Specification, Synthesis and Validation of Strategies for Collaborative Embedded Systems

Bernd-Holger Schlingloff [1,2]

[1] Humboldt-Universität zu Berlin
[2] Fraunhofer FOKUS, Berlin

Abstract. A collaborative embedded system is an autonomous component of a cyber-physical system which cooperates with other such systems in order to accomplish a common goal. In this paper, we report on approaches for the validation of such collaborative embedded systems. We describe specification methods for hierarchies of goals and targets. Using model checking of alternating signal temporal logic, we show how to construct strategies for the satisfaction of goals and targets. For runtime validation of safety properties, we give a robust monitoring procedure which can flag potential problems in advance. Our two examples are car platooning and automated guided vehicles in industrial production. In the car platooning example, autonomous vehicles collaborate to enable high-speed driving at short distances. The fleet of transport robots collaborates in loading and unloading of production machines.

1 Introduction

The validation of embedded systems is of increasing importance, since we increasingly rely on their correct functioning. It is estimated that every European citizen on average owns more than 100 such systems. In a single modern car, there are between 50 and 100 electronic control units (ECU), which provide various services and driver assistance functions. An ongoing trend is that these devices are increasingly interconnected. This holds both on the level of tightly coupled systems, such as the ECU within a vehicle, and on the level of system-of-systems, such as a group of vehicles. In order to be able to adapt to changing demands and environments, systems are given more and more autonomy in their decisions. For example, even a simple robot vacuum cleaner and lawn mower can navigate autonomously within their dedicated areas. More sophisticated industrial transport robots have complex path planning components, circumventing obstacles and road blocks. In the railway and automotive domains, autonomous driving is a major innovation factor.

Autonomous systems can increase their performance, if they join forces and form collaborative groups. For example, a group of assembly robots in an industrial production cell can collaborate and distribute the required tasks amongst themselves, to jointly minimize the assembly time. As another example, a group of autonomous trucks can form a platoon, travelling in close distance to one another in order to minimize fuel consumption. We call such a strategic alliance a *collaborative system group* (CSG), and each member of the group a *collaborative embedded system* (CES).

A defining criterion of a CSG is that all CES in the group work together to perform a common function. Thus, they share a common objective, to which each individual contributes. This is in contrast to competitive behaviour, where two or more agents have conflicting or even contradictory objectives. As an example with both cooperation and competition, consider robot soccer as practised in the annual RoboCup tournament: In each match, the two teams compete, whereas within each team the robots cooperate.

Even if several CES share a common objective, this does not exclude the possibility that each system also has individual objectives. In the platooning example, each vehicle in the platoon may have an individual destination. Conflicting objectives may result in an overall behaviour which is hard to predict. For example, we would not want a car in the middle of a platoon to follow its own route; however, at the end of a platoon such a behaviour can be acceptable.

Thus, there is a need for design methods which guarantee a reliable and safe operation of CES in a collaborating group. In this paper, we describe several approaches for informal and formal specification and validation of CSG objectives. Starting with two use cases, we describe a stepwise formalization approach for specifications: Starting from a statement of the system purposes, via the objectives of a collaborating group, to the strategies each collaborative system should follow. In order to formalize the requirements, we define a temporal specification logic as a blend of signal temporal logic and alternating-time temporal logic. Then, we use model checking and machine learning to synthesise collaboration strategies. Furthermore, we monitor properties to predict behaviour which might lead to problems.

The paper is structured as follows: First, we describe our two use cases (Section 2). After that (Section 3), we characterize objectives for these systems as goals and targets. Then, we define our specification logic ASTL* (Section 4). Subsequently, we show how this logic can be used for strategy synthesis (Section 5) and online monitoring (Section 6).

2 Use cases

Our first example is from the automotive domain. Highly automated cars can collaborate to form a platoon which travels together at high speed in short distance to one another. This reduces the air resistance for each individual car, and allows a better use of highway space. Figure 1 depicts the model cars we designed for this use case.

The model cars recognize their environment via camera, ultrasonic sensors and wheel encoders. They communicate via WLAN. There are two ECU on board: The vehicle ECU handles low-level driving functions such as adjusting the motor speed and steering direction, whereas the advanced driver assistance ECU is responsible for collaboration, image processing, and other higher-level functions. The purpose of the collaborative adaptive cruise control (CACC) soft-

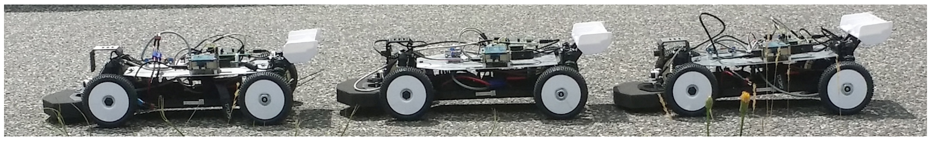

Fig. 1. Model cars for platoon driving

ware on this ECU is to control the velocity and trajectory of the vehicles such that they can safely drive in a platoon.

As our second example, we use a fleet of autonomous, collaborating transport robots in a factory environment. Figure 2 depicts some of the available robots [ProAnt].

Fig. 2. Transport robots for factory automation

Robots navigate in a production plant, using laser scanners to recognize their environment. They have a built-in floor map which allows self-localisation by comparing scan data and map data. In Fig. 3, a typical map from an actual factory is shown. Robots are only allowed to move in the area enclosed by the grey area. The current destination of the robot is indicated by a line.

The purpose of the robots is to load and unload production machines. Machines issue requests for being served, either if they are in need for new input material, or if the output buffer of produced goods is full. Robots receive the requests and decide amongst themselves which one should take the transport job. The main objective of the fleet is that every request is served in time. However, robots also have individual goals, for example never to run out of energy.

3 Natural language specifications

In a systematic design process, a first step consists of a systematic description of the functions a system shall provide. This is often done in specification docu-

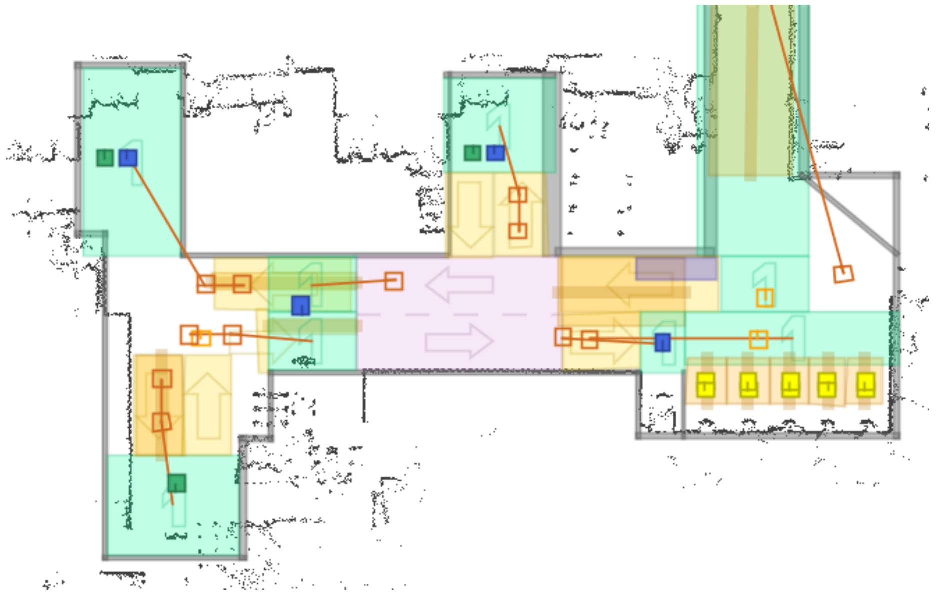

Fig. 3. Scanned floor plan with roads, robots, and docking points

ments in natural language. We describe a stepwise process which we applied for the case studies.

User stories

For specifying the different aspects of a CSG, we use controlled natural language. *User stories* [Coh04] describe the system from the viewpoint of human stakeholders. A user story describes a purpose which a user in a certain role can achieve by applying the system at a certain time, as well as the reasons for pursuing this purpose. User stories may then be exemplified via use case descriptions and scenarios, formalized in UML use case diagrams and sequence charts. We use a template sentence to formulate user stories.

> *As a* [actor/role] *I want the system to* do [function] *whenever* [trigger] occurs, *such that* [rationale] holds.

For example, the user story "Decentralized order management" describes how the robots determine which one of them accepts an order. It can be formulated as follows.

> *As a* transport system operator *I want the system to* decide autonomously which robot accepts a transport job *whenever* a job is issued by a machine, *such that* there is no need of a central control.

For the platooning use case, a typical top-level user story is the following.

> *As a* driver *I want the system to* drive automatically together with other vehicles in a platoon at close distance *whenever* a sufficiently large common route exists, *such that* fuel consumption decreases and a better traffic flow is maintained.

A subordinate user story is about leaving a platoon:

> *As a* driver *I want the system to* leave the platoon and hand over control to me *whenever* I request it, *such that* I can drive to a different destination than the platoon.

Objectives: goals and targets

From such user stories, objectives for the systems can be derived. An *objective* is a specific requirement describing a specific intention for the system. Objectives can be structured into a hierarchy, where lower levels support higher levels. Moreover, they can be ordered according to their importance, or level of contribution to the topmost objective.

For cyber-physical systems consisting of several independent agents, we have to distinguish between objectives for the collaborative system group (CSG) and for the individual collaborative embedded system (CES). The individual objectives should support the group objectives.

In the transport robot case study, the overall objective of the CSG is to provide transport services to machines. The most important high-level goal is to keep the maximal waiting time of each machine below a given threshold. That is, if the machine emits a request for a transport job, then it will be serviced by exactly one robot within this threshold. The rationale is that production machines often have a buffer for incoming and outgoing materials. If the input buffer is empty or the output buffer is full, the machine will stop its operation. This needs to be avoided. A related high-level target is to minimize the average waiting time of machines, in order to cope with varying production speed. Low-level objectives include

– *robustness and fault tolerance*, e.g., being able to deal with failures of (unloaded) robots, being able to circumvent temporary road blocks;
– *scalability and flexibility*, e.g., being able to dynamically integrate new robots into the fleet, and being able to adapt to changes in the factory topology;
– *efficiency and durability*, e.g., balancing the usage of robots for equal wear and tear; and
– *security*, e.g., ensuring that intruders and traitors cannot bring down the system.

These fleet-related objectives must be complemented with individual objectives for each CES. The topmost goal for each robot is to accomplish each transport job it has accepted, if this is within its capabilities. A related target is to accomplish the accepted transport jobs as fast as possible.

In order to support the topmost global goal “each request will be serviced”, corresponding objectives for the individual robots must be set. For example,

an individual target could be to service as many requests as possible. However, in isolation, this target might be too coarse, as it might lead robots to "self-destructive" behaviour such as neglected charging, extensive wear and tear, congestion of roads in the factory, etc. As an example, consider the case when a robot has a low battery level which would allow to finish one more transport job, but it would risk running out of energy on the subsequent way to a charging station. Here, we have a conflict between different individual objectives. Should the robot prioritize the target of servicing as many requests as possible over the goal of never running out of battery? This shows that the targets must be refined with an appropriate strategy which takes all objectives into respect.
Further objectives include

- keeping within designated floor areas,
- being able to cope with obstacles and road blocks,
- keeping battery level at 40–70%,
- minimizing the occupation time of docking and charging points,
- minimizing the number and length of empty trips, and
- avoiding rests outside designated parking areas.

Our industrial partner InSystems Automation GmbH, now ASTI Mobile Robotics, formulated a number of objectives which refines and extends this list [ZDS+17]. Analysing these objectives, we see that there are two types: Some are "sharp", for an actual implementation it is clear whether it has been reached or not. For example, requirement "Load factor" enforces that production machines are always adequately provided materials. If it is violated, machines will simply stop working.

However, most objectives are "soft" or "fuzzy", in the sense that they can be reached more or less. An example is the requirement "Minimising non-value-creating processes", which implies that the number of robots should be as low as possible. Thus, a solution with 10 robots is better than one with 20. However, a solution where 12 robots are employed may also be acceptable, it offers more options in unexpected circumstances.

We call an objective with a clear criterion whether it has been reached or not a *goal*. In analogy to an archery target disk with concentric circles, which can be hit more or less in the centre, we call soft requirements *targets* for the system. In other words, a target is an objective which can be partially met, to a higher or lower degree.

This categorization of objectives into *goals* and *targets* is essential for the design. A goal is described as a certain state of affairs which an agent strives to reach or maintain, whereas a target is a rough set of states which can be approximated more or less. An agent can be close to a target, but being close to a goal is the same as missing it. Therefore, goals usually have a long-term character, whereas targets are frequently re-evaluated.

Scenarios

Goals and targets were operationalized via so-called *scenarios*. These are procedural descriptions of sequences of actions, which illustrate one particular sequence of events within the operation of the system. There is a huge body of literature on different ways to denote scenarios, see, e.g., [?]. In our framework, each step is described by a consecutive number, the name of the agent, the action performed, the potential trigger for the action, and the rationale for the step.

An example is the distributed order management, which describes how autonomous cooperating robots are able to determine which one of them fulfils an order for transportation. If a new order is given and several robots are able to take it, it must somehow be decided which one of these robots actually will carry out the task. This is accomplished via a "bidding" or "consensus" process in which each available robot calculates its factors playing into this task, e.g. how far it is currently away from the pick-up area or how high or low its battery charging status currently is. It then sends these combined factors as information to the group as a bid. Depending on which robots can offer the most practical circumstances, it is decided which robot takes the job. The respective scenario is given in Table 1; for more information, see [Sch20].

	Who?	**What?**	**When?**	**Why?**
1	Machine	Broadcasts transportation need to robots	Every time a machine has support or dispose need (may be in advance and/or may be with priority)	The production process of the machine is not allowed to stop
2	Every Robot	Calculates a bid for this transport (may be based on individual cost and/or other criteria)	When a new transport need is notified	To get the information which robot fits the best for this transport
3	Every Robot	Determine winner by distributed leader election algorithm. If two robots bid the exact same amount, the winner is selected randomly	After bidding	
4	Robot	Bid winner adds the transport to its own transport queue	When won a bid	That the transport need is satisfied

Table 1. Scenario for decentralized order management

The first four columns are necessary for subsequent formalization steps, whereas the "why" column is for documentation purposes only.

4 Formal Specification

For an automated validation of system requirements, it is mandatory that these are denoted in a suitable formal specification language.

Temporal logics: LTL, MTL, WMTL, STL

For the formal specification of goals, temporal logics can be used. In contrast to more graphical specification languages, temporal logics are closer to textual representations. A classical example is the property "for every request there is a subsequent response". This is written in linear temporal logic (LTL, [GPSS80]) as follows[3].

$$\mathbf{G}(\mathit{request} \to \mathbf{F}\mathit{response})$$

In our setting, properties refer to real-time values. Therefore, timed temporal logics are necessary. The property that every request for service by a machine is fulfilled within 60 time units by some robot can be written in metric temporal logic (MTL, [Koy90]) as follows[4].

$$\mathbf{G}(\mathit{request}(m_i) \to \mathbf{F}_{(0,60)} \exists r_k \; \mathit{at}(m_i, r_k))$$

Other goals need spacial, epistemic, or strategic operators for formalization. It is much harder to express quantitative targets in classical or modal logics. If the bounds are made explicit (as in the example formula above), we can use these bounds in formulas. For example, we can specify performance in Weighted Metric Temporal Logic (WMTL, [BDL+12]). This logic contains an operator $\mathcal{P}$ which returns the probability of a statement within a certain time period. As an example, let the response time be the time difference between the time when a job is created and the time when the job is finished. The property "The response time within the first 1000 time units shall be less than 450 time units in 80% of all requests" can be written in WMTL as follows.

$$(\mathcal{P}_{(0,1000)}(\mathbf{G}(\mathit{Job.active} \to (\mathit{Job.clock} \leq 450))) \geq 0.8)$$

However, these specification logics do not allow to adequately translate the natural-language formulation of the targets. The numerical borders (1000 time units, 450 time units, 80%) are introduced artificially for the purpose of specification, they do not appear in the original target.

In order to formalize also targets, we need a logic which allows to reason about vagueness and strategies. Thus, we define a suitable variant of robust signal temporal logic (STL) for this purpose [DM10]. Signal temporal logic was invented to monitor the value of continuous signals in time. For example, if x is the distance of a robot to some no-go-area, then $\mathbf{G}(x \geq 10)$ means that the

[3] As a remark, this formula does not require that for each request there is a *corresponding* subsequent response, which cannot be expressed in LTL.

[4] Here, the existential quantifier is a finite disjunction ranging over the finite set r_k of robots. Implicitly, the formula is a conjunction of all formulas for machine m_i.

robot always keeps at least 10 units distance to this area. As another example, if v is the desired and y is the actual speed of the car, then $\mathbf{F}_{[0,3]}(y = v)$ requires that within 5 time units the desired speed is reached. The robustness value of a formula indicates the quality with which a formula is satisfied. A positive value means that the formula is true, with the indicated robustness. For example, the formula $(x \geq 10)$ is both true if $x = 10$ and if $x = 1000$, but in the latter case with higher robustness.

Alternating signal temporal logic ASTL*: syntax

We now define a new logic called alternating signal temporal logic (ASTL*) which is tailored for the specification of collaborative embedded systems. ASTL* is a canonical extension of both signal temporal logic (STL, [Don13]) and alternating-time temporal logic (ATL*, see [AHK02]). It is inspired by the synthesis approach for STL in [RDS+15]. In this section, we are using only the STL-part of ASTL*; strategic reasoning will be used in Section 5.

Let Σ_0 be an alphabet of *primary signals*, some of which can be *controlled* and some *observed*. For example, a variable v to adjust the speed of a motor is a controlled signal, whereas a sensor d signalling the distance to the next obstacle is an observed signal. The set of controlled variables is called Σ_c. Furthermore, let $\mathcal{F} = \{+, -, *, \sqrt{}, ...\}$ be a set of *primitive functions* on signals. We assume that $\mathcal{F}$ also contains constant functions, e.g., $0, 1, 3.14$, etc. A *derived signal* is a term built from primary signals with primitive functions. For example, if d_x and d_y are primary signals indicating the distance of an object to an origin in cartesian coordinates, then $d = \sqrt{d_x * d_x + d_y * d_y}$ is a derived signal indicating its absolute distance. The set Σ of signals consists of all primary and derived signals. An *atomic proposition* is an inequality $s \geq 0$, where s is a signal; the set of atomic propositions is denoted by $\mathbf{P}$. The syntax of ASTL is defined as follows.

$$\varphi ::= \mathbf{P} \mid |\bot \mid (\varphi \rightarrow \varphi) \mid (\varphi \ \mathbf{U}_I \ \varphi) \mid \langle\langle \Sigma_c \rangle\rangle \ \varphi$$

Here, I is a closed or open interval of $\mathbb{R}^+$, and in the formula $\langle\langle s \rangle\rangle \ \varphi$, $s \in \Sigma_c$ is a controlled variable[5]. As usual, $\neg\varphi = (\varphi \rightarrow \bot)$, $\top = \neg\bot$, $(\varphi \vee \psi) = (\neg\varphi \rightarrow \psi)$, etc. $\mathbf{F}_I \ \varphi$ is short for $(\top \mathbf{U}_I \, \varphi)$, $\mathbf{G}_I \, \varphi$ for $\neg \mathbf{F}_I \, \neg\varphi$. The unconstrained temporal operator $(\varphi \mathbf{U} \psi)$ stands for $(\varphi \mathbf{U}_{(0,\infty)} \ \psi)$, and similar for $\mathbf{F}\varphi$ and $\mathbf{G}\varphi$. Propositions $s < 0$, $s \leq c$, $s = 0$, etc., can be defined as $\neg s \geq 0$, $c - s \geq 0$, and $(s \geq 0 \wedge s \leq 0)$, respectively.

Alternating signal temporal logic ASTL*: semantics

In the semantics, each signal $s \in \Sigma_0$ is interpreted as a real-valued function over the time domain. That is, a model $\mathcal{M}$ consists of a set of functions $s^{\mathcal{M}} : \mathbb{R}^+ \rightarrow \mathbb{R}$. From the interpretation of primary signals, the interpretation of derived signals

[5] In similar strategy logics the modality is typically labelled by a set of agents which collaborate; currently, we see no need for this feature.

can be deduced. Satisfaction of a formula φ at time t in model $\mathcal{M}$ is defined as follows.

- $(\mathcal{M}, t) \models s \geq 0$ iff $s^{\mathcal{M}}(t) \geq 0$
- $(\mathcal{M}, t) \not\models \bot$, and $(\mathcal{M}, t) \models (\varphi \to \psi)$ iff $(\mathcal{M}, t) \models \varphi$ implies $(\mathcal{M}, t) \models \psi$
- $(\mathcal{M}, t) \models (\varphi \mathbf{U}_I \psi)$ iff for some $t_1 \in t + I$, $(\mathcal{M}, t_1) \models \psi$, and for all t_2 such that $t < t_2 < t_1$ it holds that $(\mathcal{M}, t_2) \models \varphi$.
- $(\mathcal{M}, t) \models \langle\!\langle s \rangle\!\rangle \varphi$ iff there is a function $s' : \mathbb{R}^+ \to \mathbb{R}$ such that $(\mathcal{M}', t) \models \varphi$, where $s^{\mathcal{M}'} = s'$ and $r^{\mathcal{M}'} = r^{\mathcal{M}}$ for all $r \neq s$.

From this definition, it follows that

- $(\mathcal{M}, t) \models \mathbf{F}_I \varphi$ iff for some $t_1 \in t + I$, $(\mathcal{M}, t_1) \models \varphi$.
- $(\mathcal{M}, t) \models \mathbf{G}_I \varphi$ iff for all $t_1 \in t + I$, $(\mathcal{M}, t_1) \models \varphi$.

In the definition of the specification logic, we are more concerned with the pragmatic, i.e., the ability to easily formulate objectives, than with questions about expressiveness and complexity.

We write $\mathcal{M} \models \varphi$ iff $(\mathcal{M}, 0) \models \varphi$. Thus, the above clauses assign a boolean truth value to a formula in a model. STL is famous for its *robust* semantics, where the "truth value" is a numerical value. Let $\overline{\mathbb{R}} = \mathbb{R} \cup \{\infty, -\infty\}$ and $(\overline{\mathbb{R}}, \leq)$ be its closure with the usual ordering relation. Further let $\sqcup : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ and $\sqcap : \overline{\mathbb{R}} \times \overline{\mathbb{R}} \to \overline{\mathbb{R}}$ be the maximum and minimum functions on the extended domain, i.e., $(x \sqcup \infty) = (\infty \sqcup y) = \infty$, $(x \sqcup -\infty) = (-\infty \sqcup x) = x$, $(x \sqcap -\infty) = (-\infty \sqcap y) = -\infty$, $(x \sqcap \infty) = (\infty \sqcap x) = x$ and for $x, y \notin \{\infty, -\infty\}$, we have $(x \sqcup y) = \max(x, y)$ and $(x \sqcap y) = \min(x, y)$. Furthermore, for any subset $X \subseteq \overline{\mathbb{R}}$, let $\bigsqcup$ and $\bigsqcap$ be the supremum and infimum functions over the set X, with $\bigsqcup \overline{\mathbb{R}} = \infty$ and $\bigsqcap \overline{\mathbb{R}} = -\infty$. The *score*, also called *spatial robustness*, of a formula with respect to a model is defined as follows.

- $\rho(\mathcal{M}, t, s \geq 0) = s^{\mathcal{M}}$
- $\rho(\mathcal{M}, t, \bot) = -\infty$
- $\rho(\mathcal{M}, t, (\varphi \to \psi)) = (-\rho(\mathcal{M}, t, \varphi) \sqcup \rho(\mathcal{M}, t, \psi))$
- $\rho(\mathcal{M}, t, (\varphi \mathbf{U}_I \psi)) = \bigsqcup_{t_1 \in t+I} \{\rho(\mathcal{M}, t_1, \psi), \bigsqcap_{t_2 \in t+I, t_2 < t_1} \rho(\mathcal{M}, t_2, \varphi)\}$
- $\rho(\mathcal{M}, t, \langle\!\langle s \rangle\!\rangle \varphi) = \bigsqcup \{\rho(\mathcal{M}', t, \varphi) \mid r^{\mathcal{M}'} = r^{\mathcal{M}} \text{ for all } r \neq s\}$

This gives

- $\rho(\mathcal{M}, t, \neg\varphi) = -\rho(\mathcal{M}, t, \varphi)$
- $\rho(\mathcal{M}, t, (\varphi \vee \psi)) = (\rho(\mathcal{M}, t, \varphi) \sqcup \rho(\mathcal{M}, t, \psi))$
- $\rho(\mathcal{M}, t, (\varphi \wedge \psi)) = (\rho(\mathcal{M}, t, \varphi) \sqcap \rho(\mathcal{M}, t, \psi))$
- $\rho(\mathcal{M}, t, \mathbf{F}_I \varphi) = \bigsqcup \{\rho(\mathcal{M}, t_1, \varphi) \mid t_1 \in t + I\}$
- $\rho(\mathcal{M}, t, \mathbf{G}_I \varphi) = \bigsqcap \{\rho(\mathcal{M}, t_1, \varphi \mid t_1 \in t + I)\}$

Robust and classical semantics are connected by the fact that $\rho(\mathcal{M}, t, \varphi) \geq 0$ iff $(\mathcal{M}, t) \models \varphi$. The score of a specification formula enables us to reason about goals and targets of a system.

Alternating signal temporal logic ASTL*: examples

Using ASTL*, we can formalize some of the goals and targets for the transport robot use case given in the previous section. For example, if *Batt.lvl* is an observable signal indicating the current battery level, then

$$\mathbf{G}(\mathit{Batt.lvl} \geq 0.4 \wedge \mathit{Batt.lvl} \leq 0.7)$$

formalizes the requirement that the battery level should be always between 40 and 70%. In each model $\mathcal{M}$, the score of this statement at time t_0 is

$$\prod\{(\mathit{Batt.lvl}^{\mathcal{M}}(t+t_0) - 0.4), (0.7 - \mathit{Batt.lvl}^{\mathcal{M}}(t+t_0)) \mid t \in \mathbb{R}^+\}.$$

If this score falls below zero, then the requirement is violated and the design of the robots must be changed.

Similarly, the score of the target

$$\mathbf{G}(\mathit{Job.active} \rightarrow \mathit{Job.clock} \leq 450)$$

which corresponds to the requirement "response time should be less than 450 time units" indicates by a numerical value whether the deadline has been kept. Here, *Job.active* is a boolean variable indicating whether a service has been requested, and *Job.clock* is a clock variable which starts to count whenever it becomes active. Such clock variables can serve to formulate also other requirements, notably "minimizing the occupation times of docking points", "minimizing the number and length of empty trips", and "avoiding rests outside of parking areas".

Alternating signal temporal logic ASTL*: application

A model for ASTL* consists of an interpretation of signals by real-valued functions. In order to generate such an interpretation which reflects the working of an actual system, we need to model it[6]. In general, a model of a collaborative system group consists of three parts:

- A model of the collaborating systems,
- a model of the static environment, and
- a model of the dynamic use of the system.

For the platooning use case, we need to model

- the CACC function,
- the road including lanes, traffic signs, etc., and
- traffic situations and platooning scenarios.

In the transport robot use case, components of the system model are

[6] Unfortunately, the word "model" is used in two different meanings: as a structure to evaluate logical formulas, and as an abstraction of a physical system. Both of these uses of the word are well-established, the meaning should be clear from context.

– the behaviour of each robot,
– the map, including no-go-areas, location of machines, charging points, etc., and
– the load, i.e., the transport jobs which are issued.

These components can be formulated in a suitable modelling language. We use deterministic hybrid automata, as realized in the Simulink® modelling language. The map is represented by a two-dimensional table. The load is a list of transport jobs, each with an ID, starting time, origin, and destination. The control algorithm of the robots is a diagram representing the behaviour and decisions. It is described in the next section.

Since the system model is a deterministic hybrid automaton, for every setup there is exactly one (infinite) run. This run defines a model for the ASTL* specification. It is represented as a discrete linear sequence of states and transitions. This sequence is directly obtained from a run of the Simulink model. For model checking such a sequence, a time-discrete semantics for ASTL* can be defined. In such a time-discrete semantics, the score of a formula is evaluated up to a certain state. Whenever the formula contains no strategic operators, the evaluation is linear in the length of the formula and the length of the run. Given a sufficiently fast computer, it can be done on-line, while the simulation is running. This approach is called *monitoring* and is elaborated in Section 6.

If the formula contains strategic operators, several alternative runs of the system need to be checked. Since there may be an infinite number of alternatives, monitoring is not sufficient. In the next section, we discuss the automatic construction of strategies for a given system model and ASTL* specification.

5 Strategy Synthesis

The control algorithm for the robots consists of a three-layered architecture. The bottom layer contains low-level control functions such as the evaluation of sensor data and driving of motors. We call this layer the *reactions* layer. In this layer, self-localization and mapping is handled: laser scan data and odometry daty are combined to calculate a most likely position for the robot in the factory. Commands to move to a specific target are translated into motor settings, and continually supervised during the movement of each robot. If an obstacle is sensed, the robot stops; if a deviation of the actual trajectory from the planned path is detected, it is corrected.

The mid-layer deals with regulations for the behaviour, such as the planning of optimal paths according to the map and current situation, maintaining a queue of assigned jobs, and navigating only in dedicated roads. This layer is called the *rules* layer.

The topmost layer deals with principles and priorities which govern the overall behaviour, such as goals and targets. In our terminology, we call this the *principles* layer. It determines the strategy according to which each robot bids for a job, or decides to drive to a power outlet.

This three-layered architecture reflects a generic scheme for reliable autonomous systems as elaborated in [FMR+20].

We wish to derive strategies for the principles layer.

For example, a strategy supporting the topmost goal of servicing each machine in time consists of an auctioning mechanism for issued transport requests, see Table 1. This strategy can be formulated in natural language as follows.

Each robot maintains a local queue of accepted transport jobs and estimated completion times. If a machine issues a new request, the robot calculates an estimated arrival time at this machine. The job mileage is the distance between the last position in this list and the location of the machine. The estimated arrival time is the estimated completion time of the last job in the task queue and the estimated travel time for the job mileage. Based on the estimated arrival time and the deadline of the job, the robot places the job mileage as a bid. Then, the robot waits and collects other bids. After all bids are placed, the robot selects and communicates the lowest bid. If more than one lowest bid arrived, one of the lowest bids is selected randomly. Otherwise, if the robot has placed the lowest bid itself, the job is appended to the task queue.

This strategy does not take into respect power consumption, battery charging, wear and tear, and other targets. Further strategies are, e.g., the following.

– Random: In the bidding, each robot bids a randomly chosen amount.
– First-come-first-served: Each robot bids the estimated completion time of its current task list.
– Shortest time: Each robot bids the estimated earliest arrival time at the machine.
– Highest energy: Each robots bid its current battery level (highest bid wins). A variant is to bid the estimated energy after completing the last job in the task list.
– Mileage: The bidding sum is a function of the job mileage, and the total mileage of the robot.

The bidding process is decisive for the performance of the whole fleet. In simulation runs, one can observe that the fleet behaviour changes significantly with the bidding strategy [Sit18]. The strategy of each robot culminates in the question which amount to bid if a new request is issued. In general, this is a function of

– current task list,
– current traffic situation,
– battery level, and
– individual history and future of the robot.

Our aim is to find a bidding strategy which satisfies the requirements for the fleet. This is where strategic quantifiers of ASTL* come into play. They allow to formulate requirements without resorting to a specific strategy.
Specifically, if *bid.i.j* is the amount robot i bids on task j, then

$$\langle\langle bid.i.j \rangle\rangle\, \varphi$$

denotes that there is a way for robot i to bid for a job which leads to the satisfaction of goal φ. We assume that *bid.i.j* is a controlled variable which can be set by robot i.

In lieu of a dedicated model checker for ASTL*, we used MCMAS [LQR17] on a scenario similar to the transport robot use case. MCMAS is a tool for model checking of strategic epistemic logic with multi-agent systems. It has been successfully applied to several academic examples.

Our scenario is similar to the well-known "Pacman" game, where agents are trying to pick up rewards on a two-dimensional grid. Strategic choices for an agent in each situation are whether to move north, south, east or west. From this, complex group strategies emerge. For example, it may not be optimal to always move towards the nearest reward, if another agent is also heading to pick it up. As expected, a state explosion occurs with the size of the board and the number of agents. Specifically, we were able to synthesize strategies for up to three robots on an 8×8 grid.

Since this is far from the size required for realistic industrial scenarios, we are using machine learning techniques to find an optimal strategy. The MAgent research platform for many-agent reinforcement learning [ZYC+17] allows to train systems with hundreds to millions of agents. We compared the quality of supervised strategy learning with reinforcement learning of strategies. In supervised learning, the MCMAS model checker guides the learning process. Since this is is computationally very expensive, the technique can be applied for off-line synthesis only. In reinforcement learning, each agent starts with a random strategy and learns from previous matches. Our experiments show some results which were surprising to us. Supervised learning scales better if only few learning iterations are allowed. However, after a few thousands of iterations, reinforcement learning results in better strategies. This is consistent with results obtained for learning games such as Go, Chess and Shogi by the AlphaZero program.

6 Online Monitoring

Online monitoring is a technique where the observed traces of a system are compared to a formal specification. Traces can be obtained from a prototypical implementation, or from a simulation as described above. As mentioned before, monitoring requires that there is a unique system run; hence is can be used to assert that a strategy automatically synthesized from a model behaves as expected also in reality. A challenge is to design a monitoring system which can detect and flag problems early, ideally even before they occur. That is, the monitor should raise an alarm even while the system is acting normally, if it has a tendency to drift into the exceptional behaviour.

To this end, we extended the semantics of ASTL*. Basically, we consider traces of finite length, which are processed one step after the other. A formula in a timed trace at a certain instant not only can be *true* or *false*, but the truth value additionally can be any real number. As long as the truth value of a formula in a model is not determined according to the standard semantics, we assign it a "likelihood" of being satisfied. This "likelihood" is calculated from the distance of the deadlines in the formula to the end of the trace, and the respective values for the sub-formulas.

We implemented an algorithm for monitoring our extended semantics [LS18]. For the evaluation of this algorithm, we collected traces from (a centralized version of) our transport robot case study. These traces covered a duration of several days up to a week of operation, and contained more than 10^6 timed events. By analysing the response time of the transport system to issued requests, we found a quite high variance of this value (between 1 and 100 min). Evaluating the MTL response property given above (every job will be fulfilled within 60 sec) revealed that property violation tended to "build up": several "near misses" were followed by a definite miss. This could not have been found by classical boolean-valued monitoring methods.

7 Conclusion and Further Work

We presented results in the specification and verification of collaborative embedded systems. As case studies, we described platooning of highly-automated vehicles, and an indoor logistics system consisting of autonomous transport robots in factories. For the specification of the systems, we used a stepwise refinement approach: From purposes via objectives to strategies. We specified the aims of the system with user stories in controlled natural language, augmented by scenarios as sequences of steps. From this, we derived formal objectives for the system. We categorized the objectives as goals or targets, depending on whether they are purely qualitative or also quantitative. Furthermore, we showed how to formalize the goals in various temporal logics. Then, we identified strategies supporting the defined goals and targets. We formalized the strategies in different state-transition-based modelling formalisms, and used simulation and model checking to analyse these models. Finally, we showed how to apply monitoring techniques to analyse long execution traces of the system for the accumulation of problems.

From our results, we can draw the following conclusions. Firstly, there is no "one size fits all" method for the formal analysis of such complex systems. We used different methods, e.g., timed automata, for different verification goals, e.g., correct timing. For the defined targets, we used quantitative analysis methods such as stochastic simulation and probabilistic model checking. However, these methods forced us to introduce artificial bounds. Therefore, we employed robust semantics which can give precise results also for fuzzy requirements and approximative targets.

Secondly, our analysis was facilitated by the fact that we were dealing with collaborative rather than competitive systems. In this paradigm, the individual strategy of each agent contributes to the aims of the whole system. The environment has no "strategy" to adverse the outcome. Therefore, by an individual optimization, the performance of the collaborative group increases. We believe that this observation is typical for a large number of similar systems. It remains further work to make this statement precise, i.e., to show that the complexity of collaborative strategy synthesis is lower than in the competitive case.

There are several other directions for further work. The complexity of our specification logic is unnecessarily high, since it employs full second-order quantification on functions. It remains to be investigated whether suitable sub-languages yield a lower complexity. Furthermore, we have not elaborated a dedicated model checking algorithm for our logic with a specific modelling framework. Hybrid automata and Simulink are very expressive formalisms which might be suitably restricted. Our hope is to find a "small" framework which is nevertheless sufficiently rich to model all important aspects of collaboration.

Furthermore, strategy learning in the context of CSG validation is a topic which needs further consideration. Although our experiments indicate that there is no "easy" way to combine model checking and machine learning for strategy synthesis, there are a number of directions we did not yet explore. Thus, the future remains exciting.

Acknowledgements: This paper is an extended and revised version of our preliminary report [Sch18]. The research was supported by the German BMBF project "CrESt" on collaborative embedded systems, FKZ 01|S16043G/E. The author would like to thank the industrial partner InSystems Automation GmbH / ASTI mobile robotics for providing the tranport robot case study, and the two anonymous referees for helpful comments and suggestions.

References

[AHK02] Rajeev Alur, Thomas A Henzinger, Orna Kupferman: Alternating-time temporal logic. JACM, Sept. 2002.

[AVM19] Houssam Abbas, Yash Vardhan Pant, Rahul Mangharam: Temporal Logic Robustness for General Signal Classes. In: Proc. ACM HSCC conference (HSCC'2019). ACM, New York, NY, USA, pp. 45–56.

[BDL+12] P. Bulychev, A. David, K. Larsen, A. Legay, G. Li, and D. Bøgsted Poulsen. Rewrite-based statistical model checking of WMTL. 3rd RV, Istanbul, 2012.

1. Cockburn-UsecasesAlistair Cockburn: Writing Effective Use Cases. Addison-Wesley, 2001.

[Coh04] M. Cohn: User Stories applied for agile software development. Addison-Wesley, Amsterdam 2004.

[DM10] Alexandre Donzé, Oded Maler: Robust Satisfaction of Temporal Logic over Real-Valued Signals, FORMATS 2010: Formal Modeling and Analysis of Timed Systems pp 92-106

[Don13] Alexandre Donzé: On Signal Temporal Logic. In: Int. Conf. Runtime Verification (RV 2013), Springer LNCS 8174, 2013

[FMR+20] Michael Fisher, Viviana Mascardi, Kristin Yvonne Rozier, Bernd-Holger Schlingloff and Michael Winikoff: Neil Yorke-Smith: Towards a Framework for Certification of Reliable Autonomous Systems. Preprint, arXiv:2001.09124v1, Jan 2020

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, Jonathan Stavi: On the temporal analysis of fairness. Proc 7th ACM POPL, pp. 163–173, 1980.

[Koy90] R. Koymans, Specifying real-time properties with metric temporal logic. Real-time systems, vol. 2, no. 4, 1990.

[LD18] Lars Lindemann, Dimos V. Dimarogonas: Robust Control for Signal Temporal Logic Specifications using Discrete Average Space Robustness. Preprint submitted to Automatica, Dec. 2018

[LQR17] Alessio Lomuscio, Hongyang Qu and Franco Raimondi: MCMAS, an open-source model checker for the verification of multi-agent systems. International Journal on Software Tools for Technology Transfer volume 19, pages9–30(2017)

[ProAnt] http://www.insystems.de/en/produkte/proant-transport-roboter/

[RDS+15] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia: Reactive Synthesis from Signal Temporal Logic Specifications. HSCC'15, April 14-16, 2015, Seattle, Washington

[Sch18] Bernd-Holger Schlingloff: Specification and Verification of Collaborative Transport Robots. Proc. EITEC 2018.

[Sch20] Bernd-Holger Schlingloff: CrESt Use Cases. In: Wolfgang Böhm, Manfred Broy, Cornel Klein, Klaus Pohl, Bernhard Rumpe and Sebastian Schröck (eds): Collaborative Embedded Systems. Springer, to appear 2020.

[Sit18] Franz Sitzmann: Simulation und Vergleich der Effektivität verschiedener Job-Scheduling-Verfahren für autonome Transportroboter. Bachelor's Thesis, Humboldt Universität zu Berlin, Institut für Informatik, 2018.

[LS18] F. Lorenz and H. Schlingloff: Online-Monitoring Autonomous Transport Robots with an R-valued Temporal Logic. CASE 2018.

[SSJ16] Bernd-Holger Schlingloff, Henry Stubert, and Wojciech Jamroga: Collaborative embedded systems – a case study. In: Proc. EITEC 2016 - 3rd Int. Workshop on Emerging Ideas and Trends in Engineering of Cyber-Physical Systems. CPS-Week, Wien, Apr. 2016.

[ZDS+17] Jan Stefan Zernickel, Susanne Dannat, André Schmiljun, Henry Stubert, Janina Samuel: CrESt UC.AP4.D1. Internal report, InSystems Automation GmbH, Berlin 2017.

[ZYC+17] Lianmin Zheng, Jiacheng Yang, Han Cai, Weinan Zhang, Jun Wang, and Yong Yu: MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. AAAI 2018, https://github.com/geek-ai/MAgent