Formal Verification of Multi-agent Plans for Vehicle Platooning

Thao Nguyen Van and Kurt Geihs(✉)

Distributed Systems Research Group, University of Kassel, Wilhelmshöher Allee 73, 34121 Kassel, Germany
{vtn,geihs}@vs.uni-kassel.de

Abstract. The collaboration and coordination of autonomous vehicles into convoys or platoons have been used on our highways. However, before deploying such vehicles on the real road, their autonomous behaviors must be certified to act safely. The vehicle platooning can be considered as a multi-agent system where each agent can make its own autonomous decisions. In order to ensure that these decision-making agents in the platoon never violate safety properties, we use the Uppaal model checker to verify them. Furthermore, to facilitate the checking process and create a consistent translation process, we have developed an automated translation program that can map our multi-agent plans to the Uppaal model checker format.

Keywords: Verification · ALICA · Model checking · Multi-agent plans

1 Introduction

The "driverless vehicles" will be critical elements of the future of transportation systems. Within these systems, vehicles perform as individual systems, and at the same time, they interact with each other as well as with the environment, i.e., pedestrians, roads, and signs to achieve common goals. Because of the legal constraints, there must always be a driver in the vehicle. However, even though having full autonomous vehicles on the road is still in the future, the automotive industry is using already "car convoys", "platoons" or "road trains". Here, a platoon consists of a leading vehicle being driven manually and one or more following vehicles automatically driving and following the leader one after another.

The platooning concept was introduced by the COMPANION project with the aim of improving the safety and efficiency of vehicles on the congested road [1]. In cooperative autonomous driving platoons, each vehicle obviously must be able to communicate with the others, at least with the leader and cars immediately in front and behind in order to share their intentions, for example, adjust the speed and the distance according to the other vehicles information. Also, the platoon leader is responsible for managing the overall platoon formation by accepting new vehicles or responding to vehicles leaving.

P. C. Vinh and A. Rakib (Eds.): ICCASA 2020/ICTCC 2020, LNICST 343, pp. 3–15, 2021.
https://doi.org/10.1007/978-3-030-67101-3_1

In this paper, we propose and verify ALICA (A Language for Interactive Cooperative Agents) as a formal description language for a specific scenario, i.e., car convoys where an unplanned platooning temporarily creates or joins in an ensemble to share part of their journey. Originally, the language was developed for robotic football in which the opponents (environment) are dynamic. They change frequently and unpredictably. Therefore, we also applied ALICA for other domains like Cooperative and Extraterrestrial Robotic Exploration Missions[1] and Cooperative Autonomous Driving Scenarios [16]. Platooning is one of the promising concepts towards autonomous vehicles in order to deal with traffic jams and, at the same time, increase the overall safety and fuel efficiency.

The ALICA plans for this work focus on a platoon scenario where the different vehicles' behaviors are organized in various modes [17]. A mode is a concept for structuring the system's overall behavior in different behaviors, and each of them is activated at a specific time. The behavior of each mode is then represented in terms of a state machine that captures the behavior of the system in a species modality, e.g., during the selection of a leader of the platoon, leaving a platoon, etc. Transitions among states can be triggered by timing constraints or external events and can lead the system to a different state or mode. Figure 1s shows a vehicle platooning scenario that involves different heterogeneous vehicles. Each vehicle is in a certain mode according to its behavior; we will describe the modes in more detail later. The communication among the vehicles is presented with dotted lines.

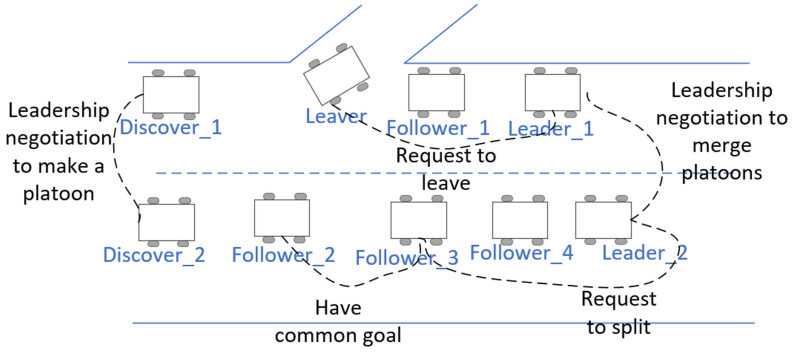

Fig. 1. Overall platoon system

As part of safety verification, we need to verify the agent's decisions, particularly when it collaborates with the other vehicles. An autonomous agent controlled by the ALICA program makes decisions based on not only its beliefs or goals but also the other agents. Our approach is to use model-checking to demonstrate that the agent always behaves according to the platoon requirements and never chooses options that end up in an unsafe situation. We verify

[1] https://www.uni-kassel.de/eecs/fachgebiete/vs/research/previous/impera.html.

the properties of the plan using the Uppaal model checker [4]. Unfortunately, designing multi-agents plans that control the agents to behave as expected is labor-intensive and time-consuming work. Moreover, the process of converting those plans into a model checking syntax to verify their properties while retaining the original characteristics of the plan is not only difficult but also requires consistency of designers. For example, a transition X that is used in plan A and plan B, when it is transferred to Uppaal, must be translated in the same way though it can be translated by different designers. Therefore, we have developed the A2U translator to solve that problem.

This paper is structured as follows: Sect. 2 presents a literature review on the verification of multi-agent systems and vehicle platooning. Section 3 describes all the modes in the ALICA plan for a platooning scenario and its requirements. The formal description of the ALICA program is presented in Sect. 4. In Sect. 5, we present the Uppaal model of vehicle platooning and properties that we checked on this model. Finally, conclusions are reported in Sect. 6.

2 Related Work

The verification for vehicle platooning is proposed in [8,9] in which the properties of the system are taken into account. Kamali et al. [8] evaluate individual autonomous decisions using the Agent Java PathFinder (AJPF) model checker and use Uppaal to verify timing behavior and the system. Similar to our approach, [9] verifies on-the-fly vehicle platooning abstracted in a unique genetic Uppaal model. However, our approach not only addresses verification of the collaboration and coordination for ALICA plans but also develops the A2U tool to map the ALICA plan into Uppaal syntax.

Answer set programming(ASP) has been used in [14] to verify abstract state machines with bounded model checking techniques. The purpose in [14] is to translate abstract state machines and linear time logic expressions to ASP and search for examples that violate the system properties that have to be validated. However, their focus is on validation, while we are focusing on verification. In [10], the authors use ASP to verify the ALICA program, but no property requirements have been carried out. Therefore, the unsafe plan can still be created. The approach in this paper addresses the verification of the platoon for different situations.

J. Campbell et al. [5] have presented a framework based on π-calculus for formal modeling of high-level decision making of platooning. In this work, the authors use hybrid automata to model closed-loop control systems. Similar to our ALICA program, the discrete and continuous aspects of the system are separated, though the verification properties are not considered in [5] to ensure the correctness of the high-level decision-making component.

The model for platooning driverless cars using an integrated formalism combining CSP and B is introduced in [6], where the system behavior is specified via the CSP and B formal methods. The vehicle behavior is modeled as a B machine, based on its stepwise refinement feature, and the communication models as a CSP controller. The focus, in [6] is on representing a correct model

of a real physical vehicle for platooning while our approach aims to verify the cooperation between vehicles in the automotive platoon. Similarly, El-Zaher et al. [7] present a compositional verification method for vehicle platooning, where feedback controllers and agent decision-making are mixed.

3 Multi-mode System

We modeled the ALICA plan for autonomous vehicles platooning, as shown in Fig. 2. The general purpose of partitioning a system in multiple modes, each of which describes a specific behavior of the system, leads to various advantages, such as reducing the software complexity and easing the addition of new features [15]. This plan can be considered multi-mode systems; if the environment changes, it switches to a corresponding model for adapting to the new conditions.

In the top-level plan, we have created the following modes:

- *Searching*: this is the first mode of the plan, where the vehicles want to search other vehicles that have the same goals to make a platoon or join an existing platoon;
- *Forming*: the two vehicles that want to form a platoon enter in this mode. To do that they decide which one will be the leader or follower of the platoon;
- *Leading*: the leader is the vehicle having the highest safety index in the platoon. Each vehicle shares its safety attributes with the other vehicles. Once the vehicle is a leader, it can steer the following vehicles, propagate information, keep track of the list of the followers, and accept new vehicles that want to join and manage the followers' leaving;
- *Following*: all the vehicles inhabiting in this mode drive in automated mode and follow the leader. A follower can receive information from the leader or transmit information to the other members of the platoon;
- *Splitting*: a platoon with at least four members and two of them have the common goal can split into smaller platoons. The new leader will be chosen from follower vehicles, which share the same goal. If the old leader no longer has any members after dividing the platoon, it will move to *dissolving* mode. Otherwise, it continues to lead the platoon with the remaining members.
- *Joining*: if the leader of an available platoon has found a *free* vehicle that wants to join, it sends a joining request to the "free" vehicle. This non-member needs to be accepted within a certain time interval;
- *Leaving*: all vehicles can request to leave platoon at any time, for example, when they have reached their goal. If a member of the platoon leaves, it must send a leaving request to the leader and leave only when it receives an authorization from the leader. When the leader leaves, the platoon will dissolve if there is only one member left and if the remaining members are greater than or equal to two and have the same common goal, they will choose the member with the highest safety index among them to be the new leader;
- *Dissolving*: a vehicle moves into the dissolving mode when (i) is a follower and it does not have a leader anymore or (ii) is a leader and it does not have followers anymore. From this it can either leave or go back to the *searching* mode and start a new platoon toward their goal;

– *Negotiation*: when a non-member vehicle wants to join an existing platoon, it has to negotiate the leadership with the current leader, either it becomes a follower or a new leader. The one with the highest safety attributes will always be the leader. The leaders of two or more platoons that want to merge also can trigger the leadership negotiation.

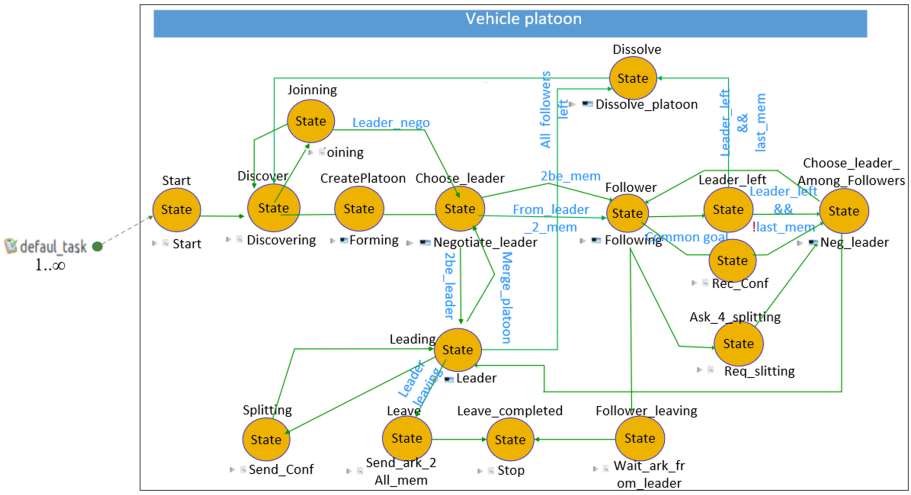

Fig. 2. ALICA plan for an autonomous platoon

4 A Language for Cooperative InteractiveAgents

ALICA is a far-reaching language to describe a sequence of actions of multi-agent systems. Its design is based on three principles, including domain independence, locality, and autonomy. Through calculating redundancy, ALICA can address the unreliable communication problem, which means all agents decide autonomously to solve mutual problems. With the necessary and available information, such as communication or action recognition, ALICA can recognize and tackle inconsistency in making the multi-agent systems' decision process. Thus, according to [13], ALICA may even work in unstable network conditions. These abilities are a crucial requirement in the domain of autonomous car driving.

All regarding agents' states are joined as a global state to execute an ALICA program. The ALICA program's hierarchical structure application allows it to overcome struggles when solving problems in a group. For any problems happening, solutions are planned and assigned to members. Besides, ALICA uses subplans to deal with sub-problems locally.

4.1 ALICA Syntax

An ALICA program includes multi-classed plans in which each plan consists of several finite state machines. $\mathcal{P}$ and $\mathcal{Z}$ represent the set of plans and states in a program, respectively. $\mathcal{L}$ is the language describing the belief base of each agent logically. The logic in $\mathcal{L}$ is not fixed, but we assume that its expression is comparable to the first-order language like C++, C#, or Java. Therefore, all conditions, for example, guards for transitions, are elements of L, and each element of $\mathcal{Z} \times \mathcal{Z} \times \mathcal{L}$ contains a transition τ.

There are two ways that ALICA uses to abstract from agents, by *tasks* and *roles*. An agent's role depends on its capability, and this assignment for each agent only changes when there is a change in team composition or its capabilities change. For example, due to being damaged, the current leader cannot continuously stay in a platoon. Hence, ALICA immediately chooses the members having the highest safety index in a platoon to become the new leader. That means ALICA changed the roles of a member from a follower to be a leader.

Tasks are used to annotate finite state machines of plans and restricted by the minimum and the maximum number of cardinalities of the assigned agents to ensure that they are capable of executing this task. For example,the finite state machine that is annotated with the task *Mer_Pla* in Fig. 3, only can be executed by two agents.

The appropriateness of a plan in a specific circumstance is estimated in two different ways. Initially, each plan ρ must satisfy a precondition $\text{Prep}(\rho) \in \mathcal{L}$, which must hold when the agents begin to execute a plan, and a run time condition $\text{Run}(\rho) \in \mathcal{L}$, which must hold during execution of a plan.

4.2 ALICA Semantics

ALICA uses a transition system [11] for its operational semantic. This transition system describes how an agent updates its internal states and when it can do, and hence how it progresses within an ALICA program. These internal states are mentioned as *agent configurations* and constrained by a set of axioms Σ_B.

Definition 1. *An agent configuration is a tuple* $(B, \Upsilon, \Xi, \theta, R)$*, where:*

- B *is the belief base of agents, it holds the current model of the environment – everything the agent believes to be true,*
- Υ *is the agent's plan base, it captures its current state*
- $\Xi \subseteq \mathcal{B}\times, \mathcal{Z}$*, is called the context of behaviors* b*. The agent executes* b *together with the state in which* b *is occupied.*
- θ *a substitution, it holds the current instantiations of plan and behavior parameters,*
- R *is a set of roles assigning to the agent based on its capabilities.*

The plan base of an agent is a set of triples (ρ, τ, z), showing that the agent is executing the behaviors in state z of task τ in plan ρ. There is at most one

triple for each plan in any plan base and it is reflected by a literal In (a,ρ, τ, z) in the belief base.

An ALICA program is structured by functions and relations between elements in a tuple $(\mathcal{P}, \mathcal{Z}, R, \mathcal{T}, \mathcal{W}, \mathcal{B}, Plantype, \rho_0, z_0, \tau_0)$ where $\mathcal{P}$ is a set of plans, $\mathcal{Z}$ a set of sates, R a set of roles, $\mathcal{T}$ a set of tasks, $\mathcal{W}$ a set of transitions, $\mathcal{B}$ a set of behavior.

For the specific ALICA program, the following function are used:

- States: $\mathcal{P} \rightarrow 2^{\mathcal{Z}}$, States($p$) denotes the states within a plan.
- Tasks: $\mathcal{P} \rightarrow 2^{\mathcal{T}}$, Task($\tau$) denotes the tasks of a plan.

When the ALICA program is implemented, each agent executes the state based on its configuration. Therefore, to inform an agent about the state of the team, we use the following predicates to is reflect in its belief base:

- In(a, p, τ, z) defined to hold iff $(p, \tau, z) \in$ PBase(a). This indicates the agent's belief about the internal state of other agents. For instance, $Bel_{a1}In(a3, vehicle\_platoon, defaultTask, leading)$ denotes that a1 believes the a3 to be committed to the *defaultTask* in a plan *vehicle_platoon* and currently inhabits state *leading*,
- HasRole(a, r), expressing that the role r is assigned to a,
- Succeeded(a, p, τ), return *true* value iff agent a successfully completed task τ in plan p,
- $Handle_f(p)$ and $Handle_f(b, z)$ which mean to hold if an agent might handle the failure of plan p or behavior b executed in state z, respectively,
- Failed(p, k), plan p failed k-times,
- Failed(b, z, k) in state z, behavior b failed k-times,
- *Alloc(z)* return *true* iff a task allocated to agents in state z is necessary.

Within ALICA, when the logical transformation rules change, the agent configuration also is changed. Explaining all rules is beyond the scope of this paper. The complete rules can be found in [12]. Therefore, we only present two rules applied in A2U translator:

The transition rule controls how and when an agent switches between two states, connected by a transition.

$$Trans\colon \frac{B \vdash \phi(p, \tau, z) \in \Upsilon(z, z', \phi) \in \mathcal{W}}{(B, \Upsilon, E, \theta, R) \rightarrow ((\mathrm{B}-\vartheta_b^-)+\vartheta_b^+, (\Upsilon-\vartheta_p^-)+\vartheta_p^+, \mathrm{E}', \theta, \mathrm{R})}$$

If an agent currently occupies in state z and it believes the condition annotating the transition to hold, an agent will follow an outgoing transition from z to z′. Moreover, this transition must not be a synchronization. The following transition only can 'fire' when all plans and behaviors of the agent, executed in the context of z, have finished or been stopped.

The Synchronized Transition Rule triggers a transition within a synchronization set when these transitions can be 'fired'. Intuitively, a synchronization models the start of cooperation, depending on the involved agents to act in a tiny time frame. The upper bound on the size of this time frame depends on the

latency, reliability of the communication, and the precision with which agents can track their teammates' intentions.

$$STrans\colon \frac{(\exists A \subseteq A)a \in A(\exists s \in \Lambda)(z, z', \phi) \in s \wedge \psi}{(B, \Psi, E, \theta, R) \rightarrow ((\mathrm{B}-\vartheta_b^-)+\vartheta_b^+, (\Psi-\vartheta_b^-)+\vartheta_b^+, E', \theta, R)}$$

Here, an agent will follow a synchronized transition, if and only if it can identify that it is part of a group of agents, called A, such that agent a believes that there is a mutual belief in A that all relevant conditions hold.

5 Uppaal Model and Platooning Verification

5.1 Uppaal Model

The Uppaal automaton models are translated from the ALICA plan of vehicle platooning by the A2U tool upgrade version. The A2U tool in [15] can only interpret the ALICA engine's simple plans, but features such as hierarchical states (plans can inhabit in a state), broadcast channels, or synchronized transitions were not allowed. However, the plan for an automotive platoon needs many features of the ALICA engine to function as the designer's requirements. For example, the leader uses the broadcast channel to transmit information to all members, or the followers use *synchronized transitions* to obligate all followers to move from *Follower* state to *Leader_left* state. Furthermore, the dynamic leader is an interesting property of the scenario, so just using a behavior like *Start* in *Start* state is not enough to handle all the cases. Instead, we have to use a hierarchical state and multiple tasks (as shown in Fig. 3) in a plan to deal with the problem.

The A2U tool to solve the multiple tasks in the ALICA plan performs two steps. First, A2U will translate the platoon vehicle plan to Uppaal like [15]. However, for states that contain plans, which have multiple tasks, these tasks will be translated and connected to the corresponding locations instead of creating a new template as in [13]. Second, A2U will delete the duplicate locations or transitions after translating. For example, the behavior that adds the old leader after negotiating between the new candidate and the leader of the existing platoon is identical to the process of merging two platoons. From there, it is possible to limit the explosion of state space.

The Uppaal model consists of nine modes shown in Fig. 4, corresponding to the modes designed on the ALICA plan (Sect. 3). Note that we did not annotate all modes for readability purposes. In order to ensure that the platoon vehicle in Uppaal is functioning as the plan made in ALICA Plan Designer, we have tested a few circumstances, e.g., if the joining time is timeout, does the vehicle candidate move from *joining* to the *discovery* mode? Or do two leaders of two platoons correctly negotiate leadership when they want to merge?

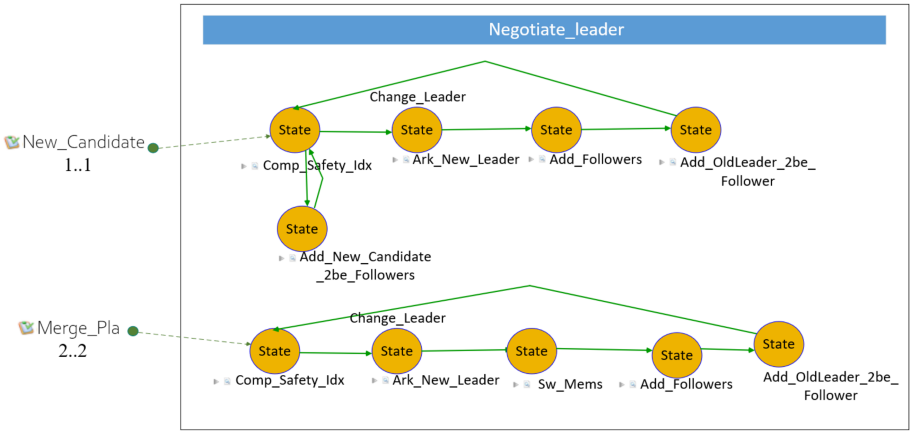

Fig. 3. Multi tasks in the ALICA plan

5.2 Platooning Verification

After translating the vehicle platooning plan, the timed automata of the platoon is now available to verify their properties using Uppaal. To create this multi-agents platoon, we used our robots for RoboCup Middle Size League (MSL). According to the MSL rules [3], robots can communicate with each other via wifi (also used for the platoon scenario), and each team has at most five members. Therefore, to facilitate the validation of the results from the Uppaal model checker, we analyze the global and timing properties of the multi-agent platoon composed of five members. In the following, we configure a few parameters, such as *joining time* and *leaving time* and set random values for each vehicle's safety index.

If an agent is a leader, then its safety index must be the highest in all other vehicles involved in its platoon. This property is formulated for agent v2 as follows (A□ φ expresses that φ should be true in all reachable states):

```
A□ (v2.leading) imply
forall (i:ID) v2.safety_idx[2] >= v2.safety_idx[i]
```

where v2 is the agent which is in *leading* location, i.e, the agent has the highest safety index, array *safety_idx* contains the safety index of all platoon members of v2.

The next interesting property is whenever the vehicle with the highest safety index starts joining in the platooning, it always ends up being a leader. To express this liveness property, we use the *lead to* or *response* form, written $\varphi \rightsquigarrow \psi$ stating that if φ is satisfied, then ψ will inevitably be satisfied.

```
v1.start --> v1.leading
```

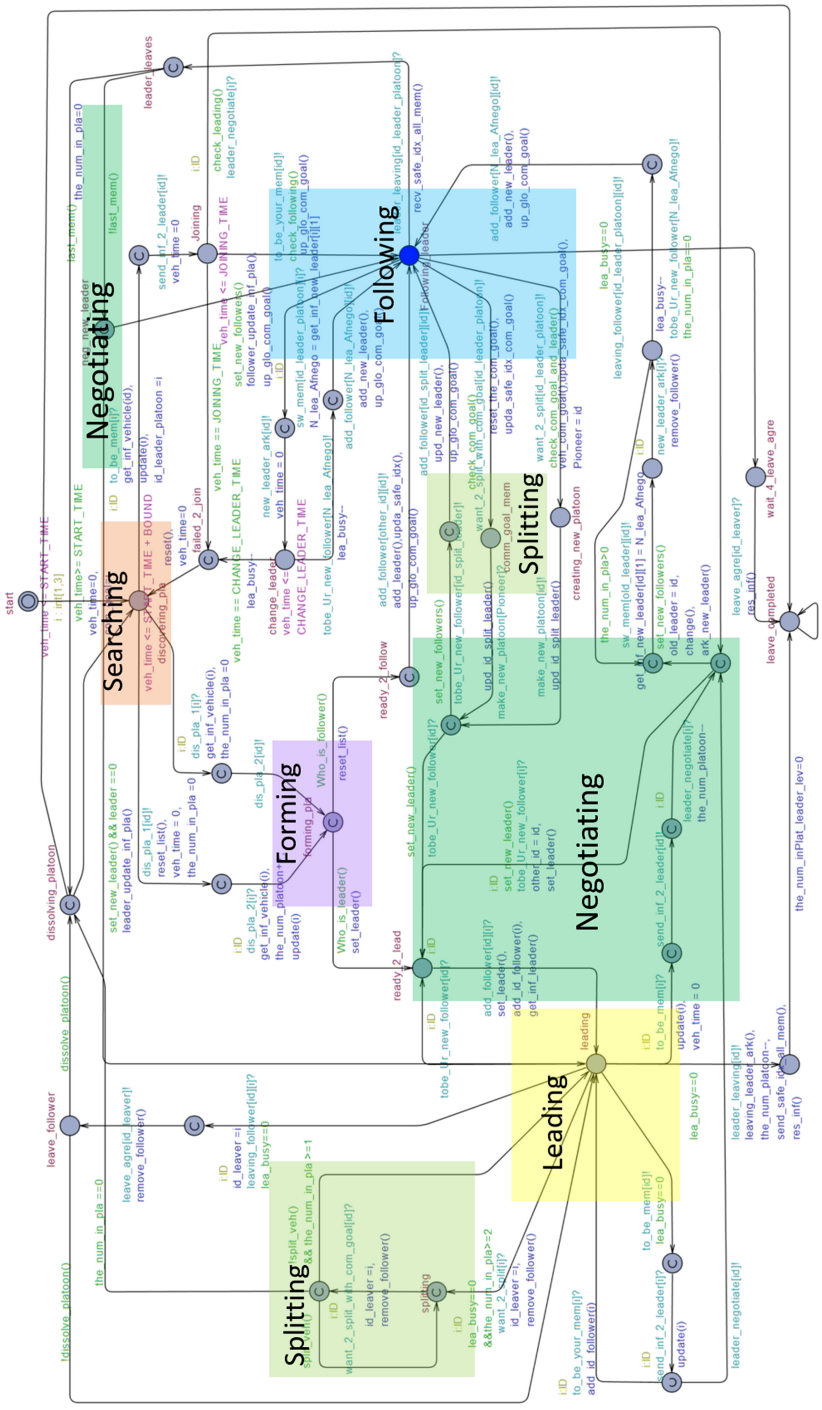

Fig. 4. Uppaal model

Similarly, we check if two platoons want to merge, the leader negotiation is triggered, and the one having a higher safety index will be the new leader.

```
v4.leader_nego && v5.leader_nego
--> v4.leading && v5.id_new_leader == 4
```

In this scenario, we assume that the vehicle v4 and v5 are the leaders of two platoon and v4 has a higher safety index than v5, variable *id_new_leader* indicates an identify number of the new leader.

To ensure that the platoon is disbanded when the leader (v4) leaves and only one member (v1) is left. At this time, the only or last member v1 must move to the *dissolving_platoon* region.

```
A□ (v4.leave_completed && v4.the_num_in_pla==1
&& (v1.id_leader_pla ==4 || v1.id_new_leader ==4))
imply (a1_leave_completed || a1.dissolving_platoon)
```

The platoon only can initiate to split, if the platoon has at least three members and two of them have the same goal.

```
A□ v1.splitting && v1.the_num_in_pla >=3
imply (v2.creating_new_platoon && v3.comm_goal_mem
&& a2.com_goal ==a3.com_goal)
```

When two platoon want to merger the leadership negotiation is triggered and then the old leader removes one by one its members, at the same time the new leader add one by one to its platoon. We are interested in verifying if the old leader accomplishes joining the platoon within an expected interval.

```
A□ (v2.time_out imply
(v2.chang_leader_time>=7 && v2.chang_leader_time<=21))
```

6 Conclusion and Future Work

Following the idea of verifying multi-agent plans during the modeling process, to save time and help the designers reduce modeling errors and improve the efficiency and consistency of the modeling process, we develop the A2U tool for mapping ALICA plans into Uppaal format. In this paper, we have used the formal verification for the ALICA plan vehicle platooning as the case study to demonstrate that our tool can handle the complicated plans.

As future work, we plan to refine the platoon model by adding ALICA's features such as *rules*, *plantypes* containing many plans, and verify more properties. Therefore, there are many issues remaining for the future. An obvious one is to continue developing the A2U translator so that A2U can map all features of the

ALICA program into Uppaal syntax. In the long term goal we plan to make our own model checker to verify hierarchical state machines without flattening them [2].

References

1. Cooperative dynamic formation of platoons for safe and energy-optimized goods transportation (2017). https://cordis.europa.eu/project/id/610990/en?format=pdf. Accessed 6 May 2020
2. Alur, R., Yannakakis, M.: Model checking of hierarchical state machines. ACM SIGSOFT Softw. Eng. Notes **23**(6), 175–188 (1998)
3. Asada, M., et al.: Middle size robot league rules and regulations for 2012 (2010). https://msl.robocup.org/wp-content/uploads/2018/08/msl-rules-2011-12-29.pdf. Accessed 01 July 2020
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
5. Campbell, J., Tuncali, C.E., Liu, P., Pavlic, T.P., Ozguner, U., Fainekos, G.: Modeling concurrency and reconfiguration in vehicular systems: A π-calculus approach. In: 2016 IEEE International Conference on Automation Science and Engineering (CASE), pp. 523–530. IEEE (2016)
6. Colin, S., Lanoix, A., Kouchnarenko, O., Souquières, J.: Using CSP——B components: application to a platoon of vehicles. In: Cofer, D., Fantechi, A. (eds.) FMICS 2008. LNCS, vol. 5596, pp. 103–118. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03240-0_11
7. El-Zaher, M., Contet, J.M., Gruer, P., Gechter, F., Koukam, A.: Compositional verification for reactive multi-agent systems applied to platoon non collision verification. Stud. Inform. Univ. **10**(3), 119–141 (2012)
8. Kamali, M., Dennis, L.A., McAree, O., Fisher, M., Veres, S.M.: Formal verification of autonomous vehicle platooning. Sci. Comput. Program. **148**, 88–106 (2017)
9. Mallozzi, P., Sciancalepore, M., Pelliccione, P.: Formal verification of the on-the-fly vehicle platooning protocol. In: Crnkovic, I., Troubitsyna, E. (eds.) SERENE 2016. LNCS, vol. 9823, pp. 62–75. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45892-2_5
10. Opfer, S., Niemczyk, S., Geihs, K.: Multi-agent plan verification with answer set programming. In: Proceedings of the 3rd Workshop on Model-Driven Robot Software Engineering, pp. 32–39 (2016)
11. Plotkin, G.D.: A structural approach to operational semantics (1981). https://web.eecs.umich.edu/~weimerw/2014-6610/reading/plotkin81structural.pdf. Accessed 04 July 2020
12. Skubch, H.: Modelling and Controlling of Behaviour for Autonomous Mobile Robots. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-658-00811-6
13. Skubch, H., Wagner, M., Reichle, R., Geihs, K.: A modelling language for cooperative plans in highly dynamic domains. Mechatronics **21**(2), 423–433 (2011)
14. Tang, C.K.F., Ternovska, E.: Model checking abstract state machines with answer set programming. Fundamenta Informaticae **77**(1–2), 105–141 (2007)
15. Van, T.N., Fredivianus, N., Tran, H.T., Geihs, K., Huynh, T.T.B.: Formal verification of alica multi-agent plans using model checking. In: Proceedings of the Ninth International Symposium on Information and Communication Technology, pp. 351–358 (2018)

16. Witsch, A., Opfer, S., Geihs, K.: A formal multi-agent language for cooperative autonomous driving scenarios. In: 2014 International Conference on Connected Vehicles and Expo (ICCVE), pp. 546–551 (2014)
17. Yin, H., Carlson, J., Hansson, H.: Towards mode switch handling in component-based multi-mode systems. In: Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering, pp. 183–188 (2012)