Engineering Collectives of Self-driving Vehicles: the SOTA Approach

Dhaminda B. Abeywickrama[1], Marco Mamei[2], and Franco Zambonelli[2]

[1] Agents, Interaction and Complexity Group, School of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom
dhaminda.abeywickrama@soton.ac.uk

[2] Dipartimento di Scienze e Metodi dell'Ingegneria, University of Modena and Reggio Emilia, Via Amendola 2, 42122 Reggio Emilia, Italy
{marco.mamei,franco.zambonelli}@unimore.it

Abstract. Future cities will be populated by myriads of autonomous self-driving vehicles. Although individual vehicles have their own goals to pursue in autonomy, they may also be part of a collective of vehicles, as in the case of a fleet of vehicles of a car sharing company. Accordingly, they may also be required to act in a coordinated way towards the achievement of specific collective goals, or to meet specific city-level objectives. This raises the issue of properly engineering the behavior of such collective of vehicles, by properly capturing their collective requirements also in consideration of their individual goals, and understanding which knowledge about the state of the collective they must be provided with. In this context, this paper shows how the SOTA model can be a very effective tool to support the engineering of self-driving vehicle collectives. SOTA, by bringing together the lessons of goal-oriented requirements engineering, context-aware systems, and dynamical systems modeling, has indeed the potential for acting as a general reference model to help tackle some key issues in the design and development of complex collective systems immersed in dynamic environments, as collectives of self-driving vehicles are.

Keywords: Self-driving vehicles · Software engineering · Self-adaptation.

1 Introduction

As we are entering the era of autonomous cars, many envision that future urban mobility will no longer be primarily supported by private vehicles, but rather by fleets of autonomous vehicles, either owned by private companies or by the municipality itself, and devoted to car or ride sharing [5, 8], and to the delivery of merchandise [16]. Thus, properly organizing and managing such fleets will be of primary importance in future cities.

Such management will have to account the diverse and mostly unpredictable demands of individual citizens, commercial activities, and industries. Also, it will have to account for resource restrictions related to, e.g., availability of parking

lots and availability of charging stations (we assume the vehicles are electric ones). In terms of objectives, the management will have to harmonize at the best with the needs of individual vehicles (that is, of the citizens that have rented a vehicle), the needs of the fleet (i.e., or of the company that owns the fleet) as a whole, and possibly the specific constraints imposed by the municipality (e.g., in terms of traffic or pollution).

Overall, then, the management of such fleets will resemble the management of collective adaptive systems that are called to operate in open-ended and unpredictable environments [4]. Accordingly, software infrastructures in charge of the fleet will have to become self-adaptive in their behavior [17], i.e., capable of dynamically adapting their behavior without human supervision. Thus, they can respond to changing situations and unexpected contingencies without suffering malfunctionings or degrading of quality of service.

In the past few years, several research works have been devoted to identify models [18, 7], languages [14, 6], and tools [2], to support the development of collective self-adaptive software systems. However, a key issue that is still open is the identification of general modeling frameworks to help tackling the many complex issues associated with the proper engineering of collective self-adaptive systems. These issues include: proper analysis and verification of functional and non-functional requirements of self-adaptation, and the analysis and identification of the knowledge requirements, i.e., of which information must be made available to a system to support its self-adaptive behavior.

To tackle this issue, we previously proposed [1] a sort of "black-box" approach to adaptation in which, abstracting from the actual mechanisms via which to achieve adaptation, we questioned about "what adaptation is for" from the viewpoint of system requirements and observable dynamic behavior of a system. The result of this process is SOTA ("State Of The Affairs"), a robust conceptual framework that, by grounding on the lessons of goal-oriented requirements engineering [12], dynamical systems modeling [20] and multidimensional context modeling [13], can provide effective conceptual support to self-adaptive software development.

In particular, the key idea in SOTA is to perceive a self-adaptive software system, like the one needed for the management of collectives of self-driving vehicles, as a sort of complex dynamic system immersed in a virtual n-dimensional phase space, each dimension being associated to either some internal software parameters or some external environmental parameters of interest for the execution of the system. The adaptive execution of the system can then be modeled in terms of movements in such space. Functional requirements (i.e., goals) are associated to areas of the phase space the system has to reach, non-functional requirements are associated to the trajectory the system should try to walk through, whereas self-adaptation is associated to the capability of the system to re-join proper trajectories when moved away from it. For example, a fleet of self-driving vehicles could have a functional requirement of arriving at destination on time, and a non-functional requirement of keeping the overall energy consumption below a certain threshold.

Indeed, in the area of complex software systems, it has been extensively argued that dynamical systems modeling can be a powerful tool to analyze the behavior of complex systems [23], and several studies exist in that direction (e.g., [19]). SOTA commits to the above perspective, but it adopts a totally different endeavour. In fact, it exploits dynamical systems as a means to model and engineer the behavioral and awareness requirements, rather than as a means to analyze the behavior of existing systems.

The exploitation of the SOTA model in the engineering of self-adaptive systems (possibly in conjunction with, and complementing, more traditional conceptual tools for goal-oriented requirements engineering [12, 15]), and in particular of collective systems of self-driving vehicles, brings several advantages:

- SOTA can be used as a tool to support the process of identifying which knowledge must be made available to the system and its components, and what degree of situation awareness they should reach to support adaptivity [22];
- SOTA can be used to early assess self-adaptation requirements via model-checking techniques [3], towards a better and more sound process of requirements engineering for self-adaptive systems.

The remainder of this paper is organized as follows. Section 2 provides an overview of the SOTA model. Section 3 shows how SOTA can be applied to a scenario of a collective of self-driving vehicles. Section 4 discusses how SOTA can be adopted to assess the knowledge or awareness requirements. Section 5 shows how the SOTA model can be an effective tool for the early assessment of requirements for self-adaptive systems via model checking. Finally, Section 6 concludes the paper.

2 SOTA Model

SOTA builds on existing approaches to goal-oriented requirements engineering [12, 15] and, for modeling the adaptation dimension, it integrates and extends recent approaches on multidimensional modeling of context, such as the "Hyperspace Analogue to Context" (HAC) approach [13]. In particular, such generalization and extensions are aimed at enriching goal-oriented and context modeling with elements of dynamical systems modeling [20], so as to account for the general needs of dynamic self-adaptive systems and components.

The term SOTA stems from "State Of The Affairs", which is a concept central in SOTA. The state of the affairs of a system is intended as any characteristics of the system itself and of the environment in which it lives and executes that may affect its behavior and that may be relevant with respect to its capabilities of achieving the objectives it was built for. In other words: (i) given a specific state of the affairs, i.e., the overall situation in which the system is; (ii) given that the state of the affairs can change due to both the internal activities of the system and the external dynamics of the environment; and (iii) a self-adaptive system must be able to trigger internal activities that enable it to achieve desirable state

of the affairs (i.e., the goals or objectives it was built for) despite the external dynamics.

2.1 SOTA Space

SOTA assumes that the current "state of the affairs" $S(t)$ at time t, of a specific entity e (let it be an individual component or an ensemble of components) can be described as a tuple of n s_i values, each representing a specific aspect of the current situation of the entity/ensemble and of its operational environment:

$$S(t) = \langle s_1, s_2, \ldots, s_n \rangle$$

As the entity executes, S changes either due to the specific actions of e or because of the dynamics of e's environment. Thus, we can generally see this evolution of S as a movement in a virtual n-dimensional space $\mathbf{S}$ (see Fig.1):

$$\mathbf{S} = \mathbf{S}_1 \times \mathbf{S}_2 \times \ldots \times \mathbf{S}_n$$

Or, according to the standard terminology of dynamical systems modeling, we can consider $\mathbf{S}$ as the phase space of e and its evolution that can be caused by internal actions or by external contingencies as a movement in such phase space.

To model such evolution of the system in terms of "transitions", $\theta(t, t+1)$ expresses a movement of e in the $\mathbf{S}$, i.e.,

$$\theta(t, t+1) = \langle \delta s_1, \delta s_2, \ldots, \delta s_n \rangle, \delta s_1 = (s_1(t+1) - s_1(t))$$

A transition can be endogenous, i.e., induced by actions within the system itself, or exogenous, i.e., induced by external sources. The existence of exogenous transitions is particularly important to account for. In fact, the identification of such sources of transitions (i.e., the identification of which dimensions of the SOTA space can induce such transitions) enables identifying what can be the external factors requiring adaptation.

2.2 Goals and Utilities

The requirements of a complex software (and more generally ICT) system can be naturally expressed in terms of the general objectives it has to achieve, which in turn typically decomposes into specific goals [9], to be achieved by either individual entities of the system or ensembles of entities.

A *goal* by definition is the eventual achievement of a given state of the affairs. Therefore, in very general terms, a specific goal G_i for the entity e can be represented as a specific point, or more generally as a specific area, in the SOTA space. That is:

$$G_i = A_1 \times A_2 \times \ldots \times A_n, A_i \subseteq \mathbf{S}_i$$

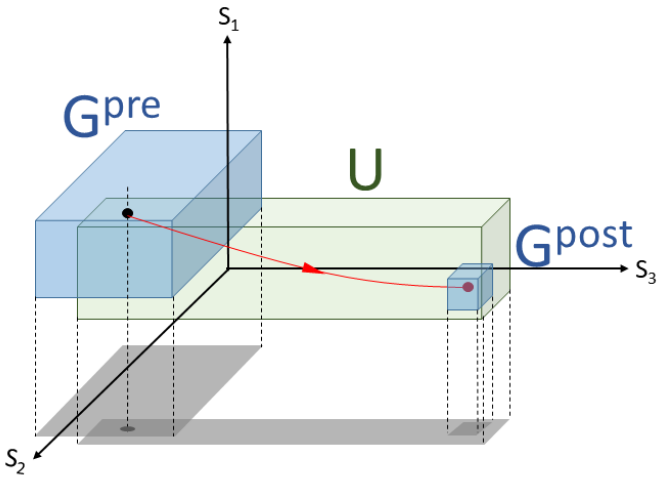

Fig. 1. The trajectory of an entity in the SOTA space, starting from a goal precondition and trying to reach the postcondition while moving in the area specified by the utility.

More specifically, a goal G_i of an entity e may not necessarily be always active. Rather, it could be the case that a goal of an entity will only get activated when specific conditions occur. In these cases, it is useful to characterize a goal in terms of a precondition G_i^{pre} and a postcondition G_i^{post}, to express when the goal has to activate and what the achievement of the goal implies. Both G_i^{pre} and G_i^{post} represent two areas or points in the space $\mathbf{S}$. In simple terms, when an entity e finds itself in G_i^{pre} the goal gets activated and the entity should try to move in $\mathbf{S}$ so as to reach G_i^{post}, where the goal is to be considered achieved (see Fig.1). Clearly, a goal with no precondition is like a goal whose precondition coincides with the whole space, and it is intended as a goal that is always active.

As goals represent the eventual state of the affairs that a system or component has to achieve, they can be considered functional requirements. However, in many cases, a system should try to reach its goals by adhering to specific constraints on how such a goal can be reached. By referring again to the geometric interpretation of the execution of an entity as movements in the space $\mathbf{S}$, one can say that sometimes an entity should try or be constrained to reach a goal by having its trajectory be confined within a specific area (see Fig. 1). We call these types of constraints on the execution path that a system/entity should try to respect as *utilities*, to reflect a nature that is similar to that of non-functional requirements.

As goals, a utility U_i can be typically expressed as a subspace in $\mathbf{S}$, and can be either a general one for a system/entity (the system/entity must always respect the utility during its execution) or one specifically associated to a specific goal G_i (the system/entity should respect the utility while trying to achieve the goal). For the latter case, the complete definition of a goal is:

$$G_i = \{G_i^{pre}, G_i^{post}, U_i\}$$

In some cases, it may also be helpful to express utilities as relations over the derivative of a dimension. That is to express not the area the trajectory should stay in but rather the *direction* to follow in the trajectory (e.g., try to minimize execution time, where execution time is one of the relevant dimensions of the state of affairs). It is also worth mentioning that utilities can derive from specific system requirements or can derive from externally imposed constraint.

A complete definition of the requirements of a system-to-be thus implies identifying the dimensions of the SOTA space, defining the set of goals (with pre- and postconditions, and possibly associated goal-specific utilities) and the global utilities for such systems, that is the sets:

$$\mathbf{S} = \mathbf{S}_1 \times \mathbf{S}_2 \times \ldots \times \mathbf{S}_n$$

$$\mathbf{G} = \{G_1, G_2, \ldots, G_m\}$$

$$\mathbf{U} = \{U_1, U_2, \ldots, U_p\}$$

Of course, during the identification of goals and utilities, it is already possible to associate goals and utilities locally to specific components of the system as well as globally, to the system as a whole. Thus, the above sets can be possibly further refined by partitioning them among local and global ones.

3 Engineering Collectives of Self-driving Vehicles

In this section, we exemplify how it is possible to address the problem of engineering collectives of self-driving vehicles (i.e., the management of a fleet of autonomous vehicles for car sharing services) using the SOTA approach. Such a scenario is characterized for being immersed in unpredictable and dynamic environment, which makes features of self-adaptation particularly important. Let us now conceptualize the problem in detail.

- A fleet F has a set of vehicles. A vehicle V_i has a set of planned rides, i.e., $R = \{R_1, R_2, \ldots, R_n\}$. Each planned ride is defined by a location L_i, a starting time ${}_iT_S^R$, and duration ${}_iD^R$. A route alternative can be provided from ride R_i to R_{i+1} as ${}_iR^D$.
- The departure time and arrival time of a vehicle are provided by ${}_iT_S^D$ and ${}_iT_E^D$, respectively.
- The battery state of charge or energy level of a vehicle V_i at departure time is defined by ${}_iE_S^D$, while ${}_iE_E^D$ specifies battery level at arrival time.
- The goal of a vehicle V_i is to arrive in time at the appointment, so that ${}_iT_E^D <= {}_iT_S^D$, and the battery level should never run out, so it is required that ${}_iE_E^D > 0$.
- The charging of a vehicle could occur during the appointment duration.

– A set of parking lots can be present where each one is defined by a name $PLname$. Similarly, a set of charging stations can be defined with each one having a name $CSname$. The available parking spaces and charging stations in location L can be defined as $ParkSpotsNum$ and $ChargeSpotsNum$.

The overall transportation system can be conceptually modeled as SOTA entities (e) moving in the SOTA space ($\mathbf{S}$). These entities could be an individual entity (e.g., vehicle), or a group of entities (e.g., fleet of vehicles; infrastructure resources, such as parking lots, charging stations and roads). Both vehicles and the fleet can be modeled in terms of entities that have goals (G_i), which can be at the individual (single vehicle) or global level (a fleet). Similarly, utilities (U_i) can be identified, related to how such goals can be achieved, also at the individual or global level. In the SOTA space, the locations, departure or arrival times, and battery energy levels correspond to the dimensions of the SOTA space, while a goal or utility can be represented as a specific point or area of the phase space.

From the point of view of a *vehicle* (V_i), a key goal is to reach the destination within time and battery energy level. We can characterize this goal in terms of a precondition and a postcondition to express when the goal has to achieve and what the achievement of the goal implies. For example, the preconditions can be (G_i^{pre}) to check whether the list of planned rides is known; the parking lots are assigned; the charging stations are assigned; and the battery state of charge level is sufficient at trip start. The postcondition (G_i^{post}) can be the actual goal itself, such as reach the destination within the time allocated and within the battery state of charge level. In the same manner, utilities (U_i) can be identified at the individual vehicle level, as a general one or as one associated to a specific goal of the vehicle.

For example, Fig. 2 illustrates a portion of the case study. A vehicle V_i starting at L_0 has the goal of completing planned ride R_i and arrive at location L_i (for simplicity of drawing we consider a one dimensional spatial extend – road – indicated by L). The figure illustrates a part of the SOTA space focusing on three dimensions: location, time and battery level. For readability, we represent both the 3D space and also 2D projections. The goal G_i of vehicle V_i is represented by the blue box: the vehicle has to be at location L_i at ${}_iT_S^R$ time, with a battery level greater than 0. The utility U_i (i.e., non-functional requirements) of V_i is represented by the green box: the vehicle battery level should not become too low. The state of vehicle V_i is a point in this space and its actions describe the red trajectory in the space: V_i reach its goal on time consuming some battery, but remaining within the non-functional requirements/boundaries.

From the viewpoint of a *fleet* (F), we can associate goals and utilities to the system as a whole, i.e., globally. Maximizing the usage of vehicles is a key requirement for the fleet. In this regard, we can identify two goals (G_i) for the fleet F. They are: (i) distributing the vehicles of the fleet fairly in the city at midnight every day; and (ii) creating and assigning trips for the vehicles. The preconditions (G_i^{pre}) for the *distribute vehicles* goal are checking whether the time is midnight, and distribution of the vehicles in the city is imbalanced. The postcondition (G_i^{post}) is the actual redistribution of the vehicles in a fair

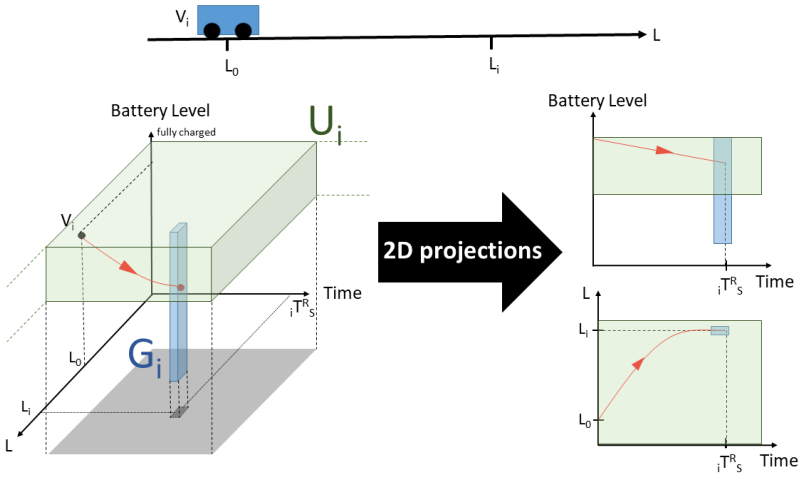

Fig. 2. SOTA space in the case study. The figure illustrates a portion of the case study. A vehicle V_i starting at L_0 has the goal of completing planned ride R_i and arrive at location L_i. The goal G_i of vehicle V_i is represented by the blue box. The utility U_i of V_i is represented by the green box.

and balanced manner. In the meantime, the precondition for the *create trips* goal can be to check whether the vehicles are available before assigning trips for them. The postconditions can be whether the rides list, parking lots and charging stations have been assigned for the vehicles.

As for global utilities (U), these can be a general one for the fleet or one specifically associated to a specific goal of the fleet. For example, there can be a utility for all the vehicles in the fleet to avoid roads with tolls or avoid localities that have disruptions. Some global utilities of the fleet which can be expressed as relations over the derivative of a dimension are: maximize usage of vehicles in the fleet, minimize journey time or cost, and minimize battery consumption.

As provided for a vehicle, we can identify the relevant dimensions ($\mathbf{S}$) of the state of affairs for a fleet. They are: current locations of the vehicles in the fleet; availability of the vehicles; availability and capacity of the infrastructure resources; battery energy levels of the vehicles; current traffic information; and journey times and costs.

4 SOTA Space and Knowledge Requirements

The "state of the affairs" is a very general concept, and its dimensions include anything relevant to keep a system up and running (i.e., hardware, software, environmental features) [1]. Therefore, identifying what are the relevant dimensions around which to model the SOTA space is a necessary activity towards the building of a self-adaptive system. However, when discussing about self-adaptive

systems, such identification also directly relates to identifying: (i) the knowledge (i.e., what dimensions) that must be made available to entities to enable self-adaptation; (ii) the type of sensors (i.e., physical or virtual) that must be available from components to gather the necessary knowledge.

4.1 Identification

For both of them, areas $A_i \subseteq \mathbf{S}_i$ can be expressed in terms of conditional expressions over the values in $\mathbf{S}_i$. In the case where one dimension S_i is not relevant for either a specific goal or utility, then $A_i \equiv \mathbf{S}_i$.

Indeed, in many practical cases, goals and utilities involve only a limited fraction of the state space. That is, G_i is expressed as a set of points or areas in an m-dimensional space (with $m < n$), projection of the original space. If we consider a base vector: $B =< b_1, b_2, \ldots b_n >, b_i \in \{0, 1\}$ such that $b_i = 0 \Leftrightarrow \forall G_i \in \mathbf{G} \wedge \forall U_i \in \mathbf{U} \longrightarrow A_i \equiv \mathbf{S}_i$, then goals and utilities can be expressed in the sub-dimensional space: $\mathbf{SS} = B \times \mathbf{S}$. The sub-dimensional space SS is important because it defines what information is relevant for the system. That is, it drives the requirements for which knowledge has to be acquired, modeled, and made available to services.

In addition, one should also account for specific contingency situations of SOTA that may affect the capability of a system of achieving its goal in respect of the utilities, and that are not explicit in either $\mathbf{G}$ or $\mathbf{U}$. It may be necessary to identify these contingencies, identify when and how they could affect the capability of the system, and turn these explicitly either as utility functions or as "impossible areas", i.e., areas of the SOTA space in which the system, however self-adaptive, will no longer be able to achieve.

Let us examine some contingency situations with respect to a vehicle and a fleet. As mentioned in Section 3, a vehicle has a goal of reaching a destination within time and battery energy level. However, during mobility it could find that the assigned parking lot is no longer available, or its battery level is running out. At the same time, a fleet has a goal of creating trips for the vehicles, but a vehicle in the fleet could leave later than its scheduled time for an appointment, thus affecting the trips of the other vehicles in the fleet. Some of these contingency situations could affect the capability of the system, and can be shown in the SOTA space as impossible areas that are no longer achievable. For instance, a fleet has a goal of distributing the vehicles of the fleet in a balanced manner in the city at midnight every day. However, there could be a disruption in the road due to maintenance work in a particular area of the city, which is unavoidable. This will result in some vehicles not being distributed fairly.

So far, we assume that all dimensions in $\mathbf{S}$ are independent from each other; that is, a movement in $\mathbf{S}_i$ does not affect the positions in the other dimensions $\mathbf{S}_j$. Unfortunately, this is not always the case: the characteristics of the domain can induce additional constraints. For instance, in a vehicle, its driving style (e.g., speed) and battery state of charge level are interlinked, a change in speed implies a change in battery state of charge. Also, the list of planned rides of a vehicle could be affected by other dimensions, such as availability of the infrastructure

resources (e.g., parking lots), and current traffic information. Similarly, in a fleet, the locations of the vehicles could be constrained by the availability of the infrastructure resources. Therefore, along with the identification of the goals and utility sets **G** and **U**, it can be useful to identify constraints on the SOTA dimensions and on the "trajectories" that a system can follow on them.

4.2 Sensors and Virtualization

Each dimension of the SOTA space implies sensors. However, this is not an issue per se: a number of different sensors are available to measure the most diverse features. For example, an e-vehicle has a range of sensors to measure different SOTA context dimensions, such as: GPS (global positioning system) sensor, accelerometer sensors, ABS (anti-lock brake system) sensors, gyrometer sensors, steering angle sensors, sensors in the battery, and temperature and humidity sensors inside the vehicle.

Most of these sensors report the values in terms of numeric time series, whereas within the SOTA space, it will be better to consider movements as represented in values which are meaningful. The problem lies in providing services with an appropriate view of what's happening, i.e., leveraging the low-level perspective of the actual sensors into that of a "virtual sensor" which is capable of providing an appropriate view representation of the values in that dimension. In general, virtual sensors are useful for: (i) grouping a number of physical sensors for the sake of fault tolerance; (ii) converting sensor readings into relevant information; and (iii) grouping different physical sensors allowing multi-modal recognition capabilities. During the modeling of a system, the issue of identifying what types of virtual sensors are required to enable and facilitate adaptation is thus necessary to properly drive activities related to knowledge modeling and processing. The latter is required to turn physical sensors into virtual ones.

Another important aspect of the virtualization process is that it detaches the provisioning of the virtual information from that of the actual sensors. Let us consider some examples of virtual sensors in the case study for a vehicle: (i) battery state of charge: the charge level of the battery which can be determined using the current and voltage measurements from the battery's sensors. Similarly, battery state of health can be calculated to indicate the overall condition of the battery; (ii) a virtual sensor to measure dynamics of a vehicle: the angle of the steering from the steering angle sensor can be used to determine where the front wheels are pointed. This measurement when combined with measurements from the yaw, accelerometer and wheel speed sensors, it is possible to measure the dynamics of the vehicle which can be used by the stability control system of the vehicle; (iii) climate comfort sensor: the temperature and humidity sensors inside the vehicle can be used to calculate climate comfort level for the user, which can be *eco* or *maximal*.

An example of a virtual sensor for the fleet is calculating and determining whether current distribution of vehicles in a fleet is fair or imbalanced, depending on the individual locations of the vehicles which can be acquired through their

individual GPS sensors and by the aggregation of all individual locations into a sort of aggregate indicator of imbalance in fleet distribution.

5 Model Checking SOTA Requirements

It is possible to adopt SOTA as an effective tool to perform an early, goal-level, *model checking analysis* [11] for self-adaptive systems. Our approach allows the developers of complex self-adaptive systems to validate the actual correctness of the self-adaptive requirements at an early stage in the software life-cycle.

SOTA supports a simple operational model [3] that makes it possible to adapt and apply existing model-checking techniques to goals and utilities, and thus assess and improve requirements identification. Our target event-based model for reasoning about goals and utilities is labeled transitions systems (LTSs), which provides a simple formalism for compositional reasoning in architectural context. This formalism is supported by a tool that provides a wide range of analysis and animation capabilities. Thus, our model checking approach is based on the formal verification, validation and simulation techniques provided by the Labeled Transition System Analyzer (LTSA) [11], and its process calculus Finite State Processes. The formalism that we use to model goals and utilities is Fluent Linear Temporal Logic (FLTL) assertions. The entities–a single *vehicle* and a *fleet*– represent the SOTA entities moving in the SOTA space. In operational terms, this can be expressed as multiple processes or LTSs, and the overall execution of the system modeled as a concurrent event-based one, in which the process transitions (of an exogenous or endogenous type) correspond to movements in the SOTA space.

As described in [3], the overall model checking process of SOTA requirements has four main stages: requirements modeling using i* framework [21], SOTA grammar and language, transform goals and utilities to asynchronous FLTL, and verification. We refer the reader to [3] on details of the operationalization of the SOTA model, and the application of the approach to simple e-mobility examples. This paper specifically focuses on the verification stage with case study exemplifications where model checking is applied to: (i) validate whether a set of required preconditions and postconditions forms a complete operationalization of a single goal (i.e., single goal operationalization); and (ii) check the satisfaction of global goals or utilities.

5.1 Validate Single Goal Operationalization

As mentioned in Section 2.2, a goal G_i can be characterized by a precondition G_i^{pre} and a postcondition G_i^{post}. Also, a goal G_i can be associated to a utility U_i that needs to be respected while trying to achieve G_i. In the SOTA model, these goals and utilities are expressed as a subspace in $\mathbf{S}$. For validating single goal operationalization, we check whether a set of preconditions, postconditions and/or a goal-associated utility forms a complete operationalization (i.e., all the conditions in the set of preconditions, postconditions and/or a goal-based utility

are satisfied) of a requirement [3]. This is to ensure that the operationalization of goals and utilities has been performed correctly by the engineer from the SOTA model.

To achieve this, the assertions created for a requirement (e.g., preconditions, postconditions) are composed with the event-based behavioral model, and then model checking can be performed using the LTSA. If the operationalization of the requirement is incomplete, for example, let us assume that a required precondition has been omitted inadvertently, then the LTSA model checker will generate a counterexample trace identifying the error. For example, as mentioned in Section 2, the reach destination goal of a vehicle (V_i) has a precondition (G_i^{pre}) to check the battery state of charge is sufficient at trip start. Also, there can be a utility (U_i) to ensure climate comfort level inside the vehicle is maintained while the vehicle reaches its destination. The postcondition of the goal (G_i^{post}) is the actual goal itself, which is to reach the destination within the battery state of charge level and on time. Now assume that the precondition for this goal has been omitted during the operationalizing process by the engineer inadvertently. This will result in violating of the assertion created for the goal. Thus, the model checker will generate a counterexample (error) trace annotated with the constraints which were violated, which can be used by the engineer to identify the error and correct the requirements model.

5.2 Validate Global Goal/Utility Satisfaction

In addition to checking single goal operationalization, we can perform model checking to check the satisfaction of global goals or global utilities by operational models that describe the behavior of multiple components. Such validation will ensure that the operationalization of the global goals and utilities from the SOTA model has been performed correctly by the engineer. For this, we check whether a set of goals or utilities forms a complete operationalization (i.e., all the goals or utilities are satisfied) of a global requirement. In SOTA terms, such validation means the checking the requirements of the set of goals G or utilities U:

$$\mathbf{G} = \{G_1, G_2, \ldots, G_n\}, |\mathbf{G}| > 1$$
$$\mathbf{U} = \{U_1, U_2, \ldots, U_n^e\}, |\mathbf{U}| > 1$$

For example, in the case study, a global goal (G) for the fleet is to maximize usage of vehicles. This global goal can be composed of two goals on distributing the vehicles of the fleet fairly in the city at midnight every day; and creating and assigning trips for the vehicles. In another example, the global utility to avoid roads with tolls for the fleet can be composed of the utilities of individual vehicles in the fleet. To validate these, the assertions created for the goals and utilities can be composed, and then model checking can be performed by the LTSA to check the overall satisfaction of the global goal or utility. If the operationalization is incomplete, for instance, a required goal or utility has been omitted by the engineer, the LTSA will produce an error trace which can be used by the engineer to locate the error and correct it.

In this manner, by performing validation of single and global goal operationalization, we identify any incompleteness of the SOTA goal-oriented requirements model. However, a typical problem that may occur in goal-oriented modeling is that an *inconsistency* or an *implicit requirement* [10] can result in a deadlock in the specification, as discussed next.

5.3 Detect Inconsistencies

An inconsistency in the specification could occur due to several reasons. These can be (i) if the postcondition of a goal does not imply its precondition then the system might be in a state where the postcondition G_i^{post} is true but the precondition G_i^{pre} is not true. So the goal needs to be satisfied but it is not, leading to an inconsistency; (ii) if the operational model is derived from conflicting goals. Therefore, it is important to detect inconsistencies in the SOTA operational model as deadlocks.

To illustrate an example of the first type, in the e-mobility case study, let us consider that the reach destination goal of a vehicle V_i has a precondition (G_i^{pre}) to check whether a parking lot and a charging station have been assigned. There could be a situation where the precondition of the goal is not satisfied, i.e., charging station has not been assigned for the trip. This is although the vehicle is able to reach the destination within the time and energy levels (i.e., the postconditions G_i^{post} are satisfied).

As for the inconsistencies that occur from conflicting goals, let us consider two entities SC_1 and SC_2 that are to be composed into an ensemble or group of entities SCE. First, assume that SC_1 and SC_2 have two shared goals G_i and G_j, which share the same n-dimensional SOTA space S. The preconditions of the two goals overlap but the postconditions do not overlap. That is:

$$G_i^{pre} \cap G_j^{pre} \neq \emptyset \wedge G_i \cap G_j = \emptyset$$

Therefore, both these goals could be activated and pursued at the same time in two paths in the SOTA space S, and this should not be the case. Second, assume that SC_1 and SC_2 have two goals G_i and G_j and the goals' preconditions and postconditions both overlap. That is:

$$G_i^{pre} \cap G_j^{pre} \neq \emptyset \wedge G_i \cap G_j \neq \emptyset$$

Therefore, both these goals could be activated and pursued at the same time in the same direction of the SOTA space S. We can perform LTSA model checking to detect such inconsistencies that arise from conflicting goals as deadlocks in the specification. Next in order to describe the inconsistencies that can occur from conflicting goals, two examples from the e-mobility case study are provided.

For the first conflicting goals situation, assume that there are two vehicle entities (V_1 and V_2) of the fleet F, which have been composed into an ensemble. Let us consider that these two vehicles have been assigned to the same user for a trip that has two planned rides (R_1 and R_2). That is, vehicle V_1 has been assigned a ride R_1 to travel to the first appointment at location L_1, and afterwards vehicle

V_2 has a ride R_2 to travel to the second appointment at location L_2. Here, both vehicles (V_1 and V_2) can have the same preconditions at the trip start, such as they are available at the time of trip creation. However, now assume that during mobility, if vehicle V_1 reaches an insufficient level of battery charge level, and it is not able to reach location L_1 in time, then the postconditions of both V_1 and V_2 entities do not match any more. That is, the goals of these entities will be conflicting. In such a situation, the system may be in a state where both operations could take place in two paths, thus leading to an inconsistency.

On the other hand, for the second conflicting goals situation, consider two vehicles V_1 and V_2 of a fleet F that have been assigned to the user for the same trip. Here, both vehicles will have overlapping preconditions and postconditions as they require to reach the same destination within the time allocated. Here the system could be in a state where both operations taking place towards the same direction, which should not be the case. This is because there is no next state that will satisfy both the postconditions of the two goals. These inconsistencies in the SOTA operational model can be overcome, first through the explicit modeling of additional constraints to handle them, and then composing them with the event-based behavioral model and performing LTSA model checking.

6 Conclusions and Future Work

Future fleets of self-driving vehicles will be examples of collective adaptive systems that are required to act in a coordinated way towards the harmonized achievement of both individual and collective goals. This paper presented the SOTA approach for the engineering of such kind of systems, focusing in particular on proper analysis and modeling of functional and non-functional requirements of self-adaptation, and the analysis and identification of which information must be made available to a system to support its self-adaptive behavior.

Future work will exploit the results of the SOTA modeling as a guide towards the adaptive identification of the most suitable coordination patterns for a fleet of self-driving vehicles, depending on their current operating conditions.

References

1. Abeywickrama, D.B., Bicocchi, N., Zambonelli, F.: SOTA: Towards a general model for self-adaptive systems. In: Proceedings of the IEEE 21st International WETICE'12 Conference. pp. 48–53 (Jun 2012)
2. Abeywickrama, D.B., Hoch, N., Zambonelli, F.: Engineering and implementing software architectural patterns based on feedback loops. Scalable Computing: Practice and Experience **15**(4) (2014)
3. Abeywickrama, D.B., Zambonelli, F.: Model checking goal-oriented requirements for self-adaptive systems. In: Proc. of the 19th IEEE International Conference and Workshops on Engineering of Computer-Based Systems. pp. 33–42 (Apr 2012)
4. Belzner, L., Hölzl, M.M., Koch, N., Wirsing, M.: Collective autonomic systems: Towards engineering principles and their foundations. Transactions on Foundations for Mastering Change **1**, 180–200 (2016)

5. Bicocchi, N., Mamei, M., Sassi, A., Zambonelli, F.: On recommending opportunistic rides. IEEE Trans. Intelligent Transportation Systems **18**(12), 3328–3338 (2017)
6. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: The SCEL language. TAAS **9**(2), 7:1–7:29 (2014)
7. Giese, H., Vogel, T., Diaconescu, A., Götz, S., Kounev, S.: Self-Aware Computing Systems, chap. Architectural concepts for self-aware computing systems, pp. 109–147. Springer (2017). https://doi.org/10.1007/978-3-319-47474-8_5
8. Hoch, N., Bensler, H.P., Abeywickrama, D.B., Bureš, T., Montanari, U.: The e-mobility case study, pp. 513–533. Springer International Publishing (2015). https://doi.org/10.1007/978-3-319-16310-9_17
9. Lapouchnian, A., Liaskos, S., Mylopoulos, J., Yu, Y.: Towards requirements-driven autonomic systems design. ACM SIGSOFT Software Engineering Notes **30**(4), 1–7 (2005)
10. Letier, E., Kramer, J., Magee, J., Uchitel, S.: Deriving event-based transition systems from goal-oriented requirements models. Automated Software Engineering **15**(2), 175–206 (June 2008)
11. Magee, J., Kramer, J.: Concurrency: State Models and Java Programs. John Wiley and Sons, second edn. (Apr 2006)
12. Mylopoulos, J., Chung, L., Yu, E.S.K.: From object-oriented to goal-oriented requirements analysis. Communications of the ACM **42**(1), 31–37 (1999)
13. Rasch, K., Li, F., Sehic, S., Ayani, R., Dustdar, S.: Context-driven personalized service discovery in pervasive environments. World Wide Web **14**(4), 295–319 (2011)
14. Salvaneschi, G., Ghezzi, C., Pradella, M.: Contexterlang: A language for distributed context-aware self-adaptive applications. Science of Computer Programming **102**(Supplement C), 20 – 43 (2015). https://doi.org/10.1016/j.scico.2014.11.016
15. Souza, V.E.S.: Requirements-based Software System Adaptation. Ph.D. thesis, DISI- University of Trento (2012)
16. Uhlemann, E.: Connected-vehicles applications are emerging [connected vehicles]. IEEE Vehicular Technology Magazine **11**(1), 25–96 (2016)
17. Weyns, D.: Software engineering of self-adaptive systems: An organised tour and future challenges. In: Taylor, R., Kang, K.C., Cha, S. (eds.) Handbook of Software Engineering. Springer (2017)
18. Weyns, D., Malek, S., Andersson, J.: Forms: Unifying reference model for formal specification of distributed self-adaptive systems. ACM Transactions on Autonomous and Adaptive Systems **7**(1) (2012)
19. Williams, R.A.: Lessons learned on development and application of agent-based models of complex dynamical systems. Simulation Modelling Practice and Theory (2017). https://doi.org/10.1016/j.simpat.2017.11.001
20. Yam, Y.B.: Dynamics of Complex Systems. Perseus Books (2002)
21. Yu, E.S.K.: Modelling strategic relationships for process reengineering. Ph.D. thesis (1995), UMI Order No. GAXNN-02887 (Canadian dissertation)
22. Zambonelli, F., Bicocchi, N., Cabri, G., Leonardi, L., Puviani, M.: On self-adaptation, self-expression, and self-awareness in autonomic service component ensembles. In: 2011 Fifth IEEE Conference on Self-Adaptive and Self-Organizing Systems Workshops. pp. 108–113 (Oct 2011). https://doi.org/10.1109/SASOW.2011.24
23. Zambonelli, F., Parunak, H.V.D.: Signs of a revolution in computer science and software engineering. In: Proceedings of the 3rd International Conference on Engineering Societies in the Agents World III. pp. 13–28. ESAW'02, Springer-Verlag, Berlin, Heidelberg (2003)