Towards Safety-Awareness and Dynamic Safety Management

Mario Trapp, Gereon Weiss
Fraunhofer ESK
Hansastrasse 32, 80686 Munich, Germany
mario.trapp | gereon.weiss@esk.fraunhofer.de

Daniel Schneider
Fraunhofer IESE
Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
daniel.schneider@iese.fraunhofer.de

***Abstract*—Future safety-critical systems will be highly automated or even autonomous and they will dynamically cooperate with other systems as part of a comprehensive ecosystem. This together with increasing utilization of artificial intelligence introduces uncertainties on different levels, which detriment the application of established safety engineering methods and standards. These uncertainties might be tackled by making systems safety-aware and enabling them to manage themselves accordingly. This paper introduces a corresponding conceptual dynamic safety management framework incorporating monitoring facilities and runtime safety-models to create safety-awareness. Based on this, planning and execution of safe system optimizations can be carried out by means of self-adaptation. We illustrate our approach by applying it for the dynamic safety assurance of a single car.** *(Abstract)*

Keywords-safety; system of systems; autonomous systems;

I. INTRODUCTION

Future safety-critical systems will be autonomous and seamlessly integrated into systems of systems and service ecosystems, which leads to several safety challenges. For example, artificial intelligence will be a core technology for autonomy leading to indeterministic, unpredictable behavior. Systems will dynamically connect to other systems, the surrounding infrastructure, and the cloud to form open systems of systems. We will hardly be able to predict in which situations they will operate, where their software will be executed, and which systems they will interact with in their lifetime.

Despite all these challenging prospects of future safety critical systems, a safety manager must complete the safety assurance at design time prior to the systems' deployment. However, compiling a complete safety assurance case prior to the systems' deployment will not be possible for future systems with too many uncertainties at design time.

Therefore, we will need to design resilient systems that dynamically manage and maintain their safety despite any kind of predicted, predictable or unpredictable situation [1] resulting from the immanent uncertainties [5] of future systems. To this end, systems need to become safety-aware and to adapt themselves dynamically to manage and maintain safety in a dynamic trade-off with other functional and non-functional properties [2][3].

This paper introduces first steps towards such a dynamic safety management framework. For this, we combine concepts taken from engineering self-adaptive systems [4][6][16], model-driven safety assurance [7][8], and safety-models at runtime [9][10][11][12][15]. We use monitoring mechanisms in combination with runtime safety-models to create safety-awareness (cf. Section II). Based on this awareness, the system dynamically assesses the currently required safety requirements as well as the system's current safety state (cf. Section III). Using self-adaptation concepts, the system dynamically self-adapts to optimize its performance while preserving its safety.

II. SAFETY-AWARENESS

The main challenge to safeguard future systems is to adequately deal with their uncertainties. In a conservative approach, we would assume worst cases, which would in many cases lead to safe but unreliable and, in particular, unavailable systems. Even worse, considering aspects such as the assurance of machine learning or open connectivity, the conservative safe answer would be not to use that technology at all.

Therefore, we need systems that resiliently react to such uncertainties resulting from untrustworthy or unpredictable elements of our system or its environment. Usually, we are only able to anticipate a small subset of uncertainties a-priori. Thus, we need to shift parts of the safety assurance process to runtime. To this end, it is necessary to create a *safety-awareness* in the systems enabling them to reason about their safety and to dynamically manage their safety – to a certain extent – autonomously at runtime.

Fig. 1 illustrates our principle idea of an architecture enabling safety-awareness as well as dynamic self-adaptation driven by such awareness. One core aspect to establish safety-awareness is to monitor the system and its environment using different monitoring mechanisms ranging from perception to (simple) error detection mechanisms. Cross-validation and aggregation could be applied to reach higher confidence and integrity. This monitoring is used as

input for a dynamic safety management component. In this component, we use model-based reflection founding on the monitored values to create safety-awareness and to derive appropriate system adaptations for optimizing the system's performance while guaranteeing its safety.

The adaptations as such can be very flexible adaptations, or – which is more likely for the short-term perspective – rather deterministic reconfigurations. This might include adapting the deployment of software to hardware nodes (e.g., if a hardware node fails), adapting the system's structure (e.g., if another algorithm, sensor etc. is more appropriate in a given context), and adapting parameters.

At system level, there will be different performance configurations for realizing graceful degradation. If the system cannot provide a performance configuration in a current situation, the system is set to a safety configuration. In detail, we require various different safety configurations to bring the system efficiently to a safe state, depending on the current situation. For example, the minimum risk maneuver of a car depends on its current driving situation and it is sufficient to find a safe configuration for enabling this specific maneuver.

III. Dynamic Safety Management

The basis for dynamic safety management is the capability of a system to analyze it´s safety at runtime. To this end, we need to shift two major elements of a safety assurance lifecycle to runtime:

(1) Analyzing the current risk and the resulting safety goals with the associated integrity levels.

(2) Analyzing the current fulfillment of safety requirements. As a basis, we are using safety engineering concepts and nomenclature as established in the automotive domain.

A. *Dynamic Risk Analysis*

Regarding the hazard analysis performed according to the automotive safety standard ISO 26262 [14], we need to regard hazardous events, i.e., a coincidence of failures and driving situations. For the risk assessment of the given hazardous event, a safety manager analyses the typical factors exposure *E*, controllability *C*, and severity *S*. Based on those three parameters, he determines the associated automotive safety integrity level (ASIL) and assigns a top-level safety-goal for mitigating the hazardous event.

Therefore, a hazardous event h analyzed in a hazard analysis and risk assessment is a tuple consisting of the driving situation d, the failure f, the exposure e, the controllability c, the severity s, the integrity level i, as well as a reference g to the assigned safety goal as top-level safety requirement:

$$h = \langle d, f, e, c, s, i, g \rangle \tag{1}$$

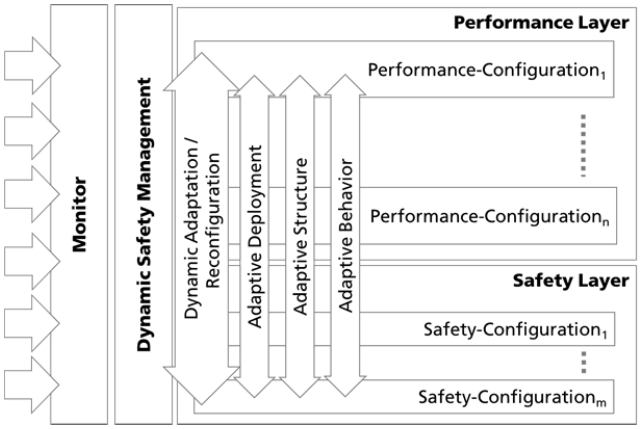

Fig. 1. General Adaptive Safety-Awareness Architecture

The set of all h defines the overall hazard analysis H. The set of all safety goals that have been referenced in the analysis defines the set of top level safety requirements the system must fulfill.

$$G = \bigcup_{h \in H}(h.g)\,; H = \{h\} \tag{2}$$

A safety goal (as any other safety requirement) consists of a functional part g_f defining the actual requirement and an integrity g_i.

$$g = \langle g_f, g_i \rangle \tag{3}$$

For each safety goal g we can derive a set H^g of these hazardous events that reference g.

$$H^g = \{h \in H \mid h.g = g\} \tag{4}$$

Based on this set, we can calculate the integrity g_i of a safety goal g as the maximum of the integrity levels of the hazardous events referencing g.

$$g.g_i = max_{h \in H^g}(h.i) \tag{5}$$

In order to enable a runtime analysis, we shift this information to a simplified runtime representation.

As a first step of the analysis, we determine the runtime state $\tilde{h}$ for each hazardous event h. For each $\tilde{h}$, we need to know the current driving situation $\tilde{d}$. Instead of a general worst-case estimation of the exposure to this situation, we calculate a probability $\tilde{p}_e$ to be in this situation. Moreover, we update the integrity level of the referenced safety goal $\tilde{g}$.

$$\tilde{h} = \langle \tilde{d}, \tilde{p}_e, \tilde{g} \rangle \tag{6}$$

For detecting a driving situation, we specify the latter using a vector of characteristic parameters $\vec{p}$ that are observable

using monitoring mechanisms. In fact, those parameters such as vehicle speed, vehicle position etc. need to be defined already at design time as a formalization of existing analysis approaches.
In addition, we define weights for each parameter in a vector $\vec{w}$. At runtime, the system's perception determines the probability of fulfillment for each parameter including a confidence $\vec{c}$ for each parameter measurement:

$$\tilde{d} = \langle \vec{p}, \vec{w}, \vec{c} \rangle \tag{7}$$

Based on those values the system can calculate a runtime probability $\tilde{p}_e$ that the system is in the situation $\tilde{d}$ that is considered in the hazardous event $\tilde{h}$:

$$\tilde{p}_e = \frac{1}{n} \sum_{i=1}^{n} p_i w_i c_i \tag{8}$$

Using this probability, the system can calculate a set $\tilde{H}$ of hazardous events that are still relevant in the car's current driving situation by ignoring all those hazardous events, whose situation is too unlikely at the current point in time. The definition of the probability threshold p_e^{min} is an important step that should be part of the safety assessment.

$$\tilde{H} = \{\tilde{h} | \tilde{h}.\tilde{p}_e \geq p_e^{min}\} \tag{9}$$

For the time being, we take a rather conservative approach and only consider the current driving situation for eliminating hazardous events that are not of relevance at the current point in time. Thus, the system will remove all hazardous elements that are unlikely in a current driving situation, which leads to a set $\tilde{G}$ of safety goals that are still relevant in the current situation.

$$\tilde{G} = \bigcup_{\tilde{h} \in \tilde{H}} (\tilde{h}.\tilde{g}) \tag{10}$$

Additionally, the system recalculates the integrity levels of each relevant safety goal. If some hazardous events could be removed in the previous step, this might lead to a reduction of the integrity level, as the safety goal always inherits the maximum integrity level from the assigned hazardous events

$$\tilde{g}.g_i = max_{\tilde{h} \in \tilde{H}^{\tilde{g}}} (\tilde{h}.i) \tag{11}$$

B. *System Safety-State Analysis*.

The first step of the analysis (as described in Subsection A) enables the system to be aware of a dynamic, situation-specific set of actually relevant safety goals. By this means, it is aware of the dynamic safety boundaries it must ensure.

The second part of the analysis creates the system's awareness of its dynamically available safety capabilities. Therefore, we map the results of monitors, such as error detection, to the fulfillment of safety requirements. Using a runtime model reflecting the dependencies between requirements allows the system to trace the impact from all monitoring results to the fulfillment of top-level safety goals.

For this, we use a modular approach (e.g. ConSerts, cf. [11][12]) and safety requirements in form of safety contracts applied to single components of a system. In general, our framework is generic to a certain extent and can be mapped to different system architecture descriptions (e.g., AUTOSAR), which in turn determine the relationships of the single elements. Following the idea of ConSerts, we describe a configuration *c* of a component as a tuple consisting of a set R_G of requirements guaranteed by the configuration, a set R_D of requirements the configuration demands, and a set *I* of invariants that must be fulfilled

$$c_M = \langle R_D, R_G, I \rangle \tag{12}$$

An invariant $i \in I$ consists of a rationale describing invariant for the engineer. In order to enable the system to evaluate invariants at runtime, the engineer must map the invariants to a set $\mathcal{M}$ of runtime monitors using a mapping function ι, taking the results of the monitors as inputs and providing a probability value as output.

$$\begin{gathered} i = \langle i_r, \iota \rangle \\ \iota \colon \mathcal{M}^n \mapsto [0,1] \end{gathered} \tag{13}$$

In fact, we model the mapping function as logical equation using a fault-tree-like Boolean calculation tree with the fulfillment of the invariant as root of the tree and the monitors as leaves of the tree. Boolean operators specify the mapping of monitors to the fulfillment of the invariant. At runtime, the system can either use the Boolean representation for a simple Boolean calculation or for a probabilistic calculation using conventional fault-tree-calculation rules and algorithms.
We extend the representation of a requirement *r* to a tuple consisting of the functional part of the requirement r_f, the integrity level r_i, a set R_d of demanded requirements (whose fulfillment is a prerequisite for fulfilling the given requirement), a set *E* of evidences that prove the fulfillment of the requirement, and a set *I* of invariants that must hold. To evaluate the fulfillment of a requirement at runtime, we additionally define an evaluation function ρ.

$$\begin{gathered} r = \langle r_f, r_i, R_d, E, I, \rho \rangle \\ \rho \colon [0,1]^n \mapsto [0,1] \end{gathered} \tag{14}$$

Defining requirements in a way to make them processable for automated analyses at runtime poses several challenges. Currently, we use the basic concepts of the ConSerts-approach [11][12] for defining requirements related to functional / logical behavior of the module. The mapping

function takes several probabilistic input values and maps them to a probabilistic output value.

Again, we define the mapping as a Boolean calculation tree. The fulfillment of the regarded requirement is the root of the tree, the fulfillment of other demanded requirements $r_d \in R_d$, the fulfillment of evidences $e \in E$, or the fulfillment of invariants $i \in I$ are the leaves of the calculation tree.

Similarly, we describe evidences $e \in E$ as a tuple of an evidence documentation e_d and a set A of assumptions that were made when the engineer generated the evidence. If the assumptions are not fulfilled at runtime, the evidences are not valid anymore.

$$e = \langle e_d, A \rangle \tag{15}$$

An assumption $a \in A$ consists of a description and again a mapping function α that maps monitor results to the fulfillment of the assumption.

$$\begin{aligned} a &= \langle a_d, \alpha \rangle \\ \alpha &: [0,1]^n \longmapsto [0,1] \end{aligned} \tag{16}$$

Obviously, this way of describing safety requirements requires more formalisms for enabling the system to reason about safety at runtime. In fact, however, this formalization is only required for those aspects subject to dynamic adaptation. At the same time, this approach provides a natural boundary of dynamic adaptation: if it is not possible to formally define the relation of monitors to the fulfillment of requirements or the fulfillment of assumptions, then a safe adaptation at runtime will not be possible and should be avoided. At runtime, a configuration is described by

$$\tilde{c}_M = \langle \tilde{R}_D, \tilde{R}_G, P(I), P(E) \rangle \tag{17}$$

Since the system does not need any kind of informal documentation at runtime, we simplify the runtime representation of invariants to a probability of the invariant to be true. To this end, we use a conjunction of the results of the mapping functions ι defined for each $i \in I$, i.e. all invariants must be fulfilled at runtime for enabling a configuration (which is again a conservative approach).

$$P(I) = P(\Lambda_i \, \iota_i) \tag{18}$$

Similarly, we reduce evidences to a probability calculation based on the mapping functions of the assumptions.

$$P(E) = P(\Lambda_i \, \alpha_i) \tag{19}$$

Moreover, we reduce requirements to the probability of fulfillment of the requirement based on its mapping function ρ. The current integrity level is derived from the fulfillment probability.

$$\begin{aligned} \tilde{r} &= \langle \tilde{r}_i, P(\rho) \rangle \\ \tilde{r}_i &= f(P(\rho)) \end{aligned} \tag{20}$$

For the analysis at runtime, we use a similar calculation scheme as we have successfully applied it for runtime safety contracts [11][12]. We create a dependency tree between the components and start with analyzing the components forming the leaves of the dependency tree. Then we step-wisely aggregate the results until eventually the guaranteed requirements of the components forming the root of the dependency tree can be directly mapped to the safety goals referenced in the dynamic hazard analysis and risk assessment.

For analyzing a single component, we inspect each single configuration. We first analyze the probabilities *P(I)* of all invariants defined for the configuration and for each requirement defined in the configuration. Moreover, we calculate the probabilities *P(E)* of all evidences for all requirements defined in a configuration. In the second step, we calculate the current integrity level and the fulfillment probability for each demanded requirement $\tilde{r}_D \in \tilde{R}_D$. In the last step, the system calculates the current integrity level and the fulfillment probability of the configuration's guaranteed requirements $\tilde{r}_G \in \tilde{R}_G$. The dependent components then use those results to continue the calculation until we have reached the top level.

Based on this analysis, the system can identify which requirements and thus, which safety goals can be fulfilled at the current point in time. It also knows which configurations are available in each module. Using calculation approaches, which we described in previous work on adaptive systems [13], this enables the system to calculate a valid safe configuration space for the overall system, hence, realizing dynamic safety management.

IV. Conclusion

Future systems need to be able to react safely, yet dynamically to uncertainties that have not been anticipated at design time. To this end, it is necessary to create safety-awareness as a subset of self-awareness and to use this as a basis for dynamic safety management. The approach described in this paper, combines and extends different existing puzzle pieces as a first step towards an overall dynamic safety management framework. This mainly shows that we have to rethink safety-engineering traditions for assuring safety of highly dynamic and highly intelligent systems of the future.

Despite these open challenges, we believe that dynamic safety management provides a huge potential for future safety assurance.

REFERENCES

[1] Laprie, J. C. (2008, June). From dependability to resilience. In 38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks (pp. G8-G9).

[2] De Lemos, R., Garlan, D., Ghezzi, C., Giese (2017). Software Engineering for Self-Adaptive Systems III. Assurances. Lecture Notes in Computer Science 9640. Springer.

[3] Cámara, J., de Lemos, R., Ghezzi, C., Lopes, A. (2013). Assurances for Self-Adaptive Systems. Lecture Notes in Computer Science 7740. Springer.

[4] De Lemos, R. et al. (2013). Software engineering for self-adaptive systems: A second research roadmap. In Software Engineering for Self-Adaptive Systems II (pp. 1-32). Springer, Berlin, Heidelberg.

[5] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., & Bruel, J. M. (2009, August). Relax: Incorporating uncertainty into the specification of self-adaptive systems. In Requirements Engineering Conference, 2009. RE'09. 17th IEEE International (pp. 79-88). IEEE.

[6] Adler, R., Schneider, D., Trapp, M. (2010). Engineering dynamic adaptation for achieving cost-efficient resilience in software-intensive embedded systems. In Engineering of Complex Computer Systems (ICECCS), 15th IEEE International Conference on (pp. 21.30). IEEE.

[7] Domis, D., Trapp, M. (2008). Integrating safety analyses and component-based design. In International Conference on Computer Safety, Reliability, and Security (pp. 58-71). Springer Berlin Heidelberg.

[8] K. Höfig, M. Trapp, B. Zimmer, P. Liggesmeyer (2012). Modeling Quality Aspects: Safety, Model-Based Engineering of Embedded Systems (pp. 107-118). Springer Berlin Heidelberg

[9] Blair, G., Bencomo, N., France, R. B. (2009). Models@ run.time. In IEEE Computer, vol. 42, no. 10, (pp. 22-27), IEEE.

[10] Trapp, M., Schneider, D. (2014). Safety assurance of open adaptive systems–a survey. In: Models@Run.Time (pp. 279-318). Springer.

[11] Schneider, D., Trapp, M. (2013). Conditional Safety Certification of Open Adaptive Systems. In ACM Transactions on Autonomous and Adaptive Systems . ACM.

[12] Daniel Schneider. "Conditional Safety Certification for Open Adaptive Systems", In PhD Theses in Experimental Software Engineering, Fraunhofer Verlag, ISBN-10: 383960690X, 03 2014.

[13] Ruiz, A., Juez, G., Schleiss, P., Weiss, G. (2015). A safe generic adaptation mechanism for smart cars. In proceedings of IEEE 26th International Symposium on Software Reliability Engineering (ISSRE) (pp. 161-171), IEEE.

[14] ISO 26262-1:2011 International Standard. Road vehicles - Functional safety

[15] Schneider, D. et al. (2015): WAP: digital dependability identities. In: 26th International Symposium on Software Reliability Engineering (pp. 324–329). IEEE

[16] Adler, R., Schneider, D., & Trapp, M. (2010, March). Engineering dynamic adaptation for achieving cost-efficient resilience in software-intensive embedded systems. In Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on (pp. 21-30). IEEE.