Today’s Challenges and Potential Solutions for the Engineering of Collaborative Embedded Systems

Marian Daun, Andrea Salmon, Bastian Tenbergen, Thorsten Weyer

paluno – The Ruhr Institute for Software Technology
University of Duisburg-Essen, Germany
{daun, salmon, tenbergen, weyer}@paluno.uni-due.de

Abstract. Traditionally, embedded systems were characterized as reactive systems, whose functionality was determined by the execution of particular control circuits. Today’s embedded systems are equipped with powerful communication abilities to form collaborative networks of embedded systems dynamically during runtime in order to fulfill an overall purpose. Due to this changing nature of embedded systems, well-established engineering approaches are no longer sufficient to guarantee a successful development and operation. In this paper, we outline different challenges for engineering processes of collaborative embedded systems originated from the specific characteristics of such systems. Based on our findings, we discuss how particular techniques from the state of the art can potentially be used to address these upcoming challenges.

Keywords: Collaborative Systems, Embedded Systems, Cyber-Physical Systems, Challenges, Context, Functional Networks, Validation, Verification

1 Introduction

Collaborative embedded systems do not only rely on their own control circuits, but also possess the ability to collaborate with other systems to achieve an overall goal. By doing so, collaborative systems form dynamic networks, which create additional functionality through the interplay of the individual systems. This functionality is not designed into one specific system, but emerges at runtime within the network of collaborating embedded systems. In particular, modern communication technologies (e.g., the future internet, or the internet of things [1]) allow for communication between distributed systems, which are not part of the same super system. This means that collaboration does not only take place within one automobile, aircraft, industrial plant, etc., but between several of them. For instance, multiple cruise control systems in different cars can communicate with one another and spontaneously form a convoy to resolve traffic jams.

However, current engineering approaches cannot easily handle functional collaboration of embedded systems in networks dynamically formed at runtime. The dynamic nature of collaborative networks makes necessary for developers to foresee the functionality that emerges this way. While contemporary engineering approaches focus on
the development of individual systems or on the definition of inputs and outputs exchanged with the system’s static operational context, the collaboration of embedded systems within dynamic networks to achieve an overall goal is not explicitly considered. In consequence, developers cannot explicitly design functionality that emerges from collaborating system networks unless the entire functional network is developed top down. Furthermore, current approaches for assuring security, reliability, safety, and functional suitability are only concerned with individual systems’ functionality, not with the emergent functionality within collaborative system networks.

Recently, many research roadmaps have investigated challenges for cyber-physical systems (e.g., [2]) or self-adaptive systems (e.g., [3]). The goal of this paper is to focus on the challenges resulting from the collaborative nature of such system types and furthermore, to provide insight into a potential solution space. Therefore, existing approaches are reviewed, which might be adapted in future work to address the challenges. To this end, we first introduce a running example, which illustrates the dynamic nature of collaborative embedded systems in Section 2. Section 3 sketches specific challenges in the development of collaborative embedded systems. In Section 4, techniques from the state of the art are sketched, which can potentially be used to address these challenges in future research. Section 5 concludes the paper.

2 Running Example

In this section, we introduce a running example of an adaptive cruise control (ACC) system for vehicles which we will use to illustrate the challenges that arise from embedded system collaboration in the following sections.

In general, an ACC is an advanced cruise control system for road vehicles that not only keeps to the driver’s desired velocity automatically, but furthermore detects vehicles driving ahead and adjusts the own car’s velocity to maintain a safe distance to the preceding vehicle. In addition, the ACC in our case example provides the additional function “autonomous driving”, which enables the own car to drive autonomously in traffic jams. To do so, multiple ACC systems of surrounding vehicles interact to determine traffic jams and to adjust the velocity of all involved vehicles to a common ‘convoy’ velocity. This behavior results in a higher overall velocity of all involved vehicles in the traffic jam, helps to avoid rear-end-collisions, and decreases strain to the environment. Further information on such advanced ACC systems with autonomous driving behavior can, for example, be found in [4] and [5].

Fig. 1 depicts a situation in which this additional function is applied. The figure shows a motorway with two distinct directions. On the bottom direction, showing two lanes, a traffic jam has formed while on the upper direction traffic is flowing normally. Each depicted vehicle possesses an ACC system that provides the function “autonomous driving”. The driver’s desired vehicle velocity (v) is set by the driver herself. Due to the established traffic jam, a decreased current velocity (v’) is adapted either by the driver or by the ACC’s “follow to stop” or “stop and go” automatic (i.e. decelerating the car till stopped and accelerating again to follow a preceding car). The collaborative nature of the ACC in this example causes v’ to be permanently transmitted...
to the adjacent vehicles, so that every vehicle is aware of its neighbors’ current velocity. When $v'$ falls below a certain threshold (i.e. if a traffic jam is approaching and the driver is decelerating), the vehicles’ ACC systems negotiate an approximate average velocity ($v''$) that every vehicle maintains in order to build a steady traffic convoy.

![Fig. 1: Adaptive Cruise Control Systems Collaborating in a Traffic Jam](image)

For example, the dark car’s regular velocity has been set on 120 km/h by its driver. Due to the traffic jam, the driver was forced to decelerate to 10 km/h. Once its velocity had been fallen below the threshold of 20 km/h, the ACC has begun negotiating a convoy velocity with the ACC of adjacent vehicles. The ACCs collaboratively adjust all vehicles’ velocities to 13 km/h, i.e. the heuristically optimal convoy velocity in this situation. In the following section, we will illustrate several challenges for the engineering of such collaborative embedded systems by referring this example.

### 3 Today’s Challenges in the Engineering of Collaborative Embedded Systems

This section outlines different challenges that contemporary engineering approaches are faced with regarding the development of collaborative embedded systems.

#### 3.1 Multiple System Instances and Multiple Configurations

Model-based engineering is commonly seen as the most appropriate way to cope with system and interaction complexity [6] and is of great importance to ensure valid functioning of high-quality systems [7]. Yet, current model-based engineering approaches are theoretically applicable for a multitude of systems on different layers of abstraction, which are related to the same super system (e.g., the entire car). The underlying assumption is that one original equipment manufacturer (OEM) is in control of the
development process of the super system and can influence how suppliers develop the subsystems. However, in case of collaborative embedded systems, the super system relates to the collaborative system network at runtime, which involves many OEMs and suppliers.

Consider the example illustrated in Fig. 2. The vehicles can be equipped with ACCs of different suppliers (i.e. supplier A or supplier B). The ACCs can also be of different model ranges (i.e. basic model 12, comfort model 19, or sport model 37), as well as of different versions (i.e. version v1, version v2, or version v3). In consequence, the functional network is likely to be highly heterogeneous at runtime in that different configurations of the concrete ACC instances are possible. Subtle differences in ACC implementations (e.g., different thresholds for the current velocity \(v\), different communication protocols) might hence impair network functionality at runtime, which must be accounted for during development.

![Fig. 2: Many Instances of Different Types of Adaptive Cruise Control Systems](image)

**Characteristics of Collaborative Embedded Systems to be Addressed**

Therefore, the model-based engineering of collaborative embedded systems is faced with specific properties regarding the type and the amount of systems, which are involved in a collaborative system network. Approaches will have to deal with:

- **Multiple Collaborating Instances.** While embedded systems are typically developed (i.e. specified, validated, tested, etc.) on a type level, collaborative embedded systems create their main benefit through interaction between multiple instances of the system under development. In consequence, documentation and analysis techniques will have to apply to an instance level and techniques must take different numbers of participating instances into account, while maintaining safety, reliability, and security of all individual instances as well as the resulting network. For example, proper functionality of the ACC must be ensured regardless whether four, five, or more instances of the ACC are in short range during a traffic jam.

- **Multiple Collaborating Instances of different System Types.** In collaborative system networks, not only systems of the same type will participate but furthermore, there is a need to cope with other systems as well. For example, the ACC may be produced in different model ranges (e.g., basic, comfort, or sport). In consequence, during runtime the heterogeneity of the dynamically created function networks increases by an order of magnitude when many different system types are present, each of which comprising a number of instances.
• **Multiple Collaborating Instances and System Types of different Suppliers.** Beside different versions of the same ACC and different ACCs of the same supplier, during runtime, one ACC will have to collaborate with ACCs of different manufacturers as well. This is in so far a challenging task, as due to protecting their intellectual property, manufacturers are unlikely to share the entire specification of their ACC implementation. Hence, only the interfaces and the guaranteed behavior and quality at the interfaces are known and other collaborating systems must be treated as black boxes. While standardization and certification of thresholds, protocols, and functionality is a viable avenue in some application domains (e.g., avionics), emergent functionality must also be certified at runtime.

3.2 **Open System Contexts**

In the engineering of collaborative embedded systems, the context can no longer be completely anticipated at design time. Systems will collaborate with other systems, which were not conceived at the time some of the collaborative systems were developed, in a partially unknown technical and physical environment during runtime. This is in so far different from the case presented in Section 3.1, as this does not only hold for the specific models or number of instances, but also for the environmental properties, exchanged signals, and the functionality offered by other systems as well. Therefore, the fundamental paradigm of a closed world assumption (cf. [8]) no longer applies, and future systems will work in open contexts in the sense of [9].

One such example is illustrated in Fig. 3. In this example, the collaborative system network not only consists of multiple ACCs in various cars, but also consists of roadside traffic guidance systems, indicated by round dots alongside the guard rail.

![Fig. 3: Newer ACC Systems Introduce Additional Functionality](image)

These traffic guidance systems monitor the current velocity $v'$ of the passing cars, e.g., by listening for $v'$ from all individual cars unbeknownst to the collaborative network, or by determining the cars’ current velocities through own sensors. When detecting a traffic jam, the traffic guidance system issues a recommended velocity $v''$ at which cars are to travel in order to resolve the traffic jam quickly. The recommended velocity $v'''$ is transmitted to all passing cars. The traffic guidance system hence
offers similar functionality as the collaborative ACC network, but does so in a different way, which must be compatible with each individual ACC as well as the collaborative ACC network. Specifically, functionality of the ACC network must not be interrupted when the traffic guidance system intercepts \( v' \), the ACCs must be able to handle the additional input \( v'' \), the ACCs must be able to decide when to favor \( v''' \) over \( v'' \), and the ACCs must be able to determine when it is safe to ignore \( v''' \).

**Characteristics of Collaborative Embedded Systems to be Addressed**

The challenging task in engineering is to empower the ACC under development to react to such new interactions and functionality, which have not been explicitly considered at design time. Therefore, the ACC must not only be able to adapt its functionality at runtime but, more importantly, must also validate the proper functioning of such new functionality and its appropriateness. Therefore, engineering approaches must be able to deal with:

- *Unknown Functionality of the Collaborative Network.* Newly developed systems and advances in system development might bring additional functionality to the system network that was unknown at the time of development of older systems, which are also part of the network. These new systems may require information from the legacy systems or provide additional input to legacy systems during runtime. Contemporary engineering approaches must hence foster the development of embedded systems to account for unknown collaborative functionality because the system currently being developed will eventually become a legacy system and must operate properly with newer systems.

- *Unknown System Types.* Many embedded systems are designed to be in operation for a long period. During this time, systems routinely undergo maintenance, in which their contexts inevitably changes. For example, systems that are part of the collaborative network may be replaced with equivalents that are more modern; they may be upgraded to offer additional functionality, or updated to fix deprecated or suboptimal behavior. To account for such runtime variety during development is particularly daunting as many of the new system types may not even be conceivable when some particular model is being developed.

### 3.3 Validation and Verification of Collaborative System Networks

Collaborative embedded systems give rise to several issues concerning the quality assurance of the functionality offered by the system network. As collaborative networks form dynamically during runtime, it must be ensured that the emerging functionality is in fact desired by the user, does not conflict with the functionality of any of the involved systems, and is inherently safe, secure, and reliable. Furthermore, correctness of the emerging behavior must be established at runtime.

The traffic jam from the running example might contain a car, which is not equipped with a compatible ACC (or none at all), as shown in Fig. 4. Since it is desirable for the concept of the dynamic convoy to allow such situations (and not stop working altogether), this car must become part of the convoy. Nevertheless, it is of
vital importance that the safety of the convoy as well as correct functional interaction between the collaborating systems remains untarnished. In Fig. 4, the lighter car is not equipped with a compatible ACC and cannot collaborate in the system network. The darker car following the lighter car must be able to detect this situation and compensate in order to guarantee correct behavior of the collaborative network. For example, the darker car could compensate by communicating with the vehicles adjacent to the lighter car, thereby taking on the responsibility of expanding the convoy around the incompatible car while at the same time, observing its behavior to ensure the convoy does not crash into it.

![Fig. 4: Collaborating ACCs with non-collaborative Cars in the Convoy](image)

Consequently, the ACC of the darker car must alter its behavior and not strictly obey the velocity $v''$ negotiated in the convoy, but must be aware of sudden velocity changes of the lighter car. It is desired to reduce the own velocity for crash prevention if the distance decreases, but the convoy velocity shall not be considered in case the distance increases. The latter will allow for suitable functionality of the system network as otherwise a rear-end-collision with the follow-up car might be provoked. The darker car, however, is now burdened with reliably detecting unsafe behavior of the lighter car, selecting a strategy to compensate, and ensuring correct convoy behavior.

**Characteristics of Collaborative Embedded Systems to be Addressed**

Assuring functional correctness, safety, etc. poses considerable challenges for contemporary engineering approaches in two distinct ways: On the one hand, validation and verification of the emerging collaborative behavior must take place at runtime as collaborative networks form dynamically. On the other hand, the prerequisite for such quality assurance must be established during design-time. Approaches must achieve:

- **Validation of the Emergent Behavior of Collaborative Networks at Runtime.** The emergent behavior of the collaborative network must be adequate in the sense that regardless of the specific operational situation, the collaborative network has the right behavior. Functional adequacy must not only be established for each system that is part of the collaborative network, but for the entire network as well. This requires real-time validation of the emergent behavior at runtime.

- **Verification of Permissible Collaborations at Runtime.** The collaborative networks must be verified in the sense that their emergent behavior must be checked for functional correctness and interface compatibility. To achieve this, an effort must
be made to only allow permissible network configurations that result in interface compatibility and, more importantly, do not result in incorrect functionality of the individual systems. This includes documenting those types of collaborations that must be avoided just as much as those that are explicitly sought after.

- **Recertification of Collaborative Networks in Changed Contexts.** Embedded systems must satisfy strict safety, security, and reliability requirements, which are typically assured explicitly in the engineering processes. However, this certification must inevitably take place during runtime. Network configurations that result in unsafe collaborative behavior leave individual systems or groups of system networks insecure or reduce their reliability and must hence be avoided.

4 Potential Techniques for Addressing the Challenges

There are several techniques in the state of the art, which can potentially be used to define solutions in order to address some of the challenges outlined in Section 3.

4.1 Instance Models at Runtime

Currently, approaches exist that suggest model-based documentation on an instance level (e.g., [10]), or that propose enhancements of current modeling languages to distinguish between model elements of a certain type and instances thereof (e.g., [11], or [12]). These approaches enable integrated model-based specification on type level and instance level in one model or diagram. While the current state of the art provides different enhancements for different modeling languages, future work will have to deal with consistent enhancement mechanisms, which can be applied to all artifacts of the engineering process (e.g., a SysML enhancement that is applicable for all SysML diagram types), which may aid in dealing with uncertainty due to unknown numbers of instances, system types, and their configurations at runtime (see Section 3.1).

4.2 Equivalence Class Generation and Validation of Instance Models

Merely being able to document instance configurations (see Section 4.1) is insufficient to deal with the sheer amount of possible instances and configurations at runtime (see Section 3.1), since documenting all possible configurations is impossible. To this end, automated model transformation approaches (e.g., [13], [14]) could in the future be adapted to foster the creation of instance models from type models. The instance models could be created for different configurations and in conclusion may serve as analysis models. This seems in so far promising as the use of dedicated review models for validation purposes has been proven to be effective (cf. [15]). Furthermore, equivalence class generation could help identifying representatives for possible instance configurations [16] which could be used for model-checking (e.g., [17]) and simulation (e.g., [18], [19]).
4.3 Collaborative Requirements Monitoring

Since uncertainty regarding the configuration of the collaborative network and the emergent functionality of this network exist during design time (see Section 3.1), approaches must be adapted to monitor the interaction between the collaborative networks at runtime. In particular, techniques for monitoring and analysis of requirements satisfaction at runtime may be of use to cope with configurations, which have not been validated at design time (cf. [20]). As literature shows, model-based monitoring (e.g., [4], [21], or [22]) and model-based adaptation (e.g., [5], [23] or [24]) are promising techniques that seem appropriate to deal with this issue.

4.4 Probabilistic Context Modeling

Some approaches explicitly define dedicated context models, which may be used for collaborative embedded systems and cyber-physical systems like (e.g., [25], or [26]). These context models in essence document the assumed behavior of the operational context of a system once it is deployed (see Section 3.2), which could potentially be used for verification (see Section 3.3), specifically, to check coherence and correctness. Furthermore, in order to allow for proper functionality of multiple instances (see Section 3.1), these context models could be annotated with probabilities regarding the assumed behavior of instances, thereby allowing on-the-fly adaptation of collaborative networks through runtime evaluation of the assumption probabilities.

4.5 Human-in-the-Loop Engineering

When deviations from the assumed behavior occur (see Section 3.3), validation approaches must take effect to confirm that the behavior is acceptable. This could be achieved by transforming techniques from the area of human computer interaction and usability engineering (e.g., [27]) into the engineering of collaborative embedded systems. For example, human users could be made a part of collaborative system networks by relaying relevant information about the status of the convoy to the user and letting users influence the behavior of the system network. In either case, a prerequisite would be that cognitive user models exist, which allow systems to anticipate what information is required, based on operational scenarios (e.g., [28]). Collaboration at runtime becomes a problem of human-in-the-loop adaptation, which may be aided through runtime model-checking (see [29]) or review models (e.g., [15], [30]).

4.6 Context Awareness

In order to counteract the inherent uncertainty involved in dynamic adaptation of service-oriented systems at runtime, a variety of approaches have been suggested (e.g., [31], [32], or [33]), which allow self-adaptive systems to be aware of uncertainty or to be able to expect certain changes in inputs, outputs, and service configurations. In the field of collaborative embedded systems, these approaches may help to account for dynamically changing system instances, instance configurations, or net-
work behavior (see Sections 3.1 and 3.2), e.g., by engineering specific sensors and control logics into individual systems, which perceive changes in the collaborative network and alter the system’s behavior accordingly.

4.7 Self-Organizing System Networks

In the field of multi-agent systems, many individual systems are interconnected in order to achieve a common goal [34]. These can exist with and without controlling instances, which govern the behavior of the emerging system network [35]. In contrast to self-adaptive systems (see Section 4.6), multi-agent systems do not observe their context and react to changes. Rather, using machine learning techniques [36] they learn the optimal behavior given context stimuli. In the past, software engineering literature has been concerned how to enable systems to decide dynamically how to adapt to given context situations (e.g., [37], [38]). However, novel approaches will have to focus on enabling the systems to learn optimal adaptation strategies and automatically reorganize at runtime.

4.8 Monitoring of Trustworthiness Violations

As collaborative networks form dynamically between basically unknown parties, a certain level of trust in the proper functioning of the network (see Section 3.2) as well as all individual systems (see Section 3.1) must be established. To this end, approaches to analyze, foster, and certify trustworthiness between the involved parties exist (e.g., [39], [40]), which may be adapted to analyze the mutual functionality of collaborative embedded systems (see Section 3.3). A prerequisite to analyze behavior at runtime and to establish trustworthiness, however, would be that uncertainty is made explicit. As shown in Section 4.4, approaches are needed to foster the explicit documentation of runtime assumptions (e.g., [41], [42]) and of uncertainty (e.g. [43], [44]).

5 Conclusion

In this paper, we have presented emerging challenges for the engineering of collaborative embedded systems. The key challenges for engineering arise due to these collaborative systems’ ability to form networks dynamically at runtime. This forces developers not only to think about the functionality of the individual system, but also about the overall functionality of a collaborative network. We have shown how contemporary approaches from the state of the art might lay the foundation to meet some of these challenges that arise due to cyber-physicality in collaborative embedded systems. However, there is a vast amount of work to be done in this area until all hypothetical scenarios depicted in this paper are sufficiently dealt with.

Acknowledgments. This research was partly funded by the German Federal Ministry of Education and Research (grant no. 01IS12005C).
References