field, as its features are much alike those of multi-agent systems. Finally, the chosen systems to implement were the *IVIS based awareness and warning information system* and the *Green Light Optimal Speed Advisory* system. The following practical part will be dedicated to the implementation methodology, introduction to the development system and the development itself.

3 Multi-agent systems

Multi-agent systems (MAS) is a broad paradigm and/or research topic. It is a subfield of Distributed problem solving. A multi-agent system can be, in a short way, described as a group of autonomous agents that act towards their objectives in an environment to achieve a common goal [17]. The agents can be defined as independent units in an environment, forming a system. They are able to act independently, possess knowledge and communicate with each other (i.e. share the knowledge). The agents should be working towards some form of a common goal, which could be achieved either by cooperating or competing. The agents usually have a perception through which they can gain knowledge from their environment.

The basic definition of MAS suggests that they should be used to model/represent a system that is not centralized but rather distributed, where autonomous, intelligent units perform actions independently. Multi-agent systems have gained popularity in the recent years, as they offer high flexibility of modelling highly non-linear systems and offering abstraction levels that make it more natural to deal with scale and complexity in these systems [18]. It is apparent that MAS excel in modeling social environments involving humans. MAS share a lot of features with human societies, as these societies also revolve around atomic units - persons, that act independently - each person makes decisions and acts upon their own beliefs. People also interact with each other, be it communication, cooperation or competition, while working towards some goal. There are numerous real-life examples that strongly resemble this MAS definition, for example sport teams, where each player has his own role, like a defender and striker. Although all players have the same intention (i.e. win the game), their specific action differ depending on the state of their environment, each of them acting upon their own beliefs, which makes the team resemble a distributed system. From more practical perspective, MAS paradigm can be used when building complex computer networks, for example.

3.1 Agent

Agents are the fundamental building blocks of a multi-agent system. While they can have many features and characteristics specific for their use case and are not possible to generalize, there are several elementary characteristics that define them in scope of MAS [17].

Situatedness - Agents are designed so that they interact with the environment through sensors, resulting in actions using actuators. The agent should be able to directly interact with its environment using actuators.

Autonomy - Agent is able to choose its actions without other agents' interference on the network.

Inferential capability - Agent is able to work on an abstract goal specifications, identifying and utilizing relevant information it gets from observations.

Responsiveness - Agent is able to respond to a perceived condition of environment in a timely fashion.

Social behaviour - Agent must be able to interact with external sources when the need arises, e.g. cooperating and sharing knowledge.

3.2 Agent type and architecture

As has been already stated, in order to model complex applications, the agent characteristics as well as their internal control architecture will differ between use cases. In an effort to standardize MAS development, several architectures that describe how agents work have been proposed. Generally, there are three classes of agent modelling architectures that are defined based on interaction complexity with external sources:

- Reactive agents
- Deliberative agents
- Interacting agents

An overview of the class characteristics and examples of agent architectures utilizing the respective classes will be given.

3.2.1 Reactive agent architectures

Having first emerged in behaviorist psychology, the concept of reactive agents is founded by agents that make their decisions based on limited information, with simple situation-action rules. The agents usually make decisions directly based on the input from their sensors. This type of agents is mostly suited for application where agent resilience and robustness is the most important factor, instead of optimal behaviour. An example of a reactive agent architecture is the Subsumption architecture.

Subsumption architecture

This architecture described by Rodney Brooks in 1986 [19] and decomposes the agent into hierarchical levels that operate in a bottom-up fashion, meaning that bottom layers that control elementary behaviour are activated by the upper layers that define more complex action or define goals for the agent. It is important to note that the behavioral modules map sensations directly to actions and can only define what the agent does, not being able to change its desires. An example of a subsumption architecture is depicted in fig. (7) with example modules from each hierarchical level, where the bottom-level module is `Avoid Objects`, which is a needed action in order to complete the `Wander Around` action, which can be induced by the general goal on the top layer `Explore World`.

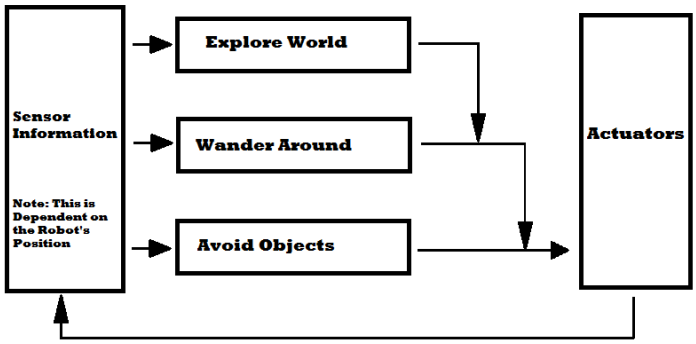

Figure 7: Example of a Subsumption architecture [20]

3.2.2 Deliberative agent architectures

Compared to a reactive agent, deliberative agents have a more complex structure and are closer to a human-like, rational behaviour. A deliberative agent is defined as one that possesses an explicitly represented, symbolic model of the world, and in which decisions are made via symbolic reasoning [21]. In other words, the agent maintains its internal representation of the external environment and thus is capable to plan its actions, while being in an explicit mental state which can dynamically change. The Beliefs, Desires and Intentions (BDI) architecture is the most widely known modelling approach of deliberative agents.

BDI architecture

The BDI paradigm has been used in various applications, such as simulating impacts of climate change on agricultural land use and production [22] or to improve internet network resilience by creating BDI agents that combats DDoS attacks [23]. The main idea behind this architecture is the emphasis on practical reasoning - the process of figuring out what to do. There are three logic components that characterize an agent:

Beliefs - The internal knowledge about the surrounding environment, which is being constantly updated by agent's perception.

Desires - What the agent wants to accomplish. An agent can have multiple desires, which can be hierarchically structured or have different priority.

Intentions - Intentions are formed when an agent commits to a plan in order to achieve a chosen goal. The plans are pre-defined within an agent, formally called a *plan library*. The plan that an agent has set to carry out can dynamically change based on updated beliefs or desires.

These components together define an agent's *reasoning engine* (fig. (8)), which drives the agent's (deliberative) behaviour.

This definition can be made clearer with a simple example scenario - a waiter in a restaurant.

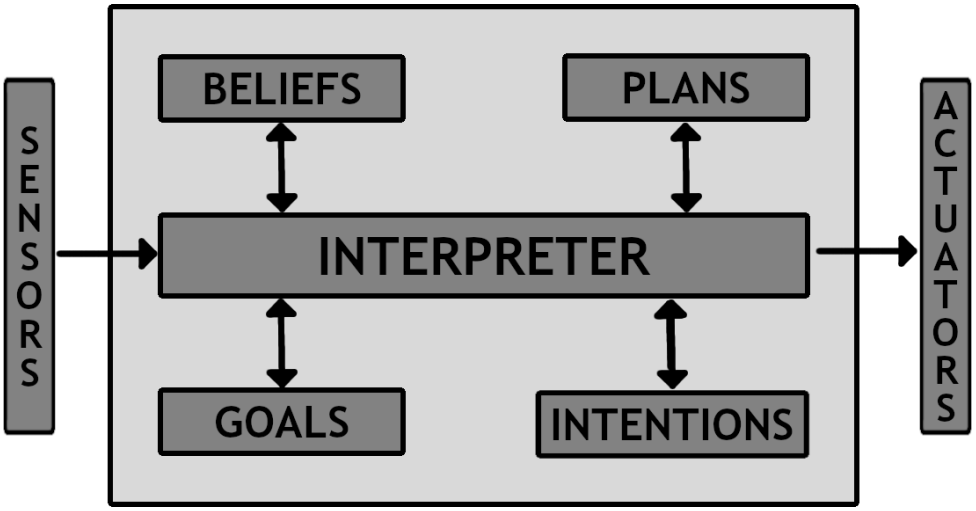

Figure 8: The BDI architecture schematic

The waiter's *beliefs* are the tables with customers and information about state of each table (i.e. choosing menu, waiting for meal, willing to pay etc.). The waiter's *desires* are to serve customers, for example accept order from a customer. The waiter carries out his desires by making a plan of *intentions* (e.g. go to table and ask if the customer wants a drink).

This architecture has its advantage in the fact that the functional decomposition of the system is clear and intuitive. However, with this architecture, there is a commitment-reconsideration tradeoff that needs to be optimized [24]. With too much commitment, there is a risk of agent overcommitment, where an agent might be trying to achieve a goal that is not longer valid. On the other hand, if an agent reconsiders too often, there is a risk that the agent will not achieve any goal because it will switch between intentions too quickly.

3.2.3 Hybrid approaches

Hybrid architectures try to utilize the best of both worlds of agent modelling. Purely reactive agents might lack the ability to solve complex tasks, whereas deliberative architectures are challenging to successfully implement on a concrete problem [25]. Pre-compiling a plan library for every possible scenario that can happen in environment with vast amount of complexities is simply not feasible, due to uncertainties of following effects when agents affect the environment.

The underlying concept of hybrid architectures is to structure agent functionalities into layers that interact with each other. This provides several advantages, namely *modularization*, which decomposes the agent's functionalities into distinct modules that have determined interfaces. This helps to deal with design complexity. Furthermore, having distinct layers enables them to run in parallel, thus increasing an agent's computational

ability as well as reactivity [26].

Generally, amongst the most widely used hybrid architectures, a *controller* layer can be found, which handles reactive tasks therefore is also connected to the sensor readings, and is hierarchically on the lowest level. Then, there are *planning* layers that handle the logic-based, deliberative tasks and often interact with the controller layer. In-between them, there is usually a *sequencing* layer that can suppress output from the reactive layer. An example of a well-known hybrid architecture is the $_3$T architecture, whose abstract model can be seen on the figure (9) below.

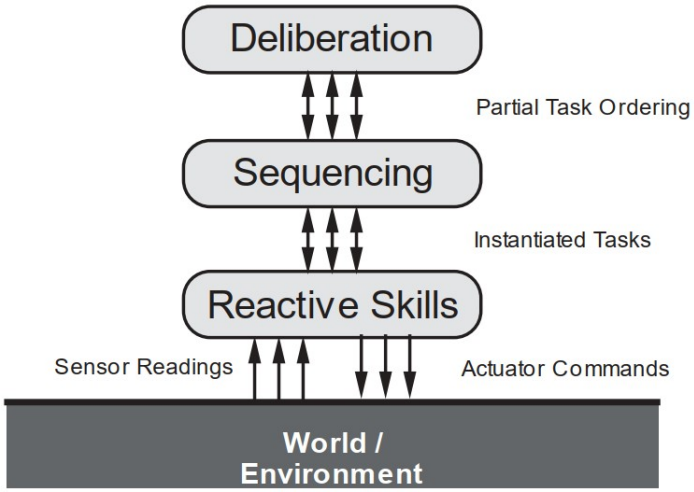

Figure 9: An architecture model of a $_3$T hybrid architecture [27]

$_3$T architecture

This architecture builds upon a predecessor architecture called Reactive Action Packages (RAPs) [28]. A RAP is essentially a process or a description of how to complete a task using discrete steps. Note that it has got no planning abilities, i.e. its actions are only based on current perceived environment state and not on an anticipated state. When a RAP is executed, it should finish only when it satisfied its outcome or else it will produce a failure state. This ensures that the agent can self-diagnose a failure and implement some fail-safe mechanisms. Individual RAPs are queued by the interpreter, in the case of the $_3$T architecture it is called a *sequencer*, which is the intermediate layer between the *reactive skills* and *deliberation* layer.

The skills layer is a collection of so-called *skills*. Skills provide an interface of an agent to its environment. They are its abilities that allow the agent to transform or maintain a particular state in the environment. Each skill has got an expected input and output, which allows to route them together.

Finally, the deliberation layer is responsible for planning on a high level of abstraction, in order to make its problem space small. Routine sequences of tasks should not be specified or dealt with at this layer.

3.2.4 Conclusion

The different architecture types that were presented can each have a different application where they excel. It is also important to note that a line between reactive and deliberative agents is not necessarily strict, hence the existence of hybrid architecture types.

Choosing the optimal agent architecture will depend on the scope of agent goals, environment and action space definition. With respect to the topic of this thesis, which is to create a framework for implementing various ITS, there isn't a clear-cut choice regarding architecture selection. Although the vast majority of ITS share the same goals (see section 2), the operating environment, underlying concepts and also used technology vary substantially.

To put things into a perspective, let's consider ITS introduced in section 2. Systems such as autonomous driving algorithms require high resilience, determinism, and high performance, all in extremely dynamic conditions. This would suggest to use a *reactive* architecture. On the other hand, there are systems such as traffic network management, where highly complex problems with lots of variables are considered. Such non-linear behaviour requires complex logic and frequent re-planning, therefore more suited for *deliberative* architectures. Furthermore, when considering the aforementioned cooperative ITS, where the emphasis on information sharing and environment awareness is given, it becomes clear that *hybrid* architectures which offer both planning and reactive behaviour would be the optimal choice.

As such, when creating the framework for ITS implementation later in this thesis, the $_3$T architecture with RAP utilization will be used.

3.3 Interaction between agents

It is safe to say that an agent-based ITS system will require some form of agent interaction and thus communication between individual agents. Interacting agents are able to interact with other agents within the environment. This concept extends an agent with an interface dedicated to communication, which makes them able to directly communicate and thus cooperate in a decentralized fashion. The concept of cooperation between agents is important in the ITS modelling context, as these systems facilitate sharing information in-between drivers and also from the road traffic environment to drivers. Therefore, interaction is a strong component when considering modelling ITS and road traffic in general. However, this interface also adds to the system's complexity.

It is expected that most agents would achieve their common goal through cooperation, where some form of forced altruism is given to the agents. With that being said there could also be situations where conflicts of interest between agents occur with no clear positive outcome, i.e. zero-sum games.

One should adhere to some communication principles to facilitate cooperation between agents. As such, agents should communicate according to Gricean maxims (see the table below) [29]. This way, the communication and performance overhead is minimized and system is more stable.

Table 2: Gricean maxims

Quantity	Say not more nor less than it is required
Relevance	Stay relevant to the topic of discussion
Manner	Avoid obscurity and ambiguity
Quality	Do not give false or unsupported information

It needs to be said that the agents developed for an ITS system will most likely not be competing amongst each other, as the agent reward should be higher the more the agents cooperate. For example, when designing cooperative intersections system, the reward function for the individual intersections (i.e. agents) should take into account not only the throughput on their own intersection but also throughput of the whole system. However, that doesn't eliminate the need to negotiate. Consider a situation where there are multiple traffic light intersections, each deciding on which phases to enable based on the incoming traffic intensity. An optimal option that minimizes cost (e.g. travel time) for one agent could have a significant negative effect on other intersections. Therefore it is important to achieve an equilibrium that minimizes the overall cost.

3.3.1 Organizational decomposition

Before defining the communication protocol, which is arguably the most important aspect of agent interaction, it is important to think about the the overall system topology and inter-agent relationships. There are two possibilities when designing agent structure.

Hierarchical organization

In hierarchical organization [30], agents are organized in a tree structure, where each level has got a different level of autonomy. The flow of control is from top to bottom, i.e. agents in lower level of hierarchy conform to decisions from higher levels.

A simple form of hierarchical organization ensures that there is a low number of conflicts and the system can be operating by relatively simpler sequential processes where the control flow is straightforward, however, this also decreases the robustness of the system, because the control and autonomy not as distributed, e.g. when a failure of a single agent with at a high hierarchical level causes the whole system to fail. A subtype of this organization - a uniform hierarchy, the authority is more distributed among the agents.

This makes the system more fault tolerant and perform graceful degradation in scenarios where one ore more parts of the system fail [17]. Here, a special attention needs to be given to conflict resolution, which is not always as straightforward, as mentioned earlier.

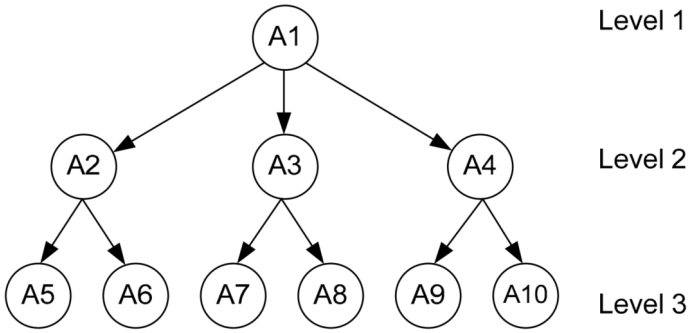

Figure 10: Hierarchical agent structure schematic [17]

Coalitions

Another useful agent organization is to organize agents into coalitions. In coalitions, a group of agents come together for a short time to increase the utility or performance of the individual agents in a group [17]. After the goal is reached or the coalition no longer becomes feasible, the coalition ceases to exist. The coalition can have a flat hierarchy, with the possibility to have one agent as the leader role for interactions outside the coalition. The use of coalitions allows for a highly dynamic system, which on the other hand increases the system's complexity.

There are more concepts for structuring a multi-agent system from organizational point of view, such as *teams* and *holons*. However, the two mentioned approaches are the most well-suited for an ITS system development, as they have been used in more works for this purpose already (e.g. [31] and [32]).

In conclusion, the usage of the aforementioned organization should ensure that the system complexity is kept as low as possible and the topology is transparent and uncluttered. The hierarchical organization can be beneficial when applied to ITS systems such as urban traffic management, where conflicts of interests need to be resolved by a capable authority. Whereas, the coalition-based organization could be applied to create virtual clusters of connected vehicles (platooning) to apply C-ITS features, e.g. dynamic navigation or GLOSA.

3.3.2 Communication between agents

As has been stated before, agent communication is a crucial component of a multi-agent system. Communication makes for inter-agent interaction beyond cues that an agent receives from the environment through its sensors, as agents can either exchange or broadcast information which could not be obtainable for the agent otherwise. This allows for deeper and more complex decision making and behaviour design - agents can act upon

the received information or update their beliefs about the world and provide distributed problem solving in general.

3.3.3 Agent Communication Language

Any language, including the one used by agents in an arbitrary system, should have defined its syntax and semantics to be an effective medium. Effectively, this means that a dedicated communication interface for each agent should be defined, with a standardized way of expressing information. To satisfy these requirements, a language developed specifically for MAS has been created, called *Agent Communication Language* (ACL) [33]. This specification proposes a standard for agent communication. Most importantly, it defines so-called *communicative act* (CA) - a special class of actions that correspond to the basic building blocks of dialogue between agents. A communicative act has a well-defined, declarative meaning independent of the content of any given act. CAs are modelled on speech act theory. Pragmatically, CA's are performed by an agent sending a message to another agent, using a specified message format. For the index of the defined CAs and their classification, see the table (3).

Using all of the defined message types is not mandatory in order to use the ACL. However, the standard introduces ACL-compliant agent requirements that need to be fulfilled. The agent requirements are in the table (4).

Apart from the mentioned communicative acts, the ACL standard also defines message parameters, which form the structure of a message (table (5)). A sample composed message structure can be found in figure (11).

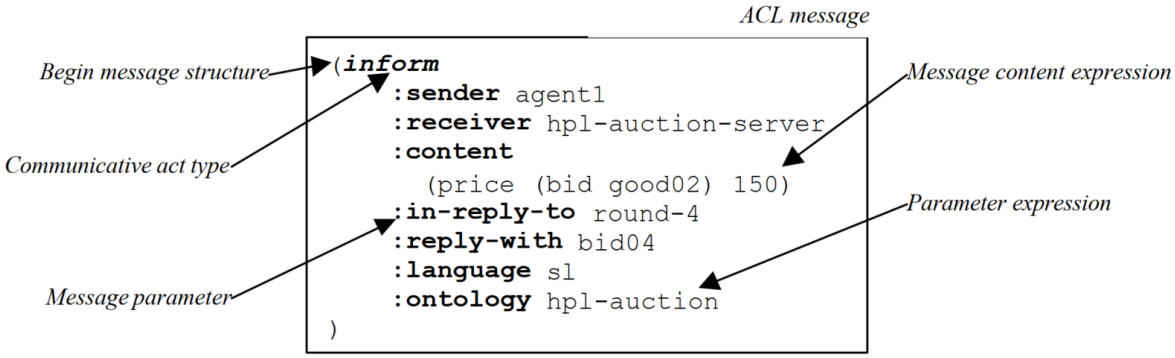

Figure 11: Main structural components of ACL message [33]

It needs to be considered that there are plenty of other standard definitions that are suited for distributed system interoperability, such as W3C, CORBA, UML and many others, so the choice of the ACL needs to be argued. A lot of these applications are based on the so called CRUD set of communication primitives - Create, Read, Update and Delete. In contrast to this, ACL defines a lot of different communication primitives (i.e. communicative acts). This leads to more complex control [34]. Also, [34] states that The

ACL model naturally allows more semantic context to be included in messages. This can give applications more understandable information about unexpected events. In addition, the richer set of primitives can lead to more flexible interaction processes.

3.3.4 Conclusion

In this section, the ways how agents interact with each other were described, with respect to the application scope of this thesis, i.e. applying MAS principles to implement an ITS simulation. The most important facts are that agent agent organization needs to be considered - despite the fact that MAS are designed to be highly distributed, giving the agents hierarchical roles can often facilitate interaction between agents and can decrease system complexity while preserving functionality, especially during negotiation between agents. Agent grouping should, while also contributing to decreased system distribution, decrease computation and communication overhead.

Next, it was found that designing how agents communicate is a crucial part of MAS development that can get complex, so it is important to keep the ways of sharing information as organized as possible, which can be done by defining the language, primarily its semantics and associated syntax. The cornerstones of inter-agent communication were discussed, and a suitable framework for the thesis' use-case of setting up communication between agents was investigated and chosen. The framework of choice was the Agent Communication Language, because its set of communication primitives already assume the MAS-based use-cases, with predefined requirements, message parameters and message types while also being open for extension.

Moving forward, the following section will be dedicated to proposing the actual system that will be implemented, subject to the topic of the thesis.

Table 3: Categories of communicative acts [33]

Communicative act	**Information passing**	**Requesting information**	**Negotiation**	**Action** performing	**Error** handling
accept-proposal			■		
agree				■	
cancel				■	
cfp			■		
confirm	■				
disconfirm	■				
failure					■
inform	■				
inform-if (macro act)	■				
inform-ref (macro act)	■				
not-understood					■
propose			■		
query-if		■			
query-ref		■			
refuse				■	
reject-proposal			■		
request				■	
request-when				■	
request-whenever				■	
subscribe		■			

Table 4: The ACL-compliant agent requirements [33]

Requirement 1: Agents should send not-understood if they receive a message that they do not recognise or they are unable to process the content of the message. Agents must be prepared to receive and properly handle a not-understood
Requirement 2: An ACL compliant agent may choose to implement any subset (including all, though this is unlikely) of the predefined message types and protocols. The implementation of these messages must be correct with respect to the referenced act's semantic definition.
Requirement 3: An ACL compliant agent which uses the communicative acts whose names are defined in the specification must implement them correctly with respect to their definition.
Requirement 4: Agents may use communicative acts with other names, not defined in the specification document, and are responsible for ensuring that the receiving agent will understand the meaning of the act. However, agents should not define new acts with a meaning that matches a pre-defined standard act.
Requirement 5: An ACL compliant agent must be able to correctly generate a syntactically well formed message in the transport form that corresponds to the message it wishes to send. Symmetrically, it must be able to translate a character sequence that is well-formed in the transport syntax to the corresponding message.

Table 5: Pre-defined message parameters [33]

Message Parameter	Meaning
`:sender`	Denotes the identity of the sender of the message, i.e. the name of the agent of the communicative act.
`:receiver`	Denotes the identity of the intended recipient of the message.
`:content`	Denotes the content of the message; equivalently denotes the object of the action.
`:reply-with`	Introduces an expression which will be used by the agent responding to this message to identify the original message. Can be used to follow a conversation thread in a situation where multiple dialogues occur simultaneously.
`:in-reply-to`	Denotes an expression that references an earlier action to which this message is a reply.
`:envelope`	Denotes an expression that provides useful information about the message as seen by the message transport service.
`:language`	Denotes the encoding scheme of the content of the action.
`:ontology`	Denotes the ontology which is used to give a meaning to the symbols in the content expression.
`:reply-by`	Denotes a time and/or date expression which indicates a guideline on the latest time by which the sending agent would like a reply.
`:protocol`	Introduces an identifier which denotes the protocol which the sending agent is employing.
`:conversation-id`	Introduces an expression which is used to identify an ongoing sequence of communicative acts which together form a conversation.

4 Requirements definition

This section will be mainly devoted to defining what should the proposed system be capable of in terms of functionality. In the previous section, general guidelines on which ITS solution the proposed framework should aim to be able to simulate were discussed. This section will expand on it, going into more details and processes how to achieve a system that is both modular enough to support a wide range of ITS solution simulation and, at the same time, offering enough facilitation value for streamlining development of ITS simulations.

Another outcome of this section will be to determine which toolset to use to facilitate building of the architecture. Ideally, an existing library for agent-based modelling could be used, which has got the elementary structure defined according to our proposed specifications.

Furthermore, regarding the topic of this thesis, the inductive approach of analyzing actors and processes of Various ITS solutions will then be used to validate the proposed framework. The framework will be designed using knowledge (e.g. optimal agent architecture) researched in the past section (3). Consequentially, an optimal ITS solution will be chosen to implement using the proposed framework, which will serve both as a proof-of-concept and as a benchmark for evaluation of the framework.

4.1 System requirements

As has been stated in the introduction, this section will be dedicated to propose logical requirements for the system architecture and model. Defining such requirements will ensure that functionality identified in both the inductive system decomposition in section (2), i.e. the ITS solutions review, and the deductive approach in section (3), which outlines the basis how multi-agent systems are modeled, will be met. This should result in a ITS virtual-deployment framework that is highly modular and capable of simulating large number of ITS. The requirements are summarized in the table (6) below.

5 Proposed system

This section is dedicated to designing the system that will be used to implement ITS in the vehicle simulator software. A framework dedicated for generic MAS-based ITS system implementation will be designed, utilizing the principles and paradigms gathered in the preceding sections (2, 3). Firstly, the *micro-architecture* will be defined, i.e. the specification of the system's actors (agents) - their inner structure will be defined, as well as interfaces to the environment, and high-level technical specification will be proposed - responsibilities of inner components and their inter-relationships. This will form the elementary foundation that will be used to build an actual system. The subsequent

Table 6: Proposed system requirements

Requirement 1: The framework should be a multi-agent based system, supporting more actors that are able to act independently, having their own logic and able to make decisions based on their own perception of the environment.
Requirement 2: The framework will be based on the $_3$T architecture. This primarily means implementation of the three layers of logic that interact between each other and thus offer complex behaviour modelling - Deliberation, sequencing and skill layers.
Requirement 4: The implementation of a particular ITS-agent's elementary capability should be done using skill modules, each of them serving one purpose. The skill modules must provide a pre-defined input and output interface.
textbfRequirement 1: An agent should be able to communicate with others through a dedicated communication module/skill that will manage the communication.
Requirement 4: Agents should be able to act upon received information from their sensory and communication interfaces and dynamically adjust their plans using the deliberation and sequencing layer.
Requirement 7: Sensory and communication capability of an agent will be incorporated as a separate skill module.
Requirement 5: Where applicable, agents should be able to detect conflicting intentions with other agents while executing their tasks. The conflict detection will be implemented as a module on the reactive layer.
Requirement 4: Agents should resolve conflicts using the ACL communication specification, and comply with its requirements, mentioned in section 3.3.3. An interface to support basic negotiations utilizing abstraction ACL messages should be provided.
Requirement 3: The framework's architecture should be modular enough to support broad spectrum of ITS implementation, including but not exclusive to C-ITS solutions.
Requirement 3: The framework should support C-ITS messaging services (Decentralized Environmental Notification Basic Service (DENM) & Cooperative Awareness Basic Service (CAM)) out-of-the-box and according to their specifications.
Requirement 2: The supported communication modes will be both direct (messaging specific agents) and indirect (broadcast & subscription).

macro-architecture proposal will mainly encompass modes of inter- agent communication. This outline will form a system specification that will be used to build the agent-based ITS framework for IVS software.

5.1 Agent/micro architecture overview

As per the previous section(s), where the individual MAS architectures have been reviewed (section 3), it was decided to utilize the hybrid *$_3T$ architecture*[1] (section 3.2.3), which will offer sufficient flexibility. Such modeling flexibility is needed primarily because there won't be a single, concrete system to model, but rather a generic system that will facilitate arbitrary agent-based ITS implementation. As such, it makes sense to choose a hybrid architecture, which will ensure there will be optimal balance between robust, reactive behaviour without giving up capabilities to model complex behaviour.

Note that the architecture will be formally assume that the its implementation will be realized in an Object-Oriented Programming (OOP) paradigm. There are multiple reasons for that. Firstly, The nature of agent based systems, having their internal logic and interacting with the surroundings through pre-defined interface, corresponds to a large degree to the concepts of OOP, especially the encapsulation mechanism.

The individual layers/components will be outlined in the following section, in a bottom-up fashion.

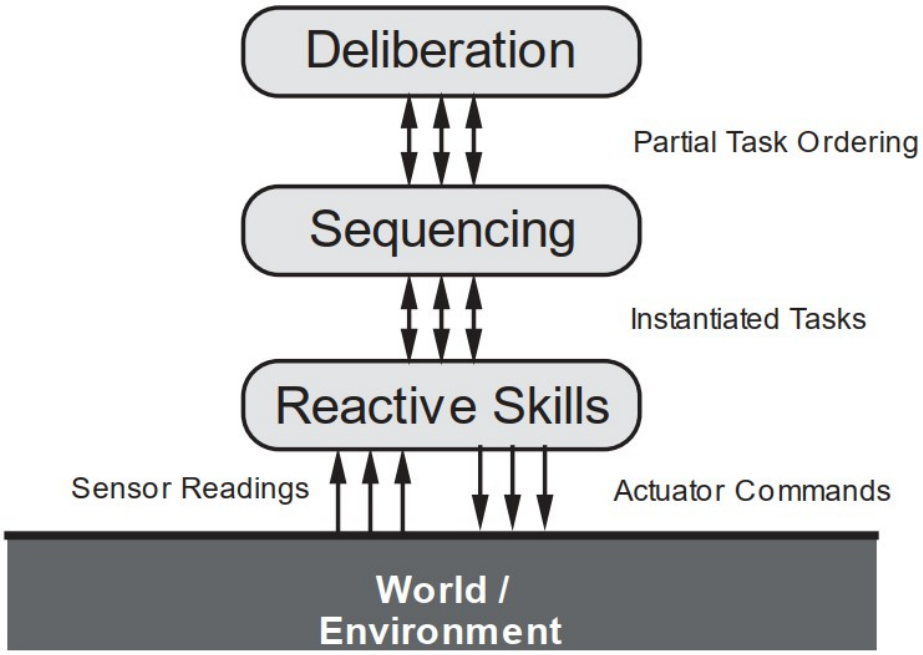

Figure 12: An architecture model of a $_3$T hybrid architecture [27]

[1]To remind the reader of the architecture's general structure, its schematic is shown below (fig. 12).

5.1.1 The architecture layers

In this section, the individual layers of the architecture will be defined. The layers' definition will adhere to the characteristics of the $_3$T architecture. For each layer, it's purpose and relation to other layers will be described, together with technical implementation guidelines, such as configuration etc.

Reactive skills layer

This layer encapsulates all the *skills* the agent is able to do. These are the most primitive types of its behaviour[2]. For example, a skill might be to slow down or to follow a vehicle in case of vehicle-based agent. Skills can be activated and deactivated based on the cognitive capabilities of higher layer (sequencing) and more than one skill can be active at any moment, and they should be independent of each other. However, one skill might use output of another skill as its input, effectively making a network of skills. By providing skills for elementary operation, a level of abstraction is created, which emphasizes focus on impacts of task sequencing without being overly concerned by how the individual skills interact with the environment. The original article proposing the architecture stresses a following canonical approach for skill specification [27]:

1. The skill's input and output specification. Each skill must provide a description of the inputs it expects and a listing of the outputs that it generates
2. A computational transform This is where the skill does its work. Once a skill is enabled, it uses this transform to continually recompute its outputs based on its current inputs.
3. An initialization routine. Each skill is given the opportunity to initialize itself when the system is started
4. An enable function. The sequencer can enable and disable skills Depending on the context, a skill is given the opportunity to perform any special start up procedures each time it is enabled.
5. A disable function When a skill is no longer needed, the sequencer will disable it and the disable function performs any necessary cleanup actions.

In practice, each skill will be an individual script that will be *asynchronously* run. Each script will have to implement a *start-up* and *clean-up* function. Inside the script, it will be possible to interact with the skills inputs and transform them to outputs. For instance, mainly will be possible to route output from sensor or communication skills to other skills that use the outputs. Skills will provide interface to the higher abstraction layer through pre-defined events specific to each skill.

[2]The original author of the architecture refers to them as Reactive Action Packages (RAPs) [28].

In relation to the lower-level skills (i.e. communication and sensory), such skills should be able to have configurable physical layer parameters such as signal range or relative service reliability (error rate). Speaking of error behaviour, a skill should also be able to return a failure state event to enable fail-safe behaviour.

Sequencing layer

The sequencing layer is responsible for *execution* of the individual skills, essentially controlling which skills to enable/disable to achieve a certain behaviour. It is the intermediate layer between the reactive skill layer and deliberative planning layer.

This layer will feature logic that will be used to accomplish a task in varying circumstances, based on the state of the environment or the agent itself. For instance, regarding a driving vehicle, a routine task to avoid an obstacle (which is defined by enabling certain skills by the sequencer) will be handled differently on a road with more than two lanes.

The sequencing layer should not be responsible for more complicated reasoning, but rather to define how to handle routine situations. The responsibility for high level planning to achieve a global goal for an agent is reserved for the upper (deliberation) layer. The sequencing layer's role in relation to the upper layer is to provide an additional layer of abstraction.

To formalize the purpose of this layer in the proposed system, it should contain definitions of which skills to enable to complete a rudimentary task, managing sequences of common skills along with logic that will account for completing them in different scenarios. The layer interact with both the lower (skill) layer - interacting through events triggered by individual skills to alter skill execution, and the upper (deliberation) layer.

Deliberation layer

The deliberation layer (also referred to as the planner) synthesizes high-level goals into a partially ordered *plan*, listing tasks that the agent has to perform, in order to achieve the specified goal. Ultimately, the main purpose of the layer is to manage tasks execution to achieve a specified goal. For example, More generally, a goal could be something that the user wants the agent to accomplish, for example, a vehicle's goal is to arrive to its final destination.

Thanks to the abstractions provided by the two lower layers, the planner can be represented as a conditional state-based model, vastly simplifying its definition [27]. In addition to this, it will be possible to add parameters to goal definition. The final version of the plan will then be generated based on parameter values and internal logic that will process them.

In practice, the most trivial specification of the deliberation layer will consist of:

a) **Primary goal** - The main objective that the agent was designed to perform.

b) **Fail-safe goal** - The desired outcome of agent's behaviour in case an unexpected failure occurs in any its components or when the agent fails to achieve the primary goal. For example performing a Minimum Risk Maneuvre in vehicle based agents [35].

For more complex behaviour, additional goals can be specified. In that case, the choice which plan to prioritize will be determined by sets of pre-conditions and constraints represented by the agent state.

To sum it up, a detailed view on the individual components of the architecture and their interface is on the figure (13) below.

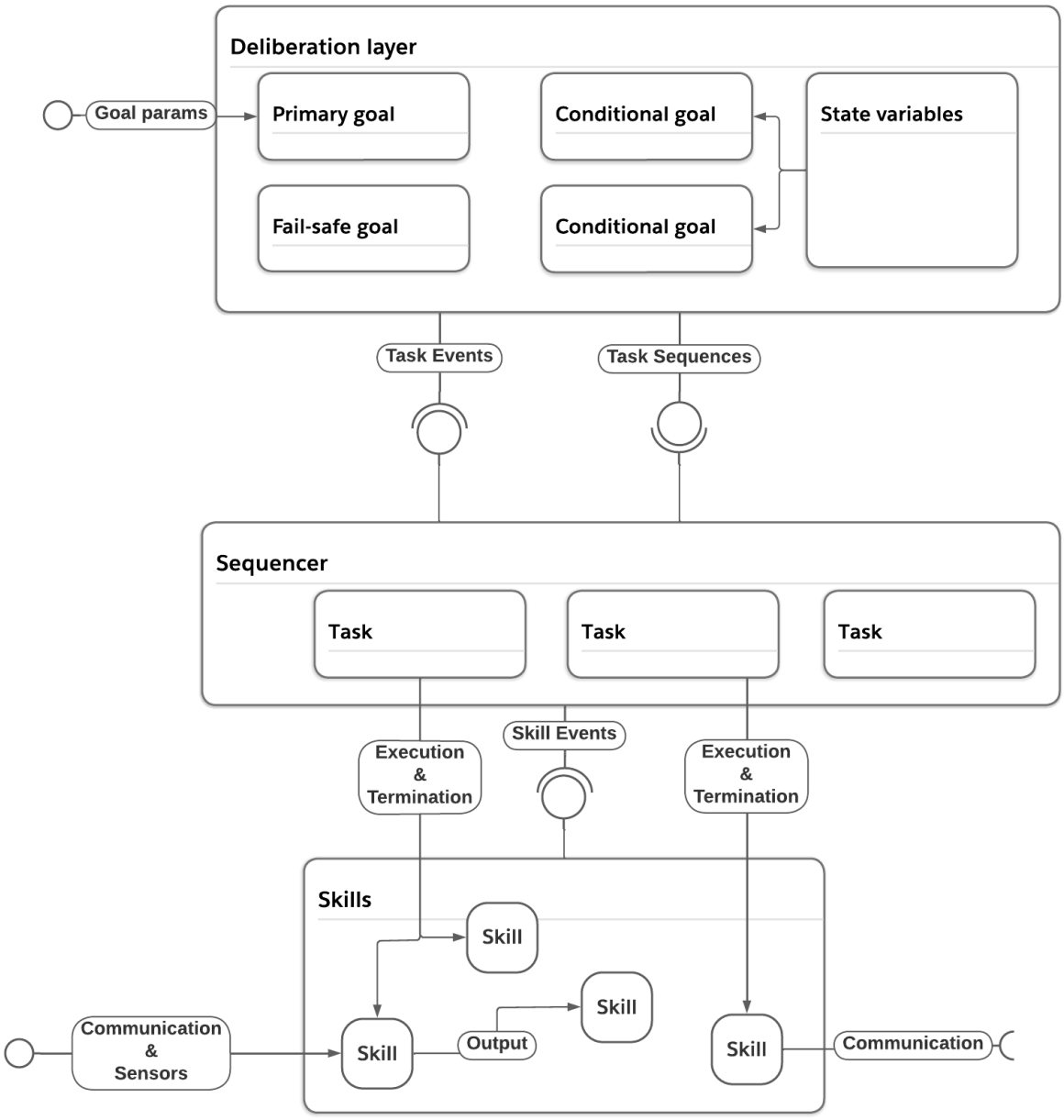

Figure 13: An overview of the proposed 3T architecture

5.2 Macro architecture

5.2.1 Communication protocol

As has been discussed in the previous sections, communication plays an important part in MAS, mainly because it vastly extends capabilities of interaction between agents and thus the ability to model more complex behaviour. In section (3.3.3), the ACL communication standard for MAS has been introduced and requirements for the protocol implementation have been presented. To be able to utilize the communication, requirements for messaging utility and high-level implementation details will be presented.

5.2.2 Broadcasting

The first type of communicating that will be implemented is message broadcasting. This is argued by the fact that ITS uses information broadcasting in a lot of cases. Generally, there are mostly two types of service messaging in ITS [36]:

- **Periodic status exchange** - ITS services often need to know about status of other actors, such as terminals or vehicles. These periodic updates often include basic status messages such as location, ID, speed, etc.
- **Asynchronous notifications** - Messages that are used to inform about specific event, usually related to a specific service and functionality.

The framework should therefore support message broadcasting for both synchronous and asynchronous events, which should ensure that all potential use-cases are covered.

5.2.3 Implementation

As has been mentioned above, the ability to broadcast and subscribe to said broadcast should be handled by a separate skill component. Upon skill initialization, the communication skill will connect to a communication channel and listen or send messages. Each communication stream will have to specify what kind (group) of messages it will be sent through it, so that agents can be set up to only use channels with messages relevant to them. The agents will be able to react to received messages by sending an internal event message to the sequencer.

Given the great utility of C-ITS system and identified similar characteristics and use-cases with agent-based systems (discussed in section 2.0.3), the framework will feature V2V communication support with standardized message structure and semantics out-of-the-box; Namely the Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM).

5.2.4 ETSI message services

Due to wide range of beneficial use-cases for the aforementioned types information sharing types (periodic status exchange & asynchronous notifications), ETSI has defined two basic messaging services; The CAM message (synchronous status update) and the DENM message (asynchronous notifications). Both message types will be described along with the implementation details.

Cooperative Awareness Message

The CAM message is specified by the Cooperative Awareness Basic Service. By definition, CAM provides by means of periodic sending of status data a cooperative awareness to neighbouring nodes. From the message propagation point of view, the CAM differs from the DENM message by only allowing *single-hop transmission*. This way, only vehicle sin the relative vicinity of the broadcasting vehicle will receive the message. CAM messages should be then processed by the CAM Management component to apply additional logic [37]. For example the CAM Management can detect a Slow Vehicle Warning from analyzing received CAM mesages from a particular vehicle. In other words, the receiver is expected to evaluate relevance of the contained information. The CAM service has its message format specification, which is found on figure (14) below.

Decentralized Environmental Notification Message

Instead of vehicle/station status update, DENM messages directly carry traffic safety-related information, characterized by *event type* identifier. Another difference from the previous message type is that DENM messages are *multi-hop*, and their range is expressed by geo-validity specification. Among other attributes, the DENM message contains attributes for message generation frequency or temporal validity threshold (expiration time). Despite specifying such attributes, it is still up to the receiving station/vehicle to determine information relevance [38] and whether to take further actions like displaying the message to the driver. The DENM service message specification is on the figure (15) below.

5.2.5 Negotiation & Agreement

I am still not sure if I should include the following sections related to conflict resolution as I don't know if pursuing this complex topic would bring substantial value to the proposed system

Negotiation between agents is important especially when wo agents run into each other, followed by the currently executing skills of the agents getting interrupted. In other words, The best course of action of one agent is not necessarily best to the other. This is more than likely to happen in environments where multiple agents share the same resource and

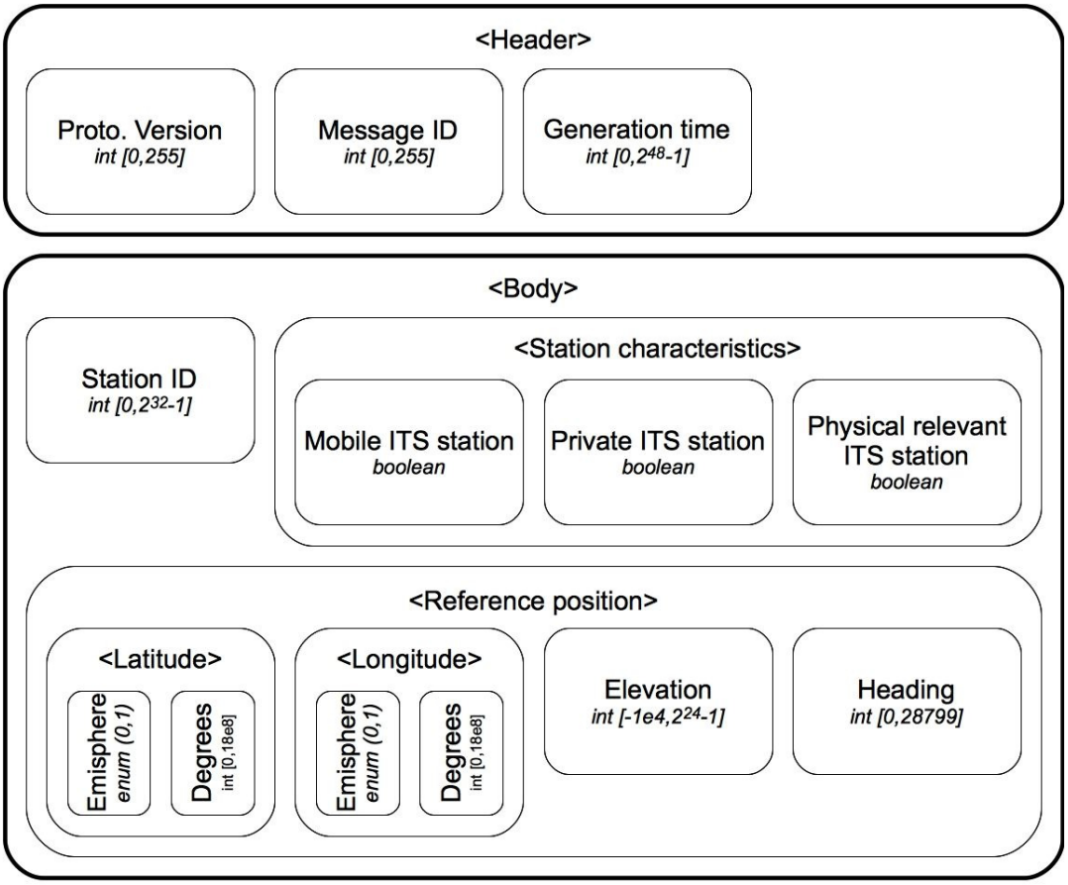

Figure 14: The CAM message specification [36]

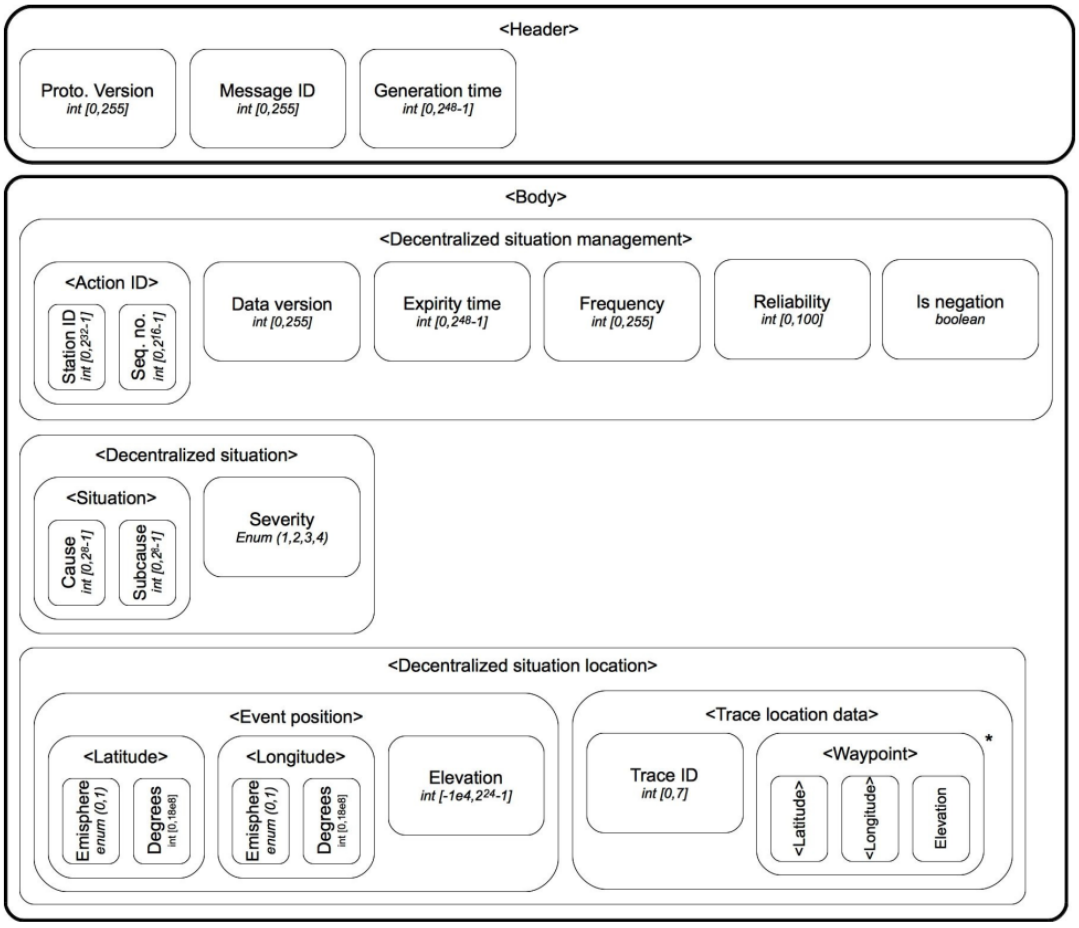

Figure 15: The DENM message specification [36]

their state is defined on the same domain, which is often the case. As such, the theory of games is formally used to model these interactions [39].

This is quite different from the principles of agent cooperation. In cooperation negotiations, the worst-case scenario improvement as opposed to working individually is net-zero. [39] defines four speech acts in agent negotiation, as seen on fig. (16). This is inherently contrary to the statement made in chapter (3.3), where it was suggested that agent cooperation should be emphasized. As agents are formally modelled as self-interested, it cannot be guaranteed that they won't encounter conflicts between each other. Therefore, modelling cooperation before figuring conflict resolution would inherently result in a more fragile system more prone to malfunction.

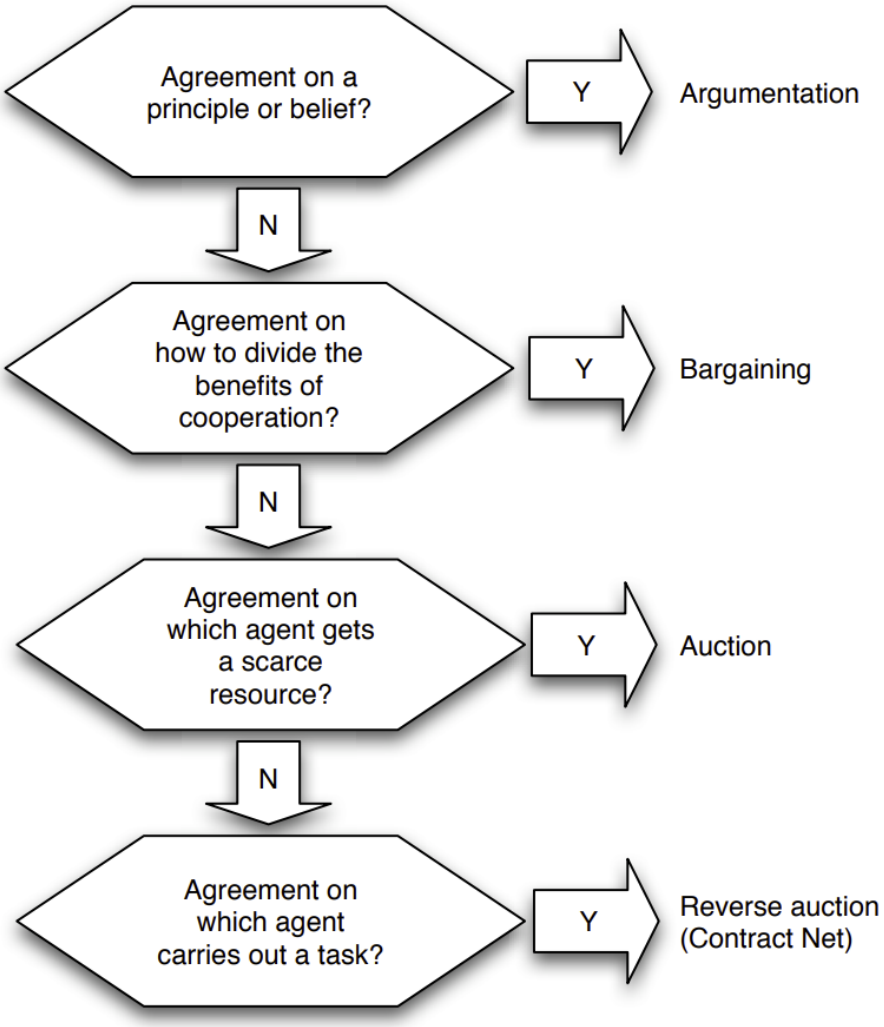

Figure 16: Types of agent negotiation [39]

5.2.6 Conflict detection

As has been argued above, because of scope limitation of this thesis, conflict resolution as a system functionality should precede cooperation. The cooperation abilities of agents can be added later by extending the framework. This decision will theoretically lead to better system stability, before offering extended behaviour possibilities.

The first step is to have a way for agent to determine a conflict has happened. Agents wouldn't be able to resolve conflicts if they didn't know it has occurred. To comply with the $_3$T architecture topology defined above, this should be handled on the reactive skills layer. Consequentially, detected conflict will be handled through a dedicated skill module. If an agent is expected to run into a conflict with another agent (of the same type), if will have to be equipped with a dedicated skill that will behave as a sensor detecting conflict with the implementation left to specific use-case. In other words, the logic for conflict detection will be left to define together will specific agent and its properties, and other modules.

It needs to be said that when one agent detects a conflict through one of its modules, it will be required to inform other agents about the conflict so that all involved agents will be aware of the situation.

5.2.7 Conflict resolution

Now that the conflict detection process was defined, in that case, it will be possible to build on it to define how conflicts will be resolved. Considering the findings in chapter 3.3.1, the simplest tool to resolve conflicts is to define hierarchy in the agent. In order to avoid more complex mathematical reasoning models [39], hierarchy, along with agent-bound state variables can be used as a bidding resource. Whichever agents is able to bid the highest value will get priority in conflict resolution (e.g. by being able to act in its own interests).

To facilitate the communication related to conflict resolution & bidding, the FIPA message standard will be used. The FIPA standard offers pre-defined messages for such situations - most importantly the `Call For Proposal (CFP)` and `Propose` messages. Although it was mentioned that agents are self-interested, in order to make things simpler, agents will assume that the counterparty in conflict is truthful and only bids resources that are available to them. Also, deciding which bid proposal has won will be handled by the agent that had called for the proposal, so the counterparty will assume that the deciding agent is truthful and decided correctly according to the bids.

5.3 Conclusion

In the preceding sections, the multi-agent system framework was proposed. This framework will be used to implement various suitable ITSs into an interactive vehicle simulator. The system was described both on micro and macro level. The micro level mainly addressed specification of inner architecture of individual agents, how they will achieve their goals using their own intelligence. For building independent agents that need to be flexible in terms of their capabilities, the $_3$T architecture was chosen and the individual layers of the architecture were specified in detail, specifying their responsibilities and capabilities.

Afterwards, the macro architecture was specified, addressing mainly the ways of how agents will interact with each other. Firstly, the communication interface was defined, including how agents will be able to use it on micro-architecture level. Building upon that, the ways of how conflicts between agents are handled was proposed, utilizing communication to resolve conflicts through resource bidding.

With the micro- and macro-architecture being specified, the final requirements for the system implementation were proposed, which aim to help ensure that the software framework will be able to facilitate ITS implementation into IVS software in an organized way, with a high system resilience.

6 Implementation toolset

With the finished system specification, it should be discussed which tools should be used to build the software framework. First off, it is important to analyze the IVS simulation system/software (i.e. the super-system) that the proposed framework will be incorporated into. An effort should be made to maximize the super-system acceptance of this system by choosing an optimal tool-set for its implementation. This, for the most part, refers to an optimal choice a programming language and a subsequent agent-based modelling library to use (if there will be any), also due to this being one of the primary tasks of this thesis.

In order to find a suitable tool for the case of MAS in the simulator software, the simulator software needs to be examined.

6.1 Simulator software

The particular simulator software that is being used at the university's faculty is being developed using the *Unity* game engine, developed by Unity Technologies. The game engine was first released in 2005 and has been used to develop numerous simulators as well as other video games ever since [40]. Unity has also been used as a tool in physical product modelling, AI & machine learning and digital twins, used in many industries [41]. The main strengths of Unity are multi-platform development support, virtual reality development support, good community support (including its own asset store) and good

documentation. The game engine has got its own development environment, which can be seen on figure (17) below.

The game engine's runtime is written in the C++ programming language, but the scripting API that it offers is in the C# language. Consequently, there will be a noticeable benefit if the chosen agent-development platform used to develop proposed system will be .NET based as well to achieve maximum customization and interoperability.

The scripting follows usual object-oriented programming paradigm, where one script contains a behaviour encapsulated in a class. Each individual script can contain reference to another script/class. Each game object can be assigned an arbitrary number of scripts that upon assigning gain access to game object's properties, such as object velocity, weight and position. Scripts can also be used to programmatically gain knowledge about the environment, for example using ray-casting to discover objects surrounding the attached game object. This object-centered development suggests that applying agent-based behaviour could be well integrated into the simulator software and offer needed flexibility and cross-integration.

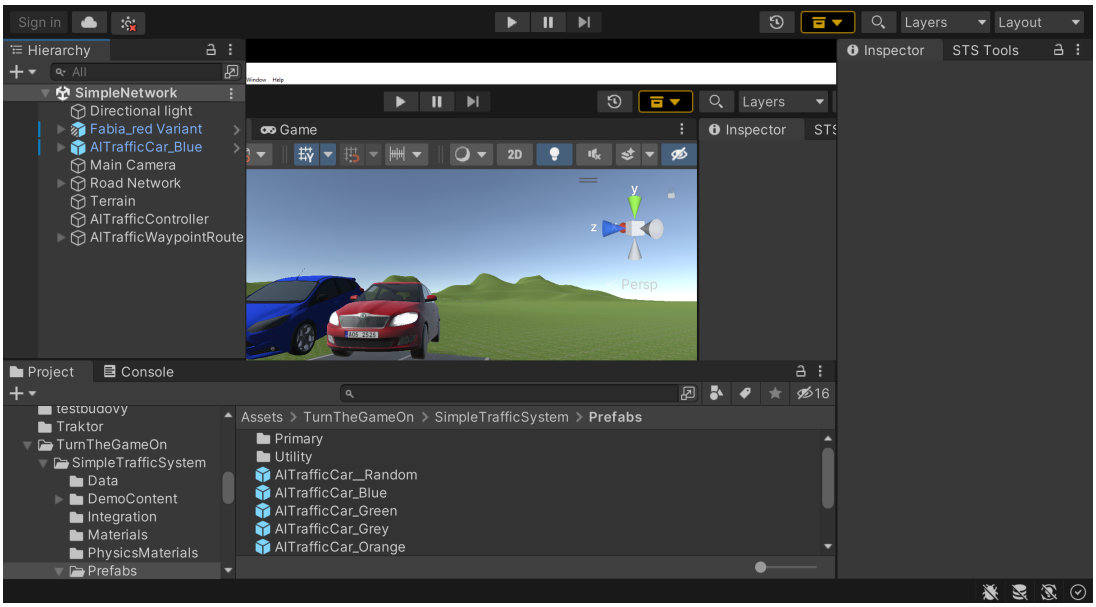

Figure 17: The Unity development platform UI

6.2 Agent Development Platforms

Having a development tool that is dedicated for agent-based modeling is an advantage as opposed to building the system on a "green field", as the tools have the common domain of agent system paradigms implemented and ready to use by a designated interface (e.g. communication, localization, state definition), ideally being built in line with widespread standards and ideally tested in practical commercial and industrial use.

An example of popular platforms, according to [39] can be found below.

JADE

Jade is a free software developed under a grant from the European Commision. The main advantage of the platform is containerization of agents, which enables distribution of agents across systems, even with cross-platform (OS) support. Configuration changes can be done in run-time. The core components of the platform are Agent Management System (AMS), Agent Communication Channel (ACC) and Directory Facilitator (DF). JADE also supports the FIPA communication standard ACL, mentioned earlier. The platform is implemented in the Java programming language.

FIPA-OS

FIPA-OS is an open-source implementation of the FIPA standard. It is a small-footprint development platform, which is loosely coupled and offers implementation also on mobile devices. The development is also done using the Java programming language.

Although these most popular platforms for agent-based development are industry-tested tools, in the current times they are obsolete, as their development was abandoned not later than in year 2005. Also, most of the advantages described, such as multi-platform support and decentralization across machines is not relevant to this thesis' use-case, as the proposed MAS will be run one one machine, preferably directly within the simulator software, to minimize system acceptance conflicts.

Another issue is that such tools have been developed to run on JVM (Java Virtual Machine), therefore the integration with the simulator software would create another layer of complexity to build an ABM framework. Related to the system implementation, it is therefore important to choose a platform supporting development in C# language as well, to ensure full and non-complex integration with the simulator software.

ActressMAS

ActressMAS is a multi-agent framework running on the .NET runtime, written in C#. The author says that its main advantages are conceptual simplicity and ease of use [42]. The framework's last release was in the year 2021, making a non-outdated option with potentially ongoing development. Notably, the framework is open-source, meaning the source code can be inspected and possibly adjusted to the needs of the developed system.

Upon inspecting the source code of ActressMAS, there are five main components that form the basic building blocks of a MAS environment - `Agent`, `EnvironmentMas`, `Container` and. The class diagram of the components can be seen on the image (18) below.

The agent environment can be distributed between more machines. Each machine will run a container which communicates with a server, as seen on fig. (18). The service handling communication between containers was is a big part of the framework, as it is composed of many classes, offering communication through web sockets and even offering to use the SSL security protocol and async version of communication, see fig. (19).

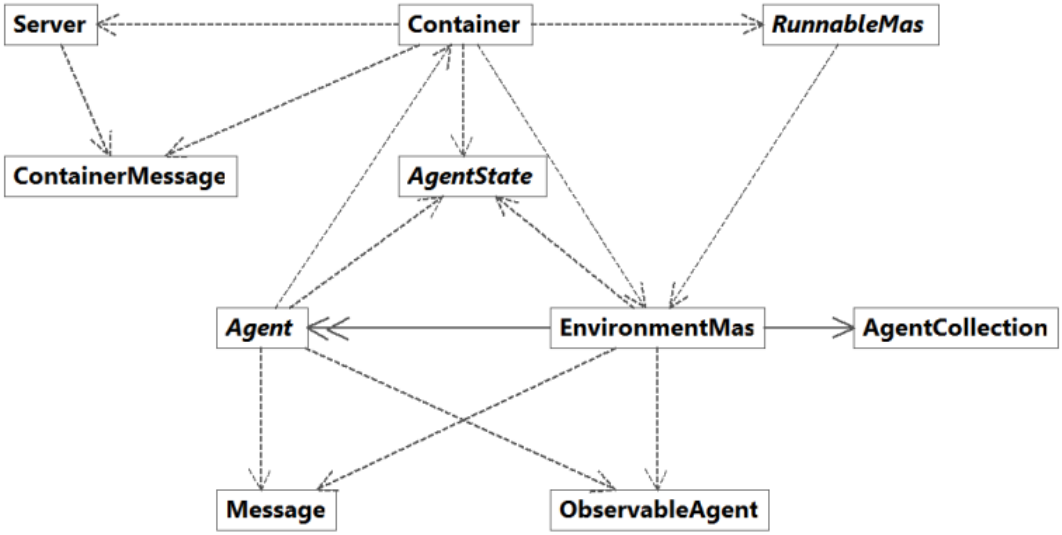

Figure 18: ActressMAS - Class diagram of framework's main components

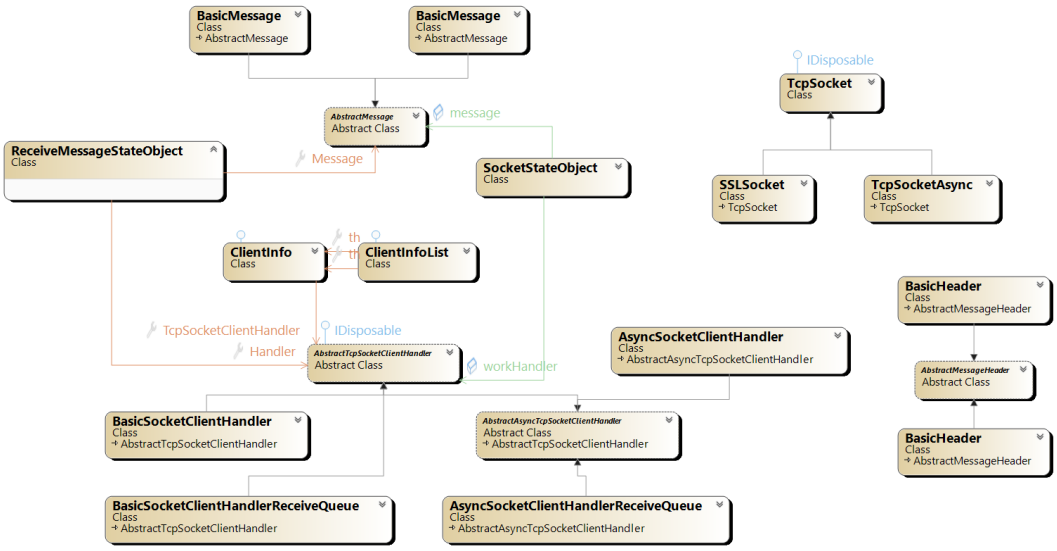

Figure 19: ActressMAS - Class diagram of distribution service

There are, however, some issues related to use-case of the proposed ITS framework. Overall, the implementation of the classes creating the local agent environment doesn't contain much logic - the logic mostly serves to distribute messages to agents. It cannot be clearly said if this is an advantage or a disadvantage, as for specific use-cases, like building an ITS multi- agent framework is expected to have specialized logic. However, The lack of behaviour implementation also means that the agent architecture will have to be build from the ground up either way, not brining any benefit to use this framework. On the second note, agents are only able to update their state once they receive a message, which doesn't meet the requirement for agents to act independently and autonomously. This also means that agents are not able to gather and act upon information from their sensors, just from messages received from other agents. This behaviour might be implemented by the user, but there might be conflicts with the current design of the interaction interface of the agents and the `EnvironmentMas` class that would add redundant effort as opposed to building an agent environment from scratch.

For the reasons explained above, a decision not to use a pre-existing implementation of MAS framework is made. This is mainly because of the perceived threat of identified drawbacks of the existing solutions overshadowing the advantages they would bring. Building the framework independent from a pre-existing MAS platform will offer maximum customization according to the system specification and its requirements.

6.3 Conclusion

In this section, the technical aspects of implementation requirements were discussed. The simulator engine has been introduced, describing the aspects of programming integration. Main aspects of integration have been discussed, which should enhance te subsequent decision about which MAS development platform to use for proposed system implementation. Consequently, MAS development platforms suggested by literature were examined. However, their review concluded not to use them as they are outdated and would not offer advantage with regard to integration with simulator software, as their focus is to integrate with distributed computer networks focusing on different problematics. Lastly, a promising agent developed framework called ActressMAS has been introduced, whose main advantage, bearing our use-case, being .NET based and open-source, allowing for adequate customization. However, after further inspection of the codebase, it was concluded that the way the agent-based system proposed in this thesis is designed, there were no significant potential benefits that implementation of the framework would bring. The ActressMAS framework's implementation revolved around deployment across multiple machines and offered minimal abstraction of agent framework itself, i.e. agent behaviour and interaction.

7 System implementation

Now that the system has been designed and the technical requirements and tools as well, it is possible to create the system implementation. In this chapter, the process of building the software package and subsequent implementation into the simulator software will be described.

The software package design will be separated from the simulator software, so that it will be implementation agnostic and could be used for other game engines or other simulation purposes.

The first section will be devoted to micro-agent implementation, where it will be described which classes and services is the agent unit comprised of. The next section will describe how the communication between agents is implemented, including implementation of the ACL and C-ITS communication standards and agent negotiation.

The way the framework is built is that it mostly consists of *abstract* classes that have got some common behaviour pre-defined, but require the user to specify given behaviour that is exclusive to their use case. This allows for a framework that has got the generic behaviour already completed but gives the user freedom and guidance with regard to detailed behaviour implementation.

7.1 Agent implementation

As has been defined in the chapter proposing the system model (5), the agent component will be composed of the three main layers, represented as standalone classes interacting with each other - *deliberation*, *sequencer* and *skill* classes.

7.1.1 Inter-layer interaction

There are two ways how the individual layers can interact and inform each other. The top-down interaction (the deliberation layer is considered to be top and skill layer the bottom one) is done by supplying the superior (i.e. top) layer with a reference to the lower layer so that its whole interface is available to use. Interaction initiated by lower layer with the upper layer is implemented in a different way, where the lower layer initiates a context-specific event with an optionally-specified content that the superior layer subscribes to. The reason behind this implementation is that this ensures that the lower layers can't control the upper layers but rather just inform about potentially important/interesting events that have taken place while operating and interacting with the environment.

Below is a class diagram of the components with marked dependencies, including their methods and properties (fig. (20)).

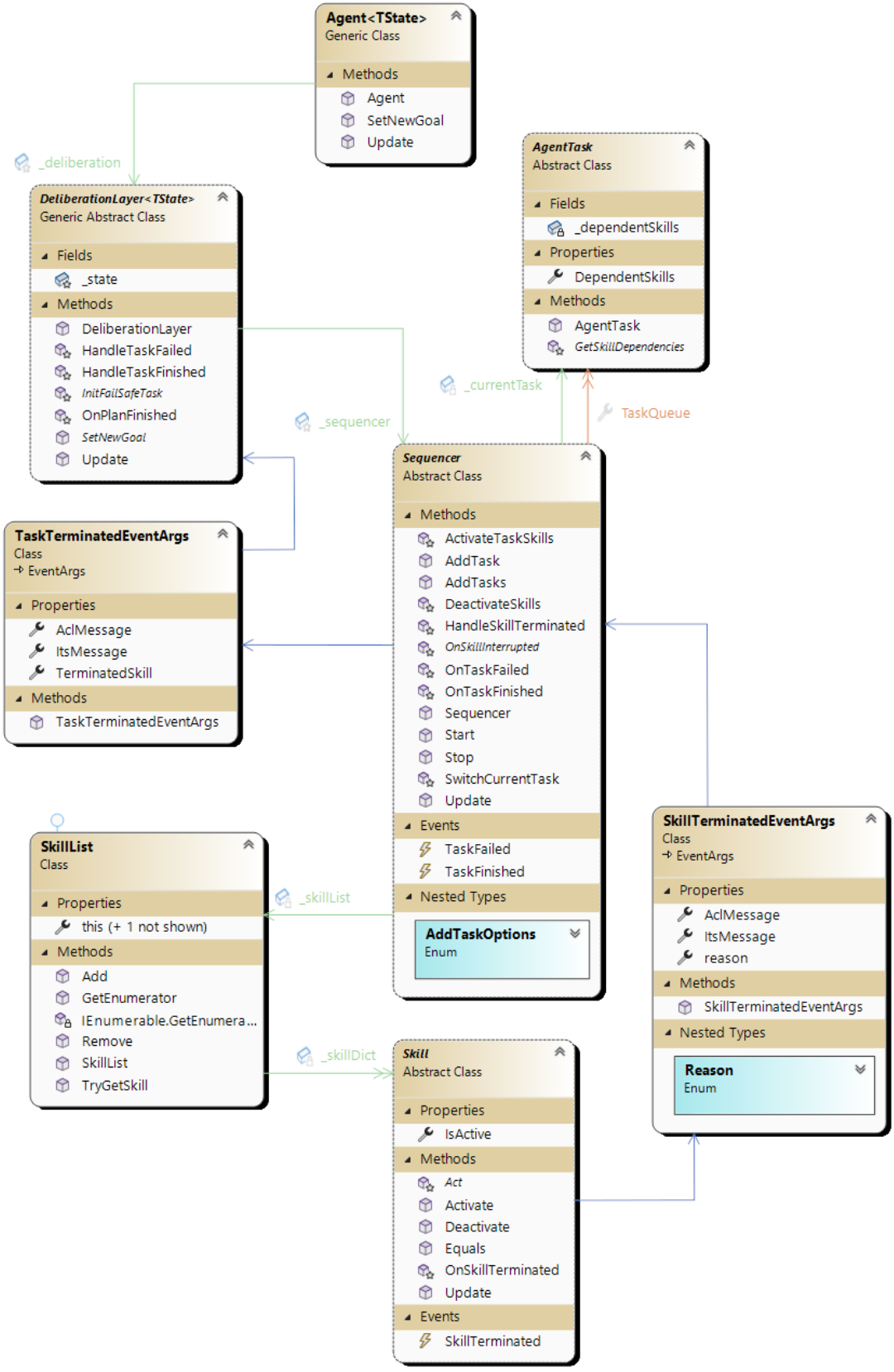

Figure 20: Class diagram of main agent framework components

7.1.2 Deliberation layer

The deliberation layer is the highest from hierarchical point of view. Its responsibility is to create a plan that satisfies a goal that the agent gets as an input. Apart from the layer being abstract (i.e. requiring further implementation details specified by inheriting from the class), it is also generic, which means that it will defer type specification of its state until its is declared by the derived class. This leaves the state specification to be implemented to the user's needs. For example, a vehicle agent can have its state as a three-dimensional vector, whereas a broadcasting station will have its state defined differently (enum etc.). The initial value of state parameter is required to be specified upon object instantiation/ creation.

Upon providing a goal to the layer by the executing client, it should to resolve it by initializing a sequence of pre-defined `AgentTask` objects whose execution should accomplish the goal. The input goal should be defined as a high level keyword, and the layer is assumed to receive only pre-defined goal keywords, which will be matched to different patterns of task resolving logic.

The `AgentTask` class encapsulates a list of Skill types that shall be activated once the Task is being executed. The skill list does not contain the actual instantiated objects, but just information about skill type (derived from the `Skill` abstract class). Such type of pseudo-contract is then transferred to the `Sequencer` layer and validated at run-time. In other words, the `Sequencer` class must contain instantiated `Skill` objects listed in the received `AgentTask`.

To offer a form of Task reuse, a parametrization of Tasks is implemented. This feature is implemented such as the task also contains an optional transform procedure delegate which gets passed to the `Skill` object once it's activated as part of the task execution. As an example, a Task which contains a general communication skill which broadcasts messages to other agents can have the broadcasted message type and content specified differently in each separate instance of resolved Tasks. The logic in the `AgentTask` type can be reused while allowing for parametrized Skill initialization (e.g. broadcasted information type).

This layer can also receive notification from the lower layer when a Task has failed or the initialized sequence of Tasks (i.e. plan) has finished. As defined in the system requirements, the user must implement behaviour when the execution of an instantiated plan has failed or has been interrupted by impacting events in the operating environment. The user has to implement a fail-safe `AgentTask` that will get activated in case the Deliberation layer is not able to achieve the specified goal.

7.1.3 Sequencer layer

The `Sequencer` class is responsible for the most amount of general logic. This is because it interacts with both the upper and lower layer and hands over information about skill execution to the Deliberation layer. Its primary responsibility is to keep track of currently executing skills and activate or deactivate them respectively to the currently executing Task.

When the Deliberation layer generates a sequence of Tasks, it is enqueued into the `Sequencer`'s internal Task queue. The Tasks in the queue get sequentially executed. Before activating skills according to the task specification, the task-specific initialization transform, if there is any, is applied to respective Skill(s). Afterwards, the Skills get activated by the Sequencer. The `Sequencer` class also exposes interface to upper (Deliberation) layer to dynamically add to or overwrite the `AgentTask` queue during runtime. This can come useful when there are immediate changes to the environment which the Deliberation layer evaluates to re-plan to reach the current goal.

Other responsibility of the `Sequencer` class, which requires implementation from the user by inheriting from the base class, is Skill termination management. The class gets notified about skill terminating by subscribing to the `SkillTerminated` event for each of the Skills bound to the layer. When one of the Skills trigger such event and terminate, the `Sequencer` gets information about the *type* of terminated Skill and *termination reason*, which is of one of three defined values - `Finished`, `Interrupted` or `Failed`. The logic implemented by the user, based on the type- and reason-based inputs, should identify whether the case of Skill termination is detrimental to the execution of the whole Task (meaning Task execution has *failed* or *finished*). If so, the `Sequencer` triggers respective Task termination event and the resolving logic is handled to the `Deliberation` class. Alternatively, the terminated Skill can be left disabled or re-started.

7.1.4 Skill layer

The Skill layer comprises of a defined collection of objects derived from the base `Skill` class. The implementation of the base abstract class is only about defining its methods that should ensure proper integration with the upper layers, such as methods for activation & deactivation of the skill exposed to the `Sequencer` class.

The implementation required by the user-specific use-case of particular skill is mainly the `Act()` method implementation, where the user is required to specify the Skill's computational transform (as specified by the requirements in section (5)).

The Skill layer is also expected to utilize chaining of Skill inputs and outputs. Skills are expected to slightly vary in abstraction level, meaning there should be Skills directly interacting with agent's sensory hardware and providing such information as an output.

Logic which makes use of agent's sensors compute additional transform should be encapsulated in a dedicated Skill to conform with each Skill having a single responsibility. Skill chaining is defined in the class derived from `Skill`. Skill outputs are defined as public class properties. To allow use of Skill `A`'s output in skill `B`, the reference of `A` is provided in `B`'s constructor. The `B` Skill can then use `A`'s exposed output property in its transform function.

A simplified activity diagram barring user-implemented logic can be seen on figure (21).

7.2 Communication implementation

As has been defined in the system requirements chapter (4), the framework will come with pre-implemented communication interface. There are two base object types handling communication between agents - an agent-bound `CommunicationSkill` class that serves as an agent's proxy to the communication space, and the `MessageBroker` class that handles message delivery to individual agents. In addition to that Message types created according to the C-ITS anc ACL standards were created to carry appropriate content. The class diagram of the communication interface can be seen on figure (22).

7.2.1 Message Broker

The `MessageBroker` is a *static* class (only one allowed instance per simulation) which manages message delivery and simulates the communication space. It manages both the ACL and C-ITS messages by providing overloads for its methods based on provided message type. Because the class is static, agents[3] can interact with it without having to hold a reference to it. Agents are able to:

- Register to message broadcasts
- Broadcast messages themselves
- Send messages directly to specific agents

When an agent registers to the class, it holds its reference in its register. That way the class can access the registered agent's inbox when a message is being delivered.

When sending/broadcasting a message, apart from providing the actual content (i.e. the message object), the message must be "wrapped" in a `tuple` which must contain the sender ID. When sending the message directly, obviously, the sender must provide the recipient agent ID.

[3]More precisely, communication skills that agents come equipped with

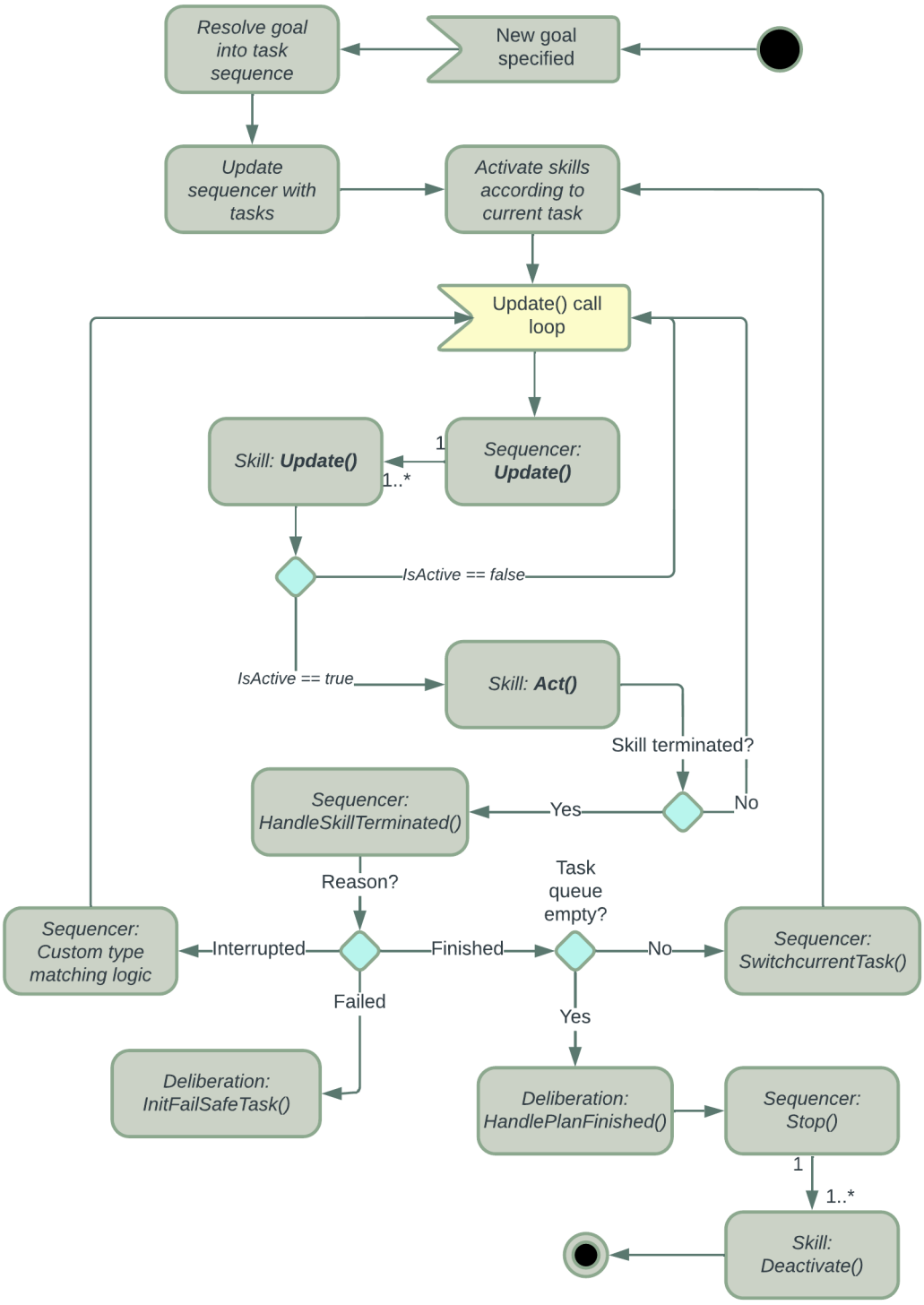

Figure 21: Trivial activity diagram of the implemented $_3$T architecture

7.2.2 Communication Skill

The `CommunicationSkill` class derives from the base `Skill` class, so it should be handled like any other skill. This type itself is *generic* and *abstract*. For the skill to be used, a derived class with specified Message type needs to be created and implemented. The pre-defined behaviour is in the `Act()` method, where all Messages to broadcast are handed to the `MessageBroker` class. The Messages to broadcast are specified upon `AgentTask` initialization by the transform procedure. That way, different `AgentTasks` can re-use the single `CommunicationSkill` class. Then, all message received in the Skill's `ReceivedMessagesQueue` are processed. The message processing logic should be implemented by the user by pattern matching. The pattern matching can be defined in the `CommunicationSkill` itself, but to honor the single responsibility paradigm, a better way is to use the pre-defined behaviour, where a `MessageReceived` event is invoked and other Skills subscribed to this event can process the message, deciding if they are interested in the contents and acting upon it accordingly.

7.2.3 Message implementation

The ACL and C-ITS (i.e. DENM and CAM services) message contents were implemented according to the official documentation. For reference, the ACL message contents can be seen on figure (11) and table (3), the C-ITS message contents can be seen on figures (14) and (15).

7.3 Qualitative analysis

Looking at the table (1), it is evident that all the tasks fall under *four* mapping types. Those will be used as entries in the qualitative analysis. Nevertheless, all types of ITS considered in the preceding sections will be examined and subsequently evaluated to decide which type of ITS would be the most optimal to implement.

The method to determine how well-suited the candidate system type is for the implementation to an IVS will be to evaluate it based on the *three* features discussed in the preceding sections:

- MAS-compatibility
- Driver engagement
- Research relevance

A set of integer values will determine significance of each feature. The significance will be given in four levels:

- None $\rightarrow$ 0
- Low $\rightarrow$ 2
- Moderate $\rightarrow$ 5

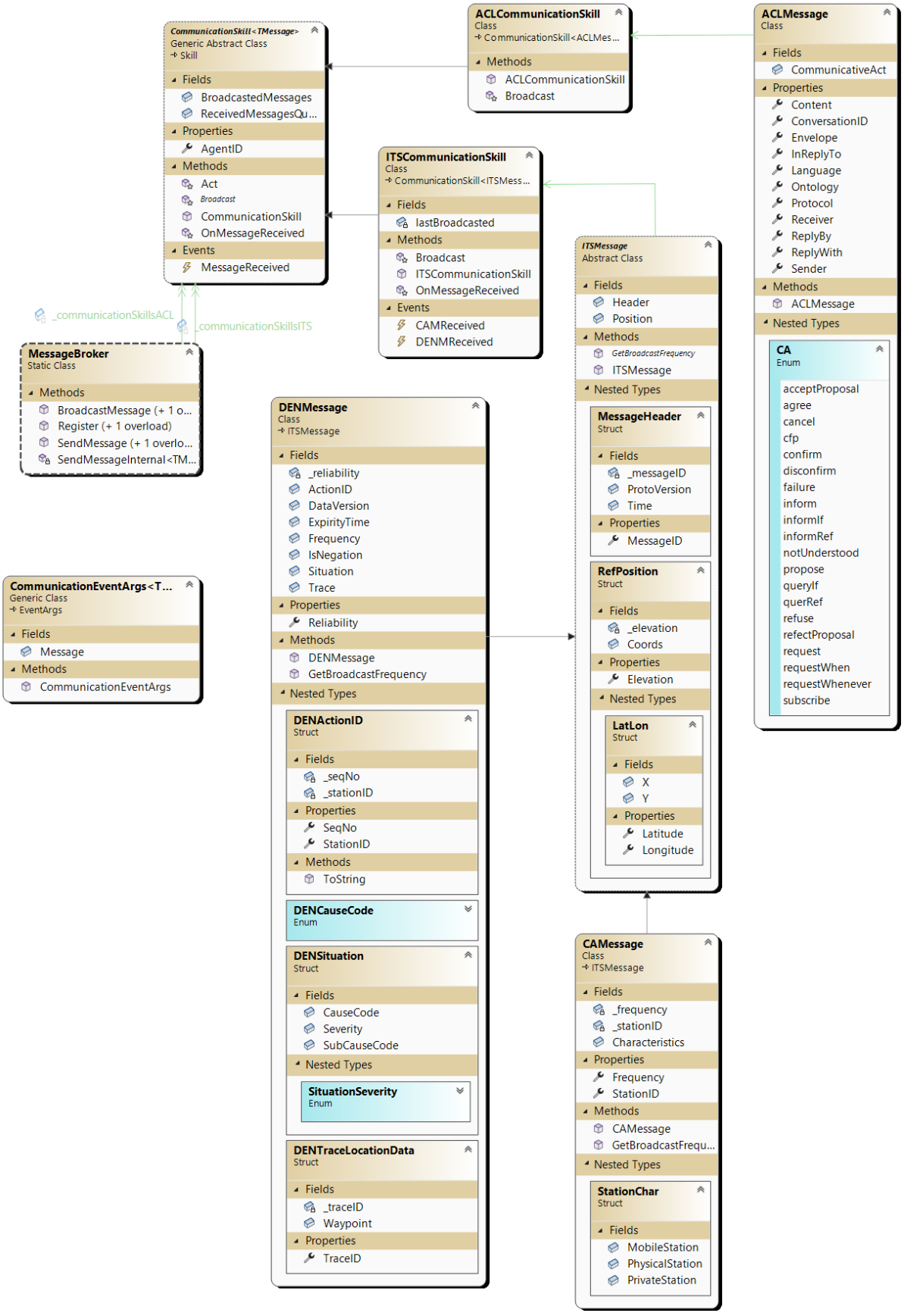

Figure 22: Class diagram of the implemented communication interface

- High → 8

Table 7: Quantitative analysis results

ITS	Features			Score
	MAS-compatibility	Driver engagement	Research relevance	
E-Call	2	2	5	9
Electronic fee collection	2	0	2	4
Parking guidance	2	8	5	15
Network traffic control	8	0	8	16
Cooperative ACC	8	8	8	**24**
European Truck Platooning	5	2	8	15
ADASIS	2	8	8	18
Mobility as a Service	8	2	8	18
Map services (Geo-fencing)	2	5	8	15
Self-driving & Platooning algorithms	8	2	8	18
IVIS	8	8	8	**24**

References

[1] European Parliament. *Road fatality statistics in the EU*. June 2021. URL: https://www.europarl.europa.eu/news/en/headlines/society/20190410STO36615/road-fatality-statistics-in-the-eu-infographic.

[2] W. Wijnen et al. *Crash cost estimatesfor European countries*. Research rep. , Deliverable 3.2 of the H2020 project SafetyCube. European Commission, 2017. URL: https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5b1e92ba3&appId=PPGMS.

[3] ETSI. *ITS schematic*. URL: https://www.etsi.org/technologies/automotive-intelligent-transport.

[4] *Intelligent Transport Systems - Road*. 2022. URL: https://transport.ec.europa.eu/transport-themes/intelligent-transport-systems/road_en.

[5] Jonathan Levy. *Emissions from traffic congestion may shorten lives*. 2011. URL: https://www.hsph.harvard.edu/news/hsph-in-the-news/air-pollution-traffic-levy-von-stackelberg/.

[6] Francesco Corman et al. "Centralized versus distributed systems to reschedule trains in two dispatching areas". In: 2 (2010), pp. 219–247. ISSN: 1866-749X. DOI: 10.1007/s12469-010-0032-7.

[7] Andy H.F. Chow, Rui Sha, and Shuai Li. "Centralised and decentralised signal timing optimisation approaches for network traffic control". In: *Transportation Research Procedia* 38 (2019). Journal of Transportation and Traffic Theory, pp. 222–241. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2019.05.013. URL: https://www.sciencedirect.com/science/article/pii/S2352146519300225.

[8] *ERTICO*. URL: https://ertico.com.

[9] Hannah Ritchie, Max Roser, and Pablo Rosado. "CO2 and Greenhouse Gas Emissions". In: *Our World in Data* (2020). https://ourworldindata.org/co2-and-other-greenhouse-gas-emissions.

[10] *C-ITS: Cooperative Intelligent Transport Systems and Services*. 2022. URL: https://www.car-2-car.org/about-c-its.

[11] *C-Roads brochure*. 2021. URL: https://www.c-roads.eu/fileadmin/user_upload/media/Dokumente/C-Roads_Brochure_2021_final_2.pdf.

[12] Anshul Saxena. *Everything You Need to Know About In-Vehicle Infotainment Systems*. URL: https://www.einfochips.com/blog/everything-you-need-to-know-about-in-vehicle-infotainment-system/.

[13] Bart van Arem, Cornelie J. G. van Driel, and Ruben Visser. "The Impact of Cooperative Adaptive Cruise Control on Traffic-Flow Characteristics". In: *IEEE Transactions on Intelligent Transportation Systems* 7.4 (Dec. 2006), pp. 429–436. DOI: 10.1109/TITS.2006.884615.

[14] Luigi Pariota et al. "Green Light Optimal Speed Advisory: a C-ITS to improve mobility and pollution". In: *2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I&CPS Europe)*. IEEE, June 2019. DOI: 10.1109/EEEIC.2019.8783573.

[15] Konstantinos Katsaros et al. "Performance study of a Green Light Optimized Speed Advisory (GLOSA) application using an integrated cooperative ITS simulation platform". In: *2011 7th International Wireless Communications and Mobile Computing Conference*. IEEE, July 2011. DOI: 10.1109/IWCMC.2011.5982524.

[16] European Commision. *Final report of the Single Platform for Open Road Testing and Pre-deployment of Cooperative, Connected and Automated and Autonomous Mobility Platform (CCAM platform)*. 2021.

[17] Balaji Parasumanna Gokulan and D. Srinivasan. "An Introduction to Multi-Agent Systems". In: vol. 310. July 2010, pp. 1–27. ISBN: 978-3-642-14434-9. DOI: 10.1007/978-3-642-14435-6_1.

[18] Birgit Burmeister, Afsaneh Haddadi, and Guido Matylis. "Application of multi-agent systems in traffic and transportation". In: *IEE Proc. Softw. Eng.* 144 (1997), pp. 51–60.

[19] R. Brooks. "A robust layered control system for a mobile robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. DOI: 10.1109/JRA.1986.1087032.

[20] R.A. Brooks. *Cambrian Intelligence: The Early History of the New AI*. Bradford book. MIT Press, 1999. ISBN: 9780262024686. URL: https://books.google.cz/books?id=LZa3QgAACAAJ.

[21] M. Wooldridge. *What agents aren't: a discussion paper*. UNICOM Seminar on Agent Software, 1996. DOI: 10.1049/ic:19960648.

[22] Philippe Caillou et al. *A Simple-to-use BDI architecture for Agent-based Modeling and Simulation*. 2017. DOI: 10.1007/978-3-319-47253-9_2.

[23] Ingrid Nunes, Frederico Schardong, and Alberto Schaeffer-Filho. "BDI2DoS: An application using collaborating BDI agents to combat DDoS attacks". In: *Journal of Network and Computer Applications* 84 (2017), pp. 14–24. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.01.035. URL: https://www.sciencedirect.com/science/article/pii/S1084804517300577.

[24] Michael Wooldridge and Simon Parsons. "Intention Reconsideration Reconsidered". In: (1999), pp. 63–79. ISSN: 0302-9743. DOI: 10.1007/3-540-49057-4_5.

[25] Patricia Anthony et al. "Agent Architecture: An Overview". In: *TRANSACTIONS ON SCIENCE AND TECHNOLOGY* (Jan. 2014), pp. 18–35.

[26] Jörg Müller. "Architectures and applications of intelligent agents: A survey". In: *The Knowledge Engineering Review* 13 (Feb. 1999), pp. 353–380. DOI: 10.1017/S0269888998004020.

[27] R. Bonasso et al. "Experiences with an Architecture for Intelligent, Reactive Agents." In: vol. 9. Jan. 1995, pp. 187–202. DOI: 10.1080/095281397147103.

[28] R. James Firby. "An Investigation into Reactive Planning in Complex Domains". In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*. AAAI'87. Seattle, Washington: AAAI Press, 1987, pp. 202–206. ISBN: 0934613427.

[29] Yoav Shoham and Kevin Leyton-Brown. *Multiagent SystemsAlgorithmic, Game-Theoretic, and Logical Foundations*. DOI: 10.1017/cbo9780511811654.020.

[30] Ariuna Damba and Shigeyoshi Watanabe. "Hierarchical Control in a Multiagent System". In: (2007). DOI: 10.1109/icicic.2007.334.

[31] P. G. Balaji et al. "Multi-agent System based Urban Traffic Management". In: (2007). DOI: 10.1109/cec.2007.4424683.

[32] Michael Vijsel and John Anderson. "Coalition Formation in Multi-Agent Systems under Real-World Conditions". In: *AAAI Workshop - Technical Report* (June 2004).

[33] Foundation for Intelligent Physical Agents. *Agent Communication Language Specification*. 2001. URL: http://www.fipa.org/specs/fipa00018/.

[34] Stefan Poslad. "Specifying Protocols for Multi-Agent Systems Interaction". In: 2 (2007), p. 15. ISSN: 1556-4665. DOI: 10.1145/1293731.1293735.

[35] Working party on Autonomous vehicles. *Proposal for the 01 series of amendments to UN RegulationNo. 157 (Automated Lane Keeping Systems)*. Standard ECE/TRANS/WP.29/2022/59/Rev.1. United Nations, May 2022.

[36] José Santa et al. "Vehicle-to-infrastructure messaging proposal based on CAM/-DENM specifications". In: *2013 IFIP Wireless Days (WD)*. 2013, pp. 1–7. DOI: 10.1109/WD.2013.6686514.

[37] ETSI. *Specification of Cooperative Awareness Basic Service*. Standard RTS/ITS-0010018. 2014.

[38] ETSI. *Specifications of Decentralized Environmental Notification Basic Service*. Standard REN/ITS-0010019. 2019.

[39] Walter Binder. "State-of-the-art in Agent-based Services". In: (Oct. 2022).

[40] Unity Technologies. *Unity*. 2022. URL: https://unity.com/.

[41] Unity Technologies. *Top Uses of Unity Solutions*. 2022. URL: https://unity.com/solutions/government-aerospace.

[42] Florin Leon. "ActressMAS, a .NET Multi-Agent Framework Inspired by the Actor Model". In: *Mathematics* 10.3 (Jan. 2022), p. 382. DOI: 10.3390/math10030382.