Use of diversity for a fail-safe computing platform

Andreas Gerstinger
Institute of Computer Technology, University of Vienna, Austria
gerstinger@ict.tuwien.ac.at

Abstract

Design diversity has been discussed in depth for at least three decades. There is a consensus on what benefit can be achieved and what cannot be achieved. In the light of modern computer systems, whose complexity is growing, and where increasingly automated tools are used for system generation, there is more potential for the successful use of diversity in its many forms. Diversity is even one of the most effective means to tolerate or detect some types of systematic faults. This potential can be shown especially well along the failure chain. A fail-safe computing platform serves as the reference system.

1. Introduction

Design Diversity [1] has been propagated as an effective means to detect and tolerate systematic faults. The assumption is, that if two or more versions of some component are developed, the faults contained in the component are different, so that the likelihood of both or all components failing is reduced. Opinions against design diversity have also been voiced quite strongly [2], but finally a generally accepted consensus on what can and what cannot be achieved with design diversity is available [3].

Computer systems evolve and it can be observed that complexity rises and the number of abstraction layers within a system increases. Indeterminism also rises in modern systems, for example it is harder to predict the exact amount of cycles an instruction needs or the exact time or sequence when tasks are scheduled. The number of possible execution paths in complex systems increases. Tools for system development are used increasingly, from compilers to high level code generators. As the testing process is improved, only the hardest to detect faults remain in systems, which only occur under rare circumstances [4]. Within this rising complexity, systematic faults which appear to occur randomly are one of the most prominent causes influencing safety. It is a well known fact that software – despite rigorous development processes – contains a large number of faults [5].

In the light of this, there is new potential for the use of diversity in many forms. We will systematically evaluate all forms of diversity based on the failure chain.

2. System Domain and Model

One system class that is particularly suitable for the use of diversity, are systems with a fail safe state, i.e. a system which has a static set of output parameters which are always safe. Some representatives of these systems are:

- Shutdown systems (the safe state is to shut down the reactor)
- Railway control systems (the safe state is to stop all traffic)
- Anti-lock brakes (the safe state is to switch back to pure mechanical braking)

Obviously reliability is also of concern, since one generally wants to minimize the cases where one unnecessarily goes into the fail safe state, but this is not within the scope of this paper.

We can also view such a system as a "black box" with input and output and an internal state. The output depends solely on the combination of input and the internal state. The combination of input and internal state will be called demand (as in [3]).

The single hazard of such a system is the output of undetectably incorrect data on some output interface.

3. Fault and Failure Model

The fault and failure model from [6] will be used to illustrate the possible introduction of diversity. This model describes the failure process as a sequence from faults which lead to errors in the system state, which in turn lead to failures.

Note that once a system is produced, the occurrence of a failure depends on two facts:

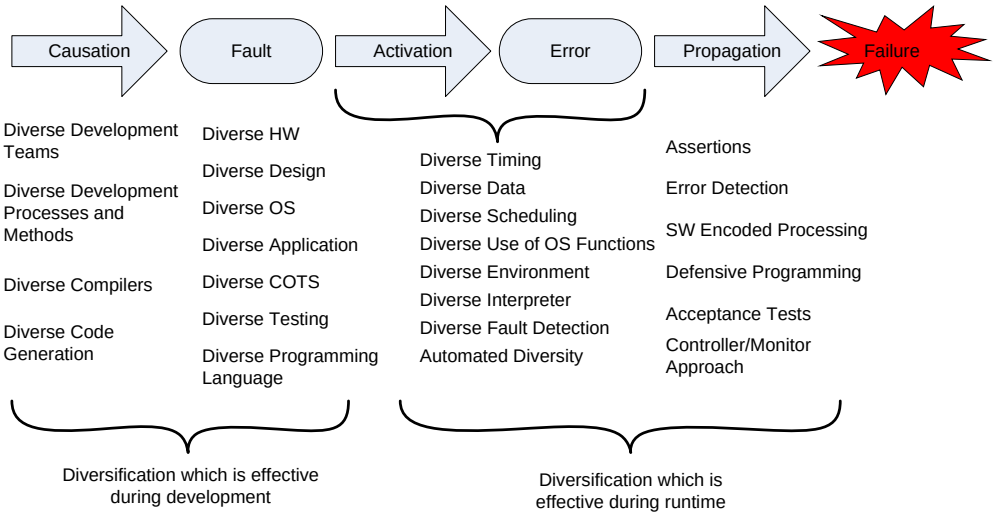

Figure 1 – Diversification methods to diversify the failure chain

(1) Presence of Faults. If no faults are present, no failures can occur. This can only be influenced during the design and development of the system.

(2) Demand: It solely depends on the demand if the faults are turned into failures. This can be influenced during runtime of the system.

4. Potential for Diversity

As we know faults and resulting failures cannot be completely avoided, the prime goal that we want to achieve with a diverse system is *failure diversity*. Even if the outputs of all channels are incorrect (i.e. they fail), this can be detected as long as the failures are diverse.

In order to achieve failure diversity, the failure chain (Figure 1) has to be diversified, so that – in case of failure – diverse failures provoked. We can start by introducing diversity during the development process to create diverse faults. Whenever already existing components are used (such as COTS – commercial off the shelf – components), we can achieve diversity by diversifying them. Diversification can also be introduced so that similar faults provoke different errors. Finally, due to the fact that a fail-safe state exists, errors can be detected so that they lead to different failures (e.g. a crash failure, which is not a hazard in a system with a fail-safe state).

5. Conclusion and Outlook

Figure 1 shows that diversity can be introduced along the failure chain. In previous diversity research, most focus was put on generating variants with diverse faults, i.e. diversity during system development. We believe that diversity which is effective at runtime, i.e. turning the same existing faults into diverse failures, deserves more attention, especially in the light of rising complexity, where many failures only occur in rare circumstances which are difficult to reproduce. At the same time, these systems offer – due to their complexity – many more possibilities for the introduction of diversity, which can even be automated is some cases.

This work will be extended so that the fault categories covered which each type of diversity will be made more explicit. This shall enable system designers to choose the diversity methods most appropriate for the expected types of faults.

References

[1] P. Bishop. Software Fault Tolerance by Design Diversity. In M. Lyu (ed.) "Software Fault Tolerance". Wiley, 1995.

[2] J. Knight, N. Leveson. An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming. IEEE Trans. on SWE, Vol. 12(1), Jan. 1996.

[3] B. Littlewood, L. Strigini. A discussion of practices for enhancing diversity in software designs. DISPO Project. Nov. 2000.

[4] M. Grottke and K. S. Trivedi. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. IEEE Computer, June 2007.

[5] B. Randell. Turing Memorial Lecture: Facing up to Faults. The Computer Journal, Vol 43, No 2. 2000.

[6] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. on Dependable and Secure Computing. Vol. 1, Iss. 1. Jan 2004.