Architecting Systems-of-Systems of Self-driving Cars for Platooning on the Internet-of-Vehicles with SosADL

Flavio Oquendo(✉)

IRISA – UMR CNRS 6074, Univ. Bretagne Sud, Vannes, France
flavio.oquendo@irisa.fr

Abstract. A Software-intensive System-of-Systems (SoS) is architecturally designed to exhibit emergent behavior from the interactions among independent constituent systems. With the upcoming generation of self-driving vehicles, an important case of emergent behavior is vehicle platooning. In a platoon, a group of vehicles (which is dynamically formed) safely travel closely together like in a convoy. It requires, on the one hand, that each vehicle in the platoon control its velocity and the relative distance to the vehicle in front of it for avoiding rear collision and, on the other hand, that vehicles coordinate for enabling other vehicles to dynamically join or leave the platoon. This paper investigates the mediated approach for architecting a platooning of self-driving vehicles, with SosADL, a novel SoS Architecture Description Language (ADL) enhanced with broadcasting for the Internet-of-Vehicles (IoV). In particular, it demonstrates how architectural mediators expressed with broadcast constructs of *SosADL for IoV* supports platooning architecture descriptions through an excerpt of a real application for architecting platoons of Unmanned Ground Vehicles (UGVs). This novel approach is supported by an integrated toolset for SoS architects.

Keywords: Software architecture · Self-driving vehicle platooning · Internet-of-Vehicles (IoV) · Systems-of-Systems (SoS) · Broadcasting · SosADL

1 Introduction

Definitely, a key facet of the design of any software-intensive system, being a single system or a System-of-Systems (SoS) [19], is its software architecture, i.e. the fundamental organization of the system embodied in its constituents, their relationships to each other, and to the environment, and the principles guiding its design and evolution, as defined by ISO/IEC/IEEE 42010 [12].

In the case of single systems, the software architecture is described in terms of components, connectors binding together these components, and their configurations [20]. The resultant system behavior is said to be aggregative, for instance, like the behavior of a car that is the result of the sum of the behaviors of its components (e.g. the moving behavior of a car is the resultant of the engine that applies a torque on the wheels turning them forward, the wheels push backwards on the road surface and, in reaction, the road surface pushes back in a forward direction).

Published by Springer Nature Switzerland AG 2020
A. Casaca et al. (Eds.): IFIPIoT 2019, IFIP AICT 574, pp. 3–20, 2020.
https://doi.org/10.1007/978-3-030-43605-6_1

Differently, in the case of SoSs, the software architecture is described in terms of constituent systems, mediators for enabling interaction among the constituents, and their coalitions [27]. The resultant system behavior is said to be emergent [16], like the behavior of a platoon of cars which results from the interactions of its constituent cars.

In a platoon, a group of vehicles (which is dynamically formed) safely travel closely together like in a convoy [13]. It requires, on the one hand, that each vehicle in the platoon controls its velocity and the relative distance to the vehicle in front of it for avoiding rear collision and, on the other hand, that vehicles coordinate for enabling other vehicles to dynamically join or leave the platoon.

Note that a car is not designed to have the platooning behavior, this emergent behavior "appears" as the result of the interactions among multiple cars. Nevertheless, the car needs to have the capabilities required for participating in platoons.

Nowadays, the Internet-of-Things (IoT) enables the engineering of SoSs, which are opportunistically constructed for achieving specified missions in specific operational environments [10, 38]. In particular, in the subset of IoT where "things" are predominantly connected vehicles (i.e. mobile "things"), the so-called Internet-of-Vehicles (IoV) [14], the challenge is to coordinate different vehicles for performing together, through emergent behavior, traffic-related missions, especially platooning. In the IoV, in a platoon, two or more self-driving vehicles are connected together in convoy using automated driving support and, possibly, wireless connectivity.

There are two main kinds of platoons of self-driving vehicles [41]: (i) platoons of stand-alone self-driving cars: they are formed and managed based only on the local sensing and actuating capabilities of each self-driving car for controlling its velocity and the relative distance to the vehicle in front of it; (ii) platoons of connected self-driving cars: they are formed and managed based, on the one hand, on the local sensing and actuating capabilities of each self-driving car and, on the other hand, on inter-vehicle communication for coordinating movements with neighboring vehicles.

Conceiving Software Architecture Description Languages (ADLs) has been the subject of intensive research in the last 25 years resulting in the definition of several ADLs for modeling initially static architectures, then dynamic architectures of (often large) single systems, and presently evolutionary architectures of SoSs [20]. However, none of the existing ADLs has the expressive power to describe the evolutionary architecture of opportunistic SoSs on the Internet-of-Vehicles [27, 36], which requires for a car to dynamically discover which other cars are in its neighborhood as well as to dynamically create a communication channel with a specific car for coordinating maneuvers while driving, such as in platoons and in crossroads.

The corresponding challenge in the architectural design of SoSs on IoV is to conceive concepts and mechanisms for describing how an SoS architecture is able to create, on the fly, and maintain emergent behaviors from connected vehicles, where the actual vehicles are not known at design time.

To fill this gap, we have enhanced SosADL [25, 32], a novel ADL specially conceived for formally describing the architecture of software-intensive SoSs, based on supervenience principles [31], with novel features for supporting SoS architecture description on the IoV, i.e. broadcast for discovering vehicles in the neighborhood and unicast for communicating over a communication channel with a specific vehicle.

SosADL for IoV brings contributions beyond the state-of-the-art to the formalization of SoS architectures exposing emergent behaviors relying on connected self-driving vehicles.

The remainder of this paper is organized as follows. Section 2 refines the notion of IoV and of vehicle platooning on the IoV. Section 3 presents how SosADL was enhanced to meet the needs of the IoV, issuing the novel *SosADL for IoV*. Section 4 introduces a field study encompassing vehicle platooning on the IoV and demonstrates how the enhanced SosADL is applied to IoV, through an excerpt of a real application, focusing on the platooning of a fleet of self-driving vehicles. In Sect. 5, we outline the implemented toolset for *SosADL for IoV*. In Sect. 6, we present the validation of *SosADL for IoV*. In Sect. 7, we compare *SosADL for IoV* with related work. To conclude, we summarize, in Sect. 8, the main contributions of this paper and outline ongoing and future work.

2 Internet-of-Vehicles and Vehicle Platooning

Let us introduce the notion of Internet-of-Vehicles and its underlying network technology, i.e. the VANET (Vehicular Ad-hoc Network).

VANETs apply the principles of Mobile Ad-hoc Networks (MANETs), i.e. the spontaneous creation of a wireless network for data exchange, to the domain of vehicles. They enable thereby the creation of the IoV, relying on both V2V communication between cars and V2I between cars and the roadside infrastructure (and more broadly on V2X, i.e. vehicle-to-everything).

Especially, VANET provides a short-range communication technology that enables vehicles to exchange information several times per second, about position, speed, acceleration, and braking. Vehicles equipped with VANET are thereby able to identify possible risks within ca. 300 m and take automatic collision-avoidance actions or alert their drivers, possibly assisted by fog/cloud computing accessed via infrastructure.

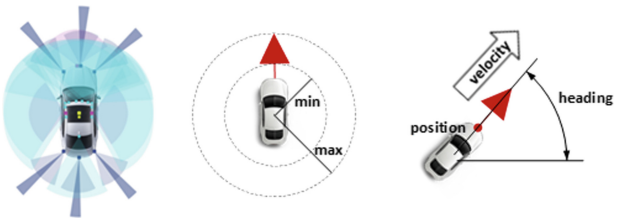

Fig. 1. Self-driving vehicles: wireless connectivity and automated driving support

As depicted in Fig. 1 (left), the upcoming generation of self-driving vehicles provides sensors, including radars and lidars, for sensing information from the environment and from other cars, processing this information, feeding it to drivers as well as communicating it to other cars, the infrastructure and, in general, with any "thing".

As also depicted in Fig. 1 (middle), there are two critical zones around a vehicle: the minimal safe separation from other vehicles and in general for any kind of obstacle

and the maximal separation (the perception range of the sensors and V2V communication enabling to coordinate with other cars and possibly the infrastructure). Note that the relative distance between two vehicles is determined using a radar or lidar (light detection and ranging sensor, usually with rotating laser beams; it measures the distance to a neighboring vehicle by illuminating that vehicle with pulsed laser light and measuring the reflected pulses).

As furthermore depicted in Fig. 1 (right), self-driving vehicles are equipped with Global Positioning System (GPS) for positioning and controlling heading and speed determining its direction and velocity. These data are generally transmitted via VANETs to the neighboring vehicles and the infrastructure. In addition to computing real time position in terms of GPS coordinates, a connected vehicle may also share the GPS coordinate with other vehicles.

VANETs make possible to set up and maintain vehicle platoons, as depicted in Fig. 2. Each vehicle communicates with some other vehicles in the platoon, in particular the ones closest in front of and behind it, but also via multihop routing to others. In a platoon, there is a "leader" vehicle that controls the speed and heading of the platoon, which drives to the destination, and "follower" vehicles (with matched acceleration and braking) that respond to the leading vehicle movements. Note that a "leader" may be explicitly or opportunistically determined.

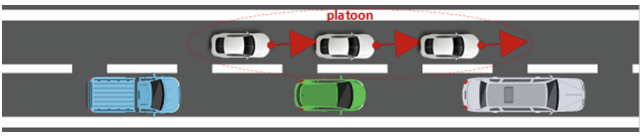

Fig. 2. Vehicles in platoon: one leader and many followers

Platooning is the process of cars autonomously following a leader to form a road convoy. It requires that each vehicle in the platoon control its velocity and the relative distance to the vehicle in front of it, and possibly also to the one behind of it. For supporting maneuvers, with vehicles joining/leaving the platoon, VANETs are used.

Vehicle platooning has several advantages: reduced risk of accidents, greater fuel economy due to reduced air resistance, higher efficiency of the vehicles and increased capacity of the highways. However, this comes with challenging issues to be considered: rear collision must be avoided in case of emergency braking, each vehicle in the platoon must have stable dynamics, and the platoon as whole must have string stability (i.e. if a deviation occurs w.r.t. the desired distance between the virtual leader and the first follower, this error should decrease towards the rear of the platoon and not the opposite). Vehicles coordinate using multi-hop routing in VANETs to ensure the string stability of the whole platoon.

By considering self-driving vehicles, it is possible to create a vehicle platoon that can travel autonomously. The platoon of vehicles travels in general on a single lane by using a longitudinal coordination strategy for each vehicle, as well as considers cases in which a vehicle has to join or leave the platoon, involving the lateral coordination of the vehicle. The platoons may also need to change lane.

3 Enhancing SosADL for the Internet-of-Vehicles

SosADL was conceived to overcome limitations of existing ADLs by providing the expressive power to describe the architectural concerns of software-intensive SoSs [29], and in particular to enable the description of emergent behaviors [30].

The core architectural concepts are the one of *system* to represent the constituents, the one of *mediator* to represent the enforced interactions among constituents, and the one of *coalition* to represent their formation as an SoS.

In SosADL, SoS architectures are represented in abstract terms (as the concrete systems which will become constituents of the SoS are not necessarily known at design-time; for instance, in vehicle platoon, the SoS architect does not know at design-time which vehicles will participate in a platoon).

Afterwards, the defined abstract architecture will be evolutionarily concretized at run-time, by identifying and incorporating concrete constituent systems; for instance, in the case of a platoon, cars which become platoonmates are discovered at run-time (see [9] for details on the automated synthesis of concrete SoS architectures from SosADL descriptions).

3.1 Extending SosADL with Digital Twins of "Things"

For extending SosADL for the IoV, we first extended SosADL for the IoT in general [32]. For achieving this aim, we investigated what are the architectural abstractions suitable for architecting SoS on the IoT.

There are indeed different notions and reference architectures for developing software-intensive systems, in particular cyber-physical systems, on the IoT.

Among them, the notion of digital twin (proposed in [7] and extended for IoT [5]) fits well the needs for coupling the physical and virtual worlds in terms of "things", as required in software-intensive SoSs on IoT.

A digital twin provides a virtual replica (in the edge, fog or the cloud) of its physical counterpart which is virtually indistinguishable from its physical twin (in the sense that it dynamically replicates the behavior and properties of the physical twin as well as enriches the replica with additional information about the physical counterpart). It is in fact a dynamic digital representation of its physical counterpart. Dynamic in the sense that the physical asset and its digital twin are connected during their whole lifecycle.

In general, the digital twin monitors the physical twin through data provided by sensors on the IoT as well as information coming from domain experts to maintain a utility replica of the physical asset. It acts back on the physical twin through actuators on IoT.

In the Internet-of-Vehicles, the digital twin of each self-driving vehicle will inspect the physical vehicle (the physical twin), which provides all the data to the digital twin.

Therefore, to enable the architectural description of enhanced IoT applications, SosADL was extended with constructs for expressing digital twins (for details, see [33]).

3.2 Constructs for Autonomous Constituent Systems in SosADL

SosADL was firstly conceived for describing SoS architectures which involve SoSs composed of autonomous systems, i.e. where constituents are systems that behave according to only its own capabilities, sensing and actuating in its local environment.

The key point to be addressed was the one of partial information, as no constituent system of the SoS has complete knowledge, each one having only partial knowledge according to the information it can perceive in its local environment.

Therefore, for handling partial knowledge in SoS architectural design and particularly in the description of the designed SoS architecture, SosADL was conceived to enable the representation of partial information as well as to reason on this partial information to influence the behavior of the SoS. By the decentralized nature of SoSs, these representation and reasoning mechanisms are expressed from the viewpoint of each constituent system and mediators enabling their interactions and thereby influencing individual behaviors as well as the raising of emergent behaviors.

For defining the formal semantics of SosADL handling partial information, we evaluated different behavior calculi developed for modeling complex systems and identified several forms of the π-Calculus [21], however none of them complied with the SoS requirements for architectural behavior description [27].

We have therefore designed a novel π-Calculus for SoS [28], which extended the original π-Calculus with mediated constraints, where mediation is achieved by constraining interactions, and where constrained interactions raise emergent behaviors [2]. More precisely, the π-Calculus for SoS generalizes the original π-Calculus with the notion of computing with partial information based on the concurrent constraint paradigm and in particular on the principles of constraint-based calculi [23].

Formally based on the π-Calculus for SoS, SosADL provides: (i) a **tell** construct for adding a constraint to the local environment; (ii) an **ask** construct for querying if a constraint can be inferred from the local environment.

These constructs enable each constituent system to behave according to the constraints imposed by other constituent systems sharing the same local environment.

Intuitively speaking, based on π-Calculus for SoS, in SosADL a constituent can publicly **tell** to the environment about the pieces of information that it knows, while maintaining private information internally. A constituent can also **ask** information from the environment that influences on its own behavior.

The communication between composed constituent systems is supported by unicast connections: (i) a **send** construct for synchronously sending a value over a connection to another constituent system; (ii) a **receive** construct for synchronously receiving a value over a connection from another constituent system.

Relying on the π-Calculus for SoS, the formal operational semantics of SosADL was defined by means of a formal transition system, expressed by labelled transition rules. For details on the formal semantics of SosADL in terms of π-Calculus for SoS, see [28].

If SosADL demonstrated to be a suitable formal ADL for SoSs, able to describe the SoS architectures of flocks of autonomous drones and platoons of autonomous self-driving cars, it is not able to support the description of SoS architectures based on wireless network communications in addition to sensing/actuating capabilities.

3.3 Extending SosADL with Constructs for Connected Constituent Systems

In this paper, we extend SosADL for describing SoS architectures which involve SoSs composed of communicating autonomous systems, i.e. where SoS constituents are connected systems to wireless local area networks, supporting in particular Vehicle-to-Vehicle (V2V) data exchange through VANETs, that behave in coordination with its own neighboring constituents in partially shared environments.

For extending SosADL for IoV, the key points to be addressed were: (i) delimiting the wireless communication range (the communication is limited to constituent systems in the neighborhood, that is determined by the maximum wireless communication range of the V2V radio technology); (ii) discovering which are the other constituent systems that are within the V2V communication range; (iii) getting specific V2V communication channels to interact with relevant neighbors.

Therefore, for handling communication supported by wireless local area networks in SoS architectural design and particularly in the description of the designed SoS architecture, SosADL was enhanced to address each one of these three points.

First, it was extended to enable to physically sense which are the neighboring constituent systems within a given radius. For instance, in the case of autonomous cars, it senses which are the other cars in front of it and behind of it in the same road lane as well as in adjacent lanes using a lidar.

Formally, we have extended SosADL with a new construct for expressing neighborhood of mobile constituents. The notion of neighborhood is given by extending digital twins accessing physical properties, i.e. every digital twin of a "thing" is able to inform about (a subset of) physical properties of its physical counterpart. These properties are available in all digital twins through built-in constraints in SosADL for IoT.

Thereby, a physical twin can **tell** about its physical properties and the digital twin can apply the **ask** construct for getting each physical property of the related physical twin: (i) **ask range for *thing*** to get the maximum V2V communication range in *meter* (the radius) of the ***thing*** in question; (ii) **ask neighborhood within *range* for *thing*** to get the set of "things" in the neighborhood of the ***thing*** in question; (iii) **ask [local] coordinate for *thing*** to get the coordinate of the ***thing*** in question (the coordinate is given in global or local coordinate systems, the default being global coordinate in terms of GPS). All physical properties are given in *SI* unit (the International System of Units) and all these *SI* datatypes are equipped with operations for manipulation and conversion.

Second, SosADL was extended to enable to communicate with all neighbors of a "thing" using *broadcasting*, supporting one-to-many communication and its opposite *collect* concept, which supports many-to-one communication.

Hereafter, we will focus on *broadcasting*, which provides the essential concept for supporting SoS architectures on the IoV. Broadcasting is the mechanism of transferring a message from a sender to all receivers, simultaneously, within the communication range. It is worth highlighting that, as demonstrated in [4], *broadcast* cannot be encoded with *unicast* and thereby is needed as a primitive concept in an ADL for IoV.

For expressing broadcast, we extend SosADL with two constructs: **talk** and **listen**.

The **talk** construct is expressed as **via** ***connection*** [**within** ***range***] **talk** ***data***, meaning that ***connection*** is used to transmit ***data*** to all "things" that are within the given ***range***. In case ***range*** is not specified, it is by default the maximum communication range. Thereby data will be transmitted to all "things" in the **neighborhood**.

The **listen** construct is expressed as **via** ***connection*** [**within** ***range***] **listen** ***placeholder***, meaning that ***connection*** is used to receive ***data*** in the ***placeholder*** talked by all "things" that are within the given ***range***. In case ***range*** is not specified, it is by default the maximum communication range. Thereby data transmitted from all "things" in the **neighborhood**, can be listened.

4 SoS Architecture Description for UGV-Based Platooning

To demonstrate how *SosADL for IoV* can be applied to architecturally describe SoS architectures on the IoV, we will present hereafter an excerpt of an SoS architecture description that we have designed in a cooperative project with stakeholders of the city of Sao Carlos as part of a pilot for a Flood Monitoring and Emergency Response SoS [26], focusing on the platooning of Unmanned Ground Vehicles (UGVs).

UGVs are self-driving (driverless) vehicles equipped with V2V wireless technology, supporting on-the-fly creation of VANETs. They are usually used for applications where it is inconvenient, dangerous, or impossible to have a human operator present.

In this pilot SoS, the UGV-based Emergency Response SoS is formed by UGVs deployed from different city councils in the metropolitan area of Sao Carlos and neighbor municipalities. Several fleets of UGVs can be activated by the gateway of the WSN-based Urban River Monitoring SoS for accomplishing disaster relief missions and search and rescue operations. UGVs drive autonomously using built-in GPS to the inundated area identified by drones.

The acquired UGVs are fully autonomous, able to self-navigate in GPS-enabled environments as well as to detect and avoid obstacles. They in particular provide a *follow* mode enabling wireless tethering to another UGV. In this mode, it reacts to its frontrunner movements and direction while having the mission route saved for autonomous execution. The *follow* mode is used by the autopilot of a UGV, when it is participating in a platoon for following the UGV that is in front of it.

Let us now focus on the supervenient emergent behavior of the UGV-based Emergency Response SoS to create and maintain platoons of UGVs through self-organization during the journey to the destination. More specifically, we will concentrate on the maneuvers for a UGV to join an ongoing platoon, where the maneuvers are coordinated between the UGV joining the platoon and UGVs that are already in the platoon using wireless communication on VANETs.

Hereafter, we will demonstrate how to apply *SosADL for IoV* to describe the SoS architecture of UGV platooning as well as explain the operational semantics of the resulting SoS architecture description during the join maneuver of a UGV into a platoon.

In the formalization with *SosADL for IoV*, we will first describe the digital twin mediators enforcing the join maneuver required for a UGV to join a platoon (we will

not present the digital twins of the UGVs due to page limit). Next, we will describe the abstract architecture of the SoS on the IoV as a whole in terms of a digital coalition.

Let us now declare, in *SosADL for IoV*, the *Platooning* mediator which will support the platooning as well as the join maneuver of a UGV in the vehicle platoon.

As shown in **Listing 1**, *Platooning* is described as a digital mediator: we declare the duties of the mediated UGV and the behavior abstractions that can be applied during the mediation, i.e. **abstraction** *Leading_platoon(…)* declares the behavior abstraction for leading the platoon; **abstraction** *Following_platoonmate(…)* declares the behavior abstraction for following the in-front platoonmate in the platoon; and **abstraction** *Steering_to_join_platoon(…)* declares the behavior abstraction for joining the platoon. The declared main behavior for self-driving the UGV is **behavior** *Self_Driving(…)*.

```
//use the predefined library for the Internet-of-Vehicles (IoV)
with IoV
//declare the self-driving vehicle mediator for platooning
digital mediator Platooning(min:Distance) is {
 //declare the duty to support the join maneuver into platoons
 duty join is {
  //declare a broadcast channel for communicating with platoonmates
  connection request_ch is broadcast{connection[connection[Thing]]}
}
duty control is {
  //declare connections for steering the mediated UGV
  connection align_x_start is out{…}
  connection align_x_end is in{…}
  connection align_y_start is out{…}
  connection align_y_end is in{…}
  connection keep_x_distance is out{…}
  connection keep_y_distance is out{…}
}
//declare the behavior abstraction for leading the UGV platoon
abstraction Leading_platoon(…) is {…}
//declare the behavior abstraction for following in the UGV platoon
abstraction Following_platoonmate(…) is {…}
//declare the behavior abstraction for joining into the UGV platoon
abstraction Steering_to_join_platoon(…) is {…}
//declare the main behavior for self-driving the UGV
behavior Self_Driving(…) is {…}
}
```

Listing 1. Digital mediator declaration for platoonmates in *SosADL for IoV*

Based on the declaration of UGVs as systems (not shown for brevity) and of the digital mediator, presented in **Listing 1**, let us now declare the *PlatooningSoS* architecture (presented in **Listing 2**). It describes which are the digital twins of the constituent systems that can participate in the SoS, the digital mediators that can be created and managed for coordinating the constituent systems via their digital twins and the digital coalitions that can be formed to achieve the SoS emergent behavior of platooning. As declared in the platoon coalition, by creating concretions, a digital mediator will be synthesized for each digital twin of a UGV that participates in the fleet of UGVs.

```
//use UGV system abstraction and Platooning mediator abstraction
  with IoV,UGV,Platooning
  architecture PlatooningSoS(min:Distance) is {
   coalition platoon is compose{
    fleet is sequence{UGV()}
    platooning is sequence{Platooning(min)}
   } binding {
      forall {ugv in fleet suchthat
        exists{one steer in platooning suchthat
          unify one{steer::join} to one{ugv::join}
          unify one{steer::control} to one{ugv::control}}}}
  }
```

Listing 2. Platooning SoS architecture declaration in *SosADL for IoV*

Let us now describe, in **Listing 2**, the SoS architecture enabling the platooning emergent behavior as well as maneuvers in the platoon.

The SoS architecture description, shown in **Listing 2**, comprises the declaration of a sequence of digital twins of constituent systems complying with the system abstraction of *UGV* and a sequence of digital mediators conforming with the mediator abstraction of *Platooning* (as declared in **Listing 1**).

Based on the digital twins of these systems and mediator abstractions, the digital coalition for creating emergent behavior is declared, named *platoon*, as shown in **Listing 2**. In particular, the digital coalition of UGVs is described as a sequence of digital twins of UGVs where each digital twin has an associated steering digital mediator created in the digital coalition, with the specified minimum separation distance (*min*) as parameter. The emergent behavior of the digital coalition, *platoon*, is giving by the macro-scale behavior created by supervenience from the mediating micro-behaviors, according to each situation.

Let us now declare the digital mediating behavior, described in **Listing 3**. For the sake of space, we will focus on the general case of the join maneuver somewhere in the middle of the vehicle platoon. In this case, a vehicle that is not in the platoon, i.e. the joining UGV, broadcasts a joining request to the platoonmates expressing its intention to join the platoon. One of the concerned platoonmates, which listened the broadcast, contact back the joining UGV to accept the request and send the information required for the UGV to join the platoon. The agreed platoonmate increases the space in front of it until enough space has been created, i.e. at least two times the min separation distance plus the size of the joining UGV, and in parallel the joining UGV aligns longitudinally itself between the agreed platoonmate and the platoonmate in front of it. Then, the joining UGV aligns laterally itself by changing its road lane to the same as the platoon. Once it merges into the platoon, being completely aligned in the lane of the platoon, it keeps driving forward while keeping a safe distance to the platoonmate in front of it. It becomes a regular platoonmate, and its autopilot (an adaptive cruise controller) takes the command (it automatically steers to maintain the platoon, regulating its distance to the platoonmate in front of it). Note that along the join maneuver, the joining UGV as well as the platoonmates drive forward with respect to the road geometry.

```
//behavior abstraction of the mediator for UGV to join the platoon
abstraction Steering_to_join_platoon(min:Distance) is {
 repeat {
  //ask the "thing" handle of the joining UGV itself
  value ugv is ask itself
  //ask the max V2V communication range of the joining UGV itself
  value ugv_range is ask ugv for range
  //request to join the platoon talking to its neighbors in platoon
  behavior Requesting_to_join(min,ugv,ugv_range)
 }
 //broadcast intention to join platoon and then wait for responses
 abstraction Requesting_to_join(min:Distance,ugv:Thing,
  ugv_range:Distance) is {
  //ask the vehicle size in the x axis of the joining UGV itself
  value ugv_size_in_x is ask ugv for size_in_x
  //declare unicast channel of channel for communicating with mate
  restrict ugv_request_ch:connection[connection[Thing]]
  //broadcast the unicast ch of ch for all neighboring platoonmates
  via join::request_ch within ugv_range talk ugv_request_ch
  replicate {//wait for the responses from neighboring platoonmates
   //receive the unicast channel from the platoonmate which agreed
   via ugv_request_ch receive platoonmate_ch:connection[Thing]
   //receive the "thing" handle of the platoonmate which agreed
   via platoonmate_ch receive platoonmate:Thing
   //receive the "thing" handle of its in-front platoonmate
   via platoonmate_ch receive frontmate:Thing
   //align in x,y the joining ugv between platoonmate and frontmate
   behavior Aligning_to_join(min,ugv_size_in_x,ugv,platoonmate,
    frontmate) }
 }
 //maneuver to align the mediated UGV for joining the platoon
 abstraction Aligning_to_join(min:Distance,ugv_size_in_x:Distance,
  ugv:Thing,platoonmate:Thing,frontmate:Thing) is {
  via control::align_x_start send [min,ugv_size_in_x,
   ugv,platoonmate,frontmate]
  via control::align_x_end receive ugv_in_x_coordinate:Coordinate
  via control::align_y_start send [ugv,platoonmate,frontmate]
  via control::align_y_end receive ugv_in_y_coordinate:Coordinate
  //once inside, follow the UGV in front of it, i.e. its frontmate
  behavior Following_platoonmate(min,ugv,front_mate)
 }
 abstraction Following_platoonmate(min:Distance,ugv:Thing,
  frontmate:Thing) is {
  choose {
     //once aligned, the joining UGV behave as a platoonmate
     via control::keep_x_distance send min
     behavior Following_platoonmate(min,ugv,frontmate)
     //in case, the UGV as platoonmate will support others joining
  or replicate { via join::request_ch listen ugv_request_ch:
                  connection[connection[Thing]
     restrict platoonmate_ch:connection[Thing]
     via ugv_request_ch send platoonmate_ch
     via platoonmate_ch send ugv
     via platoonmate_ch send frontmate
     via platoonmate_ch receive joining_frontmate
    behavior Following_platoonmate(min,ugv,frontmate) } }
 }
}
```

Listing 3. Digital mediator behavior for platoonmates in *SosADL for IoV*

It is by the application of the mediating behaviors, described in **Listing 3**, commanding the digitally mediated UGVs, that a UGV, when approaching a platoon, will perform the maneuver to join the platoon. It will stepwise behave to get closer to the neighboring UGVs in the platoon and then join the platoon. For an external observer, at the macro-scale level, a UGV somewhere in the middle of the platoon will create a space in front of it allowing the joining UGV to safely come into the platoon, while avoiding collision. The safe space is of at least two times the minimum separation distance plus the size of the UGV joining the platoon.

It is worth noting that the maneuver for joining the platoon is coordinated by the digital twins of the UGVs. Each UGV is not aware of the rest of the platoon, except for its own neighbors, generally composed of the platoonmates in front of and behind itself. For instance, a UGV in the platoon is not aware of the size of the platoon, i.e. of how many members has the platoon.

From the abstract SoS architecture described in **Listing 1**, **Listing 2**, and **Listing 3**, different concrete SoS architectures of platoons may be created based on the identified UGVs for each particular operational environment. These platoons, through the mediated platooning behavior, support the maneuvers for other UGVs joining the platoon dynamically. For instance, as mentioned, in the case of the UGV-based Emergence Response SoS of the Monjolinho river, the fleet of identified UGVs are located at different municipalities along the river. They are then commanded to drive and reach an area with high risk of flooding for an emergency response mission (triggered by the monitoring of the urban river). From each departing point, the UGVs will leave to the destination zone in platoons (economizing on fuel) and, along the way, other UGVs will join the running platoons for forming larger ones. Once in destination, they will dissolve the platooning SoSs adopted in the navigation mission to reach the destination point and then adopt another SoS architecture for the search and rescue mission.

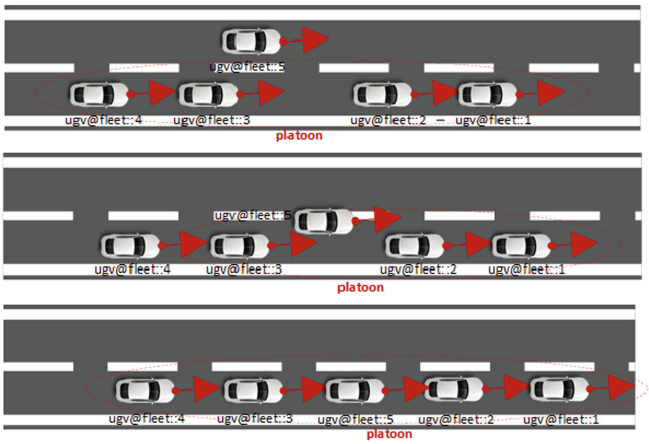

Fig. 3. Snapshots of the mediated maneuver of a UGV joining a platoon

For illustrating the platooning mediation in the support of the maneuvers for a UGV joining a platoon, let us now suppose that a UGV enters the roadway and, at some point, drives in a contiguous lane of another where there is a platoon, as shown in Fig. 3 (top). This approaching UGV can join the platoon by the application of the *Steering_to_join(...)* mediated behavior declared in **Listing 3**.

First the joining UGV broadcast its intention to join the platoon, then get a response from a member of the platoon, a platoonmate, which creates the space needed for the insertion of the joining UGV in front of it. The joining UGV will then align longitudinally to be closer to that platoonmate. When the gap appears between the near UGVs, i.e. in this case between ugv@fleet::3 and ugv@fleet::2 as shown in Fig. 3 (middle), the joining UGV, in this case ugv@fleet::5, initiates the maneuver to align laterally, as shown in Fig. 3. Smoothly, the ugv@fleet::5 maneuvers to come into the platoon, as shown in Fig. 3 (from top to bottom).

Note that other maneuvers are supported in the field study (not shown for reason of space): UGVs leaving a platoon, a platoon split in smaller platoons, and different platoons merging together to form larger ones.

5 SosADL Implementation

We have developed an SoS Architecture Development Environment, named *SosADL Studio* [34], for supporting the architecture-centric formal development of SoSs using SosADL (from missions to SoS architectures [39], and their validation/verification), while providing guarantees of correctness of the elaborated SoS architectures [35].

This toolset is constructed as plugins in Eclipse (http://eclipse.org/). It provides a model-driven architecture development environment where the SosADL meta-model is defined in EMF/Ecore (http://eclipse.org/modeling/emf/), with the textual concrete syntax expressed in Xtext (http://eclipse.org/Xtext/), the graphical concrete syntax developed in Sirius (http://eclipse.org/sirius/), and the type checker implemented in Xtend (http://www.eclipse.org/xtend/), after having being proved using the Coq proof assistant (http://coq.inria.fr/).

By applying model-to-model transformations, SoS architecture descriptions are transformed and converted to input languages of analysis tools, including UPPAAL (http://www.uppaal.org/) for extensive model checking, DEVS (http://www.ms4systems.com/) for simulation, and PLASMA (http://project.inria.fr/plasma-lab/) for statistical model checking.

The constraint solving mechanism implemented to support the **tell** and **ask** constructs are based on the Kodkod SAT-solver (http://alloy.mit.edu/kodkod/).

Of particular interest for validating SoS emergent behavior is the automated generation of concrete SoS architectures, by automated transformation from SosADL to DEVS, and the subsequent simulation in DEVS enabling to observe and tune the described emergent behavior of an SoS [6].

For supporting verification of SoS architectures, we have conceived a novel logic, named DynBLTL [37], for expressing correctness properties of evolving architectures as well as verifying these properties by model checking [3].

The *SosADL Studio for IoV* extends the core *SosADL Studio* with enhancements to the ADL, incorporating in particular the new constructs for *broadcast* (i.e. *talk* and *listen*), and their implementation in the toolchain, as well as *collect*. The different concrete syntaxes, the abstract syntax in terms of the meta-model, the type checker as well as the model-to-model transformations were extended accordingly.

In addition to these domain independent tools, the resulting *SosADL Studio for IoV* was extended with VEINS, https://veins.car2x.org/, which provides an open source framework for running vehicular network simulations, including inter-vehicular communication with VANETs.

The implemented toolchain in *SosADL Studio for IoV* provides the ability to validate and verify the studied SoS architectures on the IoV very early in the SoS lifecycle with respect to its correctness properties, in particular regarding emergent behaviors.

6 SosADL Validation for the IoV by Controlled Experiment

For validating *SosADL for IoV* and its supporting toolchain, *SosADL Studio for IoV*, we carried out a field study of a real SoS for Flood Monitoring and Emergency Response and studied its concretization in the Monjolinho river, which crosses the city of Sao Carlos (see [26] for more details on the description of the field study).

The mission of the designed SoS is to monitor potential floods and to handle related emergencies. The SoS stakeholder is the DAEE (Sao Paulo's Water and Electricity Department), a government organization of the State of Sao Paulo, Brazil, responsible for managing water resources, including flood monitoring of urban rivers. This SoS also involves as stakeholders the different city councils crossed by the Monjolinho river, the policy and fire departments of the city of Sao Carlos that own UAVs (drones) and have UGVs (self-driving vehicles) equipped with VANET. Also involved are the hospitals of the city of Sao Carlos (they have ambulances equipped with VANETs).

The aim of this field study developed conjointly with USP was to assess the fitness for purpose and the usefulness of, on the one hand, SosADL as a formal SoS architectural language, and on the other hand, of SosADL Studio as an SoS architecture development environment to support the architectural description and analysis of real SoSs.

In addition to IoT in general, to validate that the expressive power of SosADL for IoT copes with the needs of IoT, we carried out a controlled experiment for addressing IoV in particular through its challenging case of vehicle platooning, which involves ad-hoc networking and physical mobility. The controlled experiment was designed as a subset of the field study of the Flood Monitoring and Emergency Response SoS applied to the Monjolinho river in the city of Sao Carlos, as cited.

Overall, the result of the assessment based on the controlled experiment concluded that *SosADL for IoV* enables descriptions of SoS architectures on the IoV which were not possible to be described with core SosADL or even its customized *SosADL for IoT*.

The reason is that SosADL and *SosADL for IoT* (for stationary "things") are based only on unicast communication constructs, i.e. *send/receive*, while *SosADL for IoV* (for mobile "things") was extended with *broadcast* constructs. Formally speaking, the *broadcast* constructs were demonstrated to be not encodable with the *unicast* ones [4].

Intuitively, it means that the behaviors expressed in the maneuver for joining platoons cannot be specified in languages providing only unicast *send/receive* constructs, which is today the case of all existing ADLs [36].

7 Related Work

Related work on the description of behaviors shaped by software architectures is of two kinds: aggregative behavior in single systems architecture [15, 36] and emergent behavior in SoS architectures [8]. SoS architectures on the IoV imply the need to describe emergent behaviors drawn from the interaction of mobile "things".

Let us first analyze related work on emergent behavior related to SoS, of which the key ones are proposed by Wachholder and Stary [40] and Motus et al. [22].

Wachholder and Stary [40] presents an attempt for describing emergent behavior in SoSs. The proposed approach is based on Bigraphs and focuses on the modeling of both the structure and the structural dynamics of an SoS. It, however, does not address the behavioral aspects of emergent behavior, limiting the solution to the configuration and re-configuration features. The proposed solution is limited only to concrete (one-of-a-kind) SoS architectures and limited only to the endogenous approach for modeling emergent behavior in SoS. These limitations restrict the expressiveness of the proposed approach, which does not cope with the needs for architecting SoS on IoV.

Motus et al. [22] address the importance of mediating interactions among constituent systems to achieve emergent behavior. In particular, it proposes a middleware based on mediated interaction. That work is complementary to ours in the sense that SoS architectures described with SosADL can be deployed in the proposed middleware.

Let us now analyze how emergent behavior has been specifically addressed for designing platoons as SoSs. This issue has been tackled by two related works: Kumar et al. [17] and Labrado et al. [18].

Kumar et al. [17] proposed a modeling approach based on Bond Graph Theory for describing a platoon of autonomous vehicles. The proposed solution is limited to the physical model of the vehicle platoon, as well as can only be applied to a concrete (one-of-a-kind) SoS based on endogenous modeling. Its role is mainly for supervision purposes. This work is complementary to ours in the sense that SoS architectures described with SosADL for IoV can be refined to bound graph models to study the physical properties of the architected SoS platooning. It however does not address the digital counterpart.

Labrado et al. [18] proposed a testbed for simulating SoSs based on physical robots connected to a cloud. Platoon is one of the supported kinds of SoS. Again, this work is complementary to ours in the sense that SoS architectures described with SosADL for IoV can be refined to concrete implementations that can then be simulated using the proposed simulation framework. Again, it does not address the digital counterpart.

Let us now analyze related work on aggregative behavior exposed by single systems. The description of single systems architecture as based on three notions: component, connector, and configuration. The notion of connector as a first-class entity in architecture description has been proposed two decades ago [1]. Connectors were firstly

designed as static entities, that never changed during run-time, then as dynamic entities changing dynamically at run-time to support dynamic architectures [24]. More recently the automatic synthesis of connectors during design-time was proposed [11].

The notion of mediator proposed in SosADL, which provides the basis for exogenously describing SoS architectures, generalizes the notion of connector. In particular, SosADL advances the current state-of-the-art on synthesized connectors at design-time in single systems architecture exposing aggregative behavior by providing the novel concept of mediator, based on synthesized mediators at run-time, on demand, in SoS architectures exposing emergent behavior. In addition, *SosADL for IoV* encompasses mediators supporting broadcast communication, which cannot be expressed on languages based only on unicast connectors, which is the case of all existing ADLs.

Overall, currently, other ADLs are not expressive enough to be able to describe SoS architectures on the IoV such as in vehicular platooning with dynamic join/leave. *SosADL for IoV* is therefore the first ADL designed for meeting the needs of IoV, while based on a formal calculus enabling to formally express correctness properties as well as to verify these properties through automated tools, in particular statistical model checkers.

8 Conclusion and Future Work

This paper presented the novel concepts and constructs of *SosADL for IoV*, extending the core SosADL, while generalizing SosADL for IoT to both stationary and mobile "things". In particular, it demonstrated how architectural mediators expressed with *SosADL for IoV* support SoS architecture descriptions on the IoV through an excerpt of a real application for architecting a UGV-based platooning enabling dynamic maneuvers, e.g. joining/leaving, focusing on the join maneuver. It provides the first formal SoS architectural description of connected autonomous vehicles supporting dynamic maneuvers, all others being only limited to the architectural formation of SoS platooning.

The formal foundation of *SosADL for IoV* extends the π-Calculus enhanced with concurrent constraints, the π-Calculus for SoS, bringing contributions beyond the state-of-the-art by providing the first full formal ADL having the expressive power for describing emergent behavior in software-intensive SoS architectures on the IoV, in particular IoV-connected vehicle platooning with dynamic maneuvers, grounded on its constraint solving mechanism, mobile unicast, and now broadcast communication.

SosADL has been applied in several case studies and pilots where the suitability of the language and the supporting toolchain has been validated, including *SosADL for IoV* for SoS applications in smart-cities.

On-going and future work is mainly related with the application of *SosADL for IoV* to real-scale projects on the Internet-of-Vehicles. They include joint work with IBM for applying SosADL to architect smart-farms on the IoT in general and IoV in particular, and with SEGULA for applying *SosADL for IoV* to architect SoSs in the navy domain, based on 5G. Description of SoS architectures on the IoV, and their validation and verification using the *SosADL for IoV* toolchain, are main threads of these pilot projects.

References

1. Allen, R., Garlan, D.: A formal basis for architectural connection. ACM TOSEM **6**(3), 213–249 (1997)
2. Blachowicz, J.: The constraint interpretation of physical emergence. J. Gen. Philos. Sci. **44**, 21–40 (2013). https://doi.org/10.1007/s10838-013-9207-7
3. Cavalcante, E., Quilbeuf, J., Traonouez, L.-M., Oquendo, F., Batista, T., Legay, A.: Statistical model checking of dynamic software architectures. In: Tekinerdogan, B., Zdun, U., Babar, A. (eds.) ECSA 2016. LNCS, vol. 9839, pp. 185–200. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48992-6_14
4. Ene, C., Muntean, T.: Expressiveness of point-to-point versus broadcast communications. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 258–268. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48321-7_21
5. Tao, F., Zhang, M., Nee, A.Y.C.: Digital Twin Driven Smart Manufacturing. Academic Press, Cambridge (2019)
6. Graciano Neto, V.V., et al.: ASAS: an approach to support simulation of smart systems. In: 51st HICSS, Waikoloa, Hawaii, USA, January 2018
7. Grieves, M.: Virtually Perfect: Driving Innovative and Lean Products through Product Lifecycle Management. Space Coast Press, Cocoa Beach (2011)
8. Guessi, M., Graciano, V.V., Bianchi, T., Felizardo, K.R., Oquendo, F., Nakagawa, E.Y.: A systematic literature review on the description of software architectures for systems-of-systems. In: 30th ACM SAC, Salamanca, Spain, April 2015
9. Guessi, M., Oquendo, F., Nakagawa, E.Y.: Checking the architectural feasibility of systems-of-systems using formal descriptions. In: 11th IEEE SoSE, Kongsberg, Norway, June 2016
10. INCOSE, SE Vision 2025 (2014). www.incose.org/AboutSE/sevision
11. Inverardi, P., Tivoli, M.: Automatic synthesis of modular connectors via composition of protocol mediation patterns. In: 35th ACM/IEEE ICSE, May 2013
12. ISO/IEC/IEEE 42010:2011: Systems and Software Engineering – Architecture Description, December 2011
13. Jia, D., Lu, K., Wang, J., Zhang, X., Shen, X.: A survey on platoon-based vehicular cyber-physical systems. IEEE Commun. Surv. Tutor. **18**(1), 263–284 (2016)
14. Kaiwartya, O., et al.: Internet of vehicles: motivation, layered architecture, network model, challenges, and future aspects. IEEE Access **4**, 5356–5373 (2016)
15. Klein, J., van Vliet, H.: A systematic review of system-of-systems architecture research. In: 9th ACM QoSA, Vancouver, Canada, June 2013
16. Kopetz, H., Höftberger, O., Frömel, B., Brancati, F., Bondavalli, A.: Towards an understanding of emergence in systems-of-systems. In: 10th IEEE SoSE, San Antonio, Texas, USA, May 2015
17. Kumar, P., Merzouki, R., Bouamama, B.O., Koubeissi, A.: Bond graph modeling of a class of system-of-systems. In: 10th IEEE SoSE, San Antonio, Texas, USA, May 2015
18. Labrado, J.D., Erol, B.A., Ortiz, J., Benavidez, P., Jamshidi, M., Champion, B.: Proposed testbed for the modeling and control of a system of autonomous vehicles. In: 11th IEEE SoSE, Kongsberg, Norway, June 2016
19. Maier, M.W.: Architecting principles for systems-of-systems. Syst. Eng. J. **1**(4), 267–284 (1998)
20. Malavolta, I., et al.: Architectural Languages Today: The Up-to-Date List of ADLs, 7 April 2019. http://www.di.univaq.it/malavolta/al/
21. Milner, R.: Communicating and Mobile Systems: The π-Calculus. Cambridge University Press, Cambridge (1999)

22. Motus, L., Preden, J.S., Meriste, M., Pahtma, R.: Self-aware architecture to support partial control of emergent behavior. In: 7th IEEE SoSE, Genoa, Italy, July 2012
23. Olarte, C., Rueda, C., Valencia, F.D.: Models and emerging trends of concurrent constraint programming. Int. J. Constr. **18**(4), 535–578 (2013). https://doi.org/10.1007/s10601-013-9145-3
24. Oquendo, F.: π-ADL: an architecture description language based on the higher-order typed π-Calculus for specifying dynamic and mobile software architectures. ACM SEN **29**(3), 1–14 (2004)
25. Oquendo, F.: Formally describing the software architecture of systems-of-systems with SosADL. In: 11th IEEE SoSE, Kongsberg, Norway, June 2016
26. Oquendo, F.: Case study on formally describing the architecture of a software-intensive system-of-systems with SosADL. In: 15th IEEE SMC, Budapest, Hungary, October 2016
27. Oquendo, F.: Software architecture challenges and emerging research in software-intensive systems-of-systems. In: Tekinerdogan, B., Zdun, U., Babar, A. (eds.) ECSA 2016. LNCS, vol. 9839, pp. 3–21. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48992-6_1
28. Oquendo, F.: The π-Calculus for SoS: novel π-Calculus for the formal modeling of software-intensive systems-of-systems. In: Communicating Process Architectures (CPA 2016), August 2016
29. Oquendo, F.: Formally describing the architectural behavior of software-intensive systems-of-systems with SosADL. In: 21st IEEE ICECCS, Dubai, UAE, November 2016
30. Oquendo, F.: Architecturally describing the emergent behavior of software-intensive system-of-systems with SosADL. In: 12th IEEE SoSE, Waikoloa, Hawaii, USA, June 2017
31. Oquendo, F.: On the emergent behavior oxymoron of system-of-systems architecture description. In: 13th IEEE SoSE, Paris, France, June 2018
32. Oquendo, F.: Formally describing self-organizing architectures for systems-of-systems on the internet-of-things. In: Cuesta, C.E., Garlan, D., Pérez, J. (eds.) ECSA 2018. LNCS, vol. 11048, pp. 20–36. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00761-4_2
33. Oquendo, F.: Dealing with uncertainty in software architecture on the internet-of-things with digital twins. In: Misra, S., et al. (eds.) ICCSA 2019. LNCS, vol. 11619, pp. 770–786. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24289-3_57
34. Oquendo, F., Buisson, J., Leroux, E., Moguérou, G., Quilbeuf, J.: The SosADL studio: an architecture development environment for software-intensive systems-of-systems. In: SiSoS 2016, Copenhagen, DK. ACM, November 2016
35. Oquendo, F., Buisson, J., Leroux, E., Moguérou, G.: A formal approach for architecting software-intensive systems-of-systems with guarantees. In: 13th IEEE SoSE, Paris, France, June 2018
36. Ozkaya, M.: The analysis of architectural languages for the needs of practitioners. Softw. Pract. Exp. **48**, 985–1018 (2018)
37. Quilbeuf, J., Cavalcante, E., Traonouez, L.-M., Oquendo, F., Batista, T., Legay, A.: A logic for the statistical model checking of dynamic software architectures. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 806–820. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_56
38. Roca, D., Nemirovsky, D., Nemirovsky, M., Milito, R., Valero, M.: Emergent behaviors in the internet-of-things: the ultimate ultra-large-scale system. IEEE Micro **36**(6), 36–44 (2016)
39. Silva, E., Cavalcante, E., Batista, T., Oquendo, F.: Bridging missions and architecture in software-intensive systems-of-systems. In: 21st IEEE ICECCS, Dubai, UAE, November 2016
40. Wachholder, D., Stary, C.: Enabling emergent behavior in systems-of-systems through bigraph-based modeling. In: 10th IEEE SoSE, San Antonio, Texas, USA, May 2015
41. Wang, Z., Wu, G., Barth, M.J.: A review on cooperative adaptive cruise control (CACC) systems: architectures, controls, and applications. In: Intelligent Transportation Systems (ITSC 2018), Maui, HI, USA (2018)