Formal Verification of the On-the-Fly Vehicle Platooning Protocol

Piergiuseppe Mallozzi[1,2(✉)], Massimo Sciancalepore[1,2], and Patrizio Pelliccione[1,2]

[1] Chalmers University of Technology, Gothenburg, Sweden
mallozzi@chalmers.se, massimosciancalepore@gmail.com
[2] University of Gothenburg, Gothenburg, Sweden
patrizio.pelliccione@gu.se

Abstract. Future transportation systems are expected to be Systems of Systems (SoSs) composed of vehicles, pedestrians, roads, signs and other parts of the infrastructure. The boundaries of such systems change frequently and unpredictably and they have to cope with different degrees of uncertainty. At the same time, these systems are expected to function correctly and reliably. This is why designing for resilience is becoming extremely important for these systems.

One example of SoS collaboration is the vehicle platooning, a promising concept that will help us dealing with traffic congestion in the near future. Before deploying such scenarios on real roads, vehicles must be guaranteed to act safely, hence their behaviour must be verified. In this paper, we describe a vehicle platooning protocol focusing especially on dynamic leader negotiation and message propagation. We have represented the vehicles behaviours with timed automata so that we are able to formally verifying the correctness through the use of model checking.

1 Introduction

Intelligent and connected vehicles will be key elements of future of transportation systems. Within these systems, vehicles will act as standalone systems and at the same time they will interact each other as well as with pedestrians, roads, signs and other parts of the infrastructure to achieve (even temporarily) some common objectives. Future transportation systems might be then seen as Systems of Systems (SoSs) [10] in which the boundaries will change frequently and unpredictably. Moreover, these systems will need to cope with different degrees of uncertainty both at the level of single constituent systems and the entire SoS. Intelligent transport systems promise to solve issues related to road congestion, environment pollution and accidents for a better and more sustainable future [2]. In order to increase safety, reduce traffic congestion and enhance driving comfort, vehicles will cooperate exchanging information among each other and with the surrounding environment as well.

In this paper, we focus on a specific scenario, namely on-the-fly and opportunistic platooning, i.e. an unplanned platooning composed of cars that temporarily join in an ensemble to share part of their journey. Platooning is one

I. Crnkovic and E. Troubitsyna (Eds.): SERENE 2016, LNCS 9823, pp. 62–75, 2016.
DOI: 10.1007/978-3-319-45892-2_5

of the promising concepts to help us dealing with traffic jams and at the same time to increase the overall safety while driving. A platoon consists of reducing the distances among following vehicles; it consists of a *leading* vehicle driving manually and one or more *following* vehicles automatically driving and following the leader one after another. This concept has been studied and applied especially in trucks for the transportation of goods [1] with the aim of reducing the impact with air and consume less fuel, but not as much work has been done regarding normal vehicles platooning. Each vehicle must be able to communicate with the others, or at least with the cars adjacent in the platoon. The communication is important because each vehicle needs to adjust the speed and the distance according to the other vehicles information. Also, the leader of the platoon is responsible for managing the overall platoon formation, by accepting new vehicles or responding to vehicles leaving.

Platooning is also a way towards autonomous vehicles since, except for the leader, the vehicles do not need human intervention during the travel journey. Since human intervention is no longer needed, all decisions must be taken autonomously by the vehicle, and this is a huge challenge for safety assurance. Consequently, on the one side the use of platooning promises to enhance safety, and on the other side safety is exposed to new threats and challenges. It is important to notice that nowadays most of the systems are guaranteed to operate correctly only in certain configurations and within the system boundaries. When these boundaries are removed and the system is exposed to unpredictable and uncontrollable scenarios and environments, safety guarantees no longer hold. This will be one of the greatest challenges of future autonomous and connected vehicles that will cooperate with other vehicles, pedestrians, roads, etc. in a SoS setting.

Although there are different levels of autonomy of vehicles[1], autonomous vehicles can be considered as particular self-adaptive systems [4] since they are capable of adapting themselves at runtime. A connected vehicle beside being self-adaptive is also open to interactions with other vehicles and other elements of the external environment. The unpredictability and uncontrollably of the environment hamper the complete understanding of the system at design time. Often uncertainty is resolved only at runtime when vehicles will face with concrete and specific instantiations of the pre-defined environment parameters. This implies that the certification process for safety has to be extended also to runtime phases.

In this paper, we focus on a platooning scenario where the different vehicle's behaviours are organized in various modes [16]. A mode is a concept for structuring the overall behaviour of the system into a set of different behaviours, each of them activated at different times according to specific circumstances. The behaviour of each mode is then represented in terms of a state machine that captures the behaviour of the system in a specific modality, e.g. during the

[1] The National Highway Traffic Safety Administration (NHTSA) has proposed a formal classification system based on five levels: "*U.S. Department of Transportation Releases Policy on Automated Vehicle Development. National Highway Traffic Safety Administration, 2013*".

selection of a leader of the platoon, leaving a platoon, etc. Transitions among states can be triggered by timing constraints or external events. A special transition can lead the system to a different mode: in this case the two states involved are *border states* of the modes. Figure 1 shows a vehicle platooning scenario that involves different heterogeneous vehicles. Each vehicle is in a certain mode according to its behaviour; we will describe the modes in more detail later. The communication among the vehicles is represented with dotted blue lines.

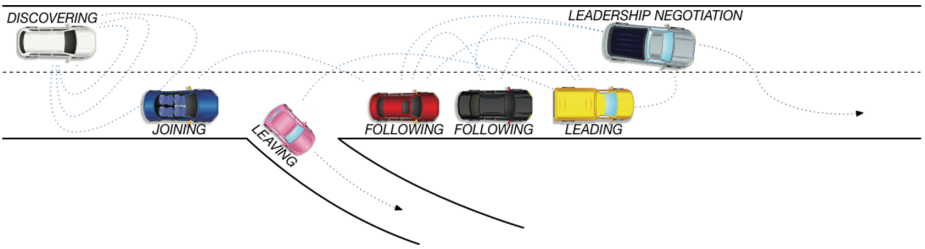

Fig. 1. Dynamic vehicle platooning scenarios. Each vehicle is in a certain mode according to its behaviour in the platoon. (Color figure online)

In this paper, we formally verify the on-the-fly vehicle platooning protocol through the use of the Uppaal [6] model checker. More precisely we verify the absence of deadlocks in the mode-switching protocol as well as other interesting properties.

The rest of the paper is structured as follows: Sect. 2 presents all the modes of our platooning scenario, Sect. 3 describes some parts of the Uppaal model, and Sect. 4 describes the properties we checked on our model. In Sect. 5 we show the results of a concrete simulation of our model in Uppal. Section 6 presents the results of the validation we performed through the use of the model checker Uppaal. Section 7 discusses works that are related to our work and finally Sect. 8 concludes the paper with directions for our future work.

2 Multi-mode System

Partitioning a system into multiple modes, each of which describing a specific behaviour of the system, is a common approach in system design. It leads to a series of advantages, such as reducing the software complexity and easing the addition of new features [16]. A self-adaptive system can be considered as a multi-mode system; if something happens in the environment, the system switches mode in order to adapt to the new conditions. This is the design strategy we follow in this paper.

We start by partitioning our system into different operational modes, recognizing different system behaviours. We have defined the different modes as a set of connected states with common behaviours. There are particular states that we

call *border states*: to pass from one mode to another, the system passes through these states. All the modes have one or more *border states* that allow the mode switching of the system. Switching from one mode to another means that the system is passing from one border state of the current mode to a border state of another mode. For each vehicle taking part in the platoon we have identified the following modes:

- *Discovering*: this is the entering mode of the vehicle that wants to take part in a platoon and searches for other vehicles that have the same goals (e.g. common destination).
- *Forming*: the first two vehicles that want to form a new platoon enter into this mode. To do that, they decide who will be the leading vehicle of the platoon.
- *Joining*: a vehicle has found an existing platoon and it wants to join it. The vehicle can be accepted in the platoon within a certain time interval;
- *Leading*: the vehicle with the best safety attributes is elected as leader of the platoon. We have assumed that each vehicle shares its safety attributes with the other vehicles. Once in this mode, the vehicle has to steer the following vehicles, propagate information, keep track of the list of the followers, accept new vehicles that want to join, and, finally, manage the leaving of the followers.
- *Following*: all the vehicles drive in automated manner and follow the leader. A follower can receive information from the leader and propagate it to the other members of the platoon. It also supports the changing of the leader and if the leader leaves then the vehicle goes into the discovering mode again.
- *Leaving*: all the vehicles can leave the platoon at arbitrary time. When the leader leaves, the platoon dissolves. When a follower leaves, it must advise the leader and receive acknowledgement.
- *Dissolving*: vehicle goes in the dissolving mode when (i) it is a follower and does not have a leader anymore or (ii) it is a leader and does not have followers anymore. From this mode, it can either leave or go back to the discovering mode and start a new platoon.
- *Negotiation*: when a new vehicle wants to take part of an existing platoon, either it becomes a follower or it has to negotiate the leadership with the current leader. The vehicle with the highest safety attributes will always be the leader. Leadership negotiation can also be triggered by two platoons that want to merge.

3 Uppaal Model Description

Our strategy to model the behaviour of the on-the-fly platooning is to build a generic Uppaal template that incorporates all the modes. This template can be then instantiated for each vehicle that will take part to a specific scenario. More precisely, this model can be instantiated by all the vehicles regardless of their role in the platoon. We can then simulate a variety of scenarios by tuning the

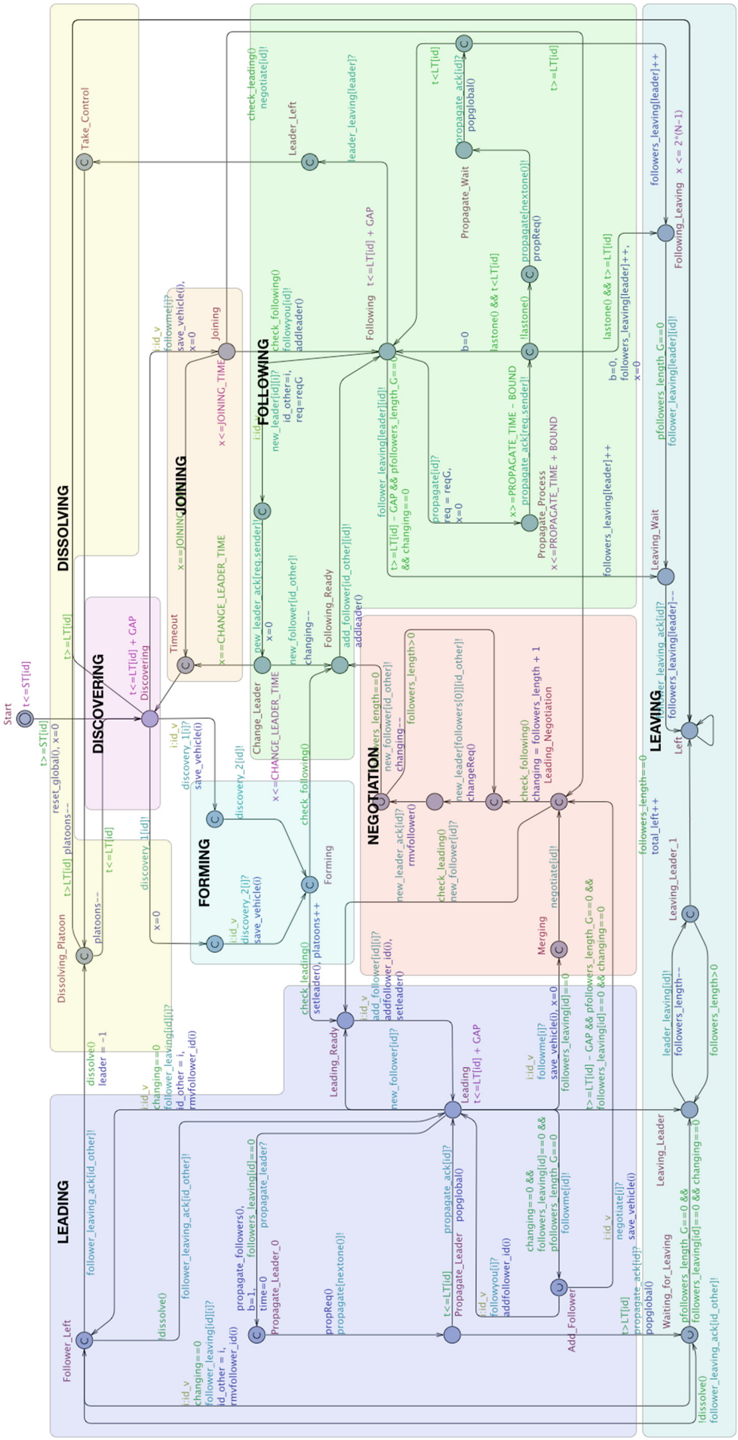

Fig. 2. Uppaal model with modes.

vehicles intrinsic properties. This solution is more scalable than having multiple models for different roles of the platoon (leader, follower) (Fig. 2).

The dynamic leader negotiation is a property of our scenario since we do not know who is going to be the leader beforehand. Furthermore, the leader can be changed during the platoon life. In order to this, we assume that each vehicle has associated a parameter representing its safety characteristics, called safety index, before it enters the platoon. Our models and protocol assure that the leader is always the vehicle with the highest safety index. The safety index it is just a value and it represents the overall safety score of the vehicle, the higher the better. We can assume that this value is calculated taking into consideration all safety-related parameters of the vehicle, either static ones such as the year of the vehicle, the size or dynamic ones taking into consideration the driver experience and the people on board.

In our model every vehicle starts from a *discovering* mode where it looks for other vehicles or platoons to join. In fact, the *formation* of a platoon can happen in different ways:

- Two vehicles negotiating with each other and forming one platoon with one leader and one follower. The two vehicles negotiate the leadership according to their safety index.
- One vehicle joining an existing platoon if there is already a formed platoon and the new vehicle is in discovery mode.
- Two existing platoons merging into one after the two leaders have performed a re-negotiation of their leadership.

If the joining of a platoon takes more than the pre-defined constant time (`JOINING_TIME`) to a vehicle, then it goes into discovering mode again. After the formation phase a vehicle can be either in *Leaving* or in *Following* state. The leader keeps track of all its followers at any time by listening to new joining or leaving requests. It can also send messages to all its followers. Message propagation can happen in two ways:

- The leader can reach all its followers and communicate with them all.
- The leader sends a message to the follower immediately behind him and then the message will propagate from follower to follower until reaching the last vehicle in the platoon.

We also take into consideration the propagation time that is needed for a vehicle to pass on the message to the next vehicle. The time is, in fact, crucial for safety-related messages; we want to be sure that the message reaches the whole platoon in the shortest time. We guarantee this by formulating and verifying time-related properties on the message propagation as described in the section below. Another feature of our model is the dynamic leader negotiation also after the platoon has been formed. This can happen in two cases:

- Two platoons want to merge. The platoon with the leader having the highest safety index will take the leadership while the other leader activates the joining procedure to the new leader that has to be completed in `CHANGE_LEADER_TIME` and afterward it becomes a follower of the newly elected leader.

– A vehicle wants to join an existing platoon and it has a safety index higher than the platoon leader. The current leader passes its followers to the new leader and itself becomes a follower.

4 Requirement Specifications Verified with Model Checking

The main purpose of a model-checker is to verify the model with respect to a requirement specification. With the timed-automata representation of the system, it is possible to verify safety and behavioural properties of our model such as the absence of deadlocks or the propagation of a safety-critical message within a certain time. Like the model, the requirement specification (or properties) to be checked must be expressed in a formally well-defined and machine readable language. Uppaal utilizes a subset of TCTL (timed computation tree logic) [3,7]. The path formulae $A <> \varphi$ (or equivalently $A <> \varphi = \neg E[\,]\neg\varphi$) expresses that φ will be eventually satisfied or more precisely that in each path will exist a state that satisfies φ. The path formulae $A[\,]\varphi$ expresses that φ should be true in all reachable states.

In order to verify the safety requirements, we have to build a scenario first, i.e., a particular instantiation of the system. Our model is made in order to be configured according to the scenario we want to verify. We first need to set the *number of vehicles* involved and for each vehicle we need to configure few parameters such as its *arrival time*, *leaving time*, and *safety index*. We have automated the configuration process by assigning random values to these values as we explain in the following section. The automation process involves also the properties that are tuned according to the scenario we want to verify. Once we have configured our scenario we can formally verify the following properties:

– *Property 1: If a vehicle is in the leading mode then its safety index is higher then all other vehicles involved in the platoon.*
Assuming a scenario where `Vehicle 3` has the highest safety index the instantiated property would be expressed as:

$$\texttt{A[] (Vehicle(3).Leading} \implies \forall\texttt{(i:id\_v) S[3]>=S[i])}$$

– *Property 2: The propagation of a message from the leader to the last follower happens in a bounded amount of time.*
The time in which the propagation has to happen varies according to the size of the platoon and the maximum acceptable delay is kept by the predefined variable `MAX_PROP_DELAY`.

$$\texttt{A[](b==1} \implies \texttt{time<=MAX\_PROP\_DELAY)}$$

A boolean variable b and a clock variable *time* are two global variables that are used to measure the propagation time from when a message is fired. In order to measure that, when a message starts propagating, the variable `b` is

set to 1 while `time` is reset. The properties assures that `time` will always be inferior to the constant `MAX_PROP_DELAY` while `b` is kept to 1. The variable `b` will be reset when the message has reached the last follower of the platoon.

– *Property 3: For each vehicle in the following state exists at least one vehicle in leading mode.*

$$\texttt{A[]}(\forall\texttt{(k:id\_v) Vehicle(k).Following} \implies \exists\ \texttt{(i:id\_v) Vehicle(i).}\mathit{All\_Leading\_States})$$

Since the leading mode is formed by a series of states this property is verified by including all the states of the leading mode (as a series of `or` elements). We did not write the full property for readability purposes.

– *Property 4: Whenever the vehicle with the highest safety index starts participating in the platooning it will eventually become the leader.*
Assuming that Vehicle 1 is the one with the highest safety index, the property becomes:

$$\texttt{Vehicle(1).Start} \implies \texttt{<> Vehicle(1).Leading}$$

– *Property 5: For all the path, the vehicle with the highest safety index goes into the leading state.*
Assuming the Vehicle 1 is the one with the highest safety index, the property becomes:

$$\texttt{A<> Vehicle(1).Leading}$$

– *Property 6: All vehicles will eventually leave the platoon.*
Since all the vehicles have a leaving time we can verify that:

$$\texttt{A<>}\ (\forall\texttt{(i:id\_v) Vehicle(i).Start} \implies \forall\texttt{(k:id\_v)Vehicle(k).Left})$$

– *Property 7: If a leader leaves the platoon then all its followers leave as well.*

$$\texttt{A[]}((\exists\texttt{(i:id\_v) Vehicle(i).Leaving\_Leader} \wedge \forall\texttt{(k:id\_v) Vehicle(k).Following}) \implies \forall\texttt{(j:id\_v) Vehicle(j).Dissolving\_Platoon})$$

– *Property 8: The model is deadlock free.*
Finally, this property assures that for all possible paths there are no deadlocks in our model:

$$\texttt{A[]}\neg\ \texttt{deadlock}$$

In Sect. 6 we present the verification times of the properties described above. We have noticed that properties apparently very similar require a very different amount of processing time in order to be verified. For example, both properties 4 and 5 verify the leadership of the vehicle with the highest safety index. Property 5 is always verified in less than 1 second, with the time increasing linearly with the number of vehicles. Property 4, instead, can take up to hundreds of seconds with an exponential increase with respect to the number of vehicles.

5 Simulation

Latest versions of Uppaal offer the possibility to perform a concrete simulation of the model. It is a verification tool that enables examination of the dynamic executions of a system. The simulation is based on concrete traces, e.g., one can choose a specific time to fire a transition. The tool helps to see at which time a transition can be fired. We have modeled some transition to fire with a uniform probability distribution. For example, in the propagation of the message, the transition will fire somewhere between `PROPAGATE_TIME-BOUND` and `PROPAGATE_TIME+BOUND` time units. We have used these time constraints to verify time properties based on the worst case scenarios when a message has to be propagated from the leader throughout the entire platoon.

In order to perform a simulation, we have to configure our model specifying parameters such as the number of vehicles, starting times, leaving times, and safety indexes. Each vehicle is an instance of the general vehicle template and by launching the simulation we can see how the vehicles interact with each other. All instances start from the same state and as the time flows Uppaal randomly selects which edge to fire among the available ones of each state. Some edges have guards and invariant in order to model the time of the transition from one state to another as a uniform probability distribution.

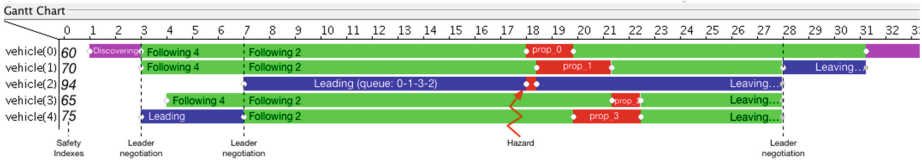

Fig. 3. Concrete simulation with Gantt Chart in Uppaal. (Color figure online)

Figure 3 shows the Gantt chart of a simulation. The horizontal axis represents the time span and in the vertical axis the list of vehicles instantiated in the simulation. A vertical line is used to represent the current time (which corresponds to the one displayed in the Simulation Trace-combo box). Horizontal bars of varying lengths and colours represent the different modes of the vehicles. Due to the limited amount of colours we are only able to show a limited amount of modes, specifically: discovering (purple), leading (blue), and following modes (green).

In the simulation showed in Fig. 3 we can see 5 vehicles participating in the platooning, each with a different safety index. `vehicle0` starts first stays in the discovering mode until other vehicles enter in the platoon. When `vehicle1` and `vehicle4` enter, the three vehicles perform a leader negotiation and `vehicle4` goes starts leading the platoon since it has the highest index. At time 4 `vehicle3` joins the existing platoon until `vehicle2` comes into play and renegotiate the leadership with `vehicle4` and so on. It is also interesting to see the message

propagation of a hazard from the leader to all its following vehicles (marked in red).

6 Verification Results

The simulation shown in Fig. 3 refers exclusively to a particular scenario. In this section, we instead report the results of an exhaustive verification that we performed on a number of different scenarios. This is obtained by automating the verification process with an external script that is able to generate different scenarios by changing the *number of vehicles* involved in the platoon and by randomly selecting independent variables within each vehicle, such as:

- *Arrival time*: the arrival time of a vehicle;
- *Leaving time*: the leaving time of a vehicle;
- *Safety index*: the safety index of a vehicle.

We are then able to verify all the properties described in Sect. 4 with a number of vehicles from 2 to 5 and for each vehicle configuration we run 100 tests with random scenarios. The height properties are verified by each generated configuration.

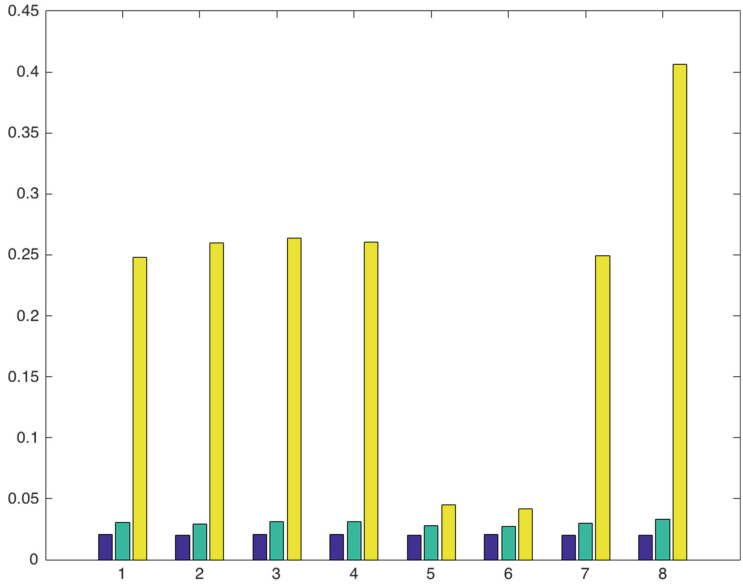

Fig. 4. Average verification times for 100 iterations. X-axes represent the property being verified. Y-axes the time to verifying it (in seconds). 2-3-4 vehicles scenario respectively

The script generates different models of the system based on a progressive number of the vehicles N and random values of some attributes. It executes

two big loops, one to change the random values and one to increment the number of vehicles N. Thanks to the standalone Uppaal verifier, the script verifies the above-mentioned properties with random attribute values of all the models generated. If one property is not satisfied, the standalone verifier generates the counterexample, which is useful to understand why the property is not satisfied. Counterexample files can be open within the GUI of Uppaal. In the end, the script generates a report of the verification, i.e., a text file that traces all the properties, both if they are satisfied or not.

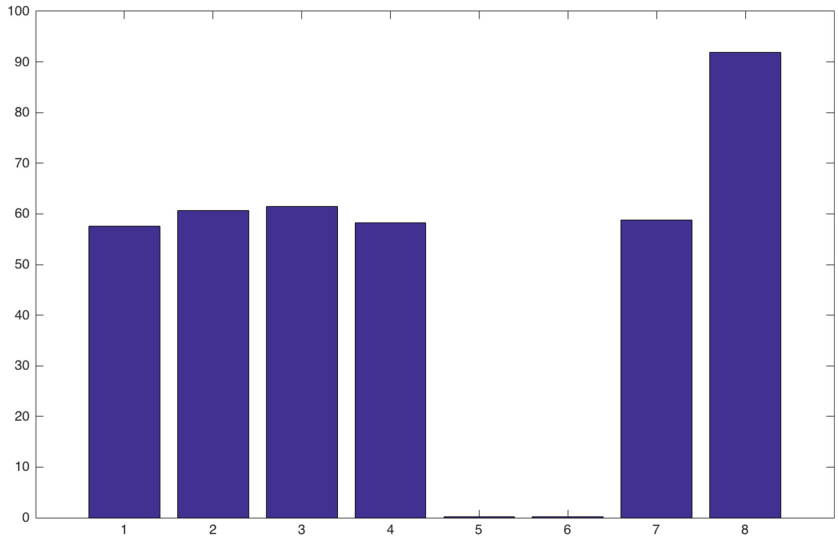

Fig. 5. Average times of 100 iterations for verifying the properties 5 vehicles

Figure 4 reports the time required to verify the 8 properties. The time shown in the figure is the average time required in 100 iterations. Since the time to complete the verification is exponential with respect to the number of vehicles the figure shows the time required by configurations of 2, 3, and 4 vehicles for verifying the 8 properties. For readability purpose, the verification time for configurations of 5 vehicles is not shown in the figure and the average times for 100 iterations are shown in Fig. 5. As we can see from the figure properties 5 and 6 have times comparable with the verifications times of 2, 3 and 4 vehicles. In fact, these two properties scale linearly while the others scale exponentially.

We have seen how changing the number of vehicles affects the verification time although these change a lot also for every configuration taken into consideration. Within the same number of vehicles, we have performed 100 iterations assigning random values to the vehicle attributes. Figure 6 shows how the verification time of a single property with a 5 vehicles configuration is affected by the random assignment of the vehicle attributes.

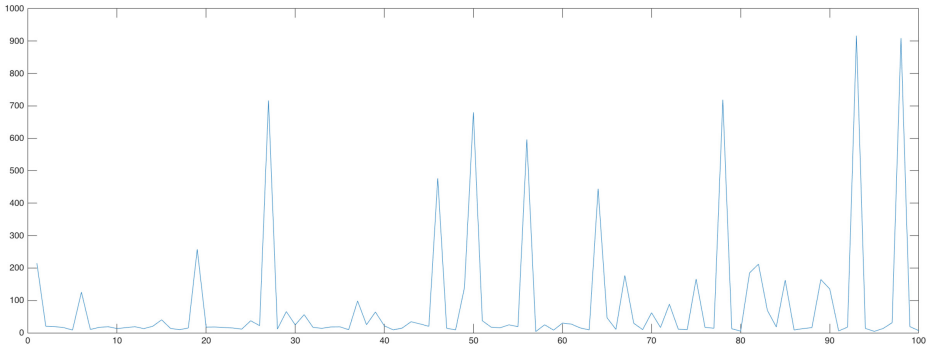

Fig. 6. Verification times of the deadlock free property for 5 vehicles scenario in 100 iterations.

7 Related Works

Kamali et al. [9] have also investigated the verification of vehicle platooning representing it as a multi-agents system. They verified the behaviour partly on the actual agent code and partly with Uppaal with timed-automata abstractions by using two different models, one for the follower and one for the leader.

One of the main challenges in open and self-adaptive systems is to certify that the system is always in a safe state. Since safety cannot be completely evaluated and assured at design time, at least part of the safety assurance must be shifted at run-time. The first ideas for certifying safety at runtime were introduced by Rushby [13,14]. He proposes an initial idea to certification based on formal analysis at runtime; however much work must be done to produce a solution that can be used concretely.

A promising approach to deal with safety certification at runtime is ConSert [15]. ConSert introduces the idea of *Conditional Safety Certificates* to facilitate the certification of open adaptive systems. Each subsystem is certified by a modular safety certificate based on a contract-like approach. The evaluation and the composition of the modular certificates happen at runtime. This framework offers flexibility as allows designers to specify safety through variable safety-certificates. Within the approach, all the configurations that a component of the system can assume must be predefined at design time in order to be certified "safe" at runtime. It allows emergent adaptive behaviours only if they can be tamed in certain boundaries with the concept of safety cages. Fully emergent behaviours are not possible to certify with ConSert hence ensuring safety in these cases is a much more difficult problem. A possible research direction can be investigating the theoretical assume-guarantee framework proposed in [8]. This framework allows one to efficiently define under which conditions adaptation can be performed by still preserving desired properties. The framework might provide the infrastructure to automatically calculate at runtime which properties are verified in specific scenarios. For instance, this might suggest excluding some

vehicles from the platooning since their inclusion might compromise important properties.

Regarding the automotive domain a more practical approach is the one proposed by Kenneth Östberg and Magnus Bengtsson [11]; they deal with run-time safety by extending the AUTomotive Open System Architecture (AUTOSAR [5]). Claudia Priesterjahnr et al. [12] tackle the runtime safety problem at a component level performing a runtime risk analysis. When a system is trying to connect to another system (for example in a platoon) it computes all reachable configurations and, for each of them, it computes the hazard probabilities at runtime in order to judge whether the configuration is safe or not.

8 Conclusion

In this paper, we have presented the formal verification of on-the-fly vehicle platooning. We have modeled the vehicle behaviours with timed-automata so that we were able to verify the correctness of the protocol with model checking. We were able to verify that some properties always hold for a different number of vehicles each with random attributes. All the vehicles are modeled with a unique generic Uppaal model that can be instantiated for each specific vehicle. In this way, it is possible to simulate different scenarios and the verification is easily scalable to more vehicles. Each scenario has been generated with a script, which changes parameters such as the number of vehicles and the attributes for each vehicle and then it verifies that all the properties hold. We have focused our attention only to some interesting part of the model such as the dynamic leader negotiation and the message propagation of the vehicles leaving other parts to be further exploited. As future work, we plan to refine our model by releasing some assumptions made during the creation of the model and verifying more properties. As a long term goal, we plan to experiment with the protocol by using a set of miniature vehicles.

Acknowledgement. This work was partially supported by the NGEA Vinnovaproject and by the Wallenberg Autonomous Systems Program(WASP).

References

1. Current State of EU Legislation - Cooperative Dynamic Formation of Platoons for Safe and Energy-optimized Goods Transportation. http://www.companion-project.eu/wp-content/uploads/COMPANION-D2.2-Current-state-of-the-EU-legislation.pdf
2. Intelligent transport systems - Innovating for the transport of the future. http://ec.europa.eu/transport/themes/its/index_en.htm
3. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, LICS 1990, pp. 414–425. IEEE (1990)
4. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Self-Adaptive Systems. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013)

5. Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., Lange, K.: Autosar-a worldwide standard is on the road. In: 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, vol. 62 (2009)
6. David, A., Behrmann, G., Larsen, K.G.: A tutorial on uppaal 4.0, 28 November 2006
7. Henzinger, T.A., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Inf. Comput. **111**(2), 193–244 (1994)
8. Inverardi, P., Pelliccione, P., Tivoli, M.: Towards an assume-guarantee theory for adaptable systems. In: Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2009, pp. 106–115. IEEE Computer Society, Washington, DC (2009)
9. McAree, O., Fisher, M., Kamali, M., Dennis, L.A., Veres, S.M.: Formal verification of autonomous vehicle platooning, 5 February 2016
10. Nielsen, C.B., Larsen, P.G., Fitzgerald, J., Woodcock, J., Peleska, J.: Systems of systems engineering: basic concepts, model-based techniques, and research directions. ACM Comput. Surv. **48**(2), 18:1–18:41, September 2015
11. Östberg, K., Bengtsson, M.: Run time safety analysis for automotive systems in an open and adaptive environment. In: SAFECOMP 2013-Workshop, NA, September 2013
12. Priesterjahnr, C.: Runtime safety analysis for safe reconfiguration, pp. 1–6, June 2013
13. Rushby, J.: Just-in-time certification. In: 12th IEEE International Conference on Engineering Complex Computer Systems, pp. 15–24. IEEE (2007)
14. Rushby, J.: Runtime certification. In: Leucker, M. (ed.) RV 2008. LNCS, vol. 5289, pp. 21–35. Springer, Heidelberg (2008)
15. Schneider, D., Trapp, M.: Conditional safety certification of open adaptive systems. ACM Trans. Auton. Adapt. Syst. **8**(2), 1–20 (2013)
16. Hansson, H., Hang, Y., Carlson, J.: Towards mode switch handling in component-based multi-mode systems. In: Proceedings of 15th International ACM SIGSOFT Symposium on Component Based Software Engineering, CBSE 2012, Bertinoro, Italy, pp. 183–188, June 2012