Towards A Holistic Software Systems Engineering Approach for Dependable Autonomous Systems

Adina Aniculaesei
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
adina.aniculaesei@tu-clausthal.de

Jörg Grieser
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
joerg.grieser@tu-clausthal.de

Andreas Rausch
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
andreas.rausch@tu-clausthal.de

Karina Rehfeldt
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
karina.rehfeldt@tu-clausthal.de

Tim Warnecke
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
tim.warnecke@tu-clausthal.de

ABSTRACT

Autonomous systems are gaining momentum in various application domains, such as autonomous vehicles, autonomous transport robotics and self-adaptation in smart homes. Product liability regulations impose high standards on manufacturers of such systems with respect to dependability (safety, security and privacy). Today's conventional engineering methods are not adequate for providing guarantees with respect to dependability requirements in a cost-efficient manner, e.g. road tests in the automotive industry sum up millions of miles before a system can be considered sufficiently safe. System engineers will no longer be able to test and respectively formally verify autonomous systems during development time in order to guarantee the dependability requirements in advance. In this vision paper, we introduce a new holistic software systems engineering approach for autonomous systems, which integrates development time methods as well as operation time techniques. With this approach, we aim to give the users a transparent view of the confidence level of the autonomous system under use with respect to the dependability requirements. We present already obtained results and point out research goals to be addressed in the future.

KEYWORDS

autonomous systems, self-learning systems, dependability, safety, security, privacy, quality assurance, runtime verification

ACM Reference Format:
Adina Aniculaesei, Jörg Grieser, Andreas Rausch, Karina Rehfeldt, and Tim Warnecke. 2018. Towards A Holistic Software Systems Engineering Approach for Dependable Autonomous Systems. In *SEFAIAS'18: IEEE/ACM 1st International Workshop on Software Engineering for AI in Autonomous Systems , May 28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3194085.3194091

SEFAIAS'18, May 28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5739-5/18/05…$15.00
https://doi.org/10.1145/3194085.3194091

1 INTRODUCTION

The advent of autonomous systems operating in environments shared by humans is arguably one of the biggest technological disruptions of the next decade. Take for instance the automotive industry, which is developing autonomous vehicles that are expected to share the road with human drivers within the next couple of years. Autonomous systems are also gaining momentum in other areas of application, ranging from autonomous transport robotics to self-adaptation in smart homes. The current success of autonomous systems can be attributed to several factors:

(1) The **growing user demand** for smart, interconnected, autonomous systems that offer efficient support for all areas of business and life.
(2) The significant **improvement in the area of data acquisition** which is enabled by a wide range of cheaper and better sensors available on the market.
(3) The **development of self-* features** supported through the usage of inferencing technologies to extract valuable information from metadata, thus improving the self-awareness of autonomous systems.
(4) The **expansion of the theory in artificial intelligence (AI)** through new concepts and methods supported by the development of powerful open source AI frameworks and the existence of cheap and powerful hardware best fitting for AI.

With the growing impact of autonomous systems in our daily lives, the need to guarantee that these systems behave correctly becomes more significant than ever. We identify three big issues which need to be addressed: *safety*, *security* and *privacy*. In this paper, we use the notion of *dependability* when we refer to the issues of safety, security and privacy as a whole. Providing proper solutions for dependability is a crucial challenge for the long-term societal acceptance of autonomous systems.

Today's conventional engineering methods are not adequate for providing guarantees with respect to dependability in a cost-efficient manner. As an example, consider the field tests in the automotive domain which require a large number of miles to be driven in order to demonstrate that a system is sufficiently safe. One strategy for reducing the cost of quality assurance is transferring a significant part of the testing effort from road tests to (system-level) simulations. However, system simulations raise several challenges.

On the one hand, the relevance of the simulation is questionable if it can only reflect a subset of all (infinite many) situations that can occur in the real operational environment of a system under test. On the other hand, there are no specifications available for AI components to derive the relevant system test cases. Based on these considerations, we must accept that system engineers will no longer be able to test and respectively formally verify autonomous systems during development time in order to guarantee the dependability requirements in advance. In order to meet these challenges, we introduce the vision of a new holistic software systems engineering approach for autonomous systems which integrates development time as well as operation time techniques. The main goal of this approach is to help developers to design, build and evolve safe autonomous systems.

The rest of the paper is structured as follows. In Section 2 we review related work regarding development of safe autonomous systems and in Section 3 we define our problem domain and formulate our research questions. Section 4 describes in detail our concept while in Section 5 we identify research goals for future work. In Section 6 we give a summary of the paper.

2 RELATED WORK

There is already a number of concepts built to increase the dependability of autonomous systems. Leitão and Restivo present in [3] the ADAptive holonic COntrol aRchitecture (ADACOR) designed for increasing the flexibility and agility of manufacturing control systems. ADACOR uses an agent-based self-learning system to provide a better response to disturbances. Cheng et al. describe in [5] an approach which shows how adaptation targets for adaptive systems can be modelled. The approach allows to model the boundary conditions that the system should consider when adapting to changes in the environment. These boundary conditions help to control the uncertainty of both system and environment. In [13], Morandini et al. present a new requirements description language for adaptive systems. Tropos4AS is able to represent requirements at runtime for the usage in adaptation decisions, similar to Cheng et al. [5]. However, these approaches demand a clear picture of the future operational environment, which in our opinion, is not feasible in uncertain environments any more. In [4], Bures et al. discuss software engineering for smart cyberphysical systems (sCPS). sCPS matches to our understanding of autonomous systems. The authors conclude that the current development process must be adapted. Firstly, the smartness and the adaptability of systems mean that testing at development time is no longer sufficient. Secondly, systems must also take into account unforeseen situations that might occur in their operational environment. Our approach will address these two issues.

In the DEIS (Dependability Engineering Innovation for Cyber Physical Systems) project [17], researchers are tackling the challenge that CPS are typically loosely coupled and build temporary system configurations. The DEIS project aims for a modelling and integration framework for assuring the dependability of CPS based on the concept of a Digital Dependability Identity. Their approach covers development and operation time. In [7], Heckemann et al. present a reference architecture for safe automotive software based on so called safety cages. Those safety cages are similar to our dependability cages with respect to their hierarchical building, to the monitoring of vehicle functions' behavior, and to the action-taking in case of possible dangerous situations. Our approach, however, goes one step further by also taking into account adaptive and changing systems and environments and providing an iterative engineering process. It combines knowledge from development, testing and operation time in a holistic software systems engineering approach.

3 CHALLENGES OF DEPENDABLE AUTONOMOUS SYSTEM DEVELOPMENT

Classical development approaches are not able to cope with the specific challenges of developing dependable autonomous systems. This fact stems from the concepts on which dependable autonomous systems are based: performing their tasks autonomously even in unknown environments (liveness properties), and at the same time satisfying their dependability requirements (safety, security and privacy properties).

In the classical software development of cyberphysical systems, the assumption is that all development artifacts - system specification, use cases, model of the operational environment - are completely documented and validated (closed-world assumption). From the development artifacts, a suitable architecture, itself a development artifact, is derived and a compliant system is built. The system is tested at development time using test cases derived from the development artifacts. Since all use cases and the environment are fully known and specified in advance, the system is stable in its operation. Therefore, the system behavior during operation time is the same as the system behavior designed and tested at development time. A further development cycle is started when new requirements are specified by the customer or the system environment is adapted in a controlled manner.

However, development processes, in our example the well established V-Model, used for building cyberphysical systems no longer scale in case of autonomous systems. There are two main reasons for this. Firstly, autonomous systems learn while they are in operation, i. e. they adapt their behavior to better interact with other systems and people or to solve problems more effectively. For the purpose of this paper, the notions of autonomous system and adaptive and learning system are henceforth used interchangeably. Secondly, autonomous systems are exposed to constantly changing and uncertain environments. Since the system environment and the behavior of the system can no longer be described entirely in advance, the formerly closed and valid development artifacts become incomplete and open. This means that the system cannot be completely designed, constructed and tested during the development phase. For this reason, it is essential that the information gained from the system in operation is used to further augment the incomplete development artifacts and to initiate further development cycles. In the following we discuss further these three areas of distinction between development of classical systems and of dependable autonomous systems and their own specific dependability risks.

3.1 Challenge 1 - Uncertain and Unknown Environments

During the design of classical cyberphysical systems the developers assume that they have complete (or at least sufficient) knowledge of the real environment in which the system operates. This assumption is not new in system development, but it becomes particularly critical in the field of autonomous systems. Instead, the operational environment of such systems is no longer controllable or predictable and therefore cannot be completely specified at design time.

Unforeseen situations in the operational environment can lead to unexpected system behavior, as the car accidents of Google [1], Tesla [8] or Uber [15] vehicles show. For example, in an accident with a Tesla, its autopilot function could not distinguish a white truck from the bright, cloudy sky behind it. Since it was developed for highways, the Tesla system was not designed to respond reliably to traffic at intersections, and therefore, it did not take into account that vehicles could visually merge with their surroundings due to their color.

At this point, we should accept that such problems are unavoidable, because it is impossible to document, test and simulate all possible, occurring scenarios. Therefore new methods are required which must consider the unpredictability of the environment in the system behavior by monitoring it, learning from it and taking it into account in the further development of the system. Based on these challenges, we formulate our first research challenge:

***Challenge 1 - Uncertain and Unknown Environments**: How can a safe and secure behavior for an autonomous system be guaranteed even in uncertain environments?*

3.2 Challenge 2 - Adaptive and Self-learning Systems

Classical systems are developed in such a way that their behavior is described by and conform to their specification. Thus they meet the requirements placed on them in the same way every time. This behavior is desirable for most types of systems, such as information systems or cyberphysical systems. However, this is not sufficient for autonomous systems, since they are not only expected to adhere to their specification, but also solve problems more effectively or aquire new skills. In order to achieve this, autonomous systems must learn from the experienced situations and adapt accordingly. This is especially necessary for very complex tasks, for which the problem domain cannot be fully specified. As an example, to grasp the complexity of autonomous driving, consider the number of years of driving experience which a person must have in order to be able to assess potentially dangerous traffic situations.

In contrast to traditional systems, adaptive and learning systems also entail risks and challenges. While the behavior of classical systems is predictable and comprehensible, the behavior of adaptive and learning systems can deviate from the behavior specified at development time. For example, in 2016, the Twitter bot TayAndYou was launched by Microsoft as an experiment to communicate with people aged 18-24 [9]. It was supposed to learn from the conversations with the Twitter users and adapt accordingly, but the bot only learned to curse and scold other people. This behavior was of course was neither planned nor expected by the system designers.

Based on the basic concepts of adaptive and learning systems, we have to accept that we cannot completely specify and predict the behavior of this type of systems. Given their incomplete specification and the uncertainty of their operational environment (see Challenge 1), it is not possible to test these systems exhaustively during their development time. Therefore, we need to provide new ways to monitor adaptive and learning systems and develop procedures to verify the correctness of their behavior. Based on these challenges, we formulate our second research challenge:

***Challenge 2 - Adaptive and Self-learning Systems**: How can we guarantee a safe and secure behavior for all parts of an autonomous system - complex functions, machine-learned functions, sensors and actuators, and the whole system - even when the system's behavior changes at operation time?*

3.3 Challenge 3 - Open and Incomplete Artifacts

The current development of cyberphysical and information systems assumes that the developmental artifacts are closed and valid. This means that the developers assume that all customer requirements are taken into account, the environment is fully specified, and all other artifacts required for development, such as simulation models, are available and valid. However, this idea does not work with autonomous systems, because on the one hand the environment has an infinite number of situations and a very high level of detail and on the other hand the behavior of the system will change or adapt at operation time through learning.

Consequently, this means the autonomous system is never stable in its implementation even after delivery. With regard to the system tests, it can also no longer be assumed whether a system has been sufficiently tested and is considered safe according to today's standards. Take for example autonomous driving, for which different degrees of autonomy from *no automation* to *full automation* are indicated by 6 SAE levels [16]. Up to level 3, the *conditional automation*, the human driver must always be available as a fallback level in order to be able to intervene in the case of an error. For these levels, it is therefore sufficient to test the system for false positives. In other words, the system does not have to react in every encountered situation. However, if a system's reaction takes place, then this reaction must be correct with respect to the system specification. In order to test systems with the level of autonomy between SAE 0 and SAE 3, it is sufficient to identify and test the situations in which a system's reaction is necessary. The circumstances change dramatically in case the driver is no longer available as the fallback performance level and the system operates fully autonomously. In addition to testing for false positives, highly automated systems as well as fully automated systems must also be tested for false negatives. This means that in every conceivable situation when it is necessary the system's reaction is compulsory and this reaction must be correct with respect to the system specification.

In consequence, state-of-the-art development time and runtime verification and testing methods are not sufficient any more and new approaches for the verification and validation of autonomous systems are needed. Furthermore, these new methods must be accompanied by a change of paradigm in the system development process. This also means, however, that we have to abandon the

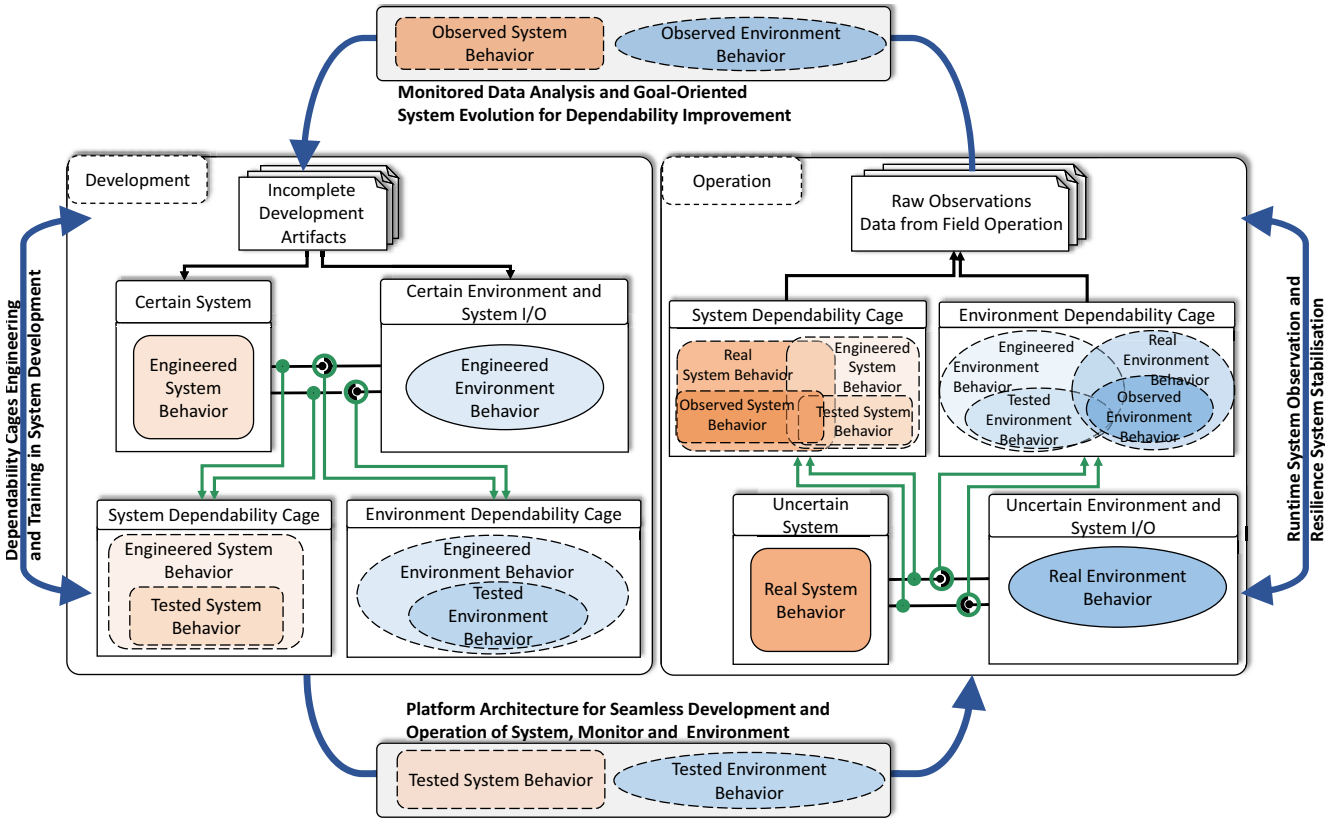

Figure 1: Dependability Cages: Overall Approach.

idea of having complete and valid development artifacts when developing autonomous systems. Based on these considerations, we formulate our third research challenge:

***Challenge 3 - Open and Incomplete Artifacts**: How can we guarantee, that we tested our system for relevant situations and how can we improve the relevance and completeness of test cases even with incomplete specifications?*

4 HOLISTIC APPROACH FOR THE DEVELOPMENT OF DEPENDABLE AUTONOMOUS SYSTEMS

In order to tackle the challenges highlighted in Section 3, we introduce a new holistic approach for the development of dependable autonomous systems. Our approach is based on the notion of *Dependability Cages*, which are derived by system engineers from the existing development artifacts. Dependability Cages are then used at development time as well as during operation time to check the correctness of the system behavior with respect to its dependability requirements. An overview of our integrated approach of Dependability Cages is given in Fig. 1.

The challenges identified in Section 3 point out to two types of risks which have to be considered through all development phases of autonomous systems: (1) external risks, due to the uncertainties in the system's real environment, and (2) internal risks, brought up by the changing behavior of the autonomous system. In order to safeguard against these risks, we define two categories of Dependability Cages: Dependability Cages developed for the system and a Dependability Cage for the system environment. In order to use the Dependability Cages, we distinguish between several types of behavior of the system and its environment, both at development time and at operation time.

For a given autonomous system as well as for its environment, we make a distinction at development time between its engineered behavior and its tested behavior (see Figure 1 left). The *engineered behavior* of the system as well as that of the system environment is the behavior specified and documented by the existing development artifacts. At development time, Dependability Cages developed both for the system and the system environment are used to test the respective engineered behavior. The behavior which is tested is denoted as *tested behavior*, and is naturally a subset of the engineered behavior.

Similarly to development time, we differentiate at operation time between the real behavior and the observed behavior of a given autonomous system and respectively of the system environment (see Figure 1 right). The *real behavior* is the behavior at operation time which may differ from the engineered behavior. It contains an uncountable number of situations, that may be caused by a number of factors. For an autonomous system, its real behavior is based

on the adaptation of the system to the constantly changing operational environment. For the environment itself, the real behavior is determined by its uncertainty due to the unforeseeable situations which may occur during system operation. The *observed behavior* is a subset of the real behavior and represents the behavior monitored at operation time through the Dependability Cages.

Once the development time tests are completed, the tested behavior is transferred into operation time via a platform envisioned for this purpose (see Figure 1 bottom). In turn, the observed behavior is channeled back to development time in order to augment the development artifacts and allow the evolution of the autonomous systems, and consequently, the improvement of their dependability through training of the Dependability Cages (see Figure 1 top).

Thus, our approach consists of four major parts which are described in detail in the following sections:

- Dependability Cages Engineering and Training in System Development
- Runtime System Observation and Resilience System Stabilization
- Monitored Data Analysis and Goal-Oriented System Evolution for Dependability Improvement
- Platform Architecture for Seamless Development and Operation of System, Monitor and Environment

4.1 Dependability Cages Engineering and Training in System Development

To handle the external and internal risks to which autonomous systems are subject during their operation, we construct several types of Dependability Cages using the existing development artifacts. Each type of Dependability Cage is tailored to address either external or internal risks.

In order to guard an autonomous system from external risks, we define and develop *Environment Dependability Cages*, illustrated in Figure 2. Such Cages are capable of testing the interface between the system and its environment at development time and constantly monitor the same interface at operation time in order to detect any evolutions which may contradict the assumptions made about the operational environment at development time.

An autonomous system is not only subject to external risks but also to internal risks which originate in the system's capability to adapt itself and learn from situations experienced in its operational environment. In order to safeguard an autonomous system from internal risks, we need to ensure the behavior of the system components and functions as well as that of the autonomous system as a whole. For this purpose, we introduce Dependability Cages for the sensors and actuators as well as for the preprocessing and postprocessing and other system components (see Figure 2).

In order to achieve their purposes, testing at development time and monitoring at operation time, Dependability Cages take into account the engineered and tested behavior as well as the real and observed behavior of both the system and the environment (see Figure 3). Furthermore, we distinguish between two different monitors in a Dependability Cage - qualitative and quantitative. While the *qualitative monitor* checks the functional correctness of the system (and thus looks inwards), the *quantitative monitor* verifies whether

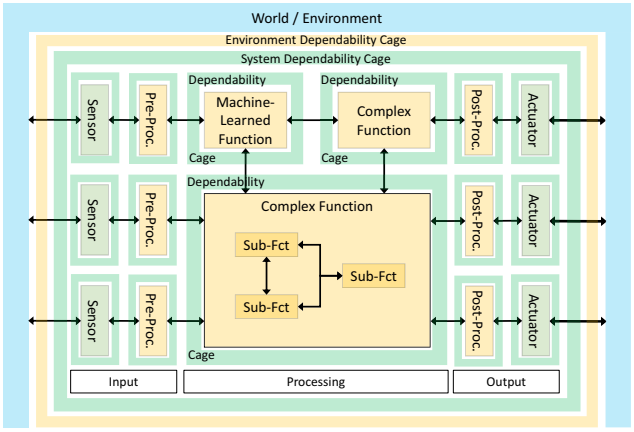

Figure 2: Autonomous System with Dependability Cages.

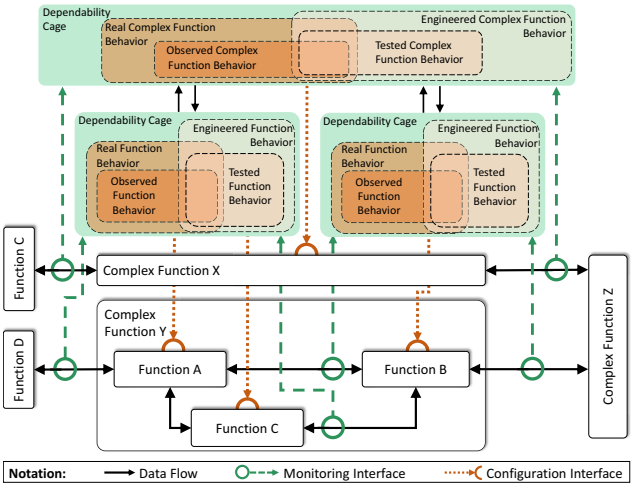

Figure 3: Engineering of Dependability Cages.

the system is still operating in a secure environmental context (and thus looks outwards).

The qualitative monitor compares the system's observed behaviour with the tested one. We consider any observed behavior identical to tested behavior a safe and correct behavior. For example, imagine a trajectory calculation. At the time of development, it was tested whether the function behaves according to its specification for different inputs. During operation time, adaptation of the overall system configuration or individual components may result in new traces of inputs for the trajectory calculation that have not been tested. In the worst case, the calculation then no longer provides outputs according to its specification. The qualitative monitor recognizes such cases.

During the operation time, an autonomous system can observe much more in its environment than it was specified at development time. A quantitative monitor observes the behavior of the environment of the autonomous system and checks whether the system is still in a context that was secured at the time when it was tested. As

an example, imagine a system designed exclusively for daytime use. When used at night, sensors may no longer function reliably due to the lack of light. In this case, the quantitative monitor detects that the context of the system is potentially unsafe, since it is untested. The quantitative monitor can be compared to the utility functions introduced by Fredericks et al. in [6]. Both monitor the changes in the operational environment of a adaptive and learning system and trigger the adaptation of development artifacts in case of changes in the operational environment.

Each Dependability Cage gathers its information by connecting to its test object through two types of interfaces: a *monitoring interface* and a *configuration interface* (see Figure 3). If the assessment of the qualitative monitor indicates a failure of a given system function, then this result is sent to the Dependability Cages of other functions that call this system function. On the basis of the received information, the Dependability Cage of the calling function has two choices. Either try a new configuration which brings the system in a correct and safe state or reflect the aggregated result at user level and ask for his or her intervention. If the quantitative monitor indicates a new behavior observed at operation time, then this behavior is logged in order to improve the development artifacts in further development cycles.

We showed a first concept for engineering an Environment Dependability Cage in [2]. The concept was evaluated using the prototype of a mobile service robot in a simulated environment.

4.2 Operation Time System Observation and Resilience System Stabilization

In order to achieve their purposes, Dependability Cages must perform several tasks. The internal structure of a Dependability Cage is shown in Figure 4, exemplary for a complex function.

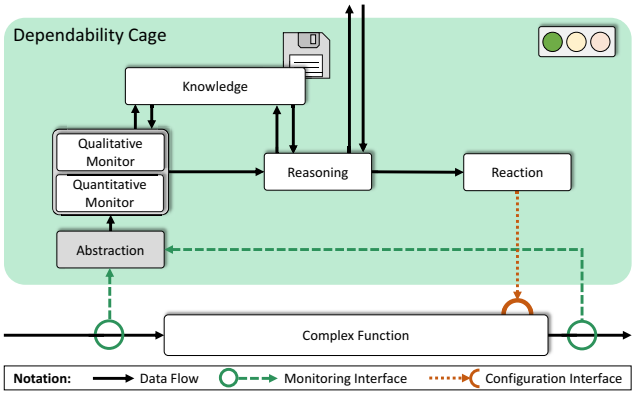

Figure 4: Internal structure of a Dependability Cage.

In order to handle the large amount of different situations, the Dependability Cage must abstract the input and output of the monitored complex function. An example for a feasible abstraction approach is the arrangement of the values in equivalence classes.

The Dependability Cage stores in a knowledge base the new situations encountered in the operational environment. In the first development cycle at operation time, the information contained in the knowledge base consists of the engineered and the tested behavior of the function. The knowledge base is augmented each time new behavior is observed. Moreover, the Dependability Cage defines compensating measures, such as adaptation, and a confidence level classification represented as a traffic light.

Once the abstraction step is finshed, the monitor component of the Dependability Cage compares the abstracted input data during operation time with tested situations to identify new observed situations. For this purpose, the monitor uses the information provided by the knowledge component, namely whether the observed situation is a known or a new one. If the monitor detects a new situation, the monitor adds the situation data to the knowledge component. Furthermore, the monitor compares the output of the complex function with the expected output to determine whether the complex function operates correctly.

If a failure, a critical deviation in signals or a new situation is detected by the monitor, the reasoning component is used to determine the cause of it. Additionally, the reasoning component determines the confidence level of the Dependability Cage. For the confidence level the reasoning component has to evaluate the current situation as safe or unsafe, with consideration of the compensatory reactions. Moreover, the reasoning component is the communication interface for the exchange of information between hierarchical Dependability Cages.

Feasible reactions in case of an implausibility or error are managed by the Dependability Cage through the reaction component. First, the application tries to reconfigure itself autonomously and automatically. This can be done, for example, by selecting those components that are more suitable in the current configuration. The objective of the adaptation is to make the system as resilient and robust as possible during operation time. In the worst case scenario, the system deactivates components or some functions to ensure a safe state of the entire system.

In [12], we introduced function monitoring for advanced driver assistance systems using Dependability Cages. We built the monitors in the Dependability Cages on the basis of safety requirements and trained them using simulated test cases. The approach was evaluated using an industrial prototype of a lane change assistant and data recorded on German highways.

4.3 Monitored Data Analysis and Goal-Oriented System Evolution for Dependability Improvement

As mentioned in Section 3, the uncertainties of the autonomous systems and their environment represent a challenge for the development of such systems. At development time it is almost impossible to have a complete picture of the operational environment and of the corresponding system requirements.

The development process of an autonomous system must therefore be rethought. Instead of developing a system once and for all, the developers and the process have to adapt to the constantly changing demands. This is the only way to guarantee that an autonomous system remains dependable. To achieve this, it is crucial that the findings from the operation time are fed back into the development artifacts.

Dependability Cages monitor the system and its correctness at operation time. The goal for a Dependability Cage is to ensure that the observed behavior stays (at least to a certain amount) inside the tested behavior. Depending on the situation, the Cages have a limited possibility to intervene and keep the system in a safe state, for example through adaptation. In case the observed behavior moves too far out of the tested behavior, i.e. the systems moves in possibly unsafe space, a new development cycle may be triggered. Then, the data from the Cages can be used to supplement the development artifacts, and thus, provide new insights for a further development cycle of the system.

The development artifacts also benefit from the fact that there is normally not only one instance of the system in operation, but entire fleets. Since autonomous systems typically operate at different temporal and spatial contexts, the Cages of the individual systems further complement each other in order to provide an overall picture of the environment as accurate as possible.

One of the challenges we have to face is how the collected data can be meaningfully used in development artifacts and further development cycles. For this purpose, the data must be readable for humans so that system developers can gain new insights from the data. Another possibility is to use raw data as new training data for machine learning functions.

4.4 Platform Architecture for Seamless Development and Operation of System, Monitor and Environment

Safeguarding autonomous systems in uncertain environments is, as already indicated in the previous sections, a multi-dimensional and complex task. The possibilities to reconfigure and change a system at operation time and the introduction of Dependability Cages requires a runtime environment respectively a platform that actively supports these capabilities. However, the design and development of such a platform for safe, autonomous systems poses new challenges, both technical and conceptual.

The first challenge is the software architecture of a safe and autonomous system. The system has to be in a safe state at all time and because of that it must be possible to adapt to new situations, as described in Section 4.2. Furthermore, the system must be able to change its internal structure, components and their connections, at operation time. The adaptability should even go as far as integrating new or updated components into the system at operation time. The integration of adaptivity into a platform which supports dependability aspects implies the use of a formalized component model which describes the structure of components, their interfaces and communication methods. Only a common understanding of these parts allows a functionally correct adaptivity.

The second challenge is the integration of the Dependability Cages into the platform. In contrast to software components that provide some kind of functionality for the overall system, Dependability Cages require the ability to observe relevant messages and values exchanged internally between different software components and externally between the autonomous system and its environment. A platform should natively support the Dependability Cages to obtain these information without help of the component developers. The Dependability Cage must also be able to actively intervene in the configuration of the system, e. g. by selecting a predefined fallback configuration if the system should not behave according to its specification. This intervention must be well coordinated with the rest of the autonomous system and must not cause any additional problems or side effects.

Another challenge is the integration of machine learned functions into autonomous systems. Such functions are trained and tested at development time with existing, possibly simulated data. As mentioned above, the operational environment is constantly changing and these functions may be subject to inputs that have not been considered before. To resolve this problem, the platform should be able to run newly integrated functions in the background and Dependability Cages should check their output. A function is only integrated into the rest of the system once it has been monitored by the Dependability Cages and classified as correct for a predefined period of time.

In order to simplify the development of software systems, a distinction is often made between test and operation mode. For example, component developers use simulated hardware or other communication technologies at development time. This is necessary because it is often very complicated to build the structure of the remaining system around the component, or because necessary development artifacts are still missing. In this case, a platform for autonomous systems should enable the developer to easily switch between test and operation mode. Technically, both modes should be identical for the developed component. Switching seamlessly between these two modes is a particular challenge.

We already have developed a dynamic-adaptive component model and middleware for autonomous systems, which is capable of adapting the structure of a system at runtime [11]. Over the years we have integrated a number of different features, including testing and verification at runtime [10, 14].

5 FUTURE WORK

A common embedded system software consists of the three sections: input, processing and output. Until now the runtime verification of the processing is approved through Dependability Cages with promising results [12].

For the verification of the entire software system a complex structure of linked Dependability Cages is required. Therefore, the design of a holistic development process is essential for successful implementation of Dependability Cages. The Dependability Cages must be developed and tested parallel to the actual system. The process has to describe, how the Cages are derived from the monitored software sections and especially how to determine the range of the monitored software sections. Additionally the consideration of the feedback inclusion from the operation time is crucial for fruitful application of the Dependability Cages.

For an effective development procedure the usage of special platforms, which are developed to support this verification approach are helpful. The interfaces for the Dependability Cages should be provided by the tool, especially to avoid retroactive effects which could disturb the reliable operation of the basic software system itself.

Our current function monitoring concept has the disadvantage that development of the Dependability Cages requires a large amount

of manual efforts. Hence, the application of machine learning approaches could be investigated for automation of abstraction and determination of the expected output.

Sensors are the senses of an autonomous system and are essential for the generation of a precise environment model. In a perfect world unexpected situations, software failures or hardware failures do not occur. Unfortunately, in the real world this is not the case. Nevertheless, a reliable operation of the sensor system is important for the entire system and its environment. Due to this fact, three challenges for Dependability Cages result: identification of a failure, detection of the failure cause, and determination of suitable compensating measures. Some sensors have overlapping detection areas, a fact which could be used to compare two different sensors. Thereby, a Dependability Cage could detect hardware failures of a sensor through comparison with other sensors. For example, a video camera and a Lidar observe both the front of a vehicle. In the overlapping area, both sensors will not have the same quality, but their data can be compared. If a Dependability Cage detects a failure, the subsequent challenge is to detect the faulty sensor. This is a Byzantine fault, because with only two sensors the failure causing sensor cannot be identified in any case. Therefore, a sound concept for Dependability Cages for sensor fault detection is needed.

A Dependability Cage can monitor the preprocessing of the sensor data as well. For example, the Dependability Cages could contain a simulation of the preprocessing based on abstracted raw data. This would enable the detection of failures in the preprocessing software functions. The sensor Dependability Cage could communicate the detected failure to a Dependability Cage of a higher hierarchy, which consolidates the current states of all sensors. The same principle could be applied on sensor fusion functions.

Currently there also exist some ideas for monitoring the output section of the software system similar to sensing. Our objective on the long term is to analyse these ideas and to write down a rigorous concept of Dependability Cages which integrates sensors, preprocessing, functions, postprocessing, and actuators for autonomous systems.

6 CONCLUSIONS

In this paper we have presented the vision of a holistic, iterative software development approach for dependable autonomous systems. Our basic assumption is that an autonomous system in operation is repeatedly confronted with situations that were not considered at development time. Because of this, the system has to adapt through adaptation and learning in order to react to the changed environment. However, this means that the system could act outside its specification. In order to identify these changes and take them into account in the further development cycles of the system, we have introduced the concept of Dependability Cages, which examines at runtime the behavior of the system and its environment against boundary conditions defined and tested at development time. In unforeseen situations, Dependability Cages can not only intervene in the configuration of the system, but also record the new situations and feed them into the iterative development process as new development artifacts. We are confident that a holistic software engineering approach as described in this paper will be necessary to develop dependable autonomous systems in the future.

ACKNOWLEDGMENTS

The authors would like to thank Prof. Dr. Falk Howar and Malte Mauritz for their contributions to the concept of Dependability Cages and the inspiring discussions regarding verification of autonomous systems.

REFERENCES

[1] Alex Davies. 2016. Google's Self-Driving Car Caused Its First Crash. https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/. (Feb 2016). [Online; last accessed 30-Jan-2018].

[2] Adina Aniculaesei, Daniel Arnsberger, Falk Howar, and Andreas Rausch. 2016. Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments. In *Proceedings of the The First Workshop on Verification and Validation of Cyber-Physical Systems, V2CPS@IFM 2016, Reykjavík, Iceland, June 4-5, 2016.* 79–90. https://doi.org/10.4204/EPTCS.232.10

[3] Paulo Leit ao and Francisco Restivo. 2006. ADACOR: A holonic architecture for agile and adaptive manufacturing control. *Computers in Industry* 57, 2 (2006), 121 – 130. https://doi.org/10.1016/j.compind.2005.05.005

[4] Tomas Bures, Danny Weyns, Bradley Schmerl, Eduardo Tovar, Eric Boden, Thomas Gabor, Ilias Gerostathopoulos, Pragya Gupta, Eunsuk Kang, Alessia Knauss, Pankesh Patel, Awais Rashid, Ivan Ruchkin, Roykrong Sukkerd, and Christos Tsigkanos. 2016. Software Engineering for Smart Cyber-Physical Systems: Challenges and Promising Solutions. (2016). https://doi.org/10.1145/3089649.3089656

[5] Betty H. C. Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. 2009. A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In *Model Driven Engineering Languages and Systems*, Andy Schürr and Bran Selic (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 468–483.

[6] Erik M. Fredericks, Byron DeVries, and Betty H. C. Cheng. 2014. Towards Runtime Adaptation of Test Cases for Self-adaptive Systems in the Face of Uncertainty. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, New York, NY, USA, 17–26. https://doi.org/10.1145/2593929.2593937

[7] Karl Heckemann, Manuel Gesell, Thomas Pfister, Karsten Berns, Klaus Schneider, and Mario Trapp. 2011. Safe automotive software. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, 167–176.

[8] Jack Stewart. 2018. People Keep Confusing Their Teslas For Self-Driving Cars. https://www.wired.com/story/tesla-autopilot-crash-dui/. (Jan 2018). [Online; last accessed 30-Jan-2018].

[9] Jane Wakefield. 2016. Microsoft chatbot is taught to swear on Twitter. http://www.bbc.com/news/technology-35890188. (Mar 2016). [Online; last accessed 30-Jan-2018].

[10] Holger Klus, Dirk Niebuhr, and Andreas Rausch. 2011. Dependable and Usage-Aware Service Binding. In *ADAPTIVE 2011, The Third International Conference on Adaptive and Self-Adaptive Systems and Applications*. 36–45. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.679.7387

[11] Holger Klus and Andreas Rausch. 2014. DAiSI - A Component Model and Decentralized Configuration Mechanism for Dynamic Adaptive Systems. *International Journal On Advances in Intelligent Systems* 7, 3 and 4 (2014), 27–36.

[12] Malte Mauritz, Falk Howar, and Andreas Rausch. 2016. Assuring the Safety of Advanced Driver Assistance Systems Through a Combination of Simulation and Runtime Monitoring. In *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer International Publishing, Cham, 672–687.

[13] Mirko Morandini, Loris Penserini, Anna Perini, and Alessandro Marchetto. 2017. Engineering Requirements for Adaptive Systems. *Requir. Eng.* 22, 1 (March 2017), 77–103. https://doi.org/10.1007/s00766-015-0236-0

[14] Dirk Niebuhr, Andreas Rausch, Cornel Klein, Jürgen Reichmann, and Reiner Schmid. 2009. Achieving dependable component bindings in dynamic adaptive systems - A runtime testing approach. In *SASO 2009 - 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 186–197. https://doi.org/10.1109/SASO.2009.40

[15] Ryan Randazzo. 2017. Self-driving Uber involved in Tempe crash. https://www.azcentral.com/story/money/business/tech/2017/09/08/self-driving-uber-involved-tempe-crash/647843001/. (Sep 2017). [Online; last accessed 30-Jan-2018].

[16] SAE J 3016. 2016. Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. (2016), 30 pages.

[17] Ran Wei, Tim P Kelly, Richard Hawkins, and Eric Armengaud. 2017. DEIS: Dependability Engineering Innovation for Cyber-Physical Systems. In *Federation of International Conferences on Software Technologies: Applications and Foundations*. Springer, 409–416.