Multi-Agent Optimization for Safety Analysis of Cyber-Physical Systems: Position Paper

Önder Gürcan, Nataliya Yakymets, Sara Tucci-Piergiovanni, Ansgar Radermacher

Multi-Agent Optimization for Safety Analysis of Cyber-Physical Systems: Position Paper

Önder Gürcan, Nataliya Yakymets,
Sara Tucci-Piergiovanni, Ansgar Radermacher

CEA, LIST, Laboratory of Model driven engineering for embedded systems,
Point Courrier 174, Gif-sur-Yvette, F-91191 France
{onder.gurcan,nataliya.yakymets,sara.tucci,ansgar.radermacher}@cea.fr

Abstract. Failure Mode, Effects and Criticality Analysis (FMECA) is one of the safety analysis methods recommended by most of the international standards. The classical FMECA is made in a form of a table filled in either manually or by using safety analysis tools. In both cases, the design engineers have to choose the trade-offs between safety and other development constraints. In the case of complex cyber-physical systems (CPS) with thousands of specified constraints, this may lead to severe problems and significantly impact the overall criticality of CPS. In this paper, we propose to adopt optimization techniques to automate the decision making process conducted after FMECA of CPS. We describe a multi-agent based optimization method which extends classical FMECA for offering optimal solutions in terms of criticality and development constraints of CPS.

Keywords: safety analysis; self-adaptation; optimization; FMEA; FMECA.

1 Introduction

Cyber-physical systems (CPS) are complex organizations of software and hardware systems (i.e. systems of systems) expected to serve, aid and cooperate with humans. Examples of CPS include large-scale engineering systems such as avionics, healthcare, transportation, automation and smart grids. CPS are expected to exceed traditional embedded systems in various aspects such as efficiency, safety, reliability, robustness, adaptability, availability and so on [1]. Since CPS are expected to interact and involve *humans*, they require a high-level of safety which can be achieved by following rigorous procedures defined in safety standards [2]. Those procedures describe application of various safety assessment (SA) methods starting from the early phases of system development life-cycle.

Over the last decade, the Model-Driven Engineering (MDE) [3] approach was widely used by design engineers to describe and analyze CPSs at the conceptual and design phases of their development cycle [4]. Using open MDE frameworks, such as Eclipse Modeling Framework and Papyrus UML/SysML modeler, CPSs

can be described in Unified Modeling Language (UML)[1], System Modelling Language (SysML)[2] or any Domain Specific Language (DSL) like AADL, RobotML, etc. and then extended to perform different types of analysis. MDE environments integrate various technologies to provide an advanced support for system requirement management [5], design [6], analysis [7–9], verification and validation, and deployment. Most of those procedures, including SA activities, can benefit from tighter coupling with MDE environments.

One of the typical SA methods recommended by safety standards [2] is Failure Mode and Effects Analysis (FMEA) and its extensions: Production FMEA (or PFMEA), Criticality FMEA (or FMECA), Diagnostic FMEA (or FMEDA). PFMEA is used to prioritise, in terms of cost, problems to be addressed in system production. FMECA [10] is used to assess the criticality of system failures and to propose improvements introduced as a list of recommended preventive actions to avoid those failures. The criticality of system failure depends on failure severity, occurrence and detectability and will be defined in the next section. By the preventive action we mean a change in system architecture that may be implemented to address a failure of system components. Another modification of FMEA called FMEDA helps safety experts to define self checking features and provides detailed recommendations for system architecture. Therefore, FMEA is used to identify causes and effects of failures that might appear across system life-cycle and, in addition, it gives a detailed specification of system failures, their criticality, cost and ways to avoid them. However, in the case of complex CPS with thousands of specified failures, multiple FMEA tables (PFMEA, FMECA, FMEDA) may lead to severe problems when choosing the trade-offs between cost, criticality and diagnostic coverage of a system. Therefore, existing methods and tools for FMEA can be improved to automate this task and to offer optimal solutions in terms of safety related constraints.

In this paper we focus on FMECA type of analysis. The classical FMECA is made in a form of the table filled in either manually or by using MDE methods and tools for SA [7, 8]. The latter approach helps to partially automate quantitative part of FMECA related to *criticality* assessment of system components. However, FMECA is not capable to detect cases when i) multiple failures can be rectified using the same preventive action or ii) several alternative prevented actions have been defined to rectify a single failure. This may lead to overuse of resources needed to increase system safety (implement redundant preventive actions) and can significantly impact the cost of CPS. The design engineers have to decide which actions might be rejected to reach a trade-off between *safety* and other development constraints. Therefore, existing methods and tools for FMECA can be improved to automate this task and to offer optimal solutions in terms of *criticality* and other constraints. In the literature, this sort of optimization is called multidisciplinary optimization (MDO). The goal of MDO is

[1] Object Management Group, The OMG Unified Modeling Language (OMG UML), Superstructure, version 2.4.1, 2011.

[2] Object Management Group. OMG Systems Modeling Language (OMG SysML), 1st Sept. 2007.

to find the configuration that maximizes (or minimizes) several objectives while satisfying several constraints [11].

We, therefore, propose to extend the classical FMECA using MDO to resolve the aforementioned problem. The proposed method analyzes recommended preventive actions associated with component failures and categorizes them according to existing constraints. As an MDO approach, we have chosen the Adaptive Multi-Agent Systems (AMAS) approach [12] since it is a good candidate for finding an optimal level of system *criticality* and *configuration* [13]. Using AMAS, we can find a set of configurations where the *criticality* of each component is below the threshold as much as possible under certain development constraints. We believe that the use of a criticality-based self-adaptation technique like AMAS, along with the adoption of SA knowledge will make it possible to harness the complexity of the given problem by finding *optimal* configurations automatically.

The remaining of the paper is organized as follows. Section 2 gives background information about FMECA and states the problem. Section 3 presents the AMAS approach and shows our method to extend classical FMECA to build safety self-adaptable CPSs. Lastly, Section 4 discusses and concludes the paper and gives some prospects for further work.

2 Failure Mode, Effects and Criticality Analysis

FMECA is an inductive bottom up approach used to identify different effects (or consequences) and causes of component failures by analyzing them from component-level up till system-level. While the qualitative FMEA [10] helps to define main causes and effects of failures, the quantitative FMECA identifies the *criticality* level of failures. The *criticality* of a failure is automatically evaluated according to (1).

$$\mathcal{C}(f) = \mathcal{S}(f) \cdot \mathcal{O}(f) \cdot \mathcal{D}(f), \tag{1}$$

where $\mathcal{C}$ is the criticality, $\mathcal{S}$ is the severity, $\mathcal{O}$ is the occurence and $\mathcal{D}$ is the detectability of failure f. The severity of failure characterizes the consequences that the failure could have on CPS or its environment. The occurence of failure characterizes the appearance of the failure and its average exposure occurrence on CPS or its environment. The detectability of the failure characterizes the means which exist to detect or plan the appearance of the feared event. The severity, occurence and detectability criteria are usually evaluated according to the matrices recommended by the domain specific standards and norms. Table 1 gives the evaluation matrices for the severity, occurence and detectability criteria adopted in our work.

During FMECA each system component is annotated with the *critical threshold*, the boundary value of the allowed *criticality* for this component. If the value of *criticality* estimated through FMECA is higher than the critical threshold, the analyzed component is considered as *critical*. Figure 1 illustrates a simple example of FMEA and FMECA: The FMEA table describes possible causes and

effects of the failure called Failure1 ("No current from Generator") of the Generator component of the train detection system. The FMECA table shows the results of the criticality analysis of Failure1 in the Generator component. The evaluation matrices scale severity, occurence and detectability criteria from 1 to 4 (Table 1). We assume that the critical threshold of each component is 2. According to (1) the estimated initial criticality of Failure1 is 6 which is higher than the threshold. Consequently, Failure1 is critical for the Generator component. The two actions recommended to tackle Failure1 of Generator include "Use of robust components" and "Introduction of hardware redundancy".

Table 1. Evaluation matrices for severity, occurence and detectability.

Severity				
Level	Negligible	Significant	Critical	Catastrophic
Rank	1	2	3	4
Descr.	Deterioration of the system with no impact on its availability neither functioning.	Deterioration of the system, which makes it not available to perform some operations.	Deterioration of the system, which leads to its unavailability, permanent or definitive.	User's deadly, potentially deadly or permanent injuries.

Occurence				
Level	Very Low	Low	Medium	High
Rank	1	2	3	4
Descr.	Less than once a week.	At least once a week.	Several times a week.	Daily.

Detectability				
Level	High	Medium	Low	Very Low
Rank	1	2	3	4
Descr.	Failure mode systematically detectable before its appearance.	Failure mode usually detectable before its appearance.	Failure mode hardly detectable before its appearance.	Failure mode not detectable before its appearance.

The FMECA process offered by most of the MDE tools for SA can be summarized in several steps: system modeling, annotation, analysis, and result generation (Figure 2). A system model is created using either UML/SysML notations (e.g., Sophia [7] or HiP-HOPS [8] tools) or formal languages like NuSMV [14], SAML [15]. Then the model is annotated (or extended) with the description of possible failures of system components for further FMECA which is conducted according to the specified failures and their severity, occurrence and detectability criteria. The results are displayed using dedicated profiles, editors, tables and report generation modules.

The classical FMECA gives a detailed description of causes and effects of single failures on component-level, checks what happens on system-level and

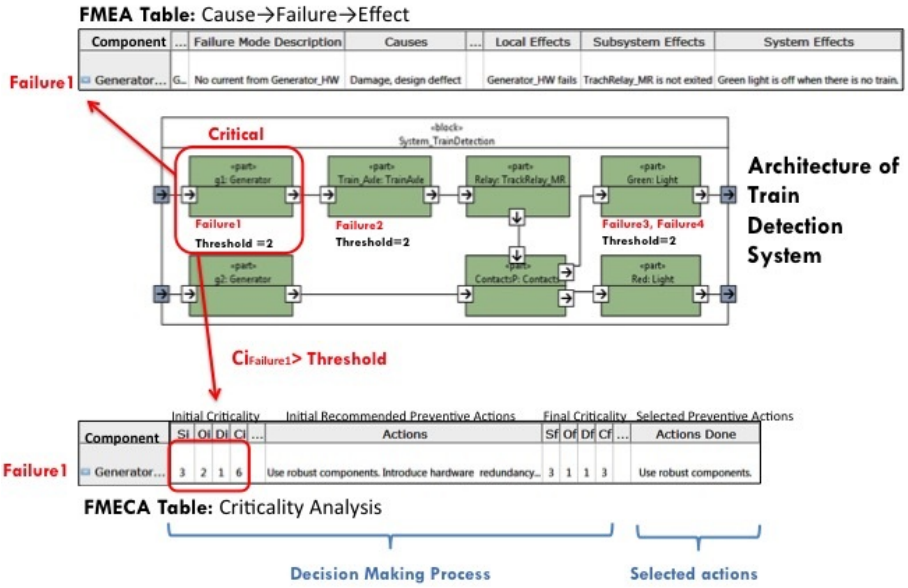

Fig. 1. A simple example of FMEA and FMECA.

guides design and safety engineers how to prevent failures. The latter is done via analysis and selection of recommended *preventive actions* associated with every failure. A preventive action is a change in system architecture that may be *selected* and further *implemented* to address a failure of system components. We consider several types of relations between recommended actions and failure modes:

- Relation type 1: one action targets one failure mode;
- Relation type 2: several complementary actions target one failure mode;
- Relation type 3: several alternative actions target the same failure mode;
- Relation type 4: one action targets several failure modes.

Those actions are defined during the FMECA process by the safety engineer and can be implemented by the design engineer to improve system safety.

However, in the case of strict specification requirements, such as *cost* (how much it will cost to implement all the preventive actions) or *time* constraints (how long it will take to implement all the preventive actions), some improvements recommended during FMECA may be rejected (for instance, when choosing between several alternative actions targeting the same failure). Taking into account a complexity of CPSs and, as a result, sophisticated FMECA (such FMECA tables can include thousands of failures and associated preventive actions) the choice of preventive actions that should be implemented is not trivial. In addition, this can significantly increase the overall *criticality* of CPS. The

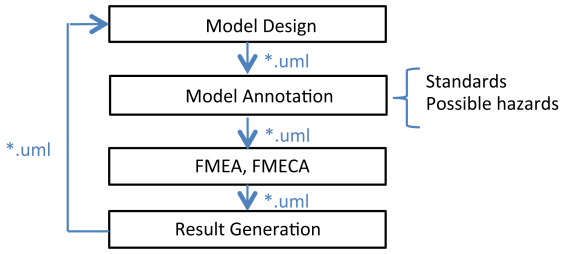

Fig. 2. The automated FMEA process.

FMECA tools do not provide information on how to optimize a set of preventive actions that should be selected for further implementation.

Based on this observation, we propose to extend classical FMECA method to automate the *decision making process* related to choosing optimal *selected actions* in terms of safety and other development constraints (Figure 1). We describe a method for optimized safety analysis based on FMECA technique and show our preliminary model that is intended to be implemented by extending our tool for SA called Sophia [7].

3 Using AMAS for Improving FMECA

For improving the FMECA process, an optimization technique, that will help finding an optimum set of recommended actions by taking into account the trade-off between criticality and cost, is needed. In this sense, the Adaptive Multi-Agents Systems (AMAS) approach [12] has been selected since it fits well to the complexity of the problem at hand [16].

3.1 The AMAS Approach

Overview In the AMAS approach, the system is composed of a set of dynamic number of autonomous agents $\mathcal{A} = \{a_0, a_1, ...\}$. According to AMAS, a system is said to be *functionally* adequate if it produces the function for which it was designed, according to the viewpoint of an external observer who knows its finality. To reach this functional adequacy, it has been proven that each autonomous agent $a_i \in \mathcal{A}$ must keep relations as cooperative as possible with its social (other agents) and physical environment [12, 17]. To do so, each agent a_i keeps tracks of its degree of criticality and tries to help the most *critical* agent in its neighborhood including itself. In other words, a critical agent is said to be the most *dissatisfied* one and its criticality has to be reduced either by itself or its neighbors[3].

[3] In certain conditions, it spontaneously communicates information to agents that it thinks the information would be useful.

Agent-Criticality Heuristic The *criticality*[4] value of an agent a_i at time t is calculated by using an *agent-criticality* function $c_{a_i}(t)$. This function may return a value ranging between 0.0 and 100.0 where 0.0 criticality indicates the highest degree of satisfaction and 100.0 criticality indicates the lowest degree of satisfaction of an agent. There is no single formula for *agent-criticality* functions. It can be defined by making use of the agent's internal parameters and state: e.g., an agent with correct internal parameter values is said to be closer to its goal and thus should be more satisfied than another agent that is still searching for its right parameter values; or an agent in a non-cooperative state should be less satisfied compared to another agent in a cooperative state. However, evaluation methods and calculation of the *agent-criticality* are specific to each type of cooperative agent and thus may change from domain to domain. Consequently, it is the designer's responsibility to identify the most appropriate *agent-criticality* function for each type of cooperative agent depending on the problem, the domain, the constraints, etc.

Non-Cooperative Situations The value of inputs coming from other agents (and physical environment) leads a_i to produce a new decision. A non-desired configuration of inputs causes a non-cooperative situation (NCS) to occur. a_i is able to memorize, forget and spontaneously send feedbacks related to desired or non-desired configurations of inputs coming from other agents. We denote the set of feedbacks as $\mathcal{F}$ and model sending a feedback $f_a \in \mathcal{F}$ using the action of the form $\mathsf{send}(f_a, \mathcal{R})$ where a is the source of f and receiver agents $\mathcal{R} \subset \mathcal{A} \setminus \{a\}$. A feedback $f_a \in \mathcal{F}$ can be about increasing the value of the input ($f_a\uparrow$), decreasing the value of the input ($f_a\downarrow$) or informing that the input is good ($f_a\approx$).

Local Solving When a feedback about a NCS is received by an agent, at any time during its life-cycle, it acts in order to avoid or overcome this situation [18] for coming back to a cooperative state. This provides an agent with learning capabilities and makes it constantly adapt to new situations that are judged harmful. In case a NCS cannot be overcome by an agent, it keeps track of this situation by using a level of annoyance value ψ_{f_a} where f_a is the feedback about this NCS. When a NCS is overcome, ψ_{f_a} is set to 0, otherwise it is increased by 1. The first behaviour an agent tries to adopt to overcome a NCS is a *tuning behaviour* in which it tries to adjust its internal parameters. If this tuning is impossible (because a limit is reached or the agent knows that a worst situation will occur if it adjusts in a given way), it may propagate the feedback (or an interpretation of it) to other agents that may handle it. If such a behaviour of tuning fails many times and ψ_{f_a} crosses the reorganization annoyance threshold $\psi_{reorganization}$ (reorganization condition), an agent adopts a *reorganisation behaviour* in which it tries to change the way of its interaction with others (e.g., by

[4] To avoid confusion of the term *criticality* between the safety analysis domain and the AMAS approach, the terms *safety-criticality* and *agent-criticality* are used respectively hereafter in this paper.

changing a link with another agent, by creating a new one, by changing the way in which it communicates with another one, etc.) In the same way this behaviour may fail counteracting the NCS and a last kind of behaviour may be adopted by the agent: *evolution behaviour*. This is detected when ψ_{f_a} crosses the evolution annoyance threshold $\psi_{evolution}$ (evolution condition). In the evolution step, an agent may create a new one (e.g., for helping itself because it found nobody else) or may accept to disappear (e.g., it was totally useless and decides to leave the system). In these two last levels, propagation of a problem to other agents is always possible if a local processing is not achieved. The overall algorithm for suppressing a NCS by an agent is given in Algorithm 1 in [19].

3.2 Identification of agents and their nominal behaviors

We designed an agent-based simulation model Sim, for optimizing FMECA described in Section 2, by basically capturing all taken design decisions based on the AMAS theory as a dynamic undirected graph $Sim(t) = (\mathcal{G}(t), \mathcal{P}(t), q)$ where $\mathcal{G}(t)$ is the set of time varying failure mode agents, $\mathcal{P}(t)$ is the set of time varying preventive action agents and q is the quality agent.

In the initial model, we only consider the *cost* constraints. Consequently, each preventive action agent $p \in \mathcal{P}(t)$ has only a cost parameter. If p is selected the cost has a non-zero value, otherwise its value is zero.

Each failure mode agent $g \in \mathcal{G}(t)$ has a set of recommended preventive action agents $\mathcal{P}_g(t) \subset \mathcal{P}(t)$ and selects preventive action agents $p \in \mathcal{P}_g(t)$ for implementation. In the initial model, we considered the cases where *one action can target one failure mode* (Section 2, relation type 1) and *several complementary actions target one failure mode* (Section 2, relation type 2). We denote the set of preventive actions of a failure mode agent g at time t as $Sel_g(t) = \{p \in \mathcal{P}_g(t) | \{p_1, p_2, .., p_n\}\}$. On the other hand, each preventive action $p \in \mathcal{P}_g(t)$ is selected by failure mode agents $g \in \mathcal{G}(t)$. We denote the set of failure mode agents of a preventive action agent p at time t as $SelBy_p(t) = \{g \in \mathcal{G}(t) | \{g_1, g_2, .., g_n\}\}$.

A failure mode agent $g \in \mathcal{G}(t)$ through its nominal behaviour aims to increase the number of selected preventive actions, as much as possible. Similarly, a preventive action agent $p \in \mathcal{P}(t)$ through its nominal behaviour aims to increase as much as possible the number of failure modes it is selected for.

The quality agent q is responsible for the satisfaction of the global quality properties like *cost* and *time* constraints. q knows list all preventive action agents $p \in \mathcal{P}(t)$. Since we only consider *cost* in this paper, the nominal behaviour of q is to continuously collect the cost information from the preventive action agents $p \in \mathcal{P}(t)$, calculate the total cost $\tau(t)$ for each time t by summing up the cost of each $p \in \mathcal{P}(t)$ and compare it with the total project budget β.

3.3 Identification of Non-Cooperative Situations and Feedbacks

The proposed agent-based model, in which the configuration of failure mode agents and preventive action agents (their number and connection) can change, is

subject to NCSs. All NCSs are identified by analyzing the possible bad situations of FMECA regarding to the explanation given in Section 2.

Bad Safety-Criticality Value If the number of selected preventive actions of a failure mode agent g is not enough and thus the safety-critical threshold is crossed, a *bad safety-criticality value NCS* is detected by g. When such a situation is detected at time t, the failure mode agent g should improve its preventive actions set (by having better preventive actions in the set). To do so, it sends an *select more* feedback ($f \in \mathcal{F}_{sel}\uparrow$) to some or all of its recommended preventive action agents $p \in P_g(t)$. Otherwise, the selection is good and an *selection good* feedback ($f \in \mathcal{F}_{sel}\approx$) is sent to $P_g(t)$.

Bad Total Cost The quality agent q continuously calculates the total cost $\tau(t)$ as mentioned before and if this cost crosses the total budget β ($\tau(t) > \beta$) a *bad total cost NCS* is detected. When such a situation is detected at time t, the quality agent q should reduce the number of selected preventive actions (and, consequently, the budget). To do so, it sends a *select less* feedback ($f \in \mathcal{F}_{sel}\downarrow$) to some or all of its preventive action agents $Pre_q(t)$. Otherwise, the costs are good and a *selection good* feedback ($f \in \mathcal{F}_{sel}\approx$) is sent to $Pre_q(t)$.

3.4 Agent-Criticality Functions

As described before, each cooperative agent a has to define an *agent-criticality* function $c_a(t)$ for calculating their *agent-criticality* value at time t.

Failure Mode Agents For a failure mode agent g_i the agent-criticality is inversely proportional to the number of its selected preventive actions, thus as an agent-criticality function we use

$$c_{g_i}(t) = \frac{1}{m_i(t)} \tag{2}$$

where m_i is the number of selected preventive actions of g_i at time t.

Preventive Action Agents Similarly, for a preventive action agent p_i the agent-criticality is inversely proportional to the number of failure modes it is selected for, thus as an agent-criticality function we use

$$c_{p_i}(t) = \frac{1}{n_i(t)} \tag{3}$$

where n_i is the number of failure modes of p_i at time t.

Quality Agent For the quality agent q, on the other hand, the agent-criticality is inversely proportional to the number of its selected preventive actions, thus as an agent-criticality function we use

$$c_q(t) = \frac{1}{n_i(t)} \tag{4}$$

where n_i is the number of failure modes of p_i at time t.

3.5 Cooperative Behaviours

There is no *tuning behaviour* for agents in our initial model since there is no parameter to tune. There is also no *evolution behaviour* for any agent type since the failure modes and recommended preventive actions are predefined. Currenty, we only defined *reorganization behaviours* as cooperative behaviours of agents.

The *reorganization behaviours* of failure mode agents and recommended preventive action agents are modelled using actions of the form $\mathsf{add}(\{g, p\})$ and $\mathsf{remove}(\{g, p\})$ for $g \in \mathcal{G}(t)$ and $p \in \mathcal{P}_g(t)$, which correspond to the formation and suppression (respectively) of a selection relation $\{g, p\}$ at time t. It is assumed that no *selection relation* is both added and removed at the same time.

NCSs are suppressed by processing these cooperative behaviours as follows. When a select more feedback ($f \in \mathcal{F}_{sel}\uparrow$) is received by a preventive action agent $p \in \mathcal{P}_g(t)$ from a failure mode agent g, p first checks if it is the most critical preventive action agent among its neighbours. If yes, it executes $\mathsf{add}(\{g, p\})$ both for helping g and reducing its criticality. If there are more critical neighbours, p forwards the incoming feedback to the most critical one.

Similarly, when a select less feedback ($f \in \mathcal{F}_{sel}\downarrow$) is received by a preventive action agent $p \in \mathcal{P}_g(t)$ from the quality agent agent q, p first checks if it is the most critical preventive action agent among its neighbours. If yes, p forwards the incoming feedback to the most critical one. If there are more critical neighbours, it executes $\mathsf{remove}(\{g, p\})$ both for helping g and itself, where $g \in SelBy_p(t)$ and g is the least critical failure mode agent.

Otherwise, if a *selection good* feedback ($f \in \mathcal{F}_{sel}\approx$) is received, p does not execute any cooperative behaviour.

4 Discussion & Conclusions

Cyber-physical systems (CPS) are constantly growing in complexity. This is accompanied by an increasing need for safety in those systems. In this paper, we propose to enrich the safety assessment process for CPSs by improving the Failure Mode, Effects and Criticality Analysis (FMECA) method. Our motivation is based on the observation that the classical FMECA becomes very complicated and time consuming when the system at hand has thousands of failure modes and thousands of corresponding recommended preventive actions. In addition to the classical FMECA, the method proposed in this paper adopts a multi-agent

approach, namely Adaptive Multi-Agent Systems (AMAS), for providing an optimal set of recommended actions to be implemented, in order to get a trade-off between system *safety-criticality* and other constraints such as *cost* and *time*.

AMAS is a promising candidate for *criticality*-based optimization problems like one arising from post analysis of FMECA results. Moreover, since AMAS is a self-organizing solution, it offers significant advantages such as increased scalability [20]. It distributes the complexity of the preventive action selection issue across agents. The scalability and success on optimization of AMAS has been shown before in various studies [21, 19, 16, 22, 13, 23, 24]. Due to the limitations coming from the current state of the art of the FMECA methods and tools, it is not possible to realize the proposed solution directly. First, we need information about constraints such as cost, time etc. which is not the scope of classical FMECA. Secondly, we need to be able to automate the calculation of *safety-criticality* per failure mode depending on the implemented actions.

In the initial model presented in this paper, we only considered the cases where *one action can target one failure mode* (Section 2, relation type 1), *several complementary actions target one failure mode* (Section 2, relation type 2) and the *cost* constraints. We will further elaborate our model for covering all the relation types between recommended preventive actions and failure modes, and the *timing* constraints. This way, we plan to obtain a highly realistic model.

Furthermore, as a future prospect, we believe that if we can integrate more safety related information (such as behavioral model) to the safety analysis process, it would be possible to realize fully-automated solutions for FMECA with high coverage of situations. For instance, such a solution may take into account *run-time* resource consumptions while deciding the number and type of replications needed. Currently, there is no such model in the literature.

References

1. Park, K.J., Zheng, R., Liu, X.: Cyber-physical systems: Milestones and research challenges. Computer Communications **36**(1) (2012) 1 – 7
2. IEC, I.E.C.: Functional Safety of Electrical, Electronic, Programmable Electronic Safety-Related Systems. IEC 61508, parts 1 to 7. Technical report (1998 and 2000)
3. Estefan, J.A.: Survey of model-based systems engineering (MBSE) methodologies. Technical Report INCOSE-TD-2007-003-01, International Council on Systems Engineering (INCOSE) (June 2008)
4. Selić, B., Gérard, S.: Modeling and Analysis of Real-Time and Embedded Systems with {UML} and {MARTE}. Morgan Kaufmann, Boston (2014)
5. Adedjouma, M., Dubois, H., Maaziz, K., Terrier, F.: A model-driven requirement engineering process compliant with automotive domain standards. In: Proceedings of the Third Workshop on Model Driven Tool and Process Integration (MDTPI), Paris, France (June 2010) 85–96
6. Gérard, S., Dumoulin, C., Tessier, P., Selic, B.: 19 papyrus: A uml2 tool for domain-specific language modeling. In Giese, H., Karsai, G., Lee, E., Rumpe, B., Schtz, B., eds.: Model-Based Engineering of Embedded Real-Time Systems. Volume 6100 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2010) 361–368

7. Yakymets, N., Dhouib, S., Jaber, H., Lanusse, A.: Model-driven safety assessment of robotic systems. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on. (Nov 2013) 1137–1142
8. Walker, M., Bottaci, L., Papadopoulos, Y.: Compositional temporal fault tree analysis. In: Proc. of the 26th Int. Conf. on Computer Safety, Reliability, and Security. SAFECOMP'07, Berlin, Heidelberg, Springer-Verlag (2007) 106–119
9. Walker, M., Reiser, M.O., Tucci-Piergiovanni, S., Papadopoulos, Y., Lnn, H., Mraidha, C., Parker, D., Chen, D., Servat, D.: Automatic optimisation of system architectures using east-adl. J of Syst. and Soft. **86**(10) (2013) 2467 – 2487
10. IEC, I.E.C.: Analysis techniques for system reliability – Procedures for FMEA. Technical report (1985)
11. Sobieszczanski-Sobieski, J., Haftka, R.: Multidisciplinary aerospace design optimization: survey of recent developments. Struct. optimization **14**(1) (1997) 1–23
12. Capera, D., Georgé, J., Gleizes, M., Glize, P.: The amas theory for complex problem solving based on self-organizing cooperative agents. In: WETICE'03: Proc. of the 20th Int. W. on Enabling Technologies, Wash., DC, USA, IEEE CS (2003) 383
13. Kaddoum, E., Georgé, J.: Collective self-tuning for complex product design. In: Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2012, Lyon, France, September 10-14, 2012. (2012) 193–198
14. Bozzano, M., Villafiorita, A.: The fsap/nusmv-sa safety analysis platform. International Journal on Software Tools for Technology Transfer **9**(1) (2007) 5–24
15. Gudemann, M., Ortmeier, F.: A framework for qualitative and quantitative formal model-based safety analysis. In: High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on. (Nov 2010) 132–141
16. Jorquera, T., George, J.P., Gleizes, M.P., Regis, C.: A natural formalism and a multi-agent algorithm for integrative multidisciplinary design optimization. In: Web Intelligence and Intelligent Agent Technologies, 2013 IEEE/WIC/ACM International Joint Conferences on. Volume 2. (Nov 2013) 146–154
17. Camps, V., Gleizes, M.P., Glize, P.: A self-organization process based on cooperation theory for adaptive artificial systems. In: 1st International Conference on Philosophy and Computer Science: Processes of evolution in real and Virtual Systems, Krakow, Poland. (1998)
18. Bernon, C., Capera, D., Mano, J.P.: Engineering self-modeling systems: Application to biology. (2009) 248–263
19. Gürcan, Ö., Türker, K.S., Mano, J.P., Bernon, C., Dikenelli, O., Glize, P.: Mimicking human neuronal pathways in silico: an emergent model on the effective connectivity. Journal of Computational Neuroscience **36**(2) (2014) 235–257
20. Martin-Flatin, J., Sventek, J., Geihs, K.: Self-managed systems and services. Communications of the ACM **49** (2006) 37–39
21. Couellan, N., Jan, S., Jorquera, T., Georg, J.P.: Self-adaptive support vector machine: A multi-agent optimization perspective. Expert Systems with Applications **42**(9) (2015) 4284 – 4298
22. Gürcan, Ö., Bernon, C., Türker, K., Mano, J.P., Glize, P., Dikenelli, O.: Simulating human single motor units using self-organizing agents. In: Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on. (Sept 2012) 11–20
23. Combettes, S., Sontheimer, T., Rougemaille, S., Glize, P.: Weight optimization of aircraft harnesses (short paper). In: International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS), Salamanca, Springer-Verlag (mars 2012) 229–232

24. Welcomme, J.B., Gleizes, M.P., Redon, R.: A Self-organising Multi-Agent System Managing Complex System Design Application to Conceptual Aircraft Design. International Transactions on Systems Science and Applications, Self-organized Networked Systems **5**(3) (november 2009) 208–221