Noname manuscript No.
(will be inserted by the editor)

The SOTA Approach to Engineering Collective Adaptive Systems

Dhaminda B. Abeywickrama · Nicola Bicocchi · Marco Mamei · Franco Zambonelli

Received: date / Accepted: date

Abstract The emergence of collective adaptive systems – i.e., computational systems made up of an ensemble of autonomous components that have to operate in a coordinated and adaptive way in open-ended and unpredictable environments – calls for innovative modeling and software engineering tools, to support their systematic and rigorous design and development. In this paper, we present a general model for collective adaptive systems called SOTA ("State Of The Affairs"). SOTA brings together the lessons of goal-oriented requirements modeling, context-aware system modeling, and dynamical systems modeling. It has the potential for acting as a general reference model to help tackling some key issues in the design and development of collective adaptive systems. In particular, as we will show with reference to a scenario of collectives of autonomous vehicles, SOTA enables: early verification of requirements, identification of knowledge requirements for self-adaptation, and the identification of the most suitable architectural patterns for self-adaptation.

Keywords Collective adaptive systems · Goal-oriented requirements engineering · Context-awareness · Architectural patterns · Model checking · Autonomous vehicles.

D. B. Abeywickrama
Department of Computer Science, University of Warwick, Coventry, United Kingdom
E-mail: dhaminda.abeywickrama@gmail.com

N. Bicocchi, M. Mamei, F. Zambonelli
Dipartimento di Scienze e Metodi dell'Ingegneria, University of Modena and Reggio Emilia, Reggio Emilia, Italy
E-mail: {name.surname}@unimore.it

1 Introduction

In the past few years, several research works have been devoted to identify models [59,22], languages [51,20], and tools [3], to support the development of collective self-adaptive software systems. That is, systems composed by a (possibly large) number of autonomous components, that are called to operate in a coordinated way in open and unpredictable environments [7]. Therefore, these systems must able to dynamically adapt their behavior without human supervision in order to respond to changing situations and unexpected contingencies, without suffering malfunctionings or degrading of quality of service [29].

However, despite the great deal of recent research, what we think is still missing is the identification of general modeling frameworks to help tackling – in a uniform way – some of the key 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; 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; and conceptual means to support designer in choosing the most suitable architecture to support adaptive behavior [58].

To tackle these issues, we previously proposed [2] a "black-box" approach 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 proposal is SOTA ("State Of The Affairs"), a robust conceptual framework that, by grounding on the lessons of goal-oriented requirements engineering [43], dynamical sys-

tems modeling [63] and multidimensional context modeling [48], can provide effective conceptual support to self-adaptive software development.

In particular, the key idea in SOTA is to model a self-adaptive software system in terms of a 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, a non-functional requirement of keeping the overall energy consumption below a certain threshold, and should be capable of adhering to the schedule despite traffic conditions.

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 [68], and several studies exist in that direction (e.g., [61]). SOTA commits to the above perspective, but it adopts a totally different perspective. 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 as a tool for the engineering of self-adaptive systems (possibly in conjunction with, and complementing, more traditional conceptual tools for goal-oriented requirements engineering [43,53], and other existing tools for modeling and engineering collective adaptive systems [7]), can bring 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 [66];
- SOTA can be used for the early assessment of self-adaptation requirements via model-checking techniques [5], towards a better and more sound process of requirements engineering for self-adaptive systems.
- SOTA can effectively support designers in the selecting the most appropriate architectural design patterns [21,14] to integrate the identified self-adaptation features in the system (in other words, supporting designers in the transition from "black box" analysis to "white box" design).

To exemplify how SOTA can be exploited and to show how it can provide the above claimed advantage, we will make reference to a case study in the area of autonomous vehicles. In particular, 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 [9,28], and to the delivery of merchandise [55]. 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 problem of managing of such fleets can be assimilated to the general problem of managing of collective adaptive systems operating in unpredictable environments and capable to adapt to changing situations.

This paper presents and consolidate the SOTA model by notably extending and rationalizing our previous work related to the SOTA model [2], model checking [5], architectural design [3], and engineering of self-driving vehicle collectives [4]. In this paper we present all these elements in a coherent conceptual framework supporting the SOTA model.

The remainder of the article is organized as follows. Section 2 provides an overview of the SOTA model. In Section 3, we show 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. How the SOTA model can be an effective tool for the early assessment of requirements for self-adaptive systems via model checking is shown in Section 5. The potentials of SOTA in guiding through the actual design of complex self-adaptive systems is shown in Section 6. Section 7 discusses related work, and finally Section 8 concludes the paper.

2 SOTA Model

SOTA builds on existing approaches to goal-oriented requirements engineering [43,53] 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 [48]. In particular, such generalization and extensions are aimed at enriching goal-oriented and context modeling with elements of dynamical systems modeling [63], 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 of these represent a specific aspect characterizing the current situation of the entity or a collective property of the system 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.

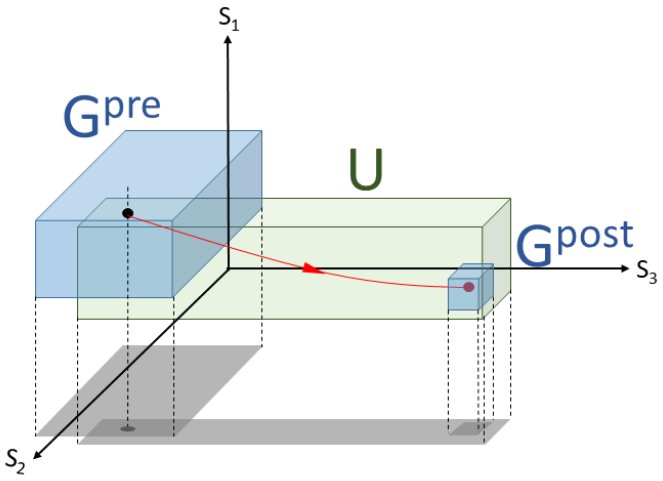

Fig. 1 The trajectory of an entity in the SOTA space, starting from a goal precondition and trying to reach the postcondition while moving in the area specified by the utility.

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 [34], 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:

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

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 collective as a whole. Thus, the above sets can be possibly further refined by partitioning them among local and global ones.

Adaptation needs, in a system modeled in SOTA terms, concerns the fact that during its evolution in the SOTA space, the system can be forced outside its planned trajectory and outside the boundaries defined by the utilities. Self-adaptation is thus the capability of the system to dynamically enact actions that enables it to restore its original trajectory and reach the goal in any case while respecting the utilities.

3 Engineering Collectives of Self-driving Vehicles

Future cities will integrate a variety of large-scale collective systems devoted to monitor and rule the different aspects of cities' life [65]. Among the others, regulating in an orchestrated way mobility services and the movements of autonomous vehicles devoted to provide them.

There are many scenarios in which collectives of self-driving vehicles will be required to dynamically and adaptively coordinate their movements: crossing intersections, entering and changing lanes in motorways, coordinating access to park spaces, as well as planning routes in order to avoid congestions and traffic jams. To keep focus without losing in generality, we focus on the specific problem of managing the collective movements of the fleet of electric vehicles of a car sharing company. There, beside the individual goal-oriented tasks that get assigned to the vehicles in the fleet (bringing clients to destination in an effective way) there are also fleet-level goals that the car sharing company may wish to achieve to maximize the effectiveness of the activities of the fleet as a whole. For example, balancing the distribution of fleet in the city according to the demand, maximizing the exploitation of owned charging stations and parking slots, minimizing the overall energy consumption of the fleet. Achieving such goals requires the collective and adaptive coordination of the vehicles in the fleet.

Let us now conceptualize the problem (deriving from a real-world case study in which we have applied SOTA [28]) 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 individual ones (e.g., a vehicle), or a group of entities (e.g., the fleet or the set 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 (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 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 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 the *fleet* (F), we can associate goals and utilities to the fleet 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 key goals (G_i). 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 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.

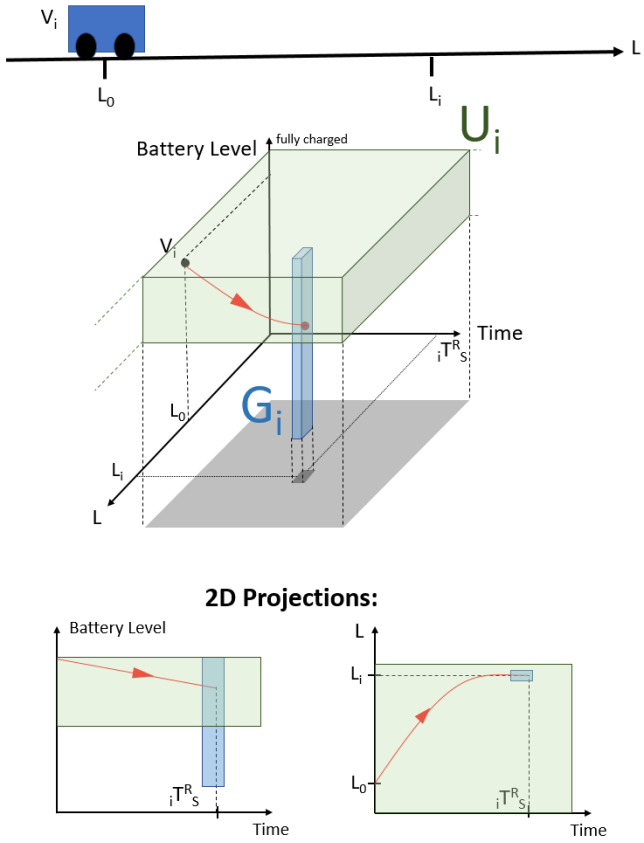

Fig. 2 SOTA space in the case study. The figure illustrates a portion of the case study. A vehicle V_i starting at L_0 has the goal of completing planned ride R_i and arrive at location L_i. The goal G_i of vehicle V_i is represented by the blue box. The utility U_i of V_i is represented by the green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

4 SOTA Space and Knowledge Requirements

The "state of the affairs" space at the basis of SOTA is a very general one, and its dimensions include anything relevant to keep a system up and running (i.e., hardware, software, environmental features) [2]. Therefore, identifying what are the relevant dimensions around which to model the SOTA space is a necessary activity towards modeling the domain of the self-adaptive system under study.

In addition, the identification of the dimension of the SOTA space also directly relates to identifying:

- the knowledge (i.e., what dimensions) that must be made available to entities to enable goal-orientedness and self-adaptation. That is, to recognize whether it is moving in the SOTA space according to the requirements, or it is getting diverted out of it;
- the type of sensors (i.e., physical or virtual) that must be available from components to gather the necessary knowledge.

4.1 Identification

In SOTA, achieving a goal implies reaching a specific position in the SOTA space (possibly, in the presence of utility, by following constrained trajectories). As a consequence, for an entity in a system, or for the collective as a whole, the capability of achieving a goal implies the capability of recognizing its position in the SOTA space.

However, after having identified the dimension composing the SOTA space, one may discover that for the identified goals and utilities some of these dimensions are not relevant. In other words, in many practical cases, goals and utilities involve only a limited fraction of the state space. More formally, when a goal 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 to be gathered during its activity. That is, it drives the requirements for which knowledge has to be acquired and processed by the entities and/or by the collective during their activities.

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 **G** or **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 in regard 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 $\mathbf{G}$ and $\mathbf{U}$, 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 that must be made available to the entities the system. 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, though, report the values in terms of low-level information (e.g., numeric time series). Instead, when reasoning about self-adaptation and goal-achievement within the SOTA space, it is more appropriate to consider movements along the dimensions of the SOTA space in terms of more expressive and meaningful representations. The problem, thus, 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 [52].

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. For instance, in the case study, cameras and other sensor within a car can exactly determine the positions and speed of other vehicles around. This is clearly necessary for supporting self-driving features. However, from the viewpoint of the SOTA goals, such detailed picture can be simply virtualized in terms of a virtual sensor in charge of detecting the overall traffic situations (e.g., fluid, intense, jammed).

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.

Concerning the collective level, 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* [38] 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 [5] 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) [38], 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 [5], the overall model checking process of SOTA requirements has four main stages: requirements modeling using i* framework [64], SOTA grammar and language, transform goals and utilities to asynchronous FLTL, and verification. We refer the reader to [5] on the details of the operationalization of the SOTA model, and the application of the approach to simple e-mobility examples. The operationalization of the SOTA model derives an event-based service component behavioral model, and FLTL assertions for goals and utilities, which is then verified using the LTSA model checker [5] (see Fig. 3). This paper specifically discusses on the verification stage of the model checking process 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); (ii) check the satisfaction of global goals or utilities; (iii) detect any inconsistencies and implicit requirements as deadlocks; and (iv) animate goal-oriented models using the standard animation and simulation features of the LTSA (see Fig. 3).

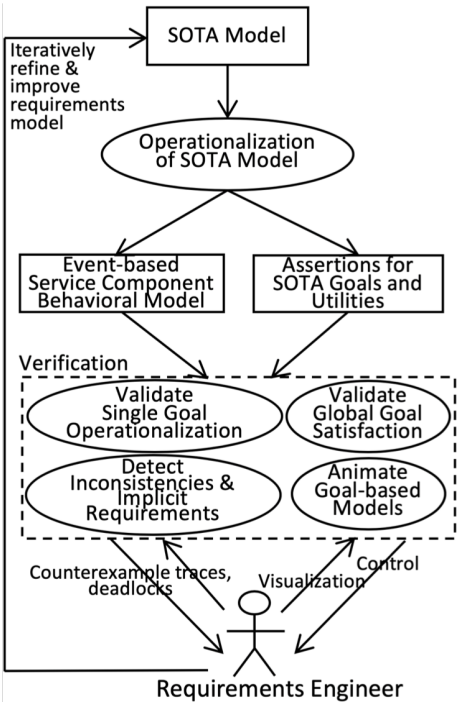

Fig. 3 Model checking SOTA goal-oriented requirements.

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 [5]. 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 3, 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* [35] 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.

5.4 Detect Implicit Requirements

Implicit requirements occur due to interactions between requirements on different goals. For example, a postcondition of a goal G_i^{post} may implicitly prevent another goal being applied even if all the preconditions for the second goal G_j^{pre} are true. This is because the actual condition G_i^{post} in which the goal is allowed to occur is stronger than the preconditions G_j^{pre} of that goal.

For example, let us consider three vehicle entities (V_1, V_2 and V_3) of a fleet F that have been composed into an ensemble for a journey. The three vehicles have been assigned three rides R_1, R_2 and R_3 respectively. The vehicle V_1 has been assigned a ride R_1 to take the user to appointment in a timely manner, but due to an unexpected event such as a disruption on the road, it is not able to arrive at appointment location L_1 in time. As a result, preventing entities V_2 and V_3 to apply their initial goal.

Like inconsistencies, implicit requirements can also cause deadlocks in the specification, if one does not specify additional properties to avoid them. Nevertheless, there is a benefit associated with implicit preconditions, as it allows requirements engineers to eventually derive a robust specification.

5.5 Animate Goal-based Models

The LTSA tool's animator can be used to explore model behaviors interactively with the stakeholders, and replay error traces generated during formal analysis. In the SOTA space, this animation provides a visualization of the evolution of the system, and it corresponds to the execution of "transitions" (i.e., $\theta(t, t+1)$) on the movement of e in the $\mathbf{S}$.

The specific advantages of animating goal-based models of SOTA are:

- animate the goals and utilities of the SOTA operational model and replay any error traces to the requirements engineer. This animation can be visualized in two ways: as a textual sequence of events, or as a graphical illustration of the LTSs;
- automatically detect any goal or utility violations during execution. To achieve this, the animated model can be composed with monitors which can be expressed as property processes. For example, as mentioned in Section 3, consider the global goal of distribute vehicles for a fleet. This goal can have preconditions to check whether it is midnight and distribution of the vehicles is imbalanced, and the postcondition is the actual redistribution of the vehicles in a balanced manner. Here, property processes can be specified to act as goal monitors for the two preconditions and the postcondition, which can then be animated using the LTSA.

In this manner, the counterexample traces generated and the deadlocks detected in the model checking process are used to iteratively refine and improve the SOTA requirements model, thus deriving a specification that is correct.

6 Self-Adaptation Patterns

The SOTA modeling approach is very useful for understanding and modeling the functional and adaptation requirements, and for checking the correctness of the specification [2]. However, when a designer considers the actual "white box" design of the system, it is important to identify which architectural schemes need to be chosen for the individual entities/components and the collective.

To this end, in previous work [14,45], we defined a taxonomy of architectural patterns for supporting adaptation at the individual and collective level. The SOTA model can be effective suggesting engineering which of this pattern better suit a system, depending on the characteristics of its SOTA space and of the identified goals and utilities. More in particular, the key questions that SOTA can help answering is: given the structure of the goals and utilities that the system-to-be has to achieve, which patterns better fit such structure?

A key assumption in the taxonomy of adaptation patterns is that self-adaptation requires the presence of a feedback loop (that is, a close control loop) [2]. A feedback loop is the part of the system that allows

for feedback and self-correction towards goals' achievement, i.e., self-adjusting behavior in response to the system's changes. A feedback loop can be directly integrated into a component (e.g., in goal-oriented software agents, the architecture of the agent embeds internal feedback loops and managing sub-systems supporting self-adaptation) or external (e.g., by way of adding to a inherently non-adaptable component one or more external managers closing the feedback loop and taking charge of adaptation for the controlled component). However, from the viewpoint of modeling and engineering adaptation provided by one or more "autonomic managers" (AMs for short), independently of whether these will be eventually embedded into the software component or implemented as external components.

Next we describe two patterns each from the single component level (i.e., autonomic component, parallel AMs component), and at the collective level (i.e., centralized ensemble, peer-to-peer AMs ensemble). Figure 4 illustrates these four patterns. Each pattern is described by providing its objective and the context it is used; the behavior; the SOTA space and the typical structures of goals and utilities satisfied by the pattern. The latter, in particular, is what can drive designer towards the choice of a specific patterns: whenever the structure of goals and utilities that can be satisfied by a pattern resembles those identified during the SOTA analysis, then such patterns is the most suitable starting point for architecting the system.

6.1 Patterns for a Single Component

6.1.1 ***Autonomic component pattern:***

For a single component to be adaptive, there is a need for at least one AM to manage it and to provide adaptation. The autonomic component (AC) pattern is composed of a controller component (CC) and an AM at the end of the loop to monitor and direct behavior. This pattern is characterized by an explicit external feedback loop (see Fig. 4-i) [14,2].

This patterns is useful to be adopted in three main situations: (i) a non-adaptable component that needs to be made adaptive; (ii) an already adaptable components that needs further adaptation capabilities to handle a different sets of utilities and goals; (iii) sharing knowledge and propagation of adaptation information is better managed using an external controller. This pattern has the capability of perceiving the SOTA space and applying actions to modify its position in the space in the medium term (i.e., it has utilities). Also, it can apply actions in the SOTA space in the long term, that is, it has goals. By using their sensors, the CC and the AM can monitor the different events and their changes in the environment. The AM has its own internal goals and utilities. Therefore, it is able to manage the adaptation inside the component, for instance, by changing the logic of choosing actions in response to a service request. The services, and goals and utilities of the CC (if any) are tightly coupled with the AM. That is:

$$\textbf{Goals} : \mathbf{G_{AC}} = \mathbf{G_{CC}} \cap \mathbf{G_{AM}}$$

$$\textbf{Utilities} : \mathbf{U_{AC}} = \mathbf{U_{CC}} \cap \mathbf{U_{AM}}$$

An example of the use of this pattern can be seen in handling of adaptation for battery state of charge in a vehicle. As a vehicle component is not able to self-adapt on its own and because of the additional requirements imposed by battery level, there is a need for an external, separate AM. Here, an instance of the pattern includes the vehicle CC and an AM for battery state of charge. A utility enforced by this pattern is, for instance, the battery level should never reach low. Therefore, it has the capability of perceiving the SOTA space and applying actions to modify its position in the space in the medium term. The vehicle CC has its own goal to reach the destination, so the pattern has goals as well.

In another example, a parking lot CC (or a charging station CC) can handle its availability through an explicit, external AM. It has its own goals and utilities, such as ensuring that it is assigned to a vehicle, and maximizing its usage. Through this, it has the capability of applying actions to modify the SOTA space.

6.1.2 ***Parallel AMs component pattern:***

This pattern is an extension of the autonomic component pattern [45]. As the component is unable to self-adapt on its own, and for this, there is a need for several external AMs that work in parallel to provide adaptation (see Fig. 4-ii). The different AMs provide different expertise and handle different aspects of adaptation of the component.

This pattern is useful in two main situations [45]. They are when: (i) there is a component which relies on different types of information for adaptation; (ii) the AMs can divide the space of the adaptation problem into different parts at the same level. Each AM manages a separate part that contains different type of context information.

This pattern is designed around several decentralized explicit feedback loops. The CC is managed by different AMs that work in parallel to provide adaptation. So there is a spatial division of the SOTA space. The SOTA description of the goals and utilities of this pattern can be provided as follows:

$$\textbf{Goals} : \mathbf{G} = \mathbf{G_{CC}} \cap \mathbf{G_{AM_1}} \cap \mathbf{G_{AM_2}} \cap ... \mathbf{G_{AM_n}} = \mathbf{0}$$

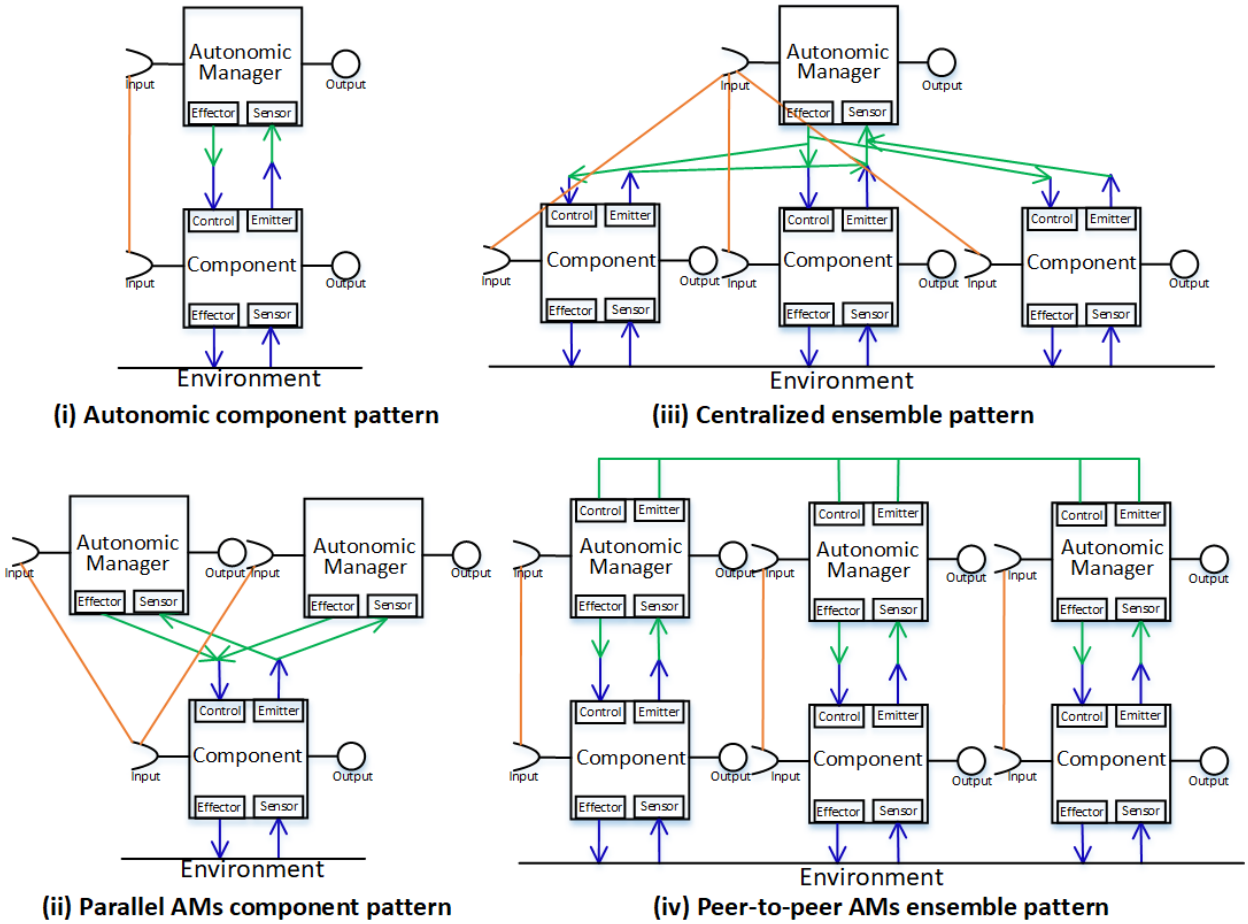

Fig. 4 SOTA self-adaptation patterns at the single component and ensemble level.

$$\textbf{Utilities} : \mathbf{U} = \mathbf{U_{CC}} \cap \mathbf{U_{AM_1}} \cap \mathbf{U_{AM_2}} \cap ... \mathbf{U_{AM_n}} = \mathbf{0}$$

Here, it shows that the goals or utilities provided by the CC and its different AMs do not overlap in the SOTA space.

An example of this pattern's use can be found in the different parallel AMs that can be defined for a vehicle component. Here, the vehicle CC can have different external AMs that provide different expertise (e.g., battery state of charge, climate comfort and driving style). As these different aspects of environment conditions determine the adaptation of the vehicle CC, they are considered parts of the same problem at the same level. The vehicle has the goal of reaching a destination within time, and to this end, each AM manages the vehicle by addressing different sub-problems. With these different AMs, there is a spatial division of the SOTA space.

As for another example, a vehicle CC can have several planned rides in a trip. One can provide an AM for each of these rides, so each AM handles different parts of the same problem of completing a trip.

6.2 Patterns for an Ensemble of Components

6.2.1 ***Centralized ensemble pattern:***

The aim of this pattern can be explained as follows. A component requires an external feedback loop to adapt, and all the components need to share knowledge and the same adaptation logic [45,3]. Therefore, they are managed by the same AM (see Fig. 4-iii).

This patterns has to be adopted when: (i) there several basic components, and a single AM is necessary to manage adaptation. For this, a centralized feedback loop is more suitable because a single AM has a global vision on the system; and (ii) indirect communication between components is necessary through a centralized AM.

This pattern is designed around a global feedback loop [3]. All the components are managed by a high-level AM or a super AM that controls the behavior of all the components, shares their knowledge, and propagates adaptation. The specification of SOTA goals and

utilities for this pattern is as follows:

$$\textbf{Goals} : \mathbf{G} = \mathbf{G_{CC_1}} \cup \mathbf{G_{CC_2}} \cup ... \cup \mathbf{G_{CC_{n1}}} \cup \mathbf{G_{AM}}$$

$$\textbf{Utilities} : \mathbf{U} = \mathbf{U_{CC_1}} \cup \mathbf{U_{CC_2}} \cup ... \cup \mathbf{U_{CC_{n1}}} \cup \mathbf{U_{AM}}$$

The goal of the ensemble (G) is provided by composing the goals of all the CCs and the AM. Similarly, utilities of the ensemble (U) are composed of the utilities of all the CCs and the AM.

As for an example, any route planning of the fleet (e.g., distributing vehicles every day, creating trips for vehicles), which has several vehicle CCs, needs to be provided by the same AM as all the vehicles need to share the same knowledge and adaptation logic. Therefore, a centralized feedback loop is necessary. The communication between the vehicles is indirect as it is through this high-level AM.

6.2.2 ***Peer-to-peer AMs ensemble pattern:***

This pattern has multiple components where each component has an explicit autonomic feedback loop for adaptation, which is provided by an AM [14]. The components communicate and coordinate with each other through their AMs (see Fig. 4-iv).

This patterns needs to be applied when (i) each basic component in the ensemble requires an external AM to manage adaptation; (ii) direct communication is needed by the components to communicate with each other through their AMs to share knowledge and propagate adaptation.

Each component is managed by an AM, and the AMs directly communicate one with each other using a peer-to-peer communication mechanism. The communication performed at the AMs' level makes it easier to share knowledge about the components and the adaptation logic. The SOTA description of the goals and utilities of this pattern:

$$\textbf{Goals} : \mathbf{G} = (\mathbf{G_{CC_1}} + \mathbf{G_{AM_1}}) \cup (\mathbf{G_{CC_2}} + \mathbf{G_{AM_2}}) \cup ... \cup (\mathbf{G_{CC_n}} + \mathbf{G_{AM_n}})$$

$$\textbf{Utilities} : \mathbf{U} = (\mathbf{U_{CC_1}} + \mathbf{U_{AM_1}}) \cup (\mathbf{U_{CC_2}} + \mathbf{U_{AM_2}}) \cup ... \cup (\mathbf{U_{CC_n}} + \mathbf{U_{AM_n}})$$

As seen above, the goal of the ensemble is composed of the goals of all the components in the ensemble. Each component is composed of a CC and an AM, thus the goal of the component is the aggregation of the goals in the CC and the AM. Similarly, utilities of the ensemble are composed of the utilities of each component of the ensemble.

An example of the peer-to-peer AMs ensemble pattern can be found for the parking lots in the case study. Assume there are a set of intelligent cameras and parking sensors to monitor the availability of the parking spaces in a parking lot. The goal of each of these cameras and parking sensors is to monitor departures and arrivals of vehicles to a parking space in a decentralized way. Each camera/sensor has a communication unit to interact with other cameras/sensors. These units can act as different AMs that can be defined for the parking lot CCs. The data observed by the collection of sensors needs to be aggregated as well.

7 Related Work

SOTA builds on the lessons of past research work, which focused on the engineering of adaptive, distributed software systems. In this section, we summarize the key works that are related to SOTA at the intersection of (i) models for self-adaptive systems and (ii) goal-oriented requirements engineering.

7.1 Modeling Approaches for Self-adaptive Systems

The IBM manifesto of autonomic computing [30] is a seminal work that identified self-adaptation as a direction to handle the software complexity and their being increasingly operating in dynamic environment, and the related modeling and engineering issues have been analyzed in a variety of later surveys (e.g., [33,37]) and roadmap papers (e.g., [58,15,19,67]).

SOTA starts from the consideration that adaptive and collective software systems are to most extent assimilable to that of a dynamical system, and that the "state of the affairs" dimensions can be a sort of phase space of the system. 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 [68], and several studies exist in that direction (e.g., [61,44]). SOTA commits to the above perspective, but adopts an entirely 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 behavior of existing systems.

In the area of autonomous agents and multi-agent systems, the issue of modeling the capability of agents capable of adaptively pursuing a goal has been widely investigated in the past. Such an issue can be approached either by adopting specific agent-based models [49,26, 69]), or by focusing on the specific agent-oriented software engineering methodology [18], or by developing appropriate agent-oriented programming languages to

account for such adaptivity [39,56,10]. However, to the best of our knowledge, a general-purpose approach to model and engineer adaptive behaviors, as we propose in SOTA, is still missing.

Some recent approaches to the engineering of self-adaptive systems have taken an architectural approach, for instance, by focusing at defining suitable architectures for supporting self-adaptation (e.g., [23,41,22,1]), or at shaping the control loops that support autonomic self-adaptive behaviors (e.g., [32,58,57,59]). For example, Glazier and Garlan [23] present an approach for meta-management of a collection of autonomous subsystems which respects local autonomy. The behavior of each subsystem has been encapsulated and abstracted as a parameterized adaptation policy, which is then used by a meta-manager to tune the subsystem adaptation. In [32], the authors propose an approach that provides parameterization for MAPE-K feedback loops in order to make the design options more explicit. They present a catalog of goal-oriented optimization patterns that enable the goal-oriented adaptation of the parameters at design time and runtime. However, these works have mostly neglected the identification of suitable general model to frame such adaptive architectural patterns on the basis of requirements, as SOTA attempts to do.

Formal approaches to model collective self-adaptive systems and associated architecture have been also explored. For instance, DEECo (Dependable Emergent Ensemble of Components) is a component system tailored for building autonomic systems [31,40] that proposes the invariant refinement method (IRM) [13] for capturing high-level goals in terms of invariants, and for identifying system components and their desired interaction using systematic refinement. Similarly to SOTA, the IRM method of DEECo can help guiding the transition from early, high-level goals, to system architecture in terms of components and ensembles. However, it lacks the simple operational nature of SOTA and does not help in identifying awareness requirements.

7.2 Goal-Oriented Requirements Engineering

In the area of requirements engineering for self-adaptive systems, goal-oriented approaches have been extensively investigated in the recent past (e.g., [42,53,17,25,6]), typically extending over existing goal-oriented approaches [64].

Early works on self-adaptive systems, such as the LoREM (Levels of Requirement Engineering for Modeling) method [25] proposes modeling the requirements of a dynamically adaptive system using the i* goal-modeling technique [64], i.e., in terms of actors, their goals, and the tasks associated to goals. However, this is not enough to capture the uncertainties and dynamic inherent to self-adaptive systems. Cheng et al. [16] describe a four stage process of goal-based modeling of environmental uncertainties in requirements by integrating the KAOS and the RELAX goal-oriented languages. The SOTA model differs to both in that modeling requirements in terms of trajectory in the SOTA space can be more expressive than simple expressing actors and goals, and it also provides a more rigorous assessment of the self-adaptation requirements via the model checking technique. Nevertheless, we think that using SOTA in conjunction with more assessed goal-oriented modeling techniques can be a viable solutions.

Many proposal for goal-oriented self- adaptation requirements have also have been provided in the context of the goal-oriented Tropos methodology. Tropos4AS (Tropos for Adaptive Systems) [42] is a framework extending Tropos for engineering agent-based self-adaptive systems founded on the Belief-Desire-Intention model. Tropos4AS aims to capture and describe at design time specific characteristics to enable self-adaptation at runtime. Qureshi et al. [47] propose continuous adaptive requirements engineering (CARE) method, which is a requirements engineering framework for self-adaptive systems that supports continuous refinement of adaptive requirements at runtime. Meanwhile, Dalpiaz et al. [17] propose an architecture that is based on requirements models, and adds self-reconfiguring capabilities to a system using a monitor-diagnose-compensate loop based on the system's requirements models expressed using Tropos. Brand and Giese [12,11] present a generic adaptive monitoring approach that is based on executed architecture runtime model queries and events. These authors' work tries to improve on existing MAPE-K loop-based approaches, which are driven by goals and require a considerable development effort each time those MAPE-K schemes are used. A key difference from SOTA is that these approaches ([12], [42], [47], [17]) support self-adaptation in runtime models, whereas SOTA focuses on supporting self-adaptation in the requirements and architecture models. However, a general reference model to help tackle issues in the engineering of self-adaptive systems is missing in those works.

RELAX [60] is a requirements language for self-adaptive systems that explicitly addresses uncertainty inherent in adaptive systems. RELAX can be used to support modeling and reasoning about uncertainty, which can be environmental or behavioral, in design time and runtime models. In RELAX, uncertainty is specified using a set of SHALL statements, aiming at capturing uncertainty declaratively with modal, temporal and ordi-

nal operators. Requirement reflection [8] supports self-adaptation by making requirements available as runtime objects (runtime goal model), and qualitative and quantitative reasoning is provided about how the goal model's organization changes over time. To deal with uncertainty, goal-oriented requirements modeling has been extended with the RELAX language. A similar approach to RELAX, but instead is based on the KAOS goal-oriented language is FLAGS [6]. FLAGS has been used to specify requirements and adaptation capabilities of self-adaptive systems in which goals have been transformed into live entities. The authors in [60] and [6] provide explicit language support for self-adaptive systems to address uncertainty. In contrast, SOTA expresses variability of the adaptation requirements using the notions of goals and utilities, and by accounting for unexpected movements in the SOTA space.

Tamersoy et al. [54] propose a goal-oriented requirements model for adaptive multi-organizational systems which consists of multiple interacting organizations. A key feature in their work is, organizations have been modeled as first-class modeling artifacts and adaptivity has been modeled in organizational layers of goals, roles and organizations. However, unlike SOTA, their proposed model has not been formalized for the modeling and analysis of adaptive requirements.

The authors in [50] have proposed a theoretical framework and a general architecture for modeling and engineering self-adaptive smart spaces. The framework, which is aimed to be independent of any specific application context, specifies the problem of proactive means-end reasoning on states of the world, goals and capabilities. The benefits of the architecture are to be domain independent and to support reusability across many application contexts. Although their work also shares the benefits of being a domain-independent, general architecture for self-adaptation, it specifically targets self-adaptive smart spaces. The SOTA model provides more broader uses, as it can be used to assess requirements with model-checking, identify knowledge requirements, and support designers in selecting architectural patterns.

In [62], the authors present a framework for software engineers to understand the aspects affecting the requirements engineering process of developing autonomic systems (e.g., system environment, system capability, level of autonomy, goal achievement alternatives). The authors propose this framework by analyzing the state-of-the-art in requirements engineering of autonomic systems. However, unlike SOTA, their work is still at an early stage of maturity and a validation of the framework has not been provided using a case study.

In summary, looking at the existing approaches in goal-oriented self-adaptive systems, to the best of our knowledge, there is a lack of of general modeling frameworks to handle the many complex issues of self-adaptive systems requirements engineering and architectural design. SOTA aims to build on existing works on goal-oriented self-adaptive systems and tries to go further, integrating the key characteristics of goal-oriented requirements approaches with concerns related to context-awareness and contextual adaptation, and with elements of dynamical systems theory.

8 Conclusions and Future Work

This paper presented the SOTA approach for the engineering of collective adaptive 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. To exemplify the key suitability and flexibility of SOTA, fleets of self-driving vehicles has been adopted as an exemplary scenario of a collective adaptive system in which the individual components (i.e., the vehicles) are required to act in a coordinated way towards the adaptive and harmonized achievement of both individual and collective goals.

Beside the case study presented in this paper, we are currently exploiting SOTA in different collective adaptive systems scenarios, such as collaborative robotics [24] and smart internet of things collectives [36]. In addition, we are continuing the framing and cataloging of relevant self-adaptive patterns [46], accurately model them in SOTA terms, and define guidelines and support to assist designers in the adoption of specific patterns. In addition, since most of future collective adaptive systems will be indeed socio-technical systems involving humans as active components of the system, we think it will be important to evaluate how SOTA could be possibly extended to account for 'supra-functional requirements" taking into account the values of the people involved [27].

References

1. N. Abbas, J. Andersson, M. U. Iftikhar, and D. Weyns. Rigorous architectural reasoning for self-adaptive software systems. In *Qualitative Reasoning about Software Architectures (QRASA)*, pages 11–18, 2016.
2. D. B. Abeywickrama, N. Bicocchi, and F. Zambonelli. SOTA: Towards a general model for self-adaptive systems. In *Proceedings of the IEEE 21st International WETICE'12 Conference*, pages 48–53, June 2012.

3. D. B. Abeywickrama, N. Hoch, and F. Zambonelli. Engineering and implementing software architectural patterns based on feedback loops. *Scalable Computing: Practice and Experience*, 15(4), 2014.
4. D. B. Abeywickrama, M. Mamei, and F. Zambonelli. Engineering collectives of self-driving vehicles: The SOTA approach. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems. ISoLA 2018.*, pages 79–93, Cham, 2018. Springer International Publishing.
5. D. B. Abeywickrama and F. Zambonelli. 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*, pages 33–42, April 2012.
6. L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 125–134, Washington, DC, USA, 2010. IEEE Computer Society.
7. L. Belzner, M. Hölzl, N. Koch, and M. Wirsing. Collective autonomic systems: Towards engineering principles and their foundations. *Transactions on Foundations for Mastering Change*, 1:180–200, 2016.
8. N. Bencomo. Requirements for self-adaptation. In Ralf Lämmel, João Saraiva, and Joost Visser, editors, *Generative and Transformational Techniques in Software Engineering IV: International Summer School*, pages 271–296, Berlin, Heidelberg, 2013. Springer.
9. N. Bicocchi, M. Mamei, A. Sassi, and F. Zambonelli. On recommending opportunistic rides. *IEEE Trans. Intelligent Transportation Systems*, 18(12):3328–3338, 2017.
10. R.H. Bordini, L. Braubach, M. Dastani, A.E.F. Seghrouchni, J.J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Special Issue: Hot Topics in European Agent Research II Guest Editors: Andrea Omicini*, 30:33–44, 2006.
11. T. Brand and H. Giese. Towards generic adaptive monitoring. In *Proceedings of the 12th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 156–161, Trento, Italy, September 2018. IEEE.
12. T. Brand and H. Giese. Generic adaptive monitoring based on executed architecture runtime model queries and events, effective and effort-reducing. In *Proceedings of the 13th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Umea, Sweden, Jun 2019. IEEE.
13. T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil. Deeco: An ensemble-based component system. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-based Software Engineering*, CBSE '13, pages 81–90, New York, NY, USA, 2013. ACM.
14. G. Cabri, M. Puviani, and F. Zambonelli. Towards a taxonomy of adaptive agent-based collaboration patterns for autonomic service ensembles. In *2011 International Conference on Collaboration Technologies and Systems, CTS 2011, Philadelphia, Pennsylvania, USA, May 23-27, 2011*, pages 508–515, 2011.
15. B. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In B.H.C. Cheng, R. de Lemos, P. Inverardi, and J. Magee, editors, *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009.
16. B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. *A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty*, pages 468–483. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
17. F. Dalpiaz, P. Giorgini, and J. Mylopoulos. An architecture for requirements-driven self-reconfiguration. In *Proceedings of the 21st International Conference on Advanced Information Systems Engineering*, CAiSE '09, pages 246–260, Berlin, Heidelberg, 2009. Springer-Verlag.
18. H. K. Dam and M. Winikoff. Towards a next-generation AOSE methodology. *Sci. Comput. Program.*, 78(6):684–694, 2013.
19. R. de Lemos, D. Garlan, C. Ghezzi, and H. Giese. Software engineering for self-adaptive systems: Assurances (Dagstuhl seminar 13511). *Dagstuhl Reports*, 3(12):67–96, 2014.
20. R. De Nicola, M. Loreti, R. Pugliese, and F. Tiezzi. A formal approach to autonomic systems programming: The SCEL language. *TAAS*, 9(2):7:1–7:29, 2014.
21. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, Reading (MA), 1995.
22. H. Giese, T. Vogel, A. Diaconescu, S. Gtz, and S. Kounev. *Self-Aware Computing Systems*, chapter Architectural concepts for self-aware computing systems, pages 109–147. Springer, 2017.
23. T. J. Glazier and D. Garlan. An automated approach to management of a collection of autonomic systems. In *Proceedings of the 4th eCAS Workshop on Engineering Collective Adaptive Systems*, Umea, Sweden, 16 June 2019.
24. K. Goldberg. Robots and the return to collaborative intelligence. *Nature Machine Intelligence*, 1(1):2, 2019.
25. H. J. Goldsby, P. Sawyer, N. Bencomo, B. H. C. Cheng, and D. Hughes. Goal-based modeling of dynamically adaptive system requirements. In *15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*, pages 36–45, March 2008.
26. Z. Graja, F. Migeon, C. Maurel, M-P Gleizes, L. Laibinis, A. Regayeg, and A. Hadj Kacem. A Pattern based modelling for self-organizing multi-agent systems with Event-B. In *International Conference on Agents and Artificial Intelligence (ICAART 2014)*, pages pp. 229–236, Angers, France, March 2014.
27. M. Harbers, C. Detweiler, and M. A. Neerincx. Embedding stakeholder values in the requirements engineering process. In Samuel A. Fricker and Kurt Schneider, editors, *Requirements Engineering: Foundation for Software Quality*, pages 318–332, Cham, 2015. Springer International Publishing.
28. N. Hoch, H-P Bensler, D. B. Abeywickrama, T. Bureš, and U. Montanari. *The e-mobility case study*, pages 513–533. Springer International Publishing, 2015.
29. S. Jähnichen, R. De Nicola, and M. Wirsing. The meaning of adaptation: Mastering the unforeseen? In *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part III*, pages 109–117, 2018.
30. J. O. Kephart and D. M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, January 2003.
31. M. Kit, I. Gerostathopoulos, T. Bures, P. Hnetynka, and F. Plasil. An architecture framework for experimentations with self-adaptive cyber-physical systems. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 93–96, May 2015.

32. V. Kles, T. Goethel, and S. Glesner. Parameterisation and optimisation patterns for mape-k feedback loops. In *Proceedings of the 2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 13–18. IEEE, Sep. 2017.
33. C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17(Part B):184 – 206, 2015.
34. A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu. Towards requirements-driven autonomic systems design. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, 2005.
35. E. Letier, J. Kramer, J. Magee, and S. Uchitel. Deriving event-based transition systems from goal-oriented requirements models. *Automated Software Engineering*, 15(2):175–206, June 2008.
36. M. Lippi, M. Mamei, S. Mariani, and F. Zambonelli. An argumentation-based perspective over the social iot. *IEEE Internet of Things Journal*, 5(4):2537–2547, 2018.
37. F. D. Macias-Escriva, R. Haber, R. del Toro, and V. Hernandez. Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267 – 7279, 2013.
38. J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, second edition, April 2006.
39. X. Mao, Q. Wang, and S. Yang. A survey of agent-oriented programming from software engineering perspective. *Web Intelligence*, 15(2):143–163, 2017.
40. V. Matena, T. Bures, I. Gerostathopoulos, and P. Hnetynka. Experimenting with adaptation in smart cyber-physical systems: A model problem and testbed. In *Engineering Adaptive Software Systems - Communications of NII Shonan Meetings*, pages 149–169. 2019.
41. F. A. Moghaddam, R. Deckers, G. Procaccianti, P. Grosso, and P. Lago. A domain model for self-adaptive software systems. In *Proceedings of the 11th European Conference on Software Architecture: Companion Proceedings*, ECSA '17, pages 16–22, New York, NY, USA, 2017. ACM.
42. M. Morandini, L. Penserini, and A. Perini. Modelling self-adaptivity: A goal-oriented approach. In *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 469–470, Oct 2008.
43. J. Mylopoulos, L. Chung, and E. S. K. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.
44. H. Van Dyke Parunak. A mathematical analysis of collective cognitive convergence. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '09, pages 473–480, 2009.
45. M. Puviani. Catalogue of architectural adaptation patterns. Technical Report TR 4.2, University of Modena and Reggio Emilia, 2012.
46. M. Puviani, G. Cabri, and F. Zambonelli. Patterns for self-adaptive systems: agent-based simulations. *EAI Endorsed Trans. Self-Adaptive Systems*, 1(1):e4, 2015.
47. N. A. Qureshi and A. Perini. Requirements engineering for adaptive service based applications. In *2010 18th IEEE International Requirements Engineering Conference*, pages 108–111, Sept 2010.
48. K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.
49. A. Ricci. Programming with event loops and control loops - from actors to agents. *Computer Languages, Systems & Structures*, 45:80–104, 2016.
50. L. Sabatucci and M. Cossentino. Self-adaptive smart spaces by proactive means–end reasoning. *Journal of Reliable Intelligent Environments*, 3(3):159–175, Sep 2017.
51. G. Salvaneschi, C. Ghezzi, and M. Pradella. Context-terlang: A language for distributed context-aware self-adaptive applications. *Science of Computer Programming*, 102(Supplement C):20 – 43, 2015.
52. D. Schuster, A. Rosi, M. Mamei, T. Springer, M. Endler, and F. Zambonelli. Pervasive social context: Taxonomy and survey. *ACM Transactions on Intelligent Systems and Technology*, 4(3):46:1–46:22, 2013.
53. V. E. S. Souza. *Requirements-based Software System Adaptation*. PhD thesis, DISI- University of Trento, 2012.
54. M. Tamersoy, E. E. Ekinci, R. C. Erdur, and O. Dikenelli. A requirements model for adaptive multi-organizational systems. In *Proceedings of the 2017 IEEE 11th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 41–50, Tucson, Arizona, USA, Sep. 2017. IEEE Computer Society.
55. E. Uhlemann. Connected-vehicles applications are emerging [connected vehicles]. *IEEE Vehicular Technology Magazine*, 11(1):25–96, 2016.
56. M. Viroli and F. Zambonelli. A biochemical approach to adaptive service ecosystems. *Inf. Sci.*, 180(10):1876–1892, 2010.
57. T. Vogel and H. Giese. Model-driven engineering of self-adaptive software with eurema. *ACM Trans. Auton. Adapt. Syst.*, 8(4):18:1–18:33, January 2014.
58. D. Weyns. Software engineering of self-adaptive systems: An organised tour and future challenges. In Richard Taylor, Kyo Chul Kang, and Sungdeok Cha, editors, *Handbook of Software Engineering*. Springer, 2017.
59. D. Weyns, S. Malek, and J. Andersson. Forms: Unifying reference model for formal specification of distributed self-adaptive systems. *ACM Transactions on Autonomous and Adaptive Systems*, 7(1), 2012.
60. J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J-M Bruel. Relax: A language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196, Jun 2010.
61. R. A. Williams. Lessons learned on development and application of agent-based models of complex dynamical systems. *Simulation Modelling Practice and Theory*, 2017.
62. M. A. Yahya, M. A. Yahya, and A. Dahanayake. Autonomic computing: A framework to identify autonomy requirements. *Procedia Computer Science*, 20(Supplement C):235 – 241, 2013.
63. Y. Bar Yam. *Dynamics of Complex Systems*. Perseus Books, 2002.
64. E. S-K Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, 1995. UMI Order No. GAXNN-02887 (Canadian dissertation).
65. F. Zambonelli. Toward sociotechnical urban superorganisms. *IEEE Computer*, 45(8):76–78, 2012.
66. F. Zambonelli, N. Bicocchi, G. Cabri, L. Leonardi, and M. Puviani. 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*, pages 108–113, Oct 2011.
67. F. Zambonelli and M. Mamei. Spatial computing: an emerging paradigm for autonomic computing and communication. In *Autonomic Communication. WAC 2004 - LNCS 3457*. Springer, 2005.

68. F. Zambonelli and H. Van Dyke Parunak. 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*, ESAW'02, pages 13–28, Berlin, Heidelberg, 2003. Springer-Verlag.
69. Y. Zhang, C. Qian, J. Lv, and Y. Liu. Agent and cyber-physical system based self-organizing and self-adaptive intelligent shopfloor. *IEEE Transactions on Industrial Informatics*, 13(2):737–747, April 2017.