Reliability Considerations for Mechatronic Systems on the Basis of a State Model

Peter Göhner, Eduard Zimmer

Institut für Automatisierungs- und Softwaretechnik
Universität Stuttgart
Pfaffenwaldring 47
70569 Stuttgart
goehner@ias.uni-stuttgart.de
zimmer@ias.uni-stuttgart.de

Talal Arnaout, Hans-Joachim Wunderlich

Institut für Technische Informatik
Universität Stuttgart
Pfaffenwaldring 47
70569 Stuttgart
arnaoutl@informatik.uni-stuttgart.de
wu@informatik.uni-stuttgart.de

Abstract[1]: The first step in analyzing a problem is to establish a valid model that would represent this problem. The model helps mainly in understanding the problem by depicting it in a visual form. Hence, in order to analyze the reliability of mechatronic systems, we need to understand first how such systems fail and how they behave in the presence of a failure. This understanding would help us later in the analysis and the development of formal solutions to achieve the demanded reliability. This could be achieved using the model that we have developed, which will be presented in this paper.

1 Introduction

Mechatronics is a field in which mechanical systems (e.g. actuators, valves, engines, etc…) interact with data processing units (e.g. controllers, microprocessors, application specific integrated circuits – ASICs, etc…), through complex communications in order to carry out their specified task. Complexity arises from the diversity of components running together or depending on each other, and from the need for interfaces and communication protocols between the different stages. Hence, this field brings together development work from three different disciplines: mechanical engineering, electronics engineering, and software engineering.

As our dependence on such systems grows at a rapid pace, faults in such systems become more and more intolerable, due to the risk that they might impose on human life. Thus, the development of fault tolerant solutions has become a major task in designing the complete system. Traditionally, development takes place within the different disciplines independently from each other, each focusing on the areas of their own interest. The terminology used is also abstracted in a way that suits each discipline. In

[1] This research work is funded by the Deutsche Forschungsgemeinschaft (DFG) under contract FOR 460/1-1

the literature, there is a lack of description of specific important terms, such as reliability, in a sense that suits mechatronics as a whole. Most definitions are focused or formulated to suit a certain perspective.

What we aim for is to come up with unified descriptions of the common terms and a model that could serve as the basis for studying the failure behavior of mechatronic systems. This paper gives a brief overview of some necessary basics in section 2, describes how we formulated the model in section 3, shows how the model can be applied to three representative examples, one from each domain, in section 4, and concludes with general remarks in section 5.

2 Basics

As we have already mentioned, the development process in mechatronics usually takes place separately within the respective domains, which is shaped by the respective ways of thinking, vocabulary of concepts, and experiences. However the integration of heterogeneous components into mechatronic systems requires broadening the concepts and cooperation between the technical disciplines involved to develop a common conception of the future product and come up with an optimized solution [GM03]. To facilitate the domain-broadening communication, a common basis should be present for the description of reliability.

There is a need for establishing a common ground for the terminology used between the three domains, which would be the basis of mechatronic studies. Therefore, we have to abstract the reliability related technical peculiarities from each discipline, and unify them in a way suitable for mechatronics as a whole.

As we are dealing with the reliability of mechatronic systems, we ought to find initially a definition that could be representative for all the functional subsystems. Searching the literature, we could realize a set of relevant definitions, which could be stated as follows:

- Reliability is the ability of an item to work properly [Bi04].
- Reliability is the probability that the system produces correct output [VP93].
- Reliability is the probability of non-failure of an item for a given period of time, and for certain operational and environmental conditions [Bi04, VDA00].
- Reliability is probability that an item performs a required function under stated conditions for a stated period of time [Bi04, Bi86, DIN95].
- Reliability is the probability of survival against failures or malfunctions, which mask out one or more functions, or limit them in unspecified ways [VDI86].

The tenor of these different definitions is the same, as they refer to the time-dependent probability that a unit remains functional. However, we have adopted the last definition as our standard definition, due to the following: First, the definition relates the reliability to the survivability of a function, which in turn governs the operability of an item to execute a specified task. Second, it expresses the effect of a malfunction, whether permanent or temporary, on the whole system and on other functions. And third, since functionality is a common concept between the three disciplines, this definition could serve as a basis for mechatronic systems.

Based on this definition, we have developed a model that could represent the *nonfunctionality* of a mechatronic system. For this model, we had the following points in mind:

- The impairment of a certain function should be considered, rather than the failure of a complete unit. This considers transient errors, which are common in electronics.

- A distinction is to be looked upon between dormant and active faults. Here software has been borne in mind, since dormant errors, i.e. bugs, can be present, and only under certain circumstances these errors result in a failure.

- The complexity of mechatronic systems is to be broken down by extracting the basic fundamental functions of the different units. This helps in contemplating the failure behavior per function and assessing the consequential influence of such a failure on other functions. Accordingly, a functional orientation is very suitable to analyze the operability.

Looking into these three points, we can deduce that a function could exist in any of three states, depending on whether a failure has occurred or not: fault free (state 0), faulty (state 1), and failure (state 2). In most cases, the system is presumed to have state 0 as an entry state to the model. In the following section, we will detail how we realized our model, which is depicted in figure 1. However, first we need to define the used terms. According to [Av97, La92, Pr96] failure, error and fault could be defined as follows:

- Failure: A failure is the inability of a function to deliver an output according to the specifications.

- Error: The error is the noticeable deviation or discrepancy in the output of the function from the specification.

- Fault: The fault is the flaw or defect that occurred to the function, causing the error at the output. It could be classified as dormant or active, and temporary or permanent.

3 Model Realization

In order to come up with a representative model that describes the behavior of failures in mechatronic systems, we focused our analysis on some particular cases and observed their respective failure behavior. It is easier to visualize the problem through an example, whose failure behavior is depicted in figure 1. We consider a CNC (computer numerical control) lathe machine, starting our analysis with the mechanical subsystem. The subsystem is initially fault-free, hence in state 0. Assume a shaft, which is currently not in use, breaks. Thus, the subsystem has acquired a fault in one of its parts, which when accessed will deliver an erroneous behavior. Hence the system moves to state 1.

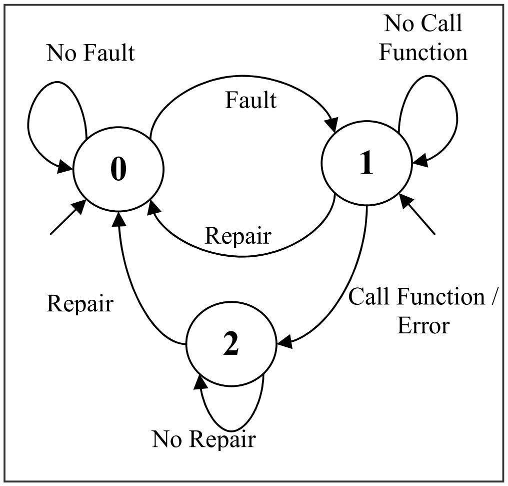

Figure 1: Functional failure behavior model

Once the shaft is used, i.e. its functionality is being requested; the shaft will not deliver its specified task and the system would be termed as failed. Here, the subsystem transits to state 2. The system would remain in a failure state as long as the shaft is not repaired. Once it is repaired, the system is fault-free again, and would shift back to state 0. If the damaged shaft was noticed and repaired before its use, the system also is back fault free, but transits from state 1 to state 0.

Analyzing the electronics subsystem, failures could be due to hard influences, i.e. physical damage, or due to soft influences such as interference or radiations, which causes a bit to flip its value. From here, and assuming no manufacturing defects, the electronic subsystem could be considered fault-free. Upon the occurrence of any fault, whether hard or soft, the system becomes faulty. Once the function that has been influenced by this fault is requested, the subsystem would generate an erroneous result. The system would have failed to deliver its task. The system remains in a failure state till it has been repaired. In such systems, removal of the source of the soft influence or the usage of some error correction mechanism to take care of soft errors could also be termed as repair. Thus, the same three states apply to electronics too.

Without loss of generality, we have assumed that the systems acquire their faults during operation, i.e. they have left the fabrication facility in a fault-free case. We will show later how the opposite case could still be represented with the proposed model.

Focusing our attention now onto the software running in the control unit, we notice that the code itself could not acquire faults during its life time. In terms of code only, assuming that the hardware on which it is residing and running on is fault free and fully compatible with it, if the code is functionally correct, it will remain always correct. Any errors generated by software are due to bugs or faults already residing in the software itself, which is often the case. In this case, the repair mechanism is the correction of the software.

Based on this analysis, we can deduce a model that depicts the behavior of system failures, which could be seen in figure 1.

We realize that state 1 could also be named as an entry state. This takes into account faults being shipped out along with the system. Such faults are ones that have not been detected during the debugging and testing phase.

4 Model Application

In what follows, we will present three examples that exemplify how the above proposed model can be applied: The Pentium bug, the Ariane 5 bug, and the failure in a generator shaft sealing. For further descriptions of the problems, the reader is referred to [SB94, Li96, Pa03].

4.1 The Pentium Bug:

Knowing that the Pentium Bug was due to a design error, which was not detected during the debugging phase, the hardware divider would enter the model from state 1. The hardware always generated correct results as long as it didn't access the faulty cells in the programmable logic array that was used to hold the lookup tables.

Once these faulty cells were accessed, the division process generated a wrong result. Hence, the access to these cells represents the "Call Function" transition. After that, the error was noticed and the failure of the division unit was detected. The Pentium's divider remained in a failure state till patches were released to fix the problem. Since these patches solved the problem, the divider was considered fault-free and transited to state 0.

4.2 The Ariane-5 Bug:

The software for the inertial reference system (IRS) was originally developed for Ariane-4. The software ran successfully in Ariane-4.

In the state model the software, concerning this function, would start from state 0. During the application of the software in the Ariane-4, it remains in state 0. Once the software, without changing the function "measurement of horizontal acceleration", was used for Ariane-5, it leaves the state 0 and shifts to state 1. The reason for this state transition is the possible explicit higher horizontal velocity reached with Ariane-5 in comparison to Arian- 4. The function of the software was however not appropriate for these high horizontal speeds. As soon as the function "Measurement of horizontal acceleration" was claimed, it came to an overflow with the conversion of 64 bits float in 16 bits signed integer and the two inertial reference systems stopped their service. The function of the IRS was claimed, however not given any longer. The software transits into state 2. Briefly after this state transition, Ariane-5 was destroyed by an error sequence.

4.3 The Failure in a Generator Shaft Sealing:

An often used example for the non classical failure of a mechanical structure is the shaft sealing of an electric power generator, which is to prevent the airflow from the inside of the generator to the external environment. The sealing is realized by a stuffing-box seal. The packaging of the stuffing-box should be in contact with a sleeve sitting on the rotating shaft. Due to the rotation of the shaft, and hence the rotation of the sleeve, frictional heat is generated. The heat is handled by a water cooling system. The sealing system enters state 0 and remains there until the packaging is worn out. The result would be that air starts flowing out of the generator. The function "prevent airflow" is no longer fulfilled. The system shifts to state 1, and as the function is continuously called, the system directly shifts to state 2. State 0 can only be reentered, if the packaging is replaced.

5 Conclusion

As human life becomes more dependant on mechatronic systems, their potential to cause harm when they malfunction increases. This risk can balance out all their benefits. Hence, understanding the failure behavior becomes an important step in designing fault tolerant mechatronic systems. Since various fault detection schemes exist, depending on whether the system is online or offline, it is important to see to what state did a certain fault drive the system. This model helps us recognize where a system's state is with respect to a certain fault, in order to decide how to manage the fault's detection and correction mechanism to establish fault tolerance. Moreover, the model helps in the debugging process, to detect inherent design errors. During design debugging or validation, special operational conditions could be applied, in order to force the system into the failure state and then observe the behavior of the system, thus exposing any inherent bugs. This could prove beneficial for software, hardware electronics, and even mechanics, especially during testing, as most of the time bugs are inherent to the system, and require special operational conditions to be exposed.

What needs to be done further is to extend the model using a hierarchical trend in order to account for some fault tolerance means such as redundancy.

References

[Av97] Avizienis, A.: Toward Systematic Design of Fault-Tolerant Systems, Computer, Vol. 30, Issue 4, April 1997

[Bi04] Birolini, A.: Reliability Engineering - theory and practice, 4ed., Springer, Berlin, 2004.

[Bi86] Bitter, P. et al.: Technische Zuverlässigkeit, 3 Auflage, Springer-Verlag, München, 1986.

[DIN95] Deutsches Institut für Normung: Internationales Elektrotechnisches Wörterbuch, Kapitel 191-Zuverlässigkeit und Dienstgüte, Beuth Verlag, Berlin 1995

[GM03] Gausemeier, J.; Möhringer, S.: Die neue Richtlinie VDI 2206 Entwicklungsmethodik für mechatronische Systeme, VDI Berichte 1753, 2003.

[La92] Laprie, J. C.: Dependability - Basic Concepts and Terminology: IFIP WG 10.4 Dependable Computing and Fault Tolerant Systems, Vol. 5, Springer-Verlag, Vienna, 1992.

[Li96] Lions, J. L.: ARIANE 5 - Flight 501 Failure, Centre National d'Etudes Spatiales & European Space Agency, Paris, 1996.

[Pa03] Pahl, G. et al.: Konstruktionslehre Grundlage erfolgreicher Produktentwicklung Methoden und Anwendungen, 5. Auflage, Springer, 2003.

[Pr96] Pradhan, D. K.: Fault-Tolerant Computer System Design, Prentice Hall, New Jersey, 1996.

[SB94] Sharangpani, H. P.; Barton, M. L.: Statistical Analysis of Floating Point Flaw in the Pentium ™ Processors, Intel White Paper, 1994.

[VDA00] Verband der Automobilindustrie e.V.: Qualitätsmanagement in der Automobilindustrie: Zuverlässigkeitssicherung bei Automobilherstellern und Lieferanten, Teil 2, 2000.

[VDI86] Verein Deutsche Ingenieure: Richtlinie 4004 Blatt 2: Zuverlässigkeitskenngrößen: Überlebenskenngrößen, 1986.

[VP93] Vaidya, N. H.; Pradhan D. K.: Fault-Tolerant Design Strategies for High Reliability and Saftey: IEEE Transactions on Computers, Vol. 42, No. 10, Oct. 1993.