CZECH TECHNICAL UNIVERSITY IN PRAGUE

FACULTY OF TRANSPORT SCIENCES

JAN MACEK

AGENT-BASED MODELING OF ITS SYSTEMS IN VEHICLE SIMULATOR

MASTER'S 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):	**Modelování 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.

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

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.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta dopravní -3-

[illegible] [illegible] v z.

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.

[illegible]

Bc. Jan Macek
Student's name and signature

Prague ....June 3, 2022

Abstract

The use of Intelligent Transport Systems (ITS) has become increasingly important in managing transportation networks and improving safety, efficiency, and sustainability of traffic. This thesis aims to advance the development of ITS through the utilization of Multi-agent Systems (MAS) in vehicle simulator software. The opportunities and underlying challenges of successful simulation of an ITS system in a vehicle simulator software are investigated and consequentially, a framework for ITS solution implementation into a simulator software is presented. The agent-based framework is able to facilitate simulating complex behaviour of various road users as well as road equipment such as (cooperative) traffic lights. The main advantage of the framework is modularization of agent's skills which makes it easier to model complex behaviour. The framework also simplifies implementation of communication interface to arbitrary agent, which makes it a suitable tool for Cooperative ITS (C-ITS) simulation. Finally, a simple ITS solution featuring a connected traffic light which provides information to connected vehicles for dynamic navigation is successfully integrated into the simulator using the framework. The framework can enable researchers to be able to implement various Intelligent Transport systems into vehicle simulator software while reducing development time and increasing maintainability of the software.

Abstrakt

Využití inteligentních dopravních systémů (ITS) je stále důležitější pro řízení dopravních sítí a zlepšování bezpečnosti, efektivity a udržitelnosti v dopravě. Tato práce si klade za cíl pokročit ve vývoji ITS prostřednictvím využití Multi-agentních systémů (MAS) v softwaru simulátoru vozidla. Dále jsou prozkoumány příležitosti a hlavní úspěšné simulace systému ITS v softwaru simulátoru vozidla a následně je prezentovaný softwarový framework pro implementaci ITS do softwaru simulátoru. Framework založený na agentech je schopen usnadnit simulaci komplexního chování různých účastníků silničního provozu a také silniční infrastruktury, jako jsou (kooperativní) semafory. Hlavní výhodou frameworku je modularizace schopností agenta, což usnadňuje modelovat komplexní chování. Framework také zjednodušuje implementaci komunikačního rozhraní, což z něj dělá vhodný nástroj pro kooperativní ITS (C-ITS) simulace. V závěru práce je do simulátoru pomocí frameworku úspěšně integrován jednoduchý ITS systém s kooperativním semaforem, který poskytuje informace připojeným kooperativním pro dynamickou navigaci. Framework může výzkumníkům umožnit implementaci různých inteligentních dopravních systémů do softwaru simulátoru vozidla a zároveň zkracuje dobu vývoje a zvyšuje míru udržitelnosti softwaru.

Keywords

Intelligent transport systems, Multi-agent systems, Vehicle simulator, Cooperative ITS, Agent-based modeling, Unity.

Klíčová slova

Inteligentní dopravní systémy, Multi-agentní systémy, Vozidlový simulátor, Kooperativní ITS, modelování na bázi agentů, Unity.

Statutory Declaration

I, Jan Macek, solemnly and sincerely declare that:

I am the author of the master's thesis titled *Agent-based Modeling of ITS Systems in Vehicle Simulator.*

The research work presented in this thesis is the result of my own original investigations, except where otherwise acknowledged.

I have appropriately acknowledged and referenced all sources used in this thesis, including but not limited to published and unpublished works, data, and other materials, in accordance with the recognized academic standards and conventions.

I have clearly identified and cited the contributions of others to my research, including collaborators, advisors, and previous researchers, through appropriate acknowledgment and citation in the relevant sections of this thesis.

I have not engaged in any form of academic misconduct, including plagiarism, fabrication, falsification, or any other fraudulent practices, in the preparation or presentation of this thesis.

This thesis has not been submitted, in whole or in part, for the award of any other degree, diploma, or qualification.

I have adhered to the ethical guidelines and principles governing research involving human subjects, and I have obtained the necessary ethical clearances and permissions, where applicable, prior to conducting any research involving human participants.

I understand that any violation of the above declarations may result in severe consequences, including the cancellation of my degree and other disciplinary actions as deemed appropriate by the academic institution.

I make this solemn declaration conscientiously, believing it to be true and knowing that it is of the same legal force and effect as if made under oath.

Signed: [illegible]

Acknowledgement

I would like to express my sincere gratitude and appreciation to all those who have contributed to the completion of this master's thesis. This work would not have been possible without the support, guidance, and encouragement from numerous individuals and institutions, and for that, I am deeply indebted.

First and foremost, I extend my heartfelt thanks to my thesis supervisor, Ing. Dmitry Rozhdestvenskiy and Clas Rydergren, PhD, for their invaluable guidance, expertise, and support throughout this research journey. Their insightful feedback, constructive criticism, and dedication have been instrumental in shaping this thesis into its final form.

I am also indebted to the faculty members and the academic staff at the Czech Technical University in Prague and the Linköping University, who have contributed significantly to my intellectual development during my time here. Their dedication and commitment to teaching has broadened my understanding of the subject matter.

Last but not least, I would like to acknowledge the unwavering support and love of my family. Their encouragement, understanding, and belief in me have been my constant motivation throughout this academic endeavor.

Contents

List of Tables

List of Figures

1 Introduction

This thesis aims to investigate the possibilities and delimitations of multi-agent systems application(s) for Intelligent Transport Systems (ITS) modeling in an IVS. Simulating traffic behavior in a virtual environment, such as an Interactive Vehicle Simulator (IVS), is a complex problem, mainly because of its highly dynamic nature. Trying to accurately simulate traffic on a microscopic level, where individual road users (e.g. vehicles) are the elementary units in a system, requires complex modelling. All vehicles need to interact with each other and act upon other drivers' actions. Because agent-based simulations have proven to model complex behavior well [1], this modeling technique seems like a suitable solution for achieving a realistic traffic environment for IVS.

In the first two chapters, a review of the state-of-art research on Intelligent Transport Systems and Multi-agent Systems (MAS) are presented and important features related to the main topic of the thesis are discussed. This research leads to the definition of system requirements that are described in the third chapter. In the fourth section, a framework architecture is developed, in line with MAS paradigms for simulation of ITS solutions. In the fifth section, development tools and their delimitations for framework development are presented. The simulator software integration, additional development platforms and libraries facilitating the framework development process are discussed. 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 framework is conducted by implementing a distributed cooperative ITS system into the IVS, using the proposed framework and assessing its performance and overall results.

1.1 Introduction to the problem

In a world where vehicle transport plays an inseparable role in the society, with an ever-increasing traffic demand, it is important to analyze and study driver behavior and inherently interactions and relationships between drivers and their environment. Even though substantial advancements in autonomous driving are being made, for the near future, vehicles will be controlled by humans. Consequentially, this makes them exposed to a substantial danger. Data shows that about 95 % of traffic accidents are a result of human error [2]. Each traffic accident has got a tremendous effect on socioeconomic growth. A study by the European Union states that accident-related expenses (including the cost of fatal accidents) cost 1,8 % of the EU's GDP [3]. A rather non-cynical point of view is that each life lost is a failure in the society itself. Therefore, an effort should be made to eliminate fatal accidents.

A method that has proven to be effective at studying driver behavior and traffic safety is performing research through the usage of an interactive vehicle simulator (IVS), which allows to undertake experiments in a safe, controlled and reproducible environment [4]. Because the vehicle simulator is a digital twin of a real vehicle, it is desirable to make the interaction between the driver and IVS as close to reality as possible. Making such effort naturally improves the trustworthiness of acquired data from experiments and potentially also range of IVS applications. The IVS has got a broad spectrum of utilization. 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 vehicle-in-the-loop software, which enables testing real hardware in virtualized traffic [5].

1.2 Aim

The goal of this thesis is to investigate multi-agent systems (MAS) and evaluate possibility of application of these systems in the scope of Intelligent Transport Systems (ITS) simulation. The objective is to find out whether the shared characteristics of MAS & ITS, e.g. distribution of intelligence, make agent-based modeling a strong tool for simulating ITS solutions. The output of the experimental/practical part of the thesis is a MAS-based simulation framework for facilitating the process of ITS implementation into an IVS software with the benefit of modular and easy-to-extend functionality. The developed system should also be able to model state-of-the-art ITS systems that utilize communication between road users, such as C-ITS systems.

1.3 Research questions

Given the main goal of the thesis, which is to empirically investigate feasibility of ITS implementation into simulation software, the research questions are defined so that they help understand the scope of the thesis and expected outcomes.

- Can agent-based modeling be applied to simulate Intelligent Transport Systems in vehicle simulators and are they a viable option?
- How would an ITS simulation be implemented into an existing IVS using agent-based modelling?
- What are the available modules and libraries for implementing MAS in different programming languages?
- What are the advantages and challenges associated with using agent-based modeling to simulate Intelligent Transport Systems in vehicle simulators, and how can they be addressed?

1.4 Methodology

The main objective of the thesis is to create a framework for ITS implementation into a vehicle simulator software. In the first step, a literature review regarding research topics of the thesis is performed - Multi-agent Systems and Intelligent Transport Systems. The idea is to gain an insight into how to approach modeling systems using MAS. Secondly, research on the current state of Intelligent Transport Systems is conducted in order to identify system features and interactions that are important when attempting to create a base model for ITS implementation.

Based on the analyzed resources, requirements for the framework's functionality are defined. With the framework's requirements in consideration, implementation options and possibilities of external software usage are discussed. With the implementation scope defined, architecture of the simulation framework is presented, including component description and its internal structure. After the framework's structure defined, integration with simulator software is discussed. With the completed framework definition, implementation using a chosen software/programming language is done through definition of individual components and their interface.

The the end goal is to implement a module/framework for general ITS implementation in a vehicle simulator. The main objective of the framework development will be to create a framework that will be sufficiently modular for a wide variety of ITS solutions to implement. At the same time, it should be easy and straightforward to use. The framework is evaluated by implementing a sample ITS simulation, which helps to empirically evaluate the implementation process and the framework's capabilities and shortcomings.

1.5 Delimitations

Research delimitations refer to the scope of the study. The delimitations for a thesis on Agent-based modeling of Intelligent Transport Systems in vehicle simulator include mainly the technical scope. The framework needs to be compatible with the Unity game development engine which is used in the IVS laboratories of the Czech Technical University University in Prague (CTU). It should be well-integrated with the simulator software so that the development using the framework offers seamless interaction with other system components.

2 Intelligent Transport Systems

This section focuses on examining various existing ITS solutions and attempting to recognize their agent-based characteristics and principles in order to determine the possible application of MAS for simulation of ITS systems.

Intelligent transport systems describe an initiative to utilize modern technology to optimize and increase the 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 technology like machine learning, computer vision and IoT. Integrating such concepts with road users to improves traffic safety and increases traffic efficiency. It also addresses environmental issues by controlling traffic to prevent congestion or lessen their adverse effects.

Looking these solutions from a systemic point of view, they can, more or less, resemble multi-agent systems. Such approach answers whether a suitable application for agent-based modeling of ITS is feasible. The insights gained by reviewing the current state of ITS will serve as a base for further sections, where considerations regarding framework design and its validation are discussed.

ITS provides itself most useful in the following aspects of transportation: *Mobility*, *Safety*, *Environment* [6]. To provide an example, regarding mobility, one of the most common systems is routing and navigation for road users. These systems can factor in time, distance and even emissions when providing route guidance [7]. 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 accident occurs. In terms of the environment protection, ITS allows for demand management through *electronic fee collection*, which makes flexible charging for road usage possible, based on vehicle type and emissions category [8]. Another, more general example of emissions reduction of ITS is through reducing congestion, as road vehicle emissions are proven to pose a significant negative factor for the environment. Especially because they produce emissions in densely-populated areas, exposing a large population to serious health risks. 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 contribute to more than 2,200 premature deaths annually, costing the healthcare system at least $18 billion [9].

2.1 Simulator-based research

Using virtual simulation tools to research and evaluate the impacts of ITS solutions greatly reduces the overall cost of product development. It also offers for fast-paced and flexible testing 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 an IVS is to research driver behavior, the scope of the current state of ITS is focused on ITS solutions that both integrate automotive transport and offer some degree of driver engagement.

To give an example, some ITS use cases with high driver engagement are listed [6]:

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

These use cases shall be considered when modeling the ITS simulation framework in the practical part of the thesis.

2.2 Intelligence distribution

It is important to acknowledge whether the underlying system control is centrally governed or whether the intelligence is distributed across multiple actors, where individual components of the system can act semi- or fully independently. Such phenomena are important to consider because ITS solutions with centralized or distributed intelligence can be fundamentally different. This leads to different system models being optimal to use when simulating a particular ITS. Intelligent transport systems are, by nature, systems combining several actors to achieve a common goal. This, however, doesn't mean that it is optimal to model *every* ITS as a multi-agent system.

Historically speaking, ITSs that proved helpful for optimizing traffic were *centralized* [10], meaning there is a single, central component in charge of the decision-making logic. A centralized traffic system organizes other units/actors within the system, while interacting with the drivers in the traffic as external actors. The main advantage of this approach is that is is less complex, therefore it is easier to develop a resilient and safe product. The main drawback of this method is that it is heavily affected by the size of instances, increasing the complexity and often resulting in exceedingly large computation time, making the solution sub-optimal [10].

As with the *distributed* systems, the logic how to solve the main problem is systematically distributed to individual agents solving a sub-problem and cooperating to achieve optimal and predictable results. Such distributed intelligence is often modeled using *Multi-agent Systems*. This topic is further discussed in the section 3. Distributed systems can be used to solve traffic optimization problems on a larger scale. These systems have seen larger usage, especially in recent times, as newly produced vehicles come equipped with powerful computers. This is leveraged to spread out the computational burden from the core computational unit in centralized systems. A practical example of distributed intelligence solutions are the state-of-art communication protocols like DSRC and ITS-G5, which enable low-latency direct communication between vehicles.

One example of 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 control elements to optimize traffic flow and decrease travel time. In a recent research that reviews signal timing optimization approaches compared centralized and distributed network traffic control systems. The final conclusion is that although the centralized system is able to achieve better performance and higher global efficiency, the decentralized solution requires significantly less computation time (40 % in their experiment case) [11].

Other decentralized systems that are more exposed to the user (i.e. driver) include the *Adaptive Cruise Control* (ACC). ACC extends the usual cruise control systems that maintain vehicle speed by adapting to the speed of the vehicle in front. Recent development has led to further improvement, making ACC a cooperative system (CACC) that enables vehicles to adopt a *driving strategy* by communicating with the infrastructure (V2I communication).

2.2.1 Current research

In general, it is important to examine systems that are time-relevant and bring benefit to current as well as future IVS and Human-Machine Interface (HMI) research activities. Therefore, this sub-section focuses on reviewing ITSs that are an ongoing subject to research and current trends in the ITS industry, exploring state-of-the-art ITS solutions. To determine which ITSs are relevant, ongoing projects of governmental bodies and their strategies as well as the current trends in automotive ITS are reviewed.

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. Such organization would closely work with the industry, helping to create conditions for deployment of novel ITS technologies.

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, the connectivity industry as well as vehicle manufacturers [12].

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

Connected Cooperative & Automated Mobility - The computational power of newly-produced vehicles is increasing dramatically with every new generation, as well as the number of sensor data. ERTICO states that their focus is on utilizing the large amounts of real-life data to deepen the machine learning models, as well as building infrastructure that will allow handling this data. C-ITS (Cooperative Intelligent Transport Systems) is also a topic that ERTICO is invested in. C-ITS is 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 the creation of the 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 [13]. Below are the four main objectives in the area of Clean & Eco-Mobility [12].

- 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 congestion 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 developing a solution for connecting logistics information systems would optimize cargo flows and facilitate supply chain management.

In conclusion, the ERTICO organization helps to reduce the time to market of innovative, state-of-the-art technologies, increasing inter-operability between individual ITS by promoting an open framework for the 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) refer 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 [14].

The concept of C-ITS was developed by the European Commission, representatives of industry and authorities in the European Union. In 2016, the bodies agreed on a coordinated establishment of intelligent transport systems in Europe. It is considered to be one of the 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 were 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 in how a MAS is described. Regarding the topic of this thesis, this could pose an argument to implement one of the C-ITS systems in the practical part of this thesis.

Vehicles and infrastructure equipped with C-ITS can, for example, communicate a traffic-related warning to each other. Upon receiving such warning, the drivers are informed about the upcoming traffic situation in time for them to take the necessary actions to avoid potential harm. Other potential benefits of the use of C-ITS include reduced congestion and improved driver comfort. Vehicles share data directly with each other (V2V) and with the infrastructure (V2I) using ad-hoc short-range telecommunication. The two types of communication 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 table 1 below.

In general, traffic safety and traffic flow improvements can be grouped based on which operational tasks they serve [15]:

- 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.

Project C-Roads The C-Roads project was established by the EU member states and road operators to study the effects and refine the future deployment process of C-ITS. The project's main objective is to harmonize C-ITS deployment activities across Europe. Within the C-Roads project, 6 work groups (WG) were established, 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
- WG2: Coordination and cooperation of R&I activities

General use case	**Implementation example**
Signalized intersections	Green Light Optimal Speed Advisory Traffic Light Prioritisation Signal Phase and Timing Information Imminent Signal Violation Warning Emergency Vehicle Priority
In-Vehicle Signage	Dynamic Speed Limit Information Dynamic Lane Management Other Signage Information
Probe Vehicle Data	Vehicle Data Collection Event Data Collection
Hazardous Location Notification	Accident Zone Traffic jam Ahead Stationary vehicle Weather condition warning Emergency vehicle approaching
Road works warning	Lane closure Road closure Road works - mobile

Table 1: C-ITS use cases [14]

- 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 because the focus of this group was, in the first place, regarding the infrastructure support for autonomous vehicles[1]. One of the objectives was combining relevant physical and digital infrastructure, assessing the relevance between them and automated vehicles, and mapping infrastructure elements to use-cases as *Generic Driving Tasks* (GDTs):

- Sensing & Perception
  - Ego localization
  - Environmental awareness (object classification and incident detection)
  - Enhanced perception (for limited visibility scenarios)
- Planning
  - (Dynamic) information and regulations

[1]Autonomous vehicles of Level 4 defined by the Society of Automotive Engineers

  - Safe and appropriate navigation plans
  - Cooperative planning
- Actuation
  - Motion Control
  - Minimum Risk Manoeuvre

It is therefore advisable to keep these GDTs 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 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 2) that links the aforementioned GDTs to existing, relevant ITS solutions. This should clear up and simplify the process of defining requirements for the proposed system, as the new framework requirements could be easily adopted from existing system documentation of the particular mapped ITS.

Table 2: 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	Vulnerable Road User
CACC	Cooperative Adaptive Cruise Control

2.3 Conclusion

In this section, Intelligent Transport Systems are introduced. ITS is an important part of traffic engineering, utilizing traffic data and mathematical modeling to reduce congestion, improve traffic safety and reduce emissions. The main focus of this section is to discuss important features of Intelligent Transport Systems that are related to the topic of this thesis. The relationship between the driver and ITS system and intelligence distribution across ITS actors is discussed. ITS projects relevant to the thesis topic are investigated, including the research and developments efforts on the EU level. The results suggested that C-ITS systems are a hot topic in current research. Furthermore, numerous projects (e.g. C-Roads) have been built and tested, paving the way for the future of interconnected vehicle mobility.

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 concisely 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 act independently, possess knowledge and communicate with each other (i.e. share knowledge). The agents are 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 it should be used to model a system that is not centralized but rather distributed, where autonomous, intelligent units perform actions independently. Multi-agent systems have gained popularity in recent years, as they offer high flexibility in modeling highly non-linear systems. They also offer abstraction levels that make it more natural to deal with scale and complexity in these systems [18]. MAS excels in modeling social environments involving humans, as MAS share a lot of features with human societies. 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 through communication, cooperation or competition, while working towards some goal. Numerous real-life examples strongly resemble this MAS definition, for example, sports teams, where each player has his role, like a defender and striker. Although all players have the same intention (i.e. win the game), their specific actions 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 a more practical perspective, the MAS paradigm can be used when building complex computer networks.

3.1 Agents

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

Situatedness - Agents directly interact with the environment through sensors and actuators.

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

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

Responsiveness - An agent is able to respond to a perceived condition of the environment in a timely fashion.

Social behaviour - An 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 differ between use cases. To standardize MAS development, several architectures that describe how agents work have been proposed. Generally, [19] defines three classes of agent modeling architectures 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 is given in the sections below.

3.2.1 Reactive agent architectures

Having first emerged in behaviorist psychology, the foundations of the reactive agent concept are that an agent makes its decisions based on limited information, with simple situation-action rules. The agent usually makes decisions directly based on input from its sensors. This type of agent is mostly suited for applications where agent resilience and robustness is the most important factor, instead of optimal behavior [19]. An example of a reactive agent architecture is the **Subsumption architecture**.

This architecture, described by Rodney Brooks [20], decomposes the agent into hierarchical levels that operate in a bottom-up fashion. In other words, the bottom layers that control elementary behavior are activated by the upper layers that define more complex actions or define goals for the agent. The behavioral modules map sensations directly to actions and can only define what the agent does, therefore the agents is not able to change its desires. An example of a subsumption architecture is depicted in fig. 1 with example modules from each hierarchical level, where the bottom-level module is `Avoid Objects`. This action is needed in order to complete the `Wander Around` action, which is consequently induced by the general goal on the top layer - `Explore World`.

3.2.2 Deliberative agent architectures

Compared to a reactive agent, deliberative agents have a more complex structure and are closer to human-like, rational behavior. A key feature of a deliberative agent is that possesses a symbolic model of the world in which decisions are made via symbolic reasoning [21]. In other words, the agent maintains its internal representation of the

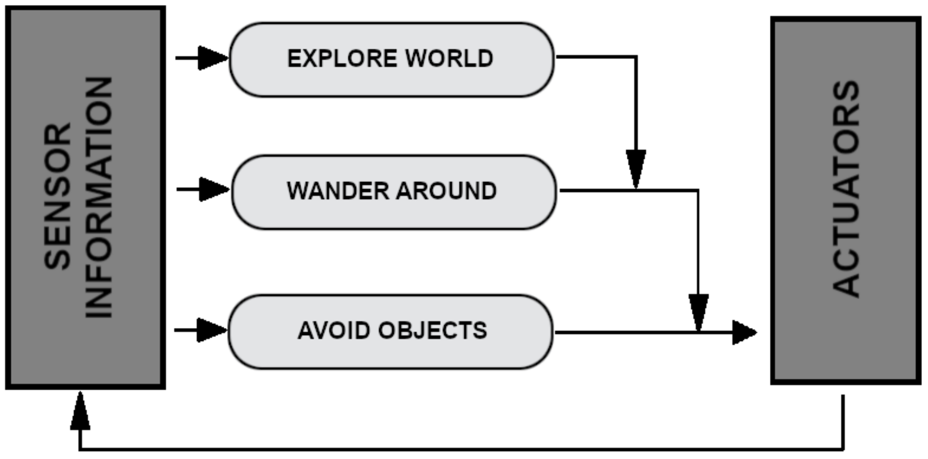

Figure 1: Example of a Subsumption architecture

external environment and thus is capable to plan its actions, while being in an explicit mental state which can dynamically change. The **The Beliefs, Desires and Intentions (BDI)** architecture is the most widely known modeling approach of deliberative agents.

The BDI paradigm has been used in various applications, such as simulating the impacts of climate change on agricultural land use and production [22] or improving internet network resilience by creating BDI agents that combat 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 the agent's perception.

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

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. 2), which drives the agent's (deliberative) behavior.

This definition can be made clearer with a simple example scenario - a waiter in a restaurant. The waiter's *beliefs* are the tables with customers and information about the state of each table (i.e. choosing the menu, waiting for a meal, willing to pay etc.). The waiter's *desires* are to serve customers, for example, accept orders from a customer. The waiter carries out his desires by making a plan of *intentions* (e.g. go to a table and ask if the customer

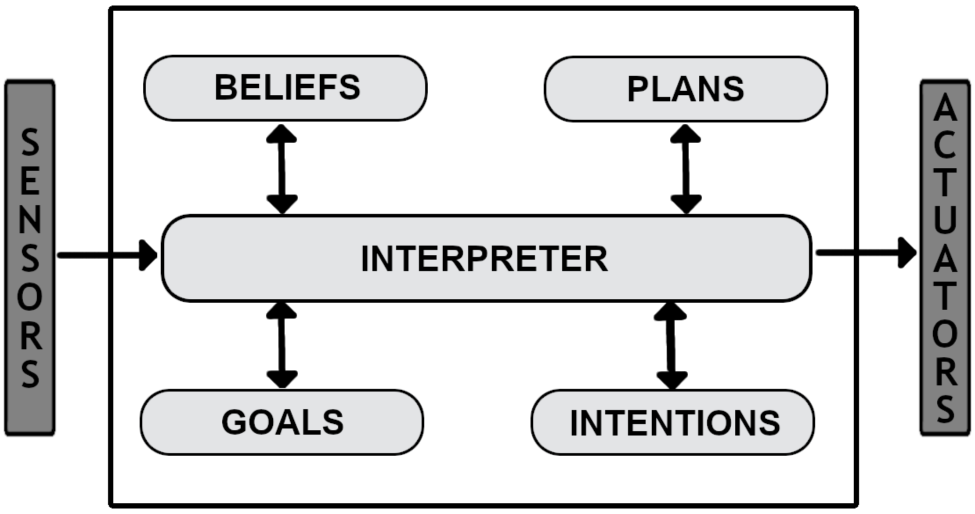

Figure 2: The BDI architecture schematic

wants a drink).

The advantage of this architecture is 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 plan commitment, there is a risk of agent overcommitment, where an agent might be trying to achieve a goal that is no 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 modeling. Purely reactive agents might lack the ability to solve complex tasks, whereas deliberative architectures are challenging to successfully implement on a concrete problem [19]. Pre-compiling a plan library for every possible scenario that can happen in an environment with a vast amount of complexities is simply not feasible, due to uncertainties of environment state changes when agents interact with the environment.

The underlying concept of hybrid architectures is to structure agent functionalities into layers that interact with each other. The most important advantage this provides is *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 [25].

The majority of hybrid architectures have a *controller* layer to handles reactive tasks. This layer is also connected to the sensor readings, and is hierarchically on the lowest level. Then there is a *planning* layer that handles logic-based, deliberative tasks and often interacts 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 hybrid architecture is the $_3$T architecture, whose abstract model can be seen in the figure 3 below.

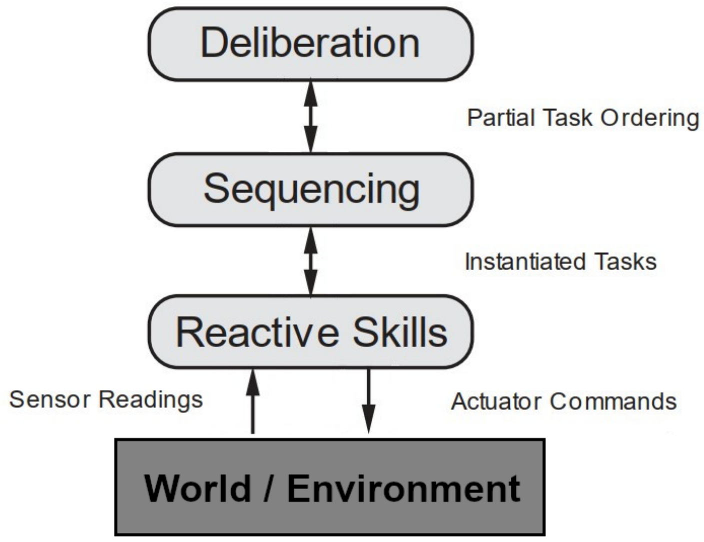

Figure 3: An architecture model of a $_3$T hybrid architecture

$_3$T architecture
This architecture builds upon a predecessor architecture called Reactive Action Packages (RAPs) [26]. A RAP is essentially a process or a description of how to complete a task using discrete steps. 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 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.

3.3 Interaction between agents

An agent-based ITS system requires some form of agent interaction and thus communication between individual agents. To make agents interact with each other, an agents needs to be extended with an interface dedicated to communication, which allows it to communicate directly and cooperate in a decentralized fashion. The concept of cooperation between agents is important in the ITS modeling context, as these systems facilitate sharing of information between drivers and also from the road traffic environment to drivers. Therefore, interaction is an important component when considering modeling ITS and road traffic in general. However, this interface also adds to the system's complexity.

Most agents achieve their common goal through cooperation, where some form of forced altruism is given to the agents. Furthermore, situations can happen where conflicts of interest between agents occur with no clear positive outcome, i.e. zero-sum games [17].

One should adhere to some communication principles to facilitate cooperation between agents. As such, agents should communicate according to Gricean maxims (see the table 3) [1]. This way, the communication and performance overhead is minimized and the system is more stable.

Table 3: 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

The agents developed for an ITS simulation should make an effort to cooperate rather than to compete, with the desired state of the system being rated "better" the more the agents cooperate. For example, when designing a cooperative intersections system, the reward function for the individual intersections (i.e. agents) should take into account not only the throughput on their intersection but also the throughput of the whole system. However, that doesn't eliminate the need to negotiate. Let's 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 the main objective would be to achieve an optimum that minimizes the overall cost.

3.3.1 Organizational decomposition

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

Hierarchical organization

In hierarchical organization [27], 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 the lower level of the 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 operated 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. To illustrate, a failure of a single agent at a high hierarchical level causes the whole system to fail. In a uniform hierarchy, the authority is more distributed among the agents. This makes the system more fault tolerant and performs graceful degradation in scenarios where one or more parts of the system fail [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 within the group [17]. After the goal is reached or the coalition is no longer feasible, the coalition ceases to exist. The coalition can have a flat hierarchy, with the possibility to have one agent as having the leader role for external interactions. 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 an organizational point of view, such as *teams* and *holons*. Though, the two mentioned approaches are the most well-suited for an ITS system development, supported by usages in other ITS-related research (e.g. [28] and [29]).

In conclusion, the usage of the aforementioned organizations 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 interest are 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

Agent communication is a crucial component of a multi-agent system. Communication improves inter-agent interaction, beyond cues that an agent receives from its sensors, as agents can either exchange or broadcast information that could not be obtainable for the agent otherwise. This allows for deeper and more complex decision-making and behavior 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 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. A language developed specifically for inter-agent communication has been created, called *Agent Communication Language* (ACL) [30]. Its 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 modeled 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 4.

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 5.

Apart from the mentioned communicative acts, the ACL standard also defines message parameters, which form the structure of a message (table 6).

One needs to consider 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 [31]. Also, [31] 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.

3.4 Conclusion

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

Choosing the optimal agent architecture depends depend on the scope of agent goals, environment and action space definition. Concerning 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 solutions 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 an example from the ITS domain. Systems such as autonomous driving algorithms require high resilience, determinism, and high performance, all in extremely dynamic conditions. This suggests using 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 behavior requires complex logic and frequent re-planning, therefore more suited for *deliberative* architectures. Furthermore, when considering the aforementioned cooperative ITS, which give emphasis on information sharing and environmental awareness, it becomes clear that *hybrid* architectures that offer both planning and reactive behavior 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 is used.

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

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 are discussed, and a suitable framework for the thesis' use case of setting up communication between agents is investigated and chosen. The framework of choice was the Agent Communication Language, because its set of communication primitives already assumes the MAS-based use cases, with predefined requirements, message parameters and message types while also being open for extension.

Table 4: Categories of communicative acts [30]

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 5: The ACL-compliant agent requirements [30]

Requirement 1: Agents should send not-understood if they receive a message that they do not recognize or they are unable to process the content of the message. Agents must be prepared to receive and properly handle a not-understood message.
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 6: Pre-defined message parameters [30]

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 that 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 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 that denotes the protocol which the sending agent is employing.
`:conversation-id`	Introduces an expression that is used to identify an ongoing sequence of communicative acts that together form a conversation.

4 Requirements definition

This section is mainly devoted to defining what should the proposed system be capable of in terms of functionality. The previous sections are devoted to discussion about the main ITS simulation aspects of the proposed framework. This section expands on it, going into more details and processes on how to achieve a system that is both modular enough to support a wide range of ITS solution simulation and offers enough facilitation value for streamlining the development of ITS simulations at the same time.

Another outcome of this section is to determine which toolset to use to facilitate the building of the framework. Ideally, an existing library for agent-based modeling should be used, which has got its structure in line with the proposed specifications in this section.

Furthermore, regarding the topic of this thesis, the inductive approach of analyzing actors and processes of various ITS solutions is then used to validate the proposed framework. The framework is designed using the obtained knowledge in the literature review researched in the previous section 3. Consequentially, an ITS solution is 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

Defining requirements ensures that functionality identified in both the inductive system decomposition in section 2, and the deductive approach in section 3 are met. This results in an ITS virtual-deployment framework that is highly modular and capable of simulating a large number of ITS. The requirements are summarized in the table 7.

Table 7: Proposed system requirements

Requirement 1: The framework shall be a multi-agent based system, supporting more actors that can act independently, have their logic and can make decisions based on their own perception of the environment.
Requirement 2: The framework shall support modular architecture by employing a layer-based architecture model
Requirement 3: The implementation of a particular ITS agent's elementary capability shall be done using skill modules, each of them serving one purpose. The skill modules must provide a pre-defined input and output interface.
Requirement 4: An agent shall be able to communicate with others through a dedicated communication module/skill that will manage the communication.
Requirement 5: Agents shall 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 6: Sensory and communication capability of an agent shall be incorporated as a pre-implemented module.
Requirement 7: Where applicable, agents shall be able to detect conflicting intentions with other agents while executing their tasks.
Requirement 8: Agents shall resolve conflicts using a standardized communication specification, and comply with its requirements, mentioned in section 3.3.3. An interface to support basic negotiations shall be provided.
Requirement 9: The framework's architecture shall be modular enough to support a broad spectrum of ITS, including but not exclusive to C-ITS solutions.
Requirement 10: The framework shall 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 11: The supported communication modes shall be both direct (messaging specific agents) and indirect (broadcast & subscription).

5 Implementation toolset

With the finished system specification, this section discusses which tools will be used to build the software framework. First, it is important to analyze the IVS simulation system/software (i.e. the super-system) that the proposed framework is incorporated into. It should 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 of a programming language and a subsequent agent-based modeling library to use.

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

5.1 Simulator software

The particular simulator software that is being used at the CTU's HMI research laboratories 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 [32]. Unity has also been used as a tool in physical product modeling, AI & machine learning and digital twins, used in many industries [33]. The main strengths of Unity are multi-platform development support, virtual reality development support, good community support (including its asset store) and detailed documentation. The game engine has got its development environment, which can be seen in figure 4.

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 should be a noticeable benefit when the chosen agent-development platform used to develop the proposed system is also written in the C# language (same as the Unity scripting API) to achieve maximum customization and interoperability.

The scripting follows the usual object-oriented programming paradigm, where one script contains a behavior encapsulated in a class. Each script can contain a reference to another script/class. Each object in the scene (referred to as a game object) can be assigned an arbitrary number of scripts that upon assigning gain access to the 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 a game object. This object-centered development suggests that applying agent-based behavior could be well integrated into the simulator software and offer needed flexibility and cross-integration.

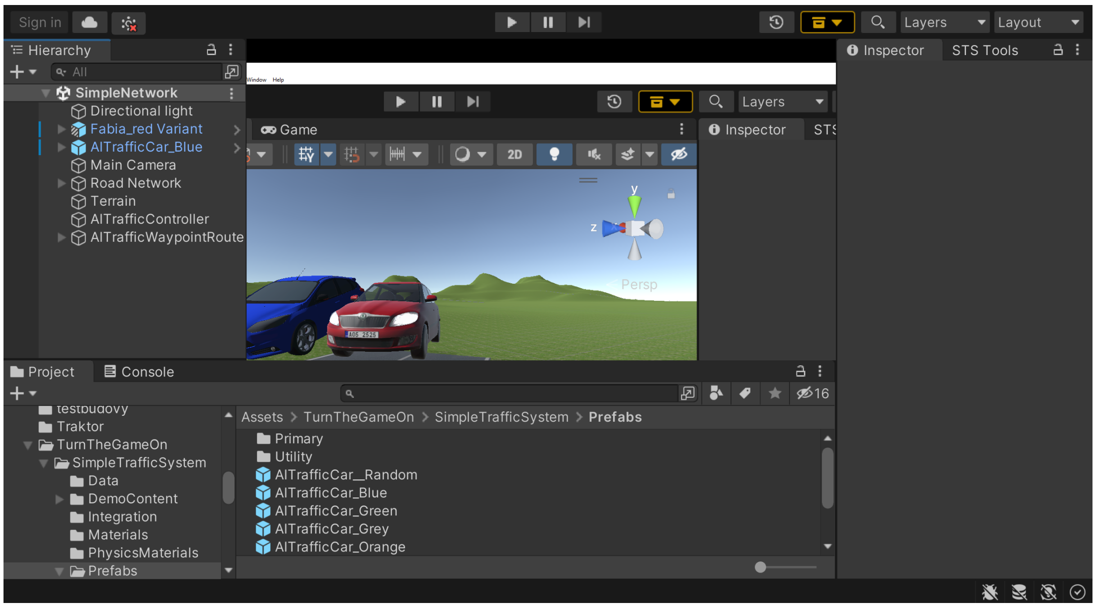

Figure 4: The Unity development platform UI

5.2 Agent Development Platforms

Having a development tool that is dedicated to 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 [34] can be found below.

JADE

Jade is a free software developed under a grant from the European Commission. The main advantage of the platform is the containerization of agents, which allows to distribute 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 with 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 Java.

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 in the 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 on 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 building 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 [35]. The framework's last release was in the year 2021, making it 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 three main components that form the basic building blocks of a MAS environment - `Agent`, `EnvironmentMas` and `Container`. The class diagram of the components can be seen in the figure 5 below.

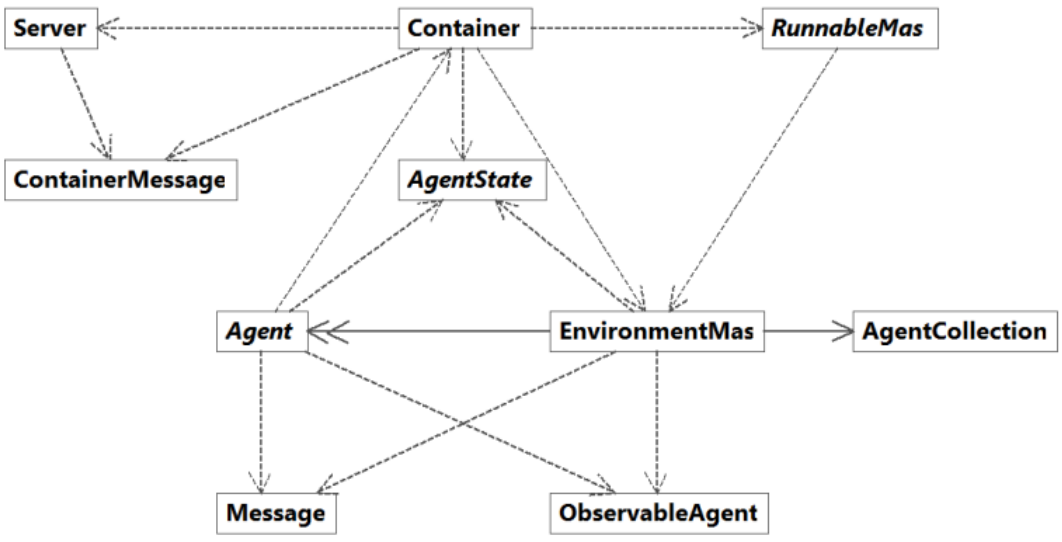

Figure 5: ActressMAS - Class diagram of framework's main components [35]

The agent environment can be distributed between more machines. Each machine runs a container that communicates with a server, as seen in fig. 5. The service handling communication between containers 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. 6.

There are, however, some issues related to the use case of the proposed framework. Overall, the implementation of the classes creating the local agent environment doesn't contain

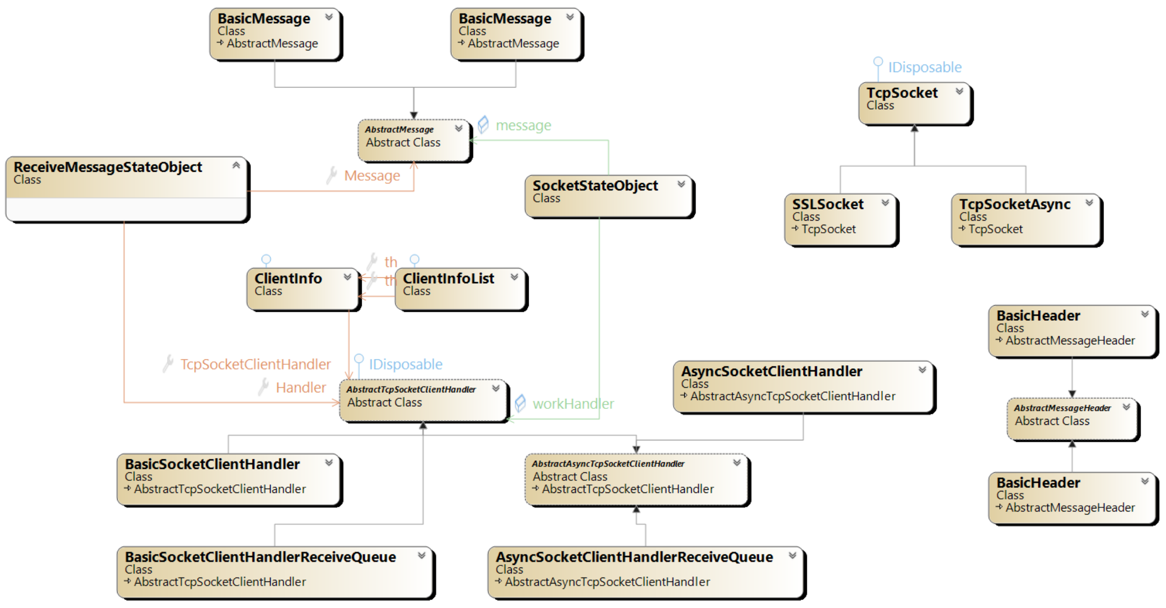

Figure 6: ActressMAS - Class diagram of distribution service [35]

much (relevant) 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 behavior implementation also means that the agent architecture has to be built from the ground up either way, not bringing any benefit of using this framework. On a second note, agents only 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 behavior 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".

5.3 Conclusion

In this section, the technical aspects of implementation requirements are discussed. The simulator engine is introduced, describing the aspects of programming integration. The main aspects of integration are discussed, which should enhance the subsequent decision about which MAS development platform to use for the proposed system implementation. Consequently, MAS development platforms suggested by the literature are examined. However, their review concludes not to use them as they are outdated and would not offer an advantage concerning integration with simulator software. Their focus is to integrate with distributed computer networks focusing on different problems. Lastly, a promising agent-developed framework called ActressMAS is introduced, whose main advantage is being .NET/C# 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 revolves around deployment across multiple machines and offered minimal abstraction of the agent framework itself, i.e. agent behavior and interaction. Therefore, building the framework independent from a pre-existing MAS platform offers maximum customization according to the system specification and its requirements.

6 Proposed framework

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

6.1 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* (section 3.2.3), which will offer sufficient flexibility. Such modeling flexibility is needed primarily because there won't be a single, particular 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 ensures that there is an optimal balance between robust, reactive behavior without giving up capabilities to model complex behavior.

The architecture is designed with a formal assumption that the implementation will be realized using the 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 a pre-defined interface, corresponds to a large degree to the concepts of OOP, especially the encapsulation principle. Secondly due to the existing IVS software being developed in an OOP way.

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

6.1.1 The architecture layers

In this section, the individual layers of the architecture are defined. The layers' definition adheres to the characteristics of the $_3$T architecture. For each layer, its purpose and relation to other layers is described, together with technical implementation guidelines, such as configuration, operation modes, inter-layer interface and internal processes within the individual layers.

Reactive skills layer

This layer encapsulates all the *skills* an agent can 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 the case of a vehicle-based agent. Skills can be activated and deactivated based on the cognitive capabilities of the 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 the 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 the impacts of task sequencing without being overly concerned by how the individual skills interact with the environment. The original article proposing the architecture outlines the following canonical approach for skill specification [36]:

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 allowed 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 allowed 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 is an individual script that runs *asynchronously.* Each script implements a *activation*, *update* and *deactivation* function. The script is used to define functionality of each particular skill with outputs from other pre-selected skills optionally available and consequentially to define its own outputs. Inside the script, it is possible to interact with the skills inputs and transform them into outputs. The primary use case of this functionality is routing output from sensors or communication skills to (multiple) other skills that could utilize the information. Skills will provide an interface to the higher abstraction layer through pre-defined events specific to each skill.

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

[2]The original author of the architecture refers to them as Reactive Action Packages (RAPs) [26].

Sequencing layer

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

This layer features logic that is 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) is 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 is to provide interface to control execution of particular skills.

To formalize the purpose of this layer in the proposed system, it should:

- Contain definitions of which skills to enable in order to complete a rudimentary task.
- Managing execution of sequences of common skills (i.e. tasks).
- Implement logic that serves to adapt to completing the tasks in different conditions.

The layer interacts 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 the execution of tasks to achieve a specified goal. A goal could be something that the user wants the agent to accomplish, for example, a vehicle's goal is to arrive at 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 [36]. In addition to this, it is possible to add parameters to the goal definition. The final version of the plan is then generated based on parameter values and internal logic that processes them.

In practice, the most trivial specification of the deliberation layer consists of:

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

b) **Fail-safe goal** - The desired outcome of the agent's behavior in case an unexpected failure occurs in any of its components or when the agent fails to achieve the primary goal. For example, performing a Minimum Risk Maneuvre in vehicle-based agents [37].

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

To sum it up, a detailed view of the individual components of the architecture and their interface is on the figure 7.

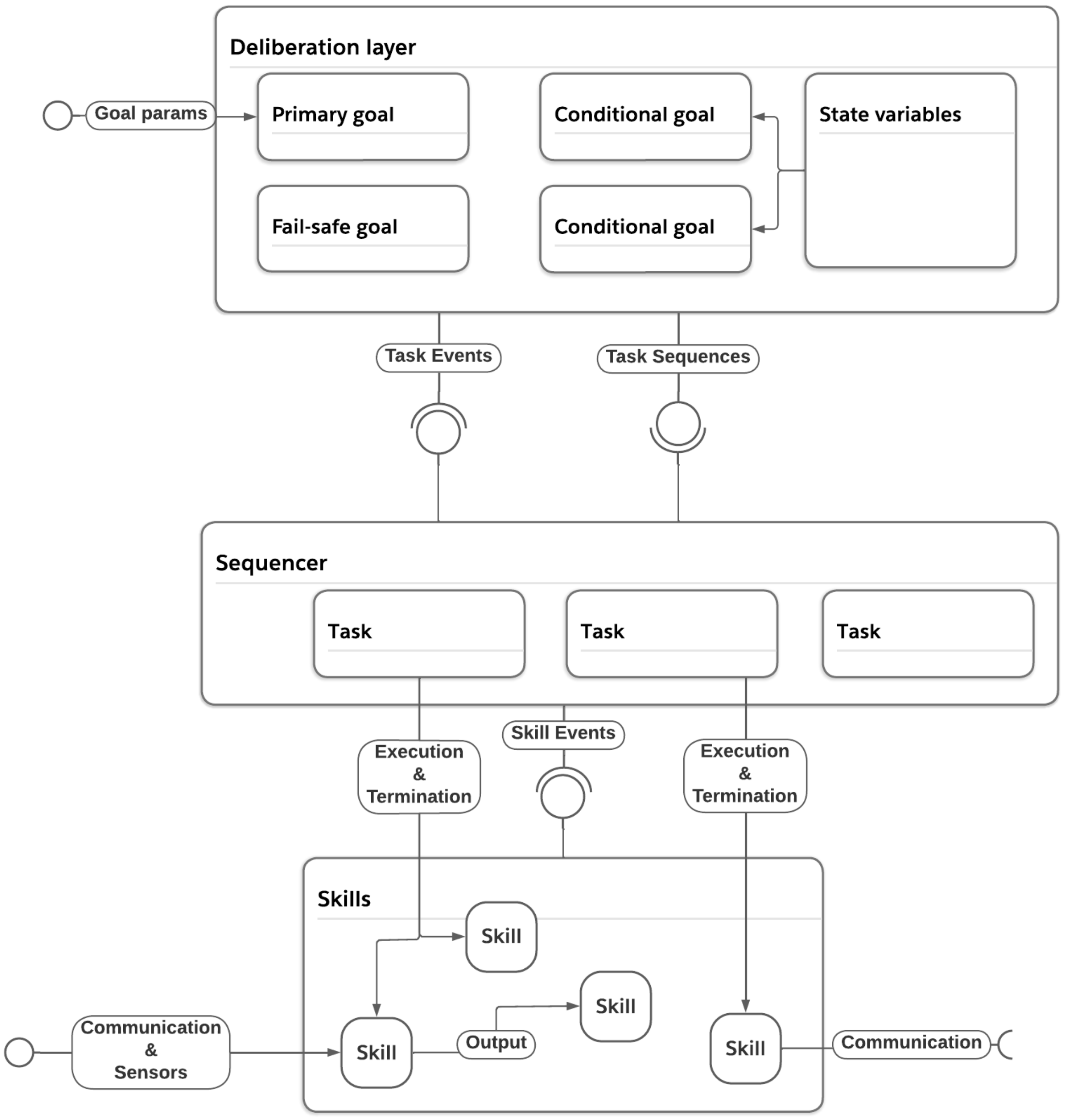

Figure 7: An overview of the proposed 3T architecture

6.2 Macro architecture

6.2.1 Communication protocol

As has been discussed in the previous sections, communication plays an important part in MAS, mainly because it vastly extends the capabilities of interaction between agents and thus the ability to model more complex behavior. 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 is presented below.

6.2.2 Broadcasting

Message broadcasting is the first type of communication that will be implemented, due to the fact that ITS uses information broadcasting in a lot of cases. Generally, there are mostly two types of service messaging in ITS [38]:

- **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 a 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.

6.2.3 Implementation

The ability to broadcast and subscribe to a broadcast should be handled by a separate skill component. Upon skill initialization, a communication skill connects to a communication channel and listen to or send messages. Each communication stream has a specified message format so that agents can be set up to process the broadcasted information. The agents are then able to react to received messages by sending an internal event message to the sequencer.

Given the great utility of the C-ITS system and identified similar characteristics and use-cases with agent-based systems (discussed in section 2.2.1), the framework features 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).

6.2.4 ETSI message services

Due to the wide range of beneficial use cases for the aforementioned types of 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 are described along with the implementation details below.

Cooperative Awareness Message

The CAM message is specified by the Cooperative Awareness Basic Service. By definition, CAM provides a means of periodic status data sending, sharing cooperative awareness to neighboring nodes. From the message propagation point of view, the CAM differs from the DENM message by only allowing *single-hop transmission*. This way, only vehicles in 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 [39]. For example, CAM management can detect a slow vehicle warning by analyzing received CAM messages from a particular vehicle. In other words, the receiver is expected to evaluate the relevance of the contained information. The CAM service has its message format specification, which is found in figure 8.

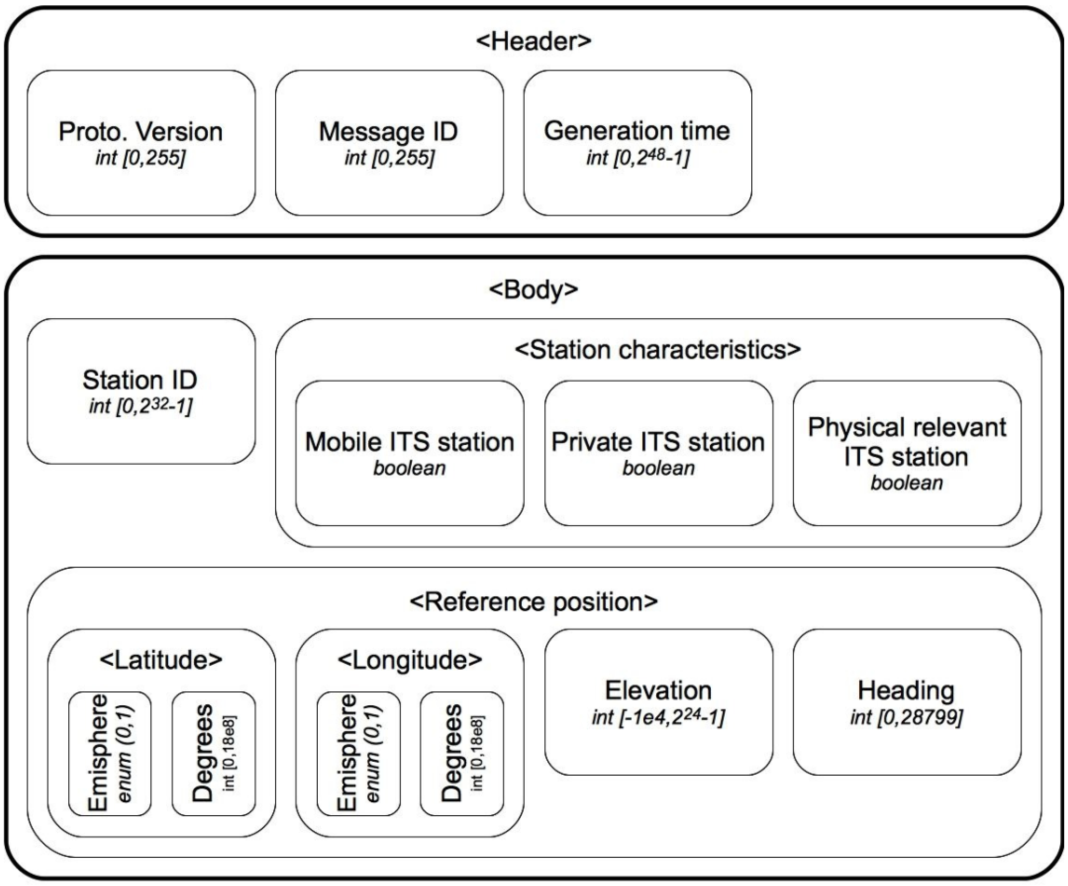
<Header>
Proto. Version int [0,255]
Message ID int [0,255]
Generation time int [0,2^48-1]
<Body>
Station ID int [0,2^32-1]
<Station characteristics>
Mobile ITS station boolean
Private ITS station boolean
Physical relevant ITS station boolean
<Reference position>
<Latitude>
Emisphere enum (0,1)
Degrees int [0,18e8]
<Longitude>
Emisphere enum (0,1)
Degrees int [0,18e8]
Elevation int [-1e4,2^24-1]
Heading int [0,28799]

Figure 8: The CAM message specification [38]

Decentralized Environmental Notification Message

Instead of vehicle/station status updates, 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 [40] and whether to take further actions like displaying the message to the driver. The DENM service message specification is presented in the figure 9.

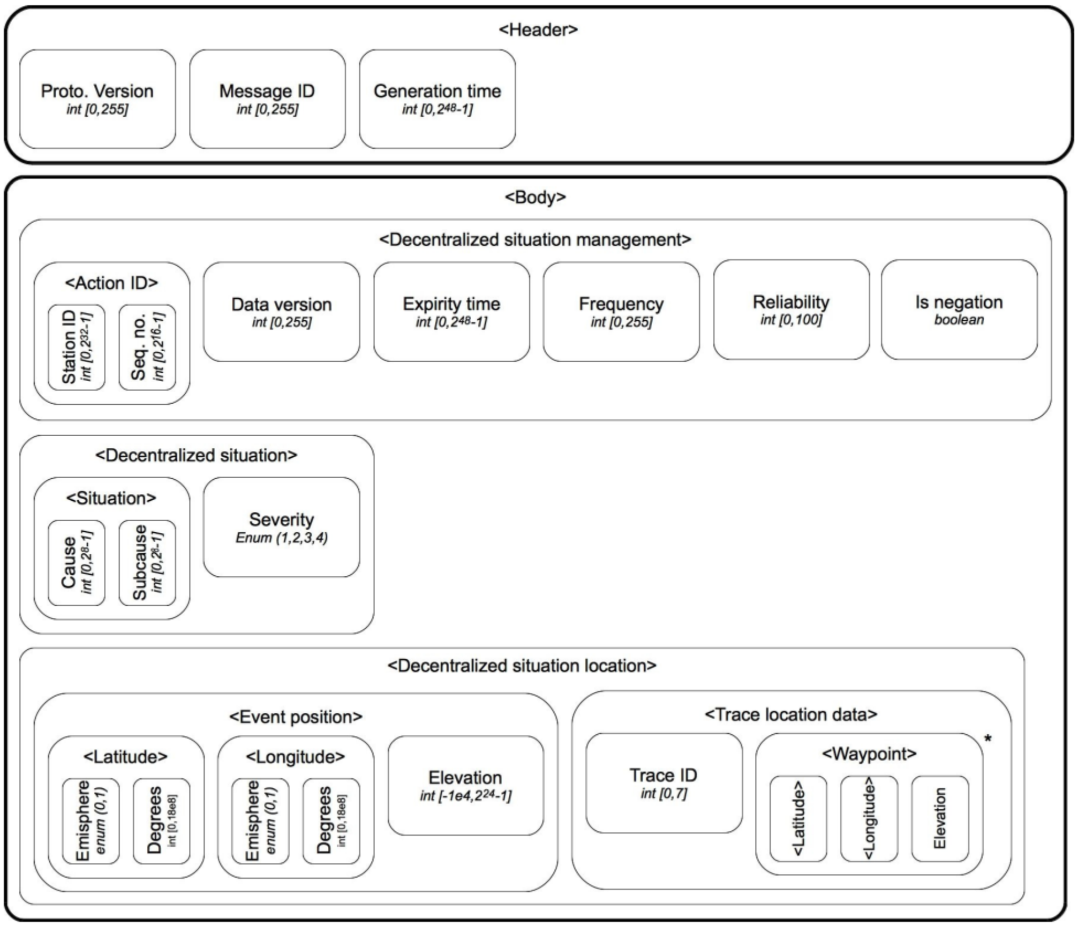

Figure 9: The DENM message specification [38]

6.2.5 Negotiation and Agreement

Negotiation between agents is important especially when agents (mutually) prevent other agents from completion of their goals. In other words, the best course of action for one agent is not necessarily the best for the other. This is more than likely to happen in environments where multiple agents share the same resource and 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 [34].

Such type of negotiation 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 [34]. This is inherently contrary to the statement made in the section 3.3, where it was suggested that agent cooperation should be emphasized. As agents are formally modeled as self-interested, it cannot be guaranteed that they won't encounter conflicts with each other. Therefore, modeling cooperation before figuring out conflict resolution would inherently result in a more fragile system.

6.2.6 Conflict detection

As has been stated above, because of the 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 behavior possibilities.

The first step is to have a way for an agent to determine if a conflict has happened. An agent is not able to resolve s conflict if it doesn'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 is handled by a dedicated skill module. If an agent is expected to run into a conflict with another agent (of the same type), it has to be equipped with a dedicated skill that behaves as a sensor detecting conflict with the implementation left to a specific use case. In other words, the logic for conflict detection is left to define together will specific agent and its properties, and other modules.

When one agent detects a conflict through one of its modules, it required to inform other agents involved about the conflict so that all involved agents are aware of the situation.

6.2.7 Conflict resolution

Now that the conflict detection process is defined, in that case, it is possible to build on it to define how conflicts are resolved. Considering the findings in section 3.3.1, the simplest tool to resolve conflicts is to define a hierarchy in the agent. In order to avoid more complex mathematical reasoning models [34], hierarchy, along with agent-bound state variables can be used as a bidding resource. Whichever agent can bid the highest value gets priority in conflict resolution (e.g. by being able to act in its interests).

To facilitate the communication related to conflict resolution & bidding, the FIPA message standard is 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 assume that the counterparties in conflict are truthful and only bid resources that are available to them. Also, deciding which bid proposal has won is handled by the agent that had called for the proposal, so the counterparties assume that the deciding agent is truthful and decides correctly according to the bids.

To better illustrate the negotiation process, an example scenario of the negotiation process is described in the *Example* part below.

Example

To illustrate how the system would work in practice a sample scenario is presented below in figure 10, where the agents are drivers/vehicles. The scenario demonstrates how agents achieve their goals using the three layers, as well as conflict detection and resolution.

There are two vehicle agents (`A`, `B`) in the scenario, which both have three routine tasks defined: `drive`, `avoid:object` and `wait` and `negotiate`. Both vehicle agents have a goal not to crash and keep a certain speed. To ensure a fail-safe behavior, a logic defined on the deliberation level is implemented to attempt to avoid objects in case the driving task fails. Furthermore, if the avoid object task fails due to another vehicle being detected, the negotiation task is executed. The execution of this task on one agent should trigger the execution of such task on the other agent in conflict as well. Based on the result of the negotiation, the agents' respective deliberation layers enqueue either the `avoid:object` or `wait` task and enqueue the basic `drive` skill to come after the current task completion.

1. Both agents do not detect any obstacles in their surroundings, so the planner initializes a plan with only the `drive` task sequenced, which gets executed.
2. Agent `A` spots an obstacle in his way (an oil spill). This causes the `drive` task to fail and `A`'s planner has to re-plan to reach its goal (not crash). The failed task's fail-safe task is triggered (`avoid:object`) and added to the new fail-safe plan. The new plan is initialized with the two following tasks sequenced: `avoid:oil_spill` → `drive`.
3. However, this plan also fails, as agent `B` detects a path conflict, consequentially informing agent `A` about it. They now have to initiate a negotiation task to reach an agreement and resolve the conflict using auction-like bidding. The agent `B` wins the bid, due to being in the right lane and so agent `A` is forced to give way. The planner creates a new plan: `wait` → `avoid:oil_spill` → `drive`.
4. The individual tasks finish successfully and the agents' goal is fulfilled.

6.3 Conclusion

In the preceding sections, the multi-agent system framework was proposed. This framework is used to implement various suitable ITSs into an interactive vehicle simulator. The system was described both on a micro and macro level. The micro level mainly addressed the specification of the inner architecture of individual agents, and how they achieve their goals using their 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.

Afterward, the macro architecture was specified, addressing mainly the ways how agents

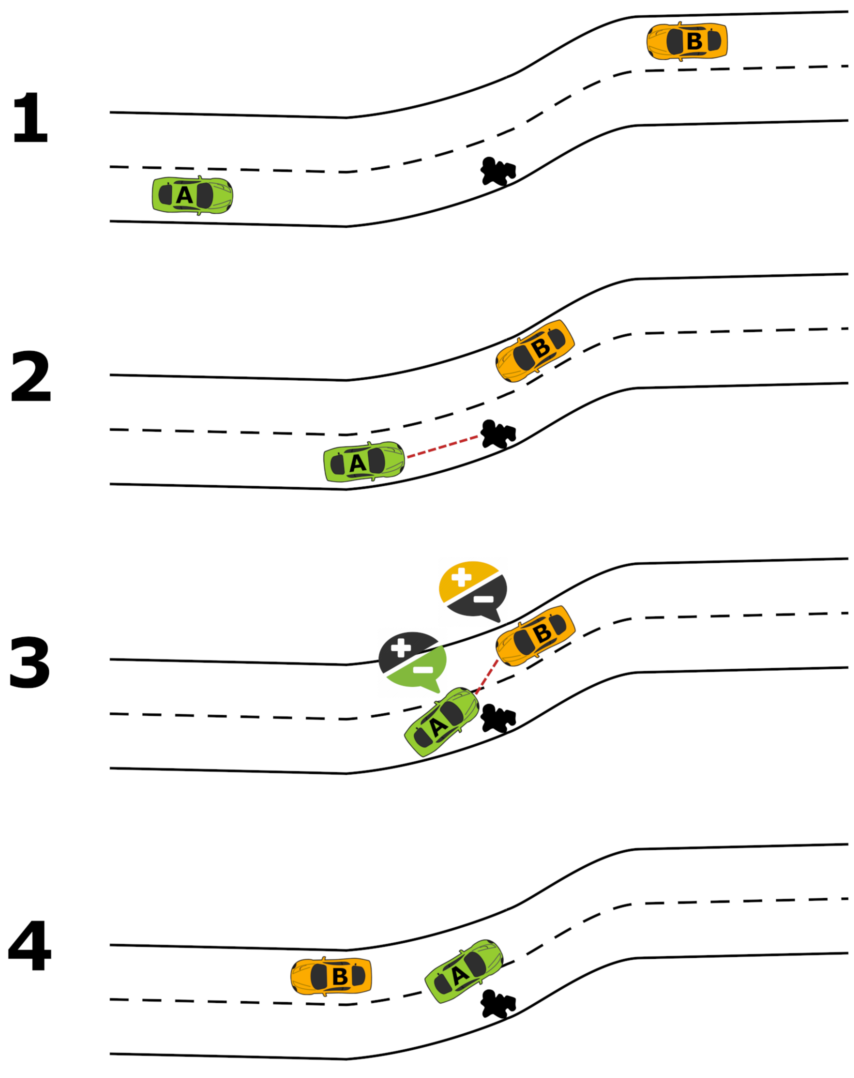

Figure 10: An example behavior of vehicle-based agents using the proposed architecture

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

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

7 System implementation

Now that the system architecture model has been designed and the technical requirements and tools considered, it is possible to create the system implementation. In this section, 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 the 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 behavior pre-defined but require the user to specify given behavior that is exclusive to their use case. This allows for a framework that has got the generic behavior already completed but gives the user freedom and guidance concerning detailed behavior implementation.

7.1 Agent implementation

As has been defined in the section 6 which proposed the system model, 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 in which the individual layers can interact and inform each other. The top-down interaction (the deliberation layer is considered to be the top and the 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 the 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 them 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. 11).

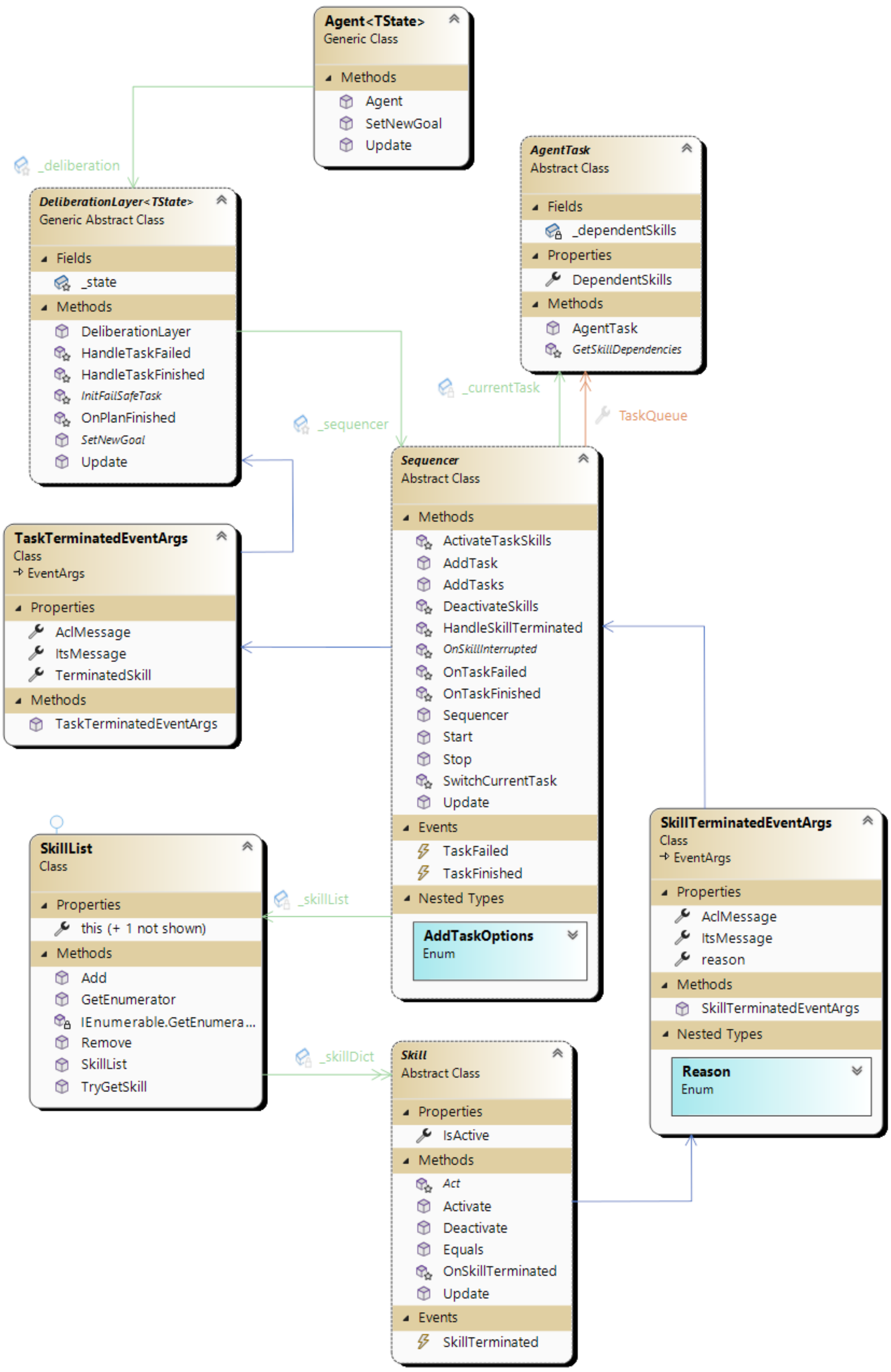

Figure 11: Class diagram of main agent framework components

7.1.2 Deliberation layer

The deliberation layer is the highest from a hierarchical point of view. Its responsibility is to create a plan that satisfies a goal that the agent gets as input. Apart from the layer being abstract (i.e. requiring further implementation details specified by inheriting from the class), it is also generic. That means that it will defer the type specification of its state until it 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 as one of its possible states (e.g. represented by a single discrete number). The initial value of the state parameter is required to be specified upon object instantiation/creation.

Upon providing a goal to the layer by the executing client, it should 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 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 in a way that 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 that contains a general communication skill that 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 behavior 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). Afterward, the Skills get activated by the Sequencer. The `Sequencer` class also exposes the interface to the 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.

Another 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 triggers such an event and terminates, 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 the 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 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 and deactivation of the skill exposed to the `Sequencer` class.

The implementation required by the user-specific use case of a 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 6).

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

The logic which makes use of the agent's sensors to 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 the 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.

An activity diagram sowing process logic and interaction between inner components can be seen in figure 12.

7.2 Communication module implementation

As has been started section 4, where the system requirements are defined, the framework will come with a pre-implemented communication interface which will allow the agents to exchange pre-defined messages. 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 and ACL standards are created to carry appropriate content. The class diagram of the communication interface can be seen in figure 13.

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 the provided message type. Because the class is static, agents[3] can interact with it without having to hold a reference to it. Agents can:

- 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, the sender must provide the recipient agent ID.

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

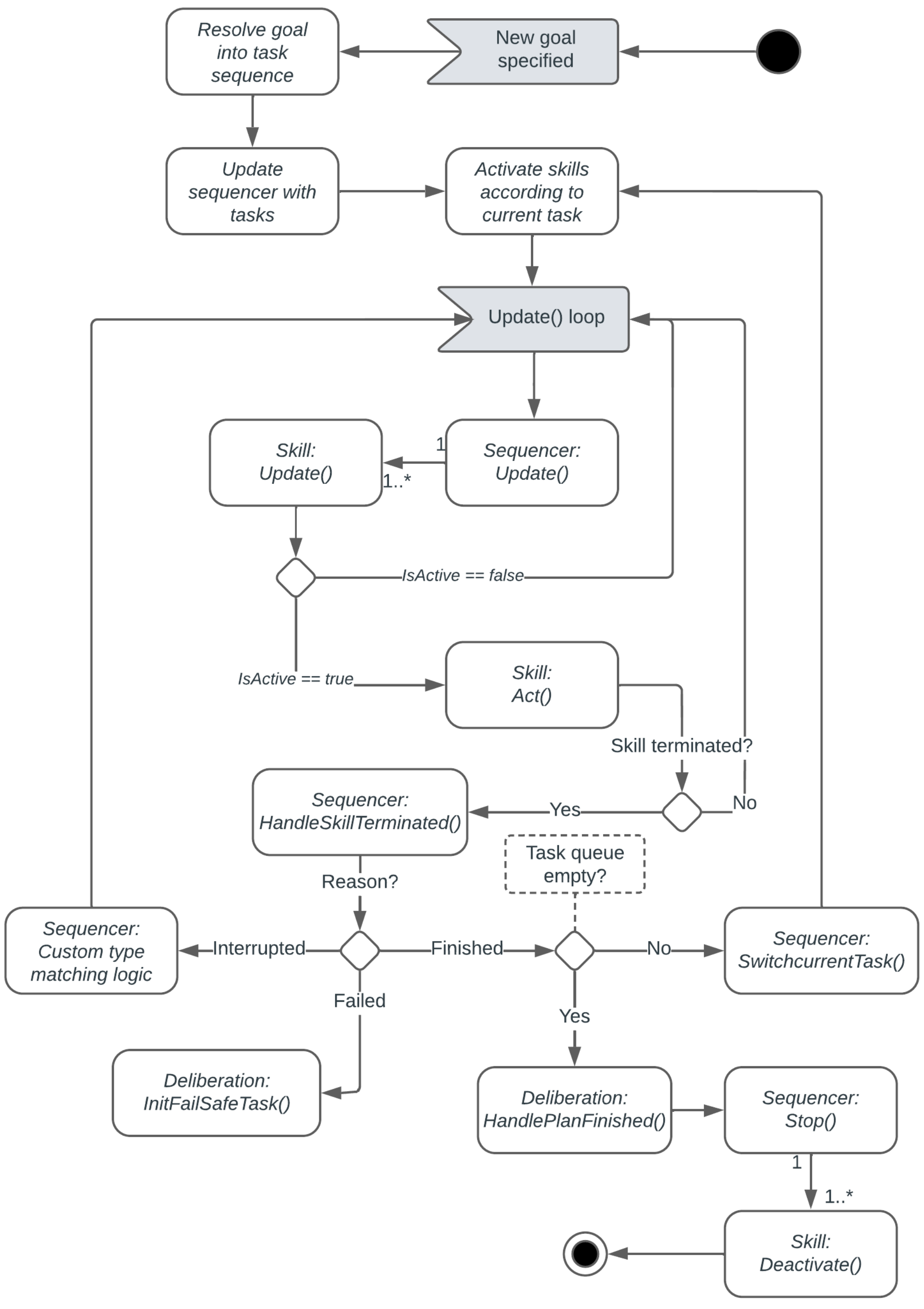
Resolve goal into task sequence
New goal specified
Update sequencer with tasks
Activate skills according to current task
Update() loop
Skill: Update()
1
1..*
Sequencer: Update()
IsActive == false
IsActive == true
Skill: Act()
Skill terminated?
Sequencer: HandleSkillTerminated()
Yes
No
Task queue empty?
Reason?
Sequencer: Custom type matching logic
Interrupted
Finished
No
Sequencer: SwitchcurrentTask()
Failed
Yes
Deliberation: InitFailSafeTask()
Deliberation: HandlePlanFinished()
Sequencer: Stop()
1
1..*
Skill: Deactivate()

Figure 12: 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 a specified Message type needs to be created and implemented. The pre-defined behavior 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 messages 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 behavior, 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 are implemented according to the official documentation. For reference, the ACL message contents can be seen in the figure **??** and table 4, the C-ITS message contents can be seen in figures 8 and 9.

7.3 Unity integration

As has been mentioned in the introduction to Unity software in the section 5, programmatically defined behavior is implemented by assigning C# scripts as individual components to game objects in the scene. The scripts are not being executed by a main client script like in conventional software, but rather by calling an `Update` method in the Unity script which is invoked by the Unity engine on each game tick. Such behavior is obtained by inheriting from the `MonoBehaviour` class, which also defines other methods that get invoked in certain scenarios. Including the `Update` method, the standard methods used in most cases are:

- `Start` method, which substitutes object initialization through the constructor and is called at the start of the script lifetime
- `OnDisable` method, which gets called when the script component or the whole game object gets disabled. Disabled components aren't rendered and cannot interact with the scene nor be interacted with.
- `OnEnable` method, which is inverse to the previous and gets called when game objects or components are enabled/activated.

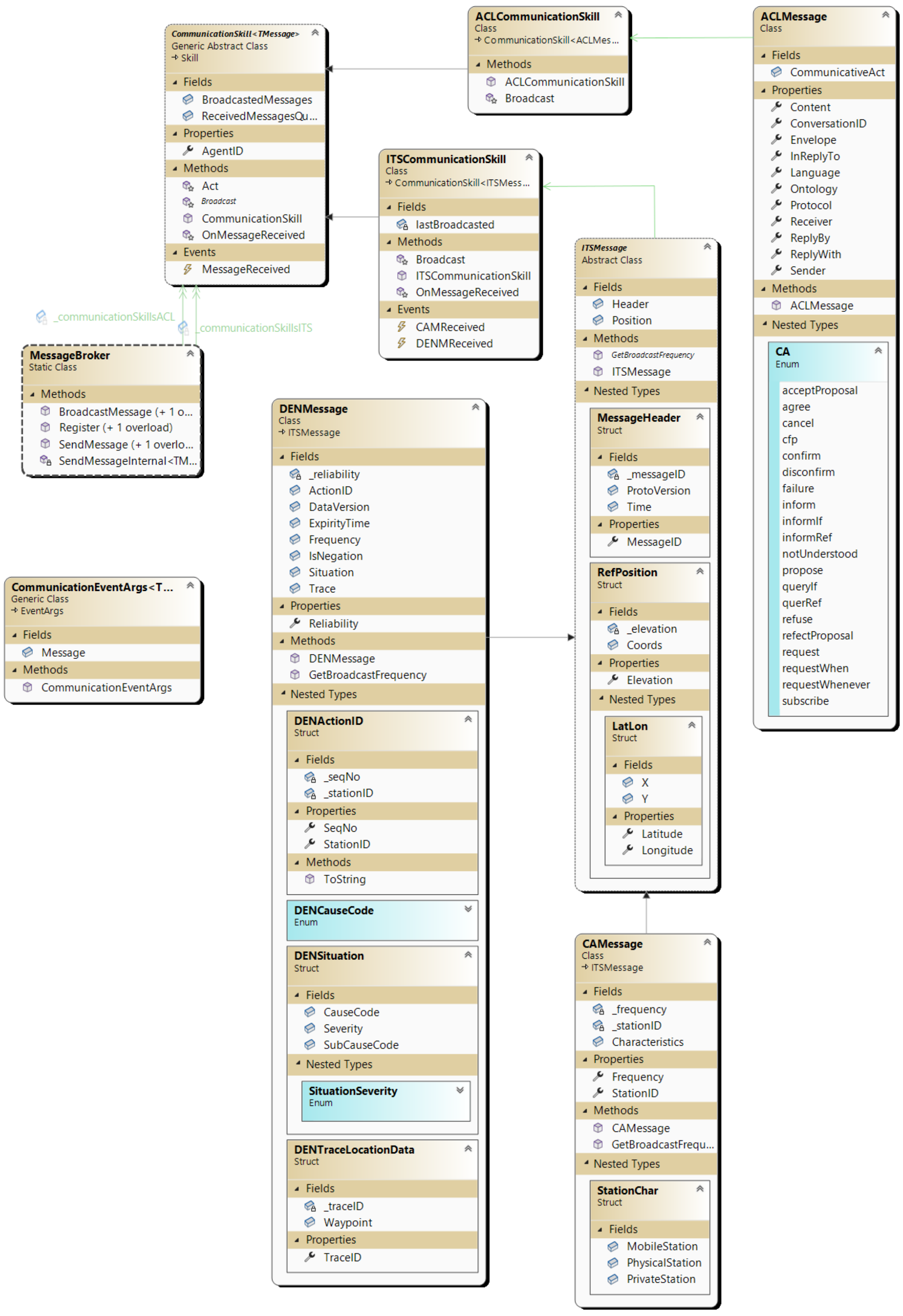

Figure 13: Class diagram of the implemented communication interface

The idea behind the proposed MAS ITS framework integration with Unity is to assign the `Agent` class to game objects as a component in order to use the game object as an agent. For instance, the game object can be a vehicle model or even a non-physical game object invisible in the scene, e.g. a traffic controller. When the `Agent` class would get assigned to a game object, it would be able to interact with its other game components and object properties through pre-defined sensor or actuator skills.

Although the `Agent` class and its components (i.e. layers) come with the `Update` method in their default implementation, it will still not get called by the Unity engine. For that reason, a *wrapper* class `AgentMonoWrapper` been created. This class will get added as a component to the game object that the user wants to induce agent behavior on. The class will initialize an agent by calling an abstract method `InitAgent()` at `MonoBehaviour.Start()` invocation and call `Agent.Update()` on `MonoBehaviour.Update()` invocation.

7.4 Conclusion

In this section, the implementation details of the proposed framework are discussed. The C# language has been used to write the framework's code, as per the discussion in the section 5. The framework's core components are defined as abstract classes, implementing general behavior and defining required behavior to be implemented by the user, which is use-case dependent. For instance, such behavior can be the `Act` method in the `Skill` layer, which serves as Skill's way of interacting with the environment, which is different for every defined Skill. The relations between classes and inner processes have been described, serving as documentation for the usage of the framework.

As per the initial requirements, the agent interaction interface has been implemented. The framework's agent architecture components have been used to create dedicated communication skills that can broadcast and directly send standardized messages to each other together with a custom messaging service. Consequentially, a C# implementation of ACL and C-ITS (CAM, DENM) message standards has been created to be used in the framework.

Lastly, integration of the framework with the Unity game engine has been discussed and described in detail, introducing an additional wrapper component to ensure proper cross-interaction.

8 System validation

This section is devoted to the evaluation and demonstration of the proposed MAS-based ITS implementation framework. An ITS solution is chosen to demonstrate implementation using the framework, serving as a validation of the system design. The chosen ITS solution is then simulated in the IVS software by using the proposed framework. The implementation process is documented, and the simulation results are discussed.

8.1 Qualitative analysis

The following paragraphs picks up on the findings in the section 2.2.1, where the table 2 maps GDTs to state-of-the-art ITS solutions. Those ITS solutions are analyzed in this section in an attempt to transparently select a system to implement using the developed framework in order to validate and demonstrate its use-case.

In order to demonstrate the framework's capabilities, a suitable ITS is chosen to implement using the framework. The goal is to identify which ITS solutions are suitable to implement using the multi-agent simulation framework. In order to achieve this, attributes reflecting the suitability to implement are defined. For each system in a selection of relevant ITS solutions, a numerical value is assigned to each identified attribute, based on how strongly the feature is represented in the system. The decision of the value of feature representation rate is decided by the thesis' author *subjectively*, based on his knowledge of the topic. This is considered as sufficient given the fact that the validation is done still in the development stage of the framework A sum of the scores for each ITS solution makes for a qualitative benchmark that is used to select an ITS solution for demonstration of the proposed framework's capabilities.

The method to determine how well-suited the candidate system type is for the implementation to an IVS is to evaluate it based on the *three* features discussed in the preceding section 2:

- Intelligence distribution rate
- Driver engagement
- Research relevance

A set of integer values determine the rates how the selected features are represented in the ITS solution in question. The representation level is expressed in a fuzzy logic given in four levels:

- None $\rightarrow$ 0
- Low $\rightarrow$ 2
- Moderate $\rightarrow$ 5
- High $\rightarrow$ 8

Because *None* representation of the feature does not bear any significance to choosing an ITS in question, it is assigned zero. *Low* representation should not have a significant impact on the suitability of ITS, therefore its value is only incremented by two as opposed to *None*. *Moderate* and *High* rates of feature representation should have a more significant influence on the final system choice, therefore their values have been assigned as increments of three instead of two.

Each system reviewed in section 2 is evaluated and the sum of individual feature evaluation determines its score. The higher the score, the better the result for a particular ITS. The system with the *highest* score is chosen to implement using the proposed framework. The results of the qualitative analysis can be seen in the table 8 below.

Table 8: Quantitative analysis results

	Features			
ITS	Intelligence distribution	Driver engagement	Research relevance	**Score**
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
In-vehicle Information System	8	8	8	**24**

8.1.1 Results discussion

The conducted analysis has determined two ITS solutions as suitable systems to demonstrate implementation - In-vehicle Information System (IVIS) awareness and warning system and Cooperative ACC. Both come from the state-of-the-art C-ITS subfield. As previously discussed, the Cooperative ACC system is using dedicated message types in its implementation - the SPaT and MAP messages. On the other hand, the IVIS system can use the pre-implemented CAM and DENM messages to share traffic-related information. For this reason, the *IVIS-based awareness and warning information system* is chosen to implement using the proposed framework. In the following section, the system is described.

The developed system should also be able to model state-of-the-art ITS systems that utilize communication between road users, such as C-ITS systems.

8.1.2 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*, which is, in most cases, a touch screen interface. It provides 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 [41]. 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. This 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 Heating, Ventilation, and Air Conditioning (HVAC) control, multimedia controls, navigation support and parking assistance (i.e. parking camera view). Despite these systems not being of value concerning 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 the IVS in the practical part because the V2X C-ITS is 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, 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 a dedicated interface for broadcasting such information, simulating message-based communication between road users & infrastructure, integrating with technical solutions to display the information on the infotainment screen.

8.2 In-Vehicle Information System implementation

With the particular ITS implementation decided, the whole process of implementation is described. Before describing the implementation process itself, a description of the implemented IVIS solution is given, together with implementation requirements and expected results. Afterward, is custom testing environment setup (i.e. the system implementation) is be presented. Finally, the implementation test results are presented and discussed.

8.2.1 System description and requirements

As has been described in the preceding section, the In-Vehicle Information System can enable the driver to control various aspects of vehicle set-up, as well as provide information about the vehicle and the environment, including traffic conditions. The state-of-the-art IVIS solutions include V2X communication where vehicles and the infrastructure can distribute otherwise locally-available information to all interested road users.

One such V2X communication use-case referred to as *Smart Routing*, is providing traffic *real-time* traffic condition information to relevant connected vehicles. For instance, suppose there is congestion forming at a particularly busy road stretch. The local roadside equipment can automatically collect accurate traffic density data of connected vehicles broadcasting awareness messages, process the collected information and in turn broadcast information about local congestion to other remote road users that can optimize their route planning using received information [42].

In order to successfully build and verify the implementation of the system in the IVS software, initial requirements and expected outputs need to be set. The implementation should conform to the following specification:

1. Intelligent agents should be created using the proposed framework's interface.
2. The agents should be able to communicate through the standard C-ITS communication protocol.
3. There should be two types of ITS agents - the connected vehicle agent and the connected infrastructure agent.
4. The infrastructure agent should be able to detect emerging congestion forming nearby.
5. Likewise, the agent should also detect that the congestion has decreased.
6. The congestion event should be specified by the appropriate Cause Code.
7. Similarly, the event status should be specified with a Sub-Cause Code value - 1 for free flow and 4 for heavy traffic jam [40].
8. The connected vehicle agent should then be able to commit to a plan where it attempts to reach a destination, potentially re-planning its route if it receives a DENM message about a downstream congestion forming.

If these conditions are met, the system should have a positive effect on traffic by decreasing the overall travel time for vehicles. The implementation is validated by collecting and analyzing traffic data from the simulation.

8.2.2 External dependencies

In order to facilitate the validation experiment setup and keep the experiment scope to C-ITS Smart Routing system implementation, additional packages available to use in the simulation software are utilized to model the traffic. Below is a brief overview of the external packages.

EasyRoads3D by AndaSoft

This package allows one to easily build roads directly in the Unity UI. The main advantage of the package is usage simplicity and friendly UI - the user can create road stretches using simple mouse-based UI with the possibility to connect road segments to create intersections and build a road network.

With this tool, a testing road network is built that the vehicle agents will drive through.

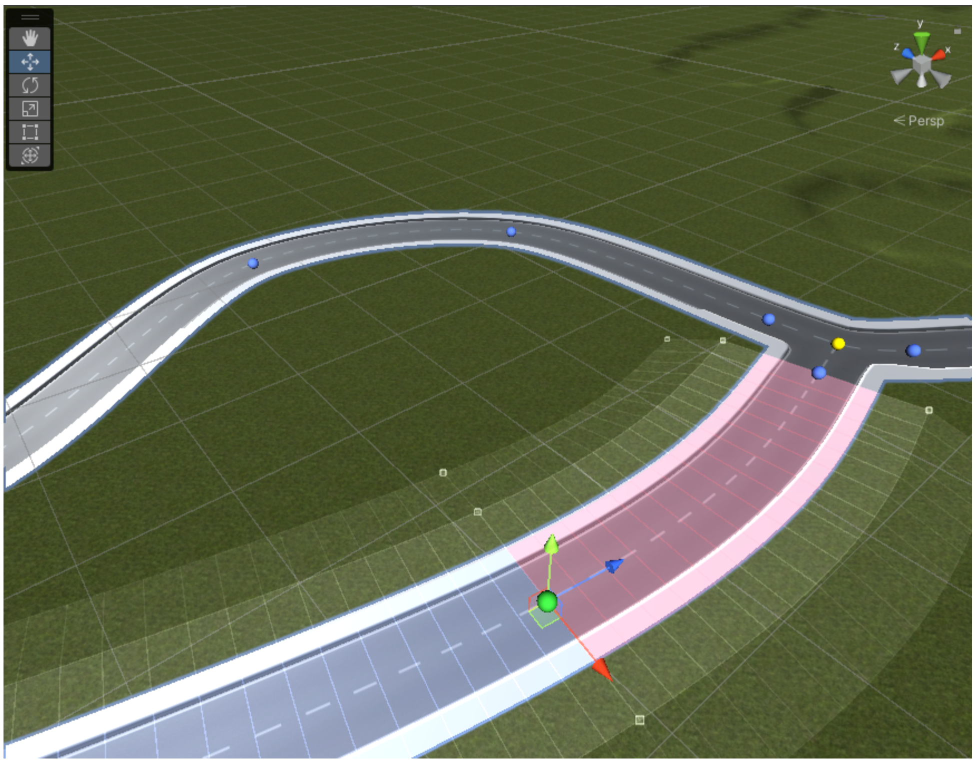

Figure 14: EasyRoads3D - The road building UI showcase

Simple Traffic System by TurnTheGameOn

This package can be used to easily generate vehicle traffic. The traffic is set up by creating waypoint-based routes that can be interconnected in a modular way to populate road networks with traffic. Vehicles are spawned to follow the route waypoints without crashing into each other.

This tool is used to create vehicles that follow a pre-defined path of points. The package also features a traffic light interaction functionality where a route can manage yielding vehicles based on the traffic light signal.

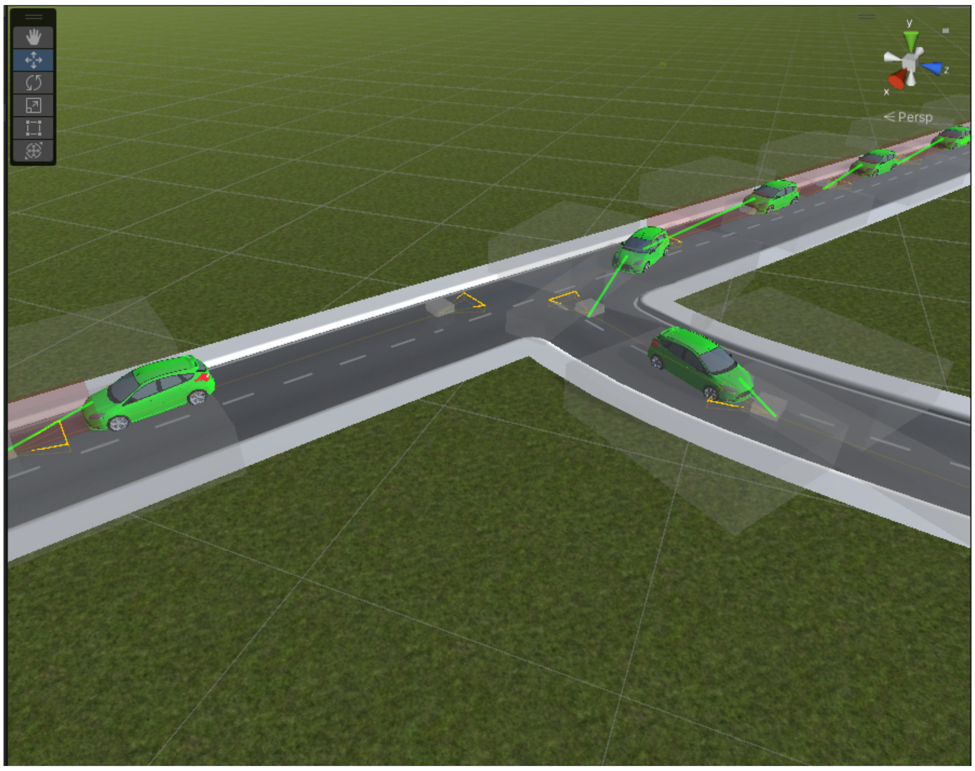

Figure 15: Simple Traffic System - A showcase of vehicles following a route

8.2.3 Agent implementation

This section covers the implementation of the chosen C-ITS solution using the proposed MAS framework. Based on the specification, two agents are created. The agents are created by inheriting classes from the MAS framework components and implementing custom behavior, as well as using its pre-defined C-ITS communication modules to simulate V2X communication.

Connected Traffic Light

The Connected Traffic Light agent is the first actor in the Smart Routing implementation. There is only one instance of this agent in the simulation which manages a single traffic light. This agent monitors the number of vehicles in the queue at a designated intersection and processes the data to evaluate if congestion is forming. It informs about its local traffic state via localized (meaning that message includes the traffic light's location) DENM messages broadcasted to all connected road users.

Since the agent's behavior is constant, apart from broadcasted DENM message content (value of Sub-Cause Code), the agent has got only one task, consisting of two Skills - `ITSCommunicationSkill` for messaging service and `MonitorQueueSkill` which changes broadcasted Sub-Cause Code based on discovered number of vehicles in the queue. For reference, the agent's composition can be seen in fig. 16 and the agent's activity diagram can be seen in fig. 17.

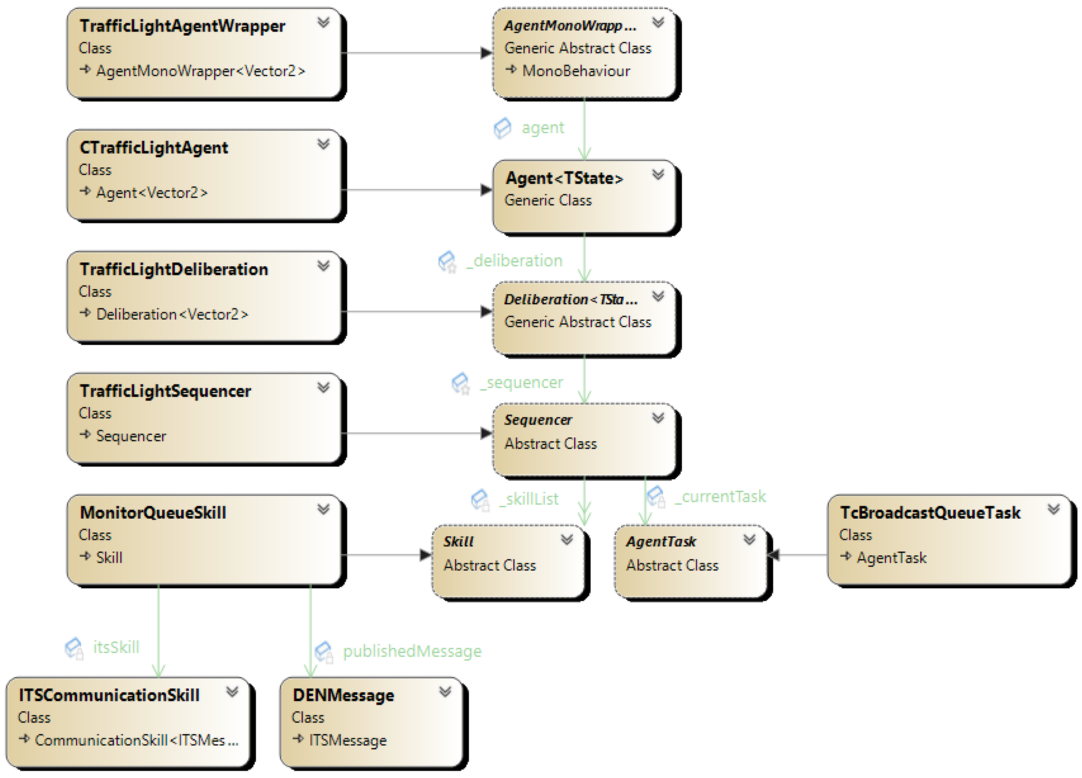

Figure 16: Class diagram of the Connected Traffic Light implementation

Connected Vehicle

The Connected Vehicle agent is the second agent type in the simulation. This agent type is added as a component of each vehicle spawned in the simulation run. The purpose of this agent module is to process received DENM messages and evaluate if the planned route needs to be changed. It also broadcasts CAM messages to other road users. The content of broadcasted CAM messages is the vehicle's location.

Although there is only a single task defined in this agent, its initialization is parametrized, allowing the agent to adapt to the current conditions. The task consists of four Skills:

- `ITSCommunicationSkill` - C-ITS messaging interface, as has been described before.
- `VehicleInfoSkill` - A Skill for retrieving information about the vehicle, namely its current position, and additional navigation properties, which is discussed separately.
- `CAMPositionUpdateSkill` - As the name suggests, this is a trivial "plumbing" Skill that takes vehicle position from the foregoing Skill and updates the broadcasted CAM message of the message service Skill accordingly.
- `CongestionMonitorSkill` - This Skill takes input from `VehicleInfoSkill` and processes received DENM messages if there are updates about traffic conditions

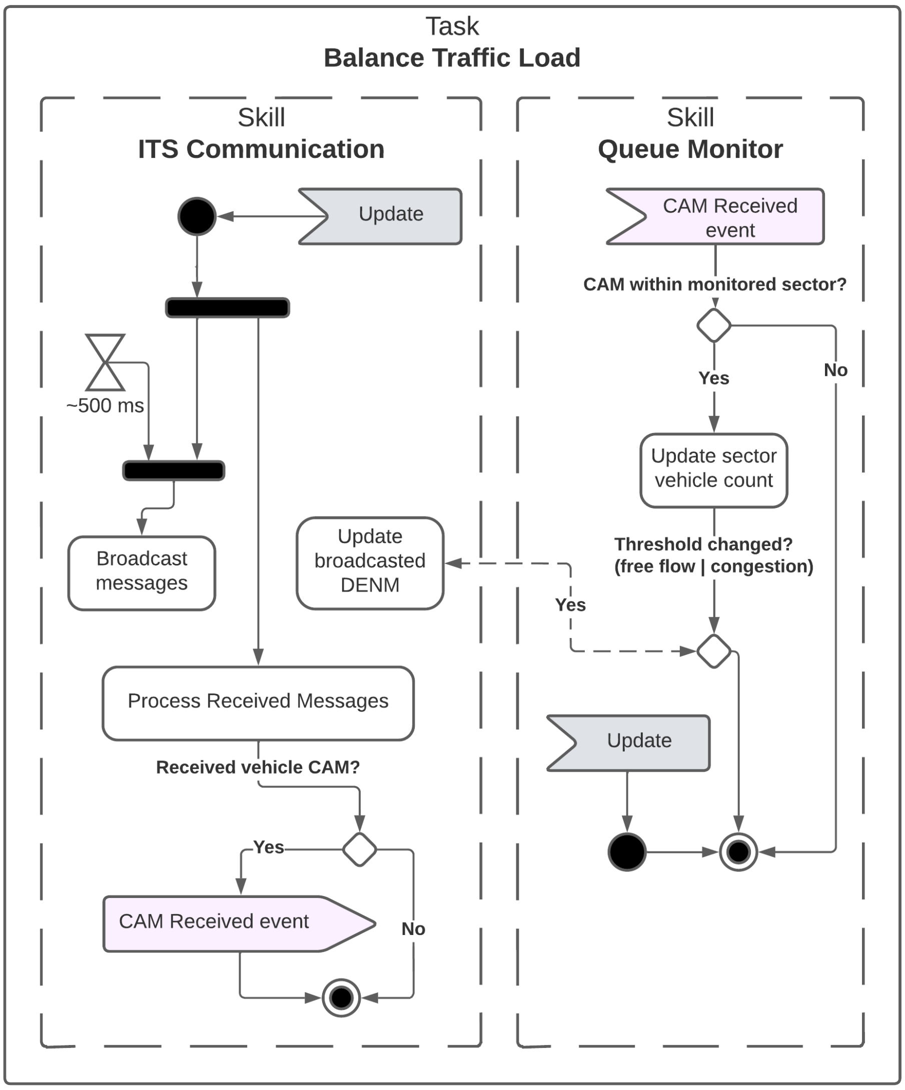

Figure 17: Activity diagram of the Connected Traffic Light agent

on the network. Initially, it checks whether the DENM event location is along the planned route and throws the message away if not. If there is congestion on the planned route, the Skill then finds out and triggers a Skill Interrupted event, which signals to the Deliberation layer to re-plan the route. Here, the parametrized initialization of the agent's task is utilized by providing a flag/switch to plan the default or an alternative route.

The agent's class diagram can be seen in figure 18 and its activity diagram on figure 19.

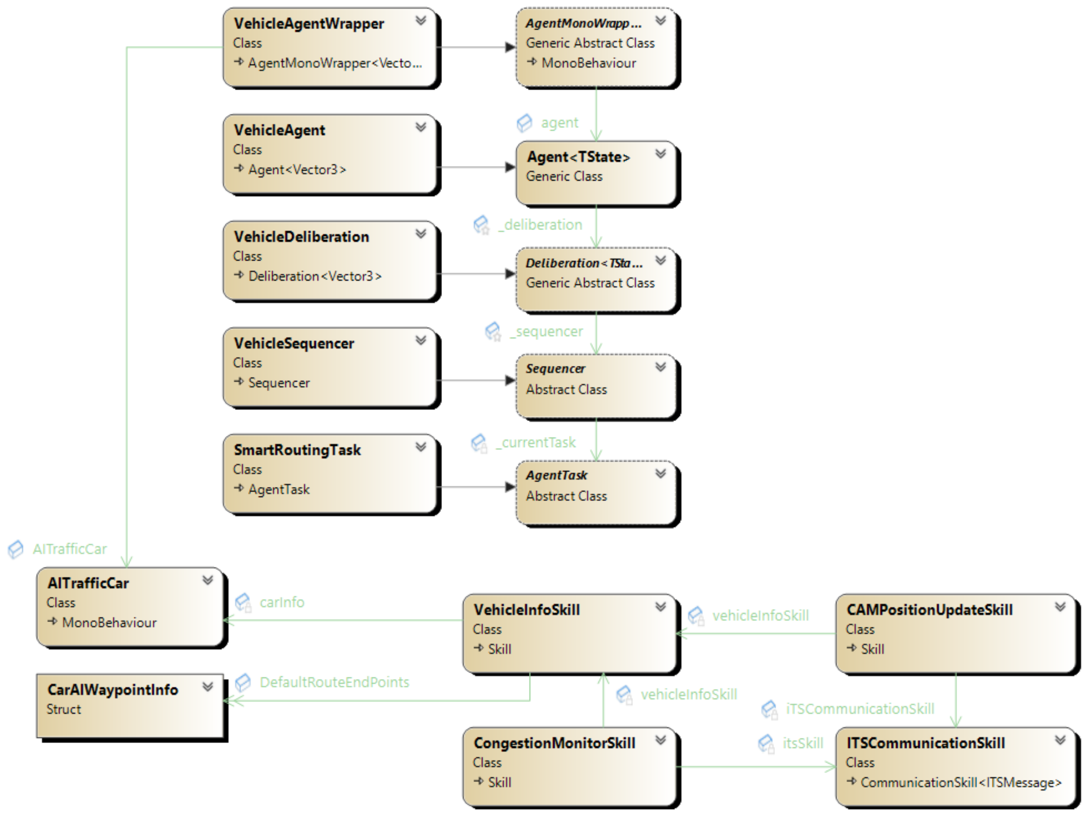

Figure 18: Class diagram of the Connected Vehicle implementation

Other modifications

Since the information about vehicle navigation and route information is managed by the *Simple Traffic System* (STS) package scripts, modifications needed to be added to the vehicle management and route management scripts. This mainly involved exposing private class fields to be accessible to other classes or adding other ways how to modify inner properties from the outside, so that the external MAS framework scripts can interact with them. The `VehicleInfoSkill` provided an interface to manipulate required properties. Additionally, the route-switching algorithm has been modified. In the original (STS) code, a vehicle chooses a turn at an intersection randomly. The performed modifications allow outside code to decide which turn shall the vehicle take.

Regarding the simulation evaluation, a logging script has been created that can be used on the vehicles in the simulation logging object uptime and writing it into a log file. As a

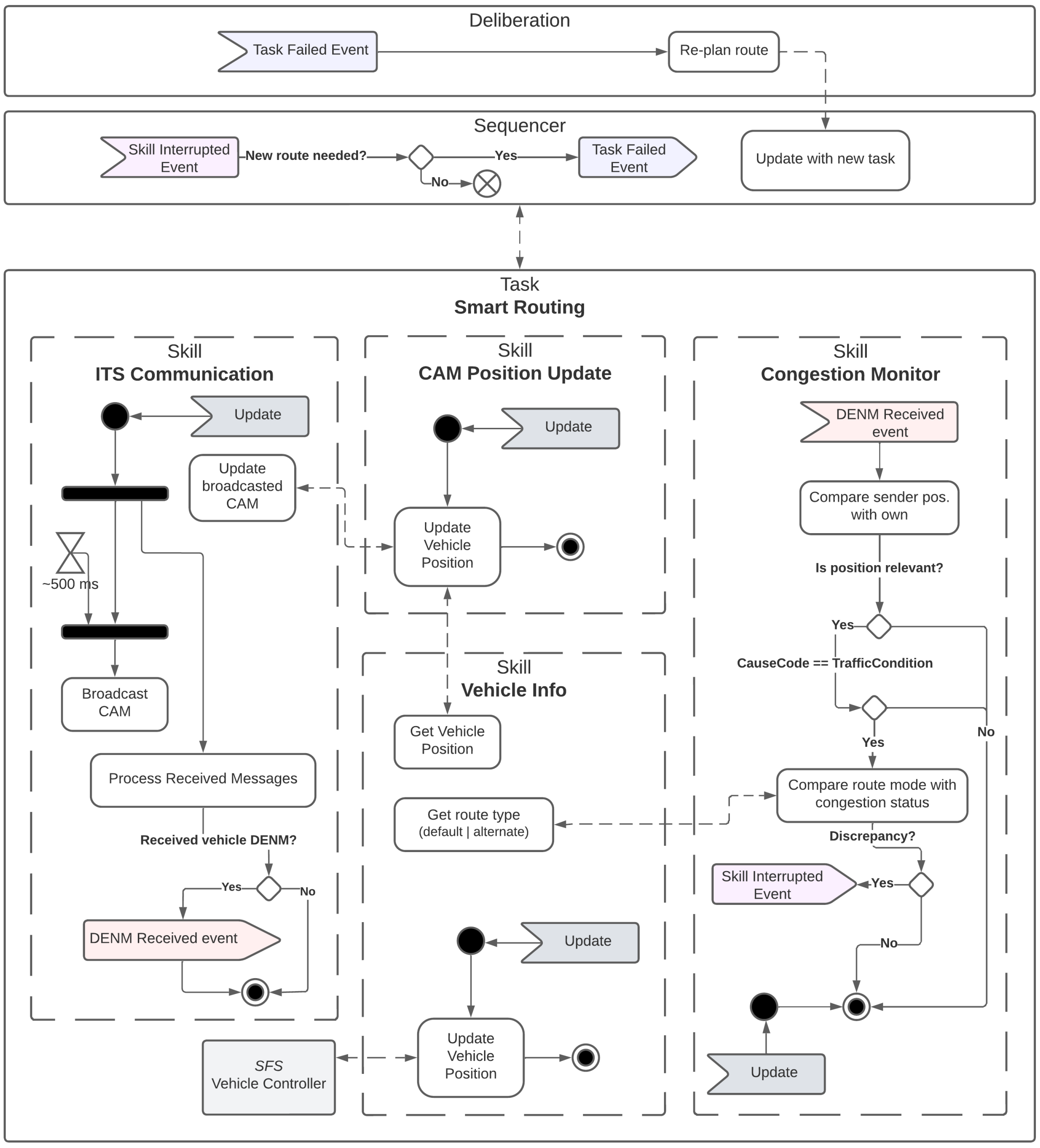

Figure 19: Activity diagram of the Connected Vehicle agent

result, the time taken by each vehicle to reach its destination can be measured and further analyzed.

8.2.4 Simulation set-up

After describing the IVIS system implementation details, the simulation set-up and evaluation methodology is described.

Using the third-party packages, a road stretch has been created to test the implementation. The road stretch features a primary route that has got a traffic light along the way, imitating an intersection. This route represents the shortest path route through an urban area. There is also a second route, which is considerably longer and winding, representing a countryside road with less average traffic intensity. A "warm-up" section where vehicles are spawned is preceding the branch-off into the two routes. For visual reference, see Figure 20.

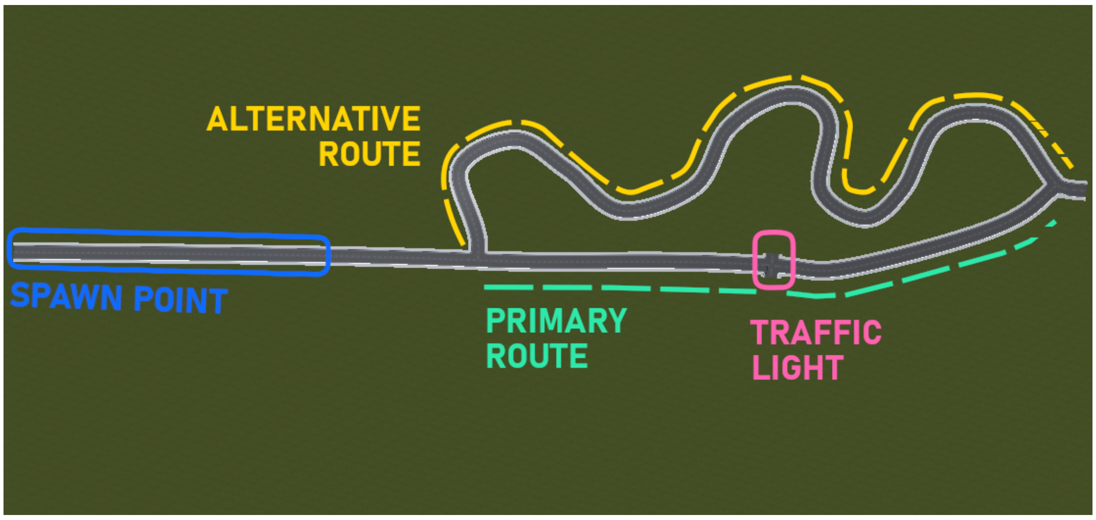

Figure 20: Road stretch for simulating the IVIS system

To verify and prove that the Smart Routing IVIS system works, simulations were run iteratively with parametrized traffic light queue size threshold for identifying congestion. In other words, the queue size threshold parameter expresses how many vehicles should be in the traffic light queue before the value of sub-cause code withing the broadcasted DEMN message changes from "free traffic" to "congestion". The number of vehicles to drive through the road stretch is chosen to be $N_v = 20$ and the queue size threshold ranged from $0..N_v$. To gather more results and avoid the risk of unknowingly analyzing an edge case parameter combination, the simulation batch was run multiple times with modified traffic flow influencing parameters. The variable parameters are:

- Alternative route speed limit (V_{alt})
- Green phase length (T_g)
- Stop phase length (T_s)

Table 9 shows the three variations of chosen simulation parameters.

Table 9: Parameter values for simulation batches

#	V_{alt} $[kmh^{-1}]$	T_g $[s]$	T_s $[s]$
1	25	6	4
2	25	7	5
3	30	7	4

8.2.5 Simulation results

The simulations were successfully run and the logs containing travel time for each vehicle were processed and evaluated. The observed variables were *Total travel time*, *Maximum travel time* (TT_{max}) and *Minimum travel time* (TT_{min}), which were recorded for simulation runs with a variable queue size threshold. The plotted simulation results are in Figures 21 and 22. Note that the plot legend conveys information about simulation parameters in format [V_{alt} - T_g - T_s].

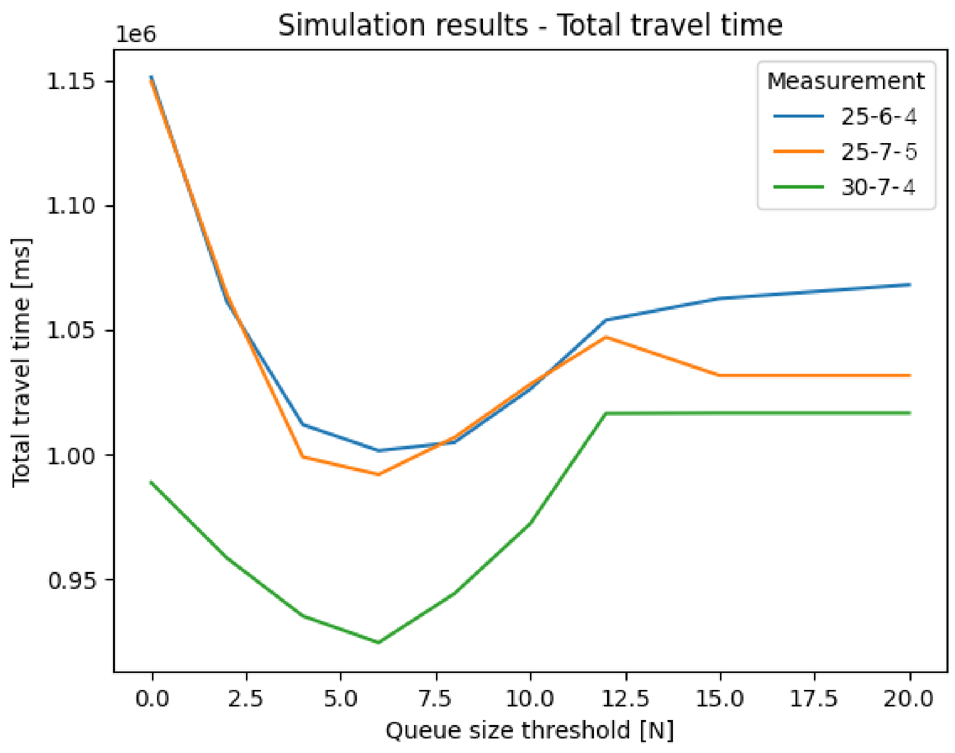

Figure 21: Simulation results - Total travel time

When evaluating the simulation results, it is clear that the threshold value does influence the travel time of vehicles, which is expected if the IVIS implementation is assumed to work. With the threshold equal to zero, all vehicles take the alternative longer path and experience large travel times. When the threshold gradually increases, allowing groups

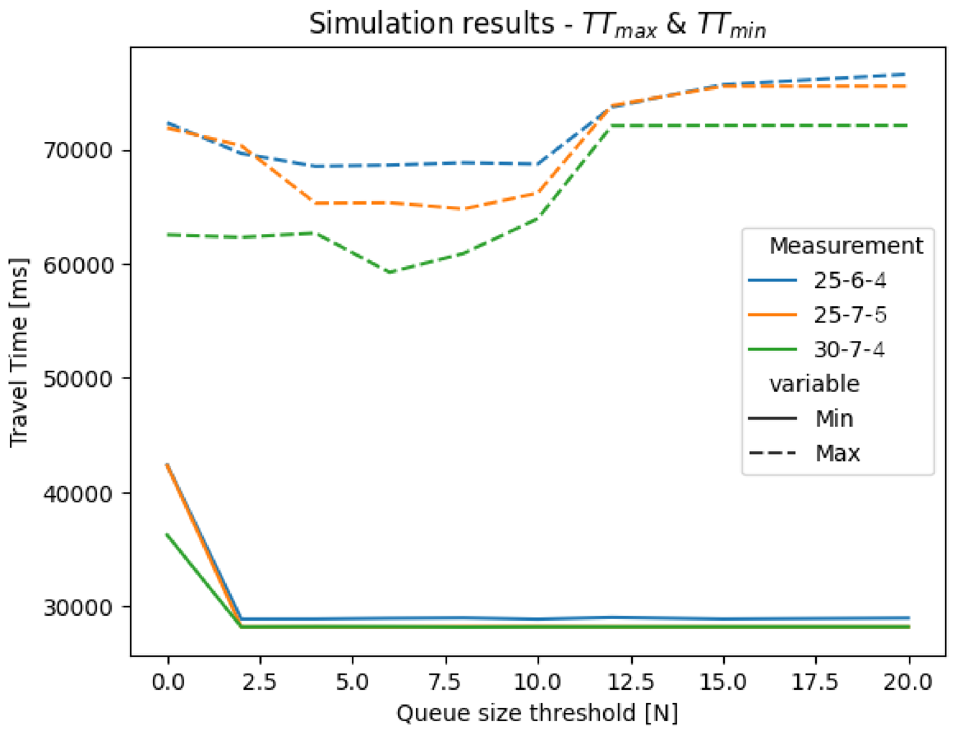

Figure 22: Simulation results - Min and Max travel time

of vehicles onto the main route, the travel time starts decreasing rapidly, as the main route has enough capacity to clear the oncoming traffic without disruption. Around the threshold level of six, the traffic flow rate reaches the route's capacity and with increasing queue threshold the travel times start to increase while rendering the alternative route under-utilized.

All three simulation batches do affect travel time (which implies that traffic flow is also affected) as expected. The absolute values of travel time have changed, however, the relative changes depending on queue threshold size were very similar in all three variations.

8.3 Conclusion

In this section, an ITS solution for the framework validation is chosen to implement using the proposed framework, based on a qualitative analysis considering criteria determined from preceding discussion and research. The system to implement is the Smart Routing C-ITS utilizing V2I communication, dynamically re-planning optimal route to the destination. Requirements for system implementation are defined to better evaluate the implementation results. Afterward, third-party packages that facilitate validation scenario development are briefly introduced, leading to agent type specification and their components, as well as action diagrams outlining their behavior, while also addressing the required IVS package code changes to ensure successful integration with the agent framework. Simulation methodology is introduced and simulations are successfully run, after which the results are analyzed. From the simulation results, it is identified that the chosen system behaved as one would expect and it is concluded that the implementation of the C-ITS system is successful.

9 Discussion

This section goes over the overall process the work that has been done regarding the thesis and evaluate the outcomes. Firstly, it should be assessed whether the thesis has fulfilled the given guidelines in the thesis assignment. The following section goes over the guidelines and discuss whether the assignment outline has been fulfilled.

Research question #1: Can agent-based modeling be applied to simulate ITS in vehicle simulators and are they a viable option?

In section 3, the literature regarding Multi-agent Systems is reviewed. The review is focused mainly on the primary characteristics of MAS and on various types and implementation methodologies of MAS. The outcome of the section is identifying important implementation details of MAS which would aid in ITS implementation using agent-based modelling.

In section 2, the current research regarding ITS systems is reviewed. The current trends in ITS suggest that modern ITS systems will rely on maximizing interoperability between individual road users, utilizing direct, non-centralized wireless communication. Such features are common in agent-based systems as well, therefore it has been concluded that MAS could represent a convenient way how to simulate such systems. Cooperative ITS (C-ITS) systems and connected In-vehicle Information Systems are identified as the most suitable systems for application using MAS.

Research question #2: How would an ITS simulation be implemented into an existing IVS using agent-based modelling?

In section 6, a system to integrate novel ITS solutions into a vehicle simulator is developed using the research findings in the preceding parts of the thesis. The system implemented the $_3$T architecture for individual agents that is supposed to offer a sufficient balance between system complexity and modularity. The architecture of the macro-system, which defined mainly how agents interact with each other, is also developed in line with requirements based on MAS research as well. The system is then implemented using the C# programming language, because of its object-oriented properties and the fact that the intended simulator software uses C# scripting API as well, allowing for deep integration with the software. The implementation comprises partially functional classes that define the system's structure but are open to detailed logic implementation based on a specific use case.

Research question #3: What are the available modules and libraries for implementing MAS in different programming languages?

As a secondary objective to building an ITS simulation framework for an IVS, it is required to review available MAS implementation modules that would facilitate building the desired framework. Multiple modules are reviewed, including their main use-cases and implementation details. After discussing the main aspects of integration into the simulator software (section 5), it is concluded that there are no suitable modules for MAS implementation to the specific IVS, as the researched modules are either outdated or focused on different aspects of multi-agent systems. Therefore, it is decided that the framework would be built from the ground up.

Research question #4: What are the advantages and challenges associated with using agent-based modeling to simulate Intelligent Transport Systems in vehicle simulators, and how can they be addressed?

The system evaluation is, in part, done by implementing a sample C-ITS system into the simulator software, with a cooperative traffic light and connected vehicles. The implementation of the sample system proves that it is possible to implement specific ITS solutions using the developed framework, therefore simulating Intelligent Transport Systems using agent-based modeling is possible.

The observed challenges are that implementing conflict resolution and cooperation between agents deems to be a very complex topic which would take a lot of additional research and effort to implement and therefore is decided that it will not be part of the developed ITS implementation framework. This could result in a less stable agent behaviour during simulation. This missing feature could be added in future improvements to the framework.

9.1 Results

Based on the designed system and its implementation, the implications of the research activity and its benefits, caveats and application possibilities are analyzed.

9.1.1 Benefits

The main benefit of developing an ITS implementation framework is obvious. In theory, the framework should facilitate ITS implementation into a vehicle simulator so that it enables the developer to focus more on the high-level implementation related to the ITS system itself, as the structure of the underlying agent system would be pre-implemented.

For instance, the developer implementing a given ITS solution into the IVS can implement a desired behavior of an actor in the system by dividing the algorithm into atomical skill modules which inherently makes developing complex behavior easier and more manageable, as well as open to extending its capabilities by iteratively adding more modules. Also, one of the greater benefits is the fact that communication API for the system developer has been implemented, according to the C-ITS ETSI standards. This makes it very easy for developers to implement C-ITS systems which rely heavily on distributed road user communication, which has been the case when developing a sample ITS simulation (dynamic vehicle navigation) in order to validate the framework's capabilities. The development process of the communication interface was simply filling the pre-defined DENM and CAM message parameters with desired information. The inner architecture of agents defined by the framework allows to develop complex, dynamic behaviour.

9.1.2 Shortcomings

When implementing the sample C-ITS system using the proposed framework, shortcomings were also identified during the process. The perceived disadvantage of using the framework is that when implementing a simple behavior that does not need complex decision-making, all the agent layers need to be implemented, although they might not be fully utilized. For instance, the *Connected traffic light* in section 8.2.3 only has one goal in the deliberation layer and also just one task to perform, which implies that the agent model could have been reduced just to a static, cyclic execution of the skills used mainly for communication with the connected vehicles. Therefore, the developed MAS framework is likely to demonstrate its advantages only when building larger-scale ITS projects in the vehicle simulator which require developing complex behavior logic for the agents.

Another potential disadvantage that was identified during the framework development is that although the 3-layer concept might be effective, it might not be intuitive to use for someone new to the concepts for multi-agent systems. For instance, it is not very clear how to decide on the atomicity of the individual actions the agent can perform. However, this problem can be solved by providing a template for a "simple" agent which has got the deliberation (and potentially also the middle sequencing layer) in a single state.

9.2 Future work and implications

Regarding future work and further improvements of the outcomes of this thesis, it would be beneficial to implement agent negotiation, which has been discussed in the theoretical part of the thesis but has not been implemented as part of the framework. The system validation section has shown that it is possible to develop fully functional simulations of Intelligent Transport Systems, however, developing a negotiation model for the agents would make the framework more robust and would allow modeling systems with greater complexity.

Apart from the improvement mentioned above, the system can already be used to model ITS systems with intelligent agents, including state-of-the-art cooperative ITS systems. To mention a few, the framework can model various in-vehicle information systems, such as the intersection collision warning, emergency vehicle warning or overtaking warning. Apart from the mentioned sensing & perception systems, planning-focused its systems such as dynamic vehicle routing or green light optimal speed advisory can be modeled. After the potential improvement of adding agent negotiation, it would also be possible to implement autonomous driving and related advanced driver assistance systems using the framework.

10 Conclusion

The goal of this thesis was to investigate Multi-Agent Systems, which is a sub-field of Artificial Intelligence research, especially research related to the application of Multi-Agent Systems in Intelligent Transport Systems simulation.

First, the area of Intelligent Transport Systems was investigated, setting delimitations and important features that were important to the topic of the thesis. The relationship between IVS and ITS systems was discussed, leading to a discussion about intelligence distribution and current research in the ITS field.

The second part was dedicated to MAS literature review and the main principles of MAS were discussed. Furthermore, individual MAS architectures were described and reviewed. Finally, the concepts of communication between agents were investigated, defining requirements on how the agents should communicate.

In the practical part of the thesis, system requirements were defined first, in order to facilitate the process of implementation and clearly define the system's capabilities. The development platform was discussed, and MAS modeling libraries that would help building the framework were discussed. It was identified that there are no suitable modules for MAS implementation to an IVS, as the research modules were either outdated or focused on different aspects of multi-agent systems. Therefore, it was decided that the framework would be built from the ground up.

Next, the actual system was proposed. The system design was divided into two parts - a micro-architecture that focused on the inner structure of individual agents, whose goal was to make the agents sufficiently modular as well as to actually facilitate the ITS implementation. The architecture was designed based on the literature review in the preceding chapters. The second part was dedicated to the macro architecture, which described how the individual agents should interact with each other. The integral part of agent interaction was communication specification, how agents will send and receive information. Based on the preceding research on ITS and MAS, the ETSI messaging standards (CAM & DENM) were decided to be implemented.

After the system for MAS-based ITS systems implementation has been specified, the next part was devoted to the system implementation. The outcome is a highly modular framework that is integrated with the chosen simulator software, with general behavior specific to MAS & ABM and the architecture pre-implemented. The framework was implemented using the C# programming language and the simulator engine (Unity) API.

The developed framework was then validated by choosing a case appropriate for demonstration of the framework's utility. The selected system for the demonstration was a dynamic routing system whose purpose was to improve traffic conditions by harmonizing a road network's load by the use of cooperative traffic lights and connected vehicles. The system was successfully implemented and the analysis of the simulation showed that the system is behaving as expected, i.e. reducing the overall travel time when calibrated.

Overall, the goals specified by the thesis guidelines have been fulfilled. The outcome of the thesis is a modular C# framework/module which facilitates implementation of various (Cooperative) Intelligent Transport Systems solutions implementation into a vehicle simulator software.

11 Attachments

The attachment of this thesis is a repository with the ITS implementation framework named "ITS-MAS-Framework" and the validation project (in folder "Validation Project") hosted on the GitHub site.

The repository is found on the following link:
`https://github.com/Honzeak/ITS-MAS-Framework`

References

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

[2] 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.

[3] 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.

[4] Joost de Winter, P.M. Leeuwen, and Riender Happee. "Advantages and Disadvantages of Driving Simulators: A Discussion". In: Jan. 2012.

[5] Márton Horváth et al. "Vehicle-In-The-Loop (VIL) and Scenario-In-The-Loop (SCIL) Automotive Simulation Concepts from the Perspectives of Traffic Simulation and Traffic Control". In: *Transport and Telecommunication Journal* 20 (Apr. 2019), pp. 153–161. DOI: 10.2478/ttj-2019-0014.

[6] Taras Lishchenko. *Intelligent Transport System: Trends, best practices, success stories*. N-iX. Jan. 2021. URL: https://www.n-ix.com/intelligent-transport-system/.

[7] P.E. Firmin. "Satellite Navigation Technology Applications for Intelligent Transport Systems: A European Perspective." In: May 2006.

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

[9] 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/.

[10] 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.

[11] 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.

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

[13] 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.

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

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

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

[20] 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.

[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] 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.

[26] 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.

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

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

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

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

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

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

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

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

[35] 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.

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

[37] 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.

[38] 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.

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

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

[41] 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/.

[42] Areti Kotsi, Evangelos Mitsakis, and Vasileios Psonis. *Coordinated provision of C-ITS services for Dynamic Traffic Management*. 2020. DOI: 10.1109/itsc45102.2020.9294559.