26th International System Safety Conference

Hosted by the System Safety Society • Sheraton Wall Centre • Vancouver, Canada

25 - 29 August 2008

Sponsors

Technical Papers

Posters

Search

Conference Proceedings

The Next Generation of System Safety Professionals

Editor: Rodney J. Simmons, Ph.D., CSP Associate Editors: David J. Mohan and Monica Mullane

ISBN: 0-9721385-8-7

Diversity for Safety and Security Improvement

Andreas Gerstinger; Institute of Computer Technology; University of Technology, Vienna, Austria

Thomas Novak; Institute of Computer Technology; University of Technology, Vienna, Austria

Keywords: safety, security, diversity

Abstract

Design diversity has been discussed in depth for at least three decades. There now seems to be a broadly accepted consensus on what benefit can be achieved and what cannot be achieved. However, the ideas behind diversity have much more potential than what is expected at first glance. This paper shows, that safety-critical systems as well as security-critical systems might benefit from various variations of diversity. We will show methods and approaches how and where to introduce diversity, which is much broader than pure design diversity, and show how the safety integrity as well as the resistance to security issues can be improved by these means. This topic is of high interest, as safety-critical systems are increasingly being used in security-critical domains, which is for example always the case if a system is connected to a public or private network. On the other hand, modern systems whose complexity is by orders of magnitude bigger than older systems, provide more possibilities for the introduction of diversity.

Introduction

Safety-critical system development is faced with many challenges today. Embedded computer systems with safety-responsibility are becoming more and more ubiquitous, and the range of potential applications and ideas for new applications seems to increase almost daily. At the same time, society has high expectations for safety-critical systems.

First and foremost, safety-critical systems are expected to be sufficiently safe, so that the remaining risk is acceptable. Risk acceptability is a complex issue [Nor01] and will not be discussed here, but trends indicate that risk acceptability varies regionally and that risk acceptance decreases over time. Therefore, safety-requirements are becoming more stringent over time.

Security and safety are becoming increasingly dependent on each other. Security consciousness has especially increased in the last decade for various reasons, such as the threat of terrorism and frequent reports of intrusions into vital computer systems. Additionally, systems are networked intensively and therefore such threats result in a high risk. It is also recognized that safety and security should not be segregated since both domains deal with risk reduction, but that an integrative approach is beneficial [Nov&07].

Technology changes, computer systems are becoming increasingly complex and powerful, and at the same time smaller in physical size. Therefore, our expectations regarding the functionality also increases, and more and more functionality is put into single systems. This does not make safety certification and security evaluation easier.

Finally, to remain competitive, safety-critical and security-critical systems developers are faced with economic challenges. Although financial considerations should never compromise the target level of safety and security, each company has to find ways to achieve the required levels and remain competitive at the same time.

Challenges in Safety Critical Systems Development Today

Apart from these general challenges, there are more concrete technical challenges which must be faced. These challenges are introduced here.

Compliance to Safety Standards: Safety-critical system design is governed by a large number of standards and guidelines. The situation concerning standards for safety-critical systems is today similar to the situation concerning software development standards. In this area, it has already been detected that there are too many,

sometimes even conflicting, guidelines. This situation has been suitably termed as a "quagmire" [She01]. A similar quagmire can be detected in the safety-critical standards area (Figure 1).

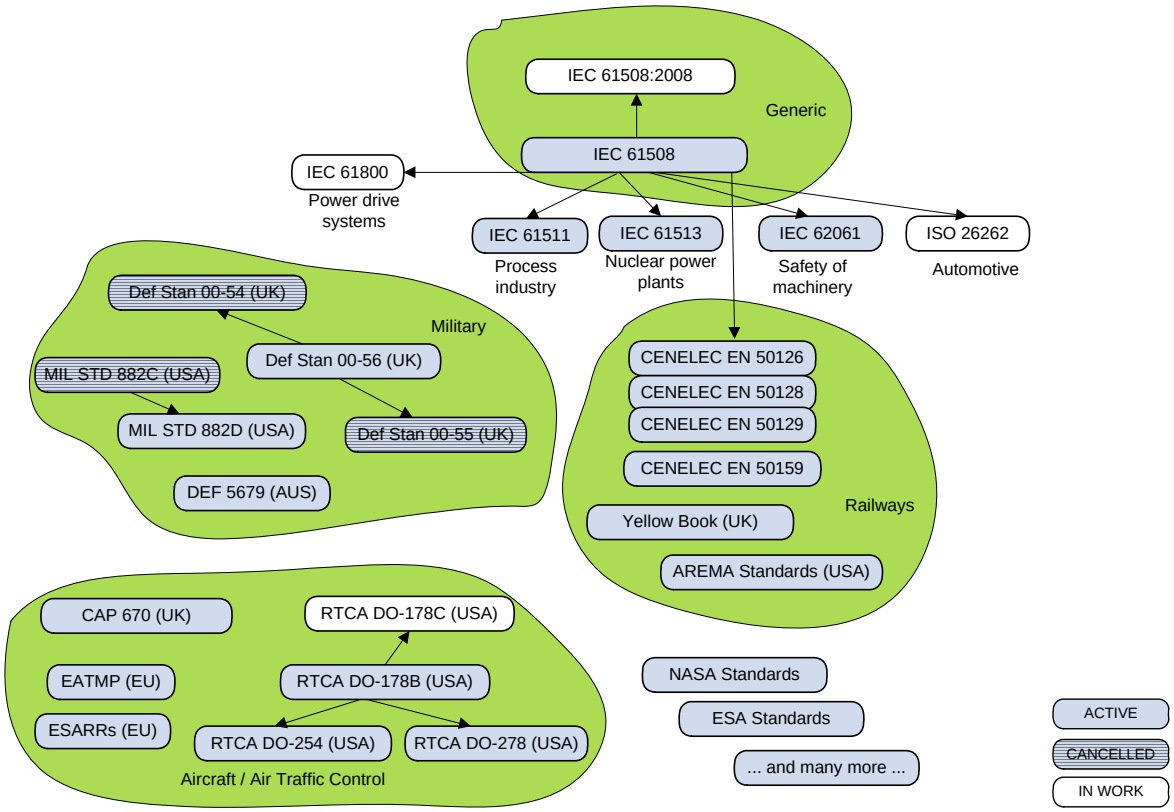

Figure 1 — Safety Standards Quagmire

The problem is that standards vary along several dimensions. First, each *domain* – railways, aircraft, air traffic control, nuclear sector, etc. – seems to have their own standards. Secondly, each *country or region* has their own standards. And thirdly, standards vary over *time*, they are regularly updated and requirements change. This is the reason for the existence of so many different standards. Nevertheless, there are many principles that are shared by the standards. For example, the requirement that a sound development lifecycle has to be followed is shared by practically all frameworks. Differences are then contained in the details, such as if it is required to use formal methods for a high criticality system. There is no consensus on this and many similar issues [McD&01].

Increasing Complexity: Complexity of computer systems has increased in the last decades massively (see e.g. [Hol07]). Although too much complexity is often an indication of bad system design, there is some level below that the complexity of a system cannot be reduced, if it shall fulfill a specific task. So, the problem that shall be solved by a system prescribes a certain lower limit on the complexity of the system. An arbitrarily complex problem cannot be solved by an arbitrarily simple system. This lower limit on the system complexity increases with the problem complexity. It follows that as we try to solve increasingly complex problems with computer systems, the complexity of these systems will also grow.

It follows that it is becoming increasingly difficult for a single person to understand every aspect of a computer system. A programmer does not need to understand every detail of the hardware architecture of the CPU, nor does the CPU designer have to understand every aspect of object oriented development. The interfaces between the components of the systems become more important – it is essential that they are clearly defined.

Usage of "Off The Shelf" Components: Increasing complexity leads to the fact that reusing previously developed software and hardware components is becoming increasingly important. Even in the safety-critical community it is impossible to redesign everything from scratch for each application. The usage of COTS (commercial off the shelf) components is becoming increasingly popular also in safety-critical projects, but has also provoked many discussions about their suitability (see for example [Bat&01] for the use of COTS CPUs and [Lin&00] for COTS software).

The conclusions that are drawn in research about off the shelf components range from [Lin&00] where it is stated that "the general consensus seems to be that COTS should not be used for components with a safety integrity level greater than 2", to [Pro&96], where an approach is presented how arbitrary off-the-shelf components can be used even for the highest safety integrity.

Indeterminism: It is now a well-established fact that indeterminism in computer systems has increased over the last years. This is due to the increased complexity described above, as well as the continuous improvement of their (average) performance. The increased indeterminism is mainly introduced by the hardware as well as operating systems.

Issues that increase this indeterminism in hardware are the execution order of instructions, the caches, the varying clock rate and the quality of branch prediction. In the operating system, indeterminism is increased due to features such as flexible scheduling policies, asynchronous interrupts and memory swapping mechanisms.

This indeterminism makes analysis more tedious. For example worst case execution time analysis is complex [Kir&07]. On the other hand, this indeterminism also shows that it is relatively easy to introduce differences or variation in systems. This artificially introduced variation can also be considered as a type of diversity. If a certain failure is due to a special combination of internal states, it may well be that this special combination exists only in one system, and a redundant – and diverse – system will not exhibit this failure.

Importance of Security Aspects: Security Aspects are becoming increasingly important in safety-critical system development. As soon as a safety-critical system is connected to some network, security issues emerge, which must be analyzed and dealt with accordingly. Security and safety considerations can go hand in hand, such as the introduction of a password protection, such that unauthorized users cannot change potentially safety-related parameters in an operational system. On the other hand, a password protection may in some cases decrease safety, e.g. in the case of an emergency when a legitimate operator needs access to a system as quickly as possible.

The relevance of security aspects in safety-critical system is well known now (see e.g. [Nor08]), which is also reflected by the fact that security plays a role in safety standards (e.g. IEC61508 which is currently being revised will contain a full appendix on security). A harmonized safety and security lifecycle is proposed in [Nov&07].

Changes in Dominant Fault Types: The faults which safety-critical system developers are confronted with today can be classified according to many dimensions, such as the ones defined in [Avi&04].

Random faults which occur in hardware can generally only be predicted with statistics – it cannot be known when exactly a resistor fails, but we can easily make quite accurate predictions about its reliability in terms of mean values, such as the mean time between failures. The occurrence of random faults in more complex components, such as a CPU or memory are also predictable, but its effects are less predictable – the effect may range from "no effect" to incorrect calculations or total crash. Certain classes which may become more common in future are transient faults in highly integrated components, such as e.g. "bit-flips" in memory. They may have devastating effects in safety-critical systems, if the flipped bit represents some vital safety-related information. The reason why these faults may become more common are manifold, such as the lower voltage levels used and continuing miniaturization leading to higher integration density.

Systematic faults in software and hardware are also quite well researched, and it is a mater of fact that complex systems and especially software-based complex systems contain faults. Predictions about their frequency are much harder than with random faults. Some studies give numbers on how many faults are contained per 1000 lines of code [Wan&05]. What can be inferred is that systematic faults can hardly be eliminated totally, and that even a

system developed according the best engineering standards may contain a large number of systematic faults. Nevertheless, advanced development and verification and validation methods have the potential to effectively fight this class of faults.

A class of faults which is in the middle between the extremes of random and systematic faults is the class of faults which are systematic in their cause (e.g. a programming bug) but which seem to occur randomly. We call these faults quasirandom faults or Heisenbugs, since from a phenomenological point of view they are random, but their root cause is systematic. Typical examples of these faults are race conditions or memory leaks in software-based systems. They occur only under very rare and exceptional circumstances, and are hardly ever reproducible. They are hardly caught by traditional verification and validation methods. On the other hand, their likelihood increases with system complexity. Therefore, we must face this class of faults in complex systems, and devise effective means against them.

Introduction to Diversity

Design Diversity ([Bis95]) 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 ([Kni&86]), but finally a generally accepted consensus on what can and what cannot be achieved with design diversity is available ([Lit&00]).

The general consensus is that diversely developed systems are not truly independent. Their faults are still correlated, which is because humans tend to make mistakes at similar tasks. For example, if software is developed, it is more likely that the software fails on some rare and "difficult" inputs than on a standard input. Theoretical models have also been developed to show this varying "difficulty" ([Lit&01]). However, although independence cannot be assumed, the benefit of diversity – if applied correctly – is also generally agreed. It is simply a question of "cost vs. benefit", i.e. is the cost of diversity smaller or greater than the benefit that is expected ([Lit&00]).

How Diversity Can Help to Meet the Challenges

We do not simply want to propagate diversity here, but we would like to show that the challenges mentioned above seem to favor the use of diversity. Several arguments can be derived:

Number of Abstraction Levels: The increasing complexity of systems leads to the fact that the number of abstraction layers in a system also increases. Four basic layers which are often present are the hardware, the operating system, some middleware and the application. Often, additional layers can be identified, such as a hardware abstraction layer or a runtime framework for applications as used by the programming language Java.

Diversity was focused primarily on the most obvious targets, i.e. application layer and on the hardware, with the other layers not being considered for diversification. As the number of layers increases, the potential and the possibilities to introduce diversity also increase. Operating systems and middleware such as application frameworks are viable targets for diversification, not just to tolerate and detect their faults but also other faults.

So, the increased complexity here goes hand in hand with the increased potential for diversification, suggesting that complex systems are a suitable target for the use of diversity.

High system complexity --> Systems have many layers --> More potential to introduce diversity

Indeterminism Increases: The increasing complexity also leads to increased indeterminism as discussed above. This means that slight variations which do not affect the result of an operation can lead to significant differences in how some operation is performed internally. These may be different execution sequences, use of different resources, use of different data, etc. It may be that some internal states give rise to failures, wherever if the same operation is re-executed with slight variations, no failure occurs. A sufficiently complex computer system may never be in the same state during its complete lifetime. The occurrence of failures depends on the state and input to a system, so a

system may never be confronted with the same input and state combination during its lifetime. If the inputs and states of a system are sufficiently different, the chances that they lead to different failure behavior also increase.

High system complexity --> Increased indeterminism --> Small variations in input --> large differences in internal state --> "Easier" to introduce diversity

Automated Translations Getting More Common: Whenever some translation of an artifact to a representation on a lower abstraction level takes place, it is possible to introduce diversity. Most of those translations are performed manually. A specification is translated to the design, which is translated to an implementation. These translations can be performed in many different ways, e.g. a specification can be translated to many different (and all equally correct) designs. Translating a given common specification to more than one design is called design diversity. However, these translations are increasingly carried out automatically with the help of tools. The one step that is virtually always tool-based is the compilation from a high level language to machine code. However, more than one of these automatic translations are often used, such as the translation from a processor independent byte code to machine code (as in Java) or the translation from a design representation to source code (as can be done automatically e.g. with state diagrams in the Unified Modeling Language). All of these automated translation steps increase the potential to introduce diversity automatically at various points in time.

Automated translations becoming more common --> More potential to introduce diversity automatically

Dominant Fault Types Change: As shown in above, for various reasons the large number of quasirandom faults in today's systems must not to be neglected. This is especially true for the types of complex computer-based systems. Verification and validation, i.e. testing, is good at catching systematic faults, but quasirandom faults typically only develop into failures in operational systems. Therefore it is especially important to provide defenses against this fault class, and diversity is likely to be a very efficient defense against these faults. Again, high complexity causes this fault class to be prominent, but this complexity also provides more possibilities for the introduction of diversity, which in turn is an effective measure against quasirandom faults. This slightly intricate relationship can be visualized as shown in Figure 2.

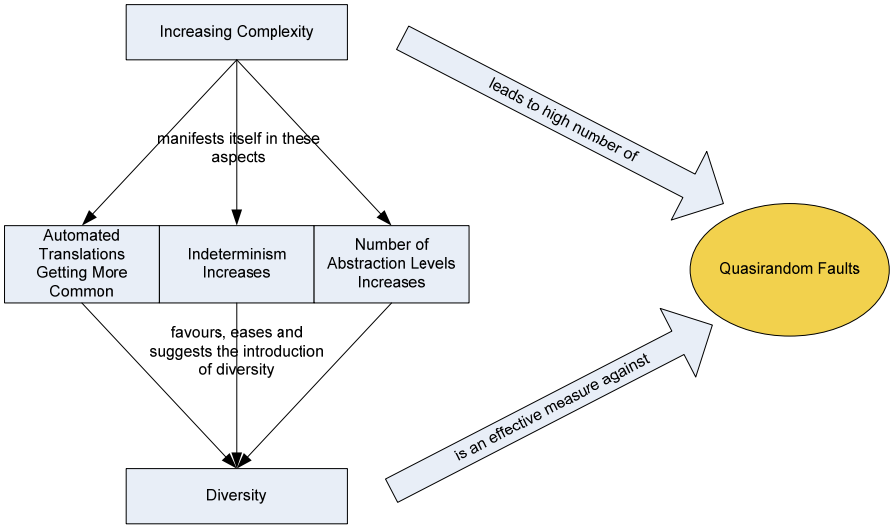

Figure 2 — Relationship between Complexity, Diversity and Quasirandom Faults

Redundancy: Redundancies are often available anyway for various reasons, such as to increase reliability in case of failure of one system. Therefore, it makes sense to make redundant components diverse if this is easily achievable, to increase their resilience against design faults. The introduced diversity can be very easy to achieve. Consider a redundant computer system consisting of two computing channels with two software applications. One can imagine diversifying these two applications simply by using two different compilers. With this method, we have a measure against compiler faults, which comes at almost no additional cost.

Finally, diversity is the only method which can detect or tolerate design faults *during operation*. All other methods against design faults are only effective during development. However, it is a matter of fact that design faults are made and systems *are* faulty, even if they are developed according to the most rigorous standards and with the best methods available. Therefore, the detection and tolerance of design faults during operation is necessary and useful.

Diversity Introduction and Effect Model

We have indicated that there are many ways to introduce diversity. It will now be shown more specifically how this can be done. As a basis for the introduction of diversity, the failure chain from [Avi&04] will be used. The goal for our purposes is to avoid failures. According to the failure chain model, the occurrence of failures in an existing system depends solely on two issues:

(1) Presence of Faults. If no faults are present, no failures can occur. The number of faults present is correlated with the number of failures which occur (but one fault can cause many failures and some faults may never cause a failure). The number and nature of faults can be influenced during the design and development of the system.

(2) Inputs and Environment: It solely depends on the inputs and environment of the system if the faults which are present are turned into failures. The internal state of the system and the input to the system can be influenced during the runtime of the system.

The first point to break the failure chain is obviously at the fault causation. If the cause for a fault is removed, the failure chain is broken at the very beginning. Typically, this category contains all fault avoidance measures which one performs during the design and development of a system. Also measures to avoid random faults in hardware can be counted in this category, such as the adherence to specified environmental conditions, in order to avoid these fault causes.

If already existing products are selected for two diverse channels (such as a database component or a library), it would be a possibility to select two diverse products which likely possess diverse faults. Diverse verification and validation strategies during the development also lead to a reduction and diversity of faults, i.e. the failure chain is broken or diversified.

As stated earlier, faults will still be present in the system. Therefore, the goal must now be that the faults are either not activated in both channels, or that they are activated differently so that diverse errors result. In this case, the failure chain "diverges" (i.e. branches) at the error activation state and the likelihood of a diverse failure is increased. An example would be the use of a diverse timing – a race condition fault might not be activated with different timings.

In order to break or diversify the failure chain at the error activation state, methods such as assertions, defensive programming or a controller/monitor approach may be suitable. The illustration of the introduction of diversity along the failure chain opens the way to a wide spectrum of possibilities for diversity. When considering the possibilities to break or diversify the failure chain, it is evident that diversity has a broad application spectrum, much broader than pure design diversity.

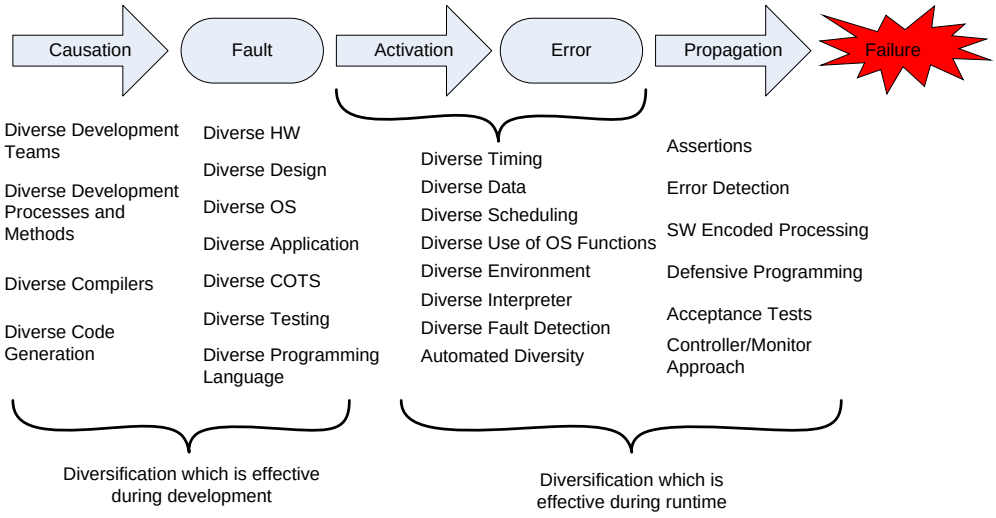

Figure 3 — Examples how to break or diversify the failure chain

Development diversity is the most obvious way to introduce diversity – this can have the form of either additional channels which perform some operations in a different way, or a diverse monitoring channel. These decisions during development can be of various nature, such as to develop two separate specifications in different notations, use diverse development methods for the design stage, diverse programming languages during implementation, diverse tools during any of the stages, etc. All these decisions have some impact on the diversity of the developed product and they diversify the fault causes. Therefore, a certain "fault diversity" is expected in the product. The expectation is, that the faults in one channel are – to a certain extent – dissimilar to the faults in the diverse channels. Based on the failure chain considerations (Figure 3), this fault diversity increases the possibility that failures are also diverse – which is the actual goal.

The term "assurance" as it is used here subsumes all measures taken to assure that the system is safe to be used. This includes all verification and validation measures, such as all tests, as well as static verification measures such as correctness checks and reviews. It also includes all specific safety activities, such as the generation of a rigorous and sound safety case containing convincing arguments. All these assurance measures can also benefit from diversity: first, because if using diverse methods, it is more likely that more faults are detected and corrected. Each assurance method has its inherent strengths and weaknesses, and using diverse methods helps to exploit all their strengths combined. Similarly, if diverse channels are developed and for each channel the assurance methods also vary, this further increases their diversity. Therefore, diverse assurance is expected to lead to a reduction of faults as well as a further diversification of faults.

It was shown that the failure chain can also be "broken" or "diversified" at the error activation and failure propagation stage with the help of diversification. In order to achieve this, the runtime environment has to be diversified, so that different faults are activated or faults are activated differently – this is diversity during the operation phase. Therefore, runtime diversity includes all measures that do not influence the fault diversity in the product, but which influence the evolution of faults into failures. This can include presenting the system to different environments, different input data or different timings. Runtime diversity can either contribute to diverse internal errors in the product or to diverse failure occurrences.

Figure 4 shows these three anchor points where diversity can be introduced and how they affect the development of a fault into a failure.

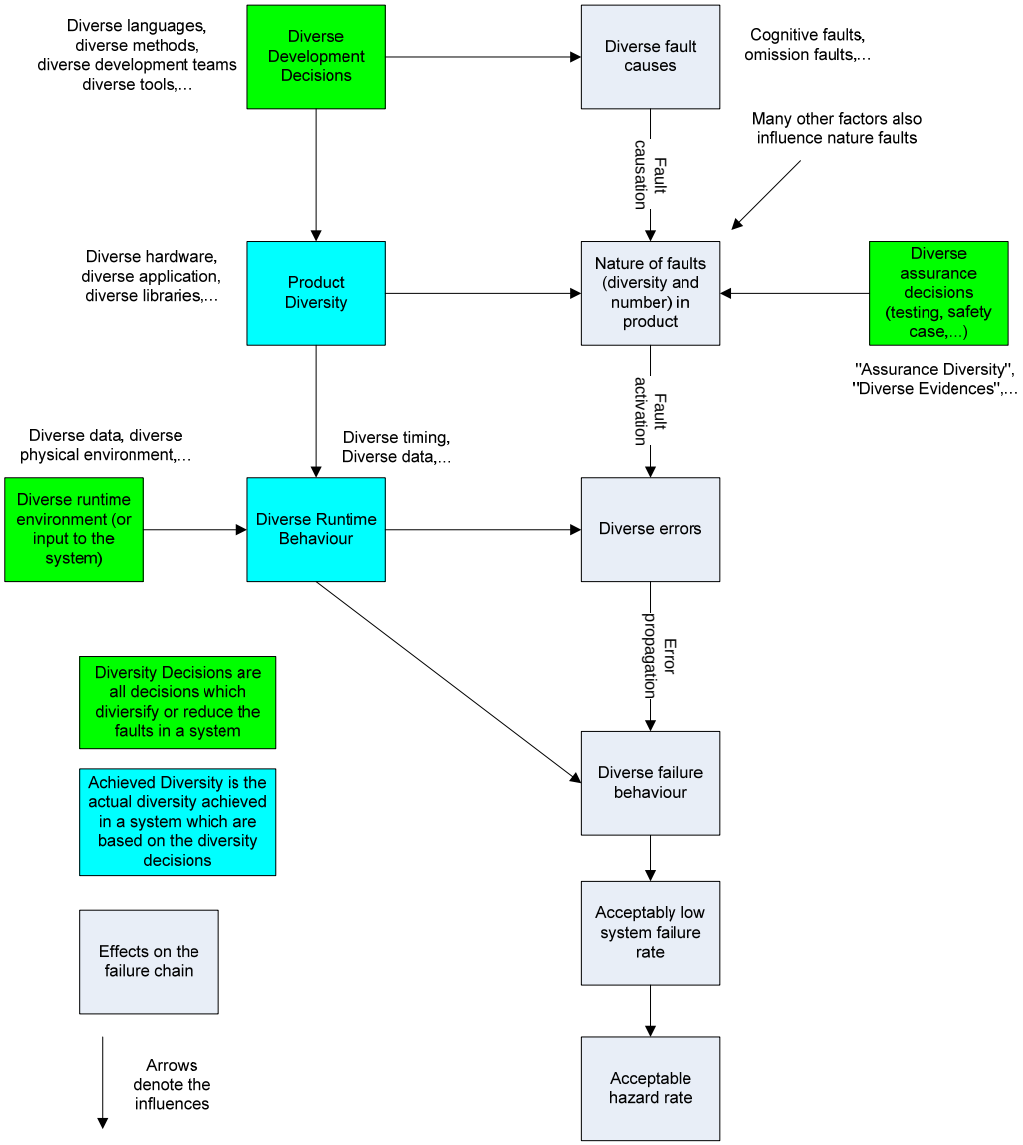

Figure 4 — Diversity Introduction and Effect Model

Diversity for Security

Diversity has the potential to increase resistance against security attacks. This is best summarized by the statement in [Sai04]: "Diversity is a successful strategy that has kept humans alive. For instance, every human has a different immune system, so some will resist viruses that others won't. It's easier to run a network where all the systems are the same, but if you have a network with a lot of different operating systems, platforms and application software, only part of it will be vulnerable to specific attacks."

Attacks against the security of a system share some properties with systematic software faults. Security attacks and software faults have some deterministic cause. In one case it is a fault, in the other case it is a certain intention. the root causes are deterministic, but the process when and how these root causes have an effect is not known and appears to be random. The only difference is that security attacks are intentional and performed due to a malicious motivation, but software faults are unintentional.

The similarities encourage to treat security attacks similarly to software faults. Some examples are:

- The use of diverse operating systems. Operating systems are a critical part of a system, since they generally have power over the complete hardware, and a failure of an operating system leads to the total loss of the computer system. Planned attacks often use (known or still unknown) vulnerabilities to intrude into a system. A redundant system configuration which uses different operating systems makes attacks much harder, since the potential attacker has to find vulnerabilities of both systems at the same time.
- The selection of diverse application or middleware software. Diverse application software or middleware also protects against security problems in the same way as operating systems. [Mon&02] presents a method how this selection can be performed.
- Code Randomization. Software code can be randomized in such a way, that each machine has a totally different pattern of software machine code, such that the common class of code injection attacks (a very common class of security problems which is often achieved by the exploitation of buffer overflows) is ineffective. With this approach, the machine code of all machines are so diverse, that a code injection attacks works on one machine only. Viruses and worms have no chance to propagate.
- Diverse intrusion detection mechanisms. The use of diverse intrusion detection mechanisms helps to identify potential attackers effectively. A single intrusion detection system generally has its weaknesses. It is expected that by the combination of several sufficiently diverse intrusion detection systems, the common weaknesses are minimized.

Conclusion

This paper has presented the challenges faced by developers of safety-critical computer systems. It was then argued, that the use of diversity in various forms is a possible way to meet these challenges. By thinking about diversity in a broader sense instead of pure design diversity to avoid common faults, new ways open up, which show how diversity can be employed to provide a substantial safety benefit. Most of these new ways do not require a manual generation of a diverse software channel, which suggests that they can be employed cost effectively. Security – which is becoming more and more related to safety, as systems must often be safe and secure at the same time – can also be increased by methods related to diversity.

There are still open questions – some of the benefits of employing diversity have been described in this paper. However, it is a topic area that still provides many open questions, which deserve to be adequately addressed.

References

[Avi&04] Algirdas Avizienis, Jean-Claude; Laprie, Brian Randell, Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing. Volume 1, Issue 1, Jan 2004.

[Bat&01] Iain Bate, Philippa Conmy, Tim Kelly and John McDermid. Use of Modern Processors in Safety-Critical Applications. The Computer Journal 44(6):531-543. 2001.

[Bis95] Peter Bishop. Software Fault Tolerance by Design Diversity. In M. Lyu (ed.) "Software Fault Tolerance". Wiley, 1995.

[Hol07] Gerard J. Holzmann. Conquering Complexity. IEEE Computer. December 2007.

[Kir&07] Raimund Kirner, Peter Puschner. Time-Predictable Task Preemption for Real-Time Systems with Direct-Mapped Instruction Cache. 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07) pp. 87-93. 2007.

[Kni&86] John C. Knight, Nancy G. Leveson. An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming. IEEE Transactions on Software Engineering, Volume 12(1), January 1986.

[Lin&00] Peter Lindsay, Graeme Smith. Safety Assurance of COTS Software. Technical Report no. 00-17, Software Verification Research Centre, School of Information Technology, The University of Queensland. 2000.

[Lit&00] Bev Littlewood, Lorenzo Strigini. A discussion of practices for enhancing diversity in software designs. DISPO project technical report, Centre for Software Reliability, City University, 2000..

[Lit&01] Bev Littlewood, Peter Popov, Lorenzo Strigini. Modelling Software Design Diversity – A Review. ACM Computing Surveys, Vol. 33, No. 2, June 2001.

[McD&01] John A McDermid, David J Pumfrey. Software Safety: Why is there no Consensus? Proceedings of the 19th International System Safety Conference (ISSC). Huntsville, Alabama, 2001.

[Mon&02] Marco Casassa Mont, Adrian Baldwin, Yolanta Beres, Keith Harrison, Martin Sadler, Simon Shiu. Towards Diversity of COTS Software Applications: Reducing Risks of Widespread Faults and Attacks. Hewlett-Packard Company 2002.

[Nov&07] Thomas Novak, Albert Treytl, Peter Palensky. Common Approach to Functional Safety and System Security in Building Automation and Control Systems. In Proceedings of the 12th IEEE International Conference on Emerging Technologies and Factory Automation, pp. 1141-1148, 2007.

[Nor01] Odd Nordland. When is risk acceptable? Proceedings of the 19th International System Safety Conference (ISSC). Huntsville, Alabama, 2001.

[Nor08] Odd Nordland. Making Safe Software Secure. In "Improvements in System Safety: Proceedings of the Sixteenth Safety-critical Systems Symposium". Springer-Verlag, 2008.

[Pro&96] Joseph A. Profeta III, Nikos P. Andrianos, Bing Yu, Barry W. Johnson, Todd A. DeLong, David Guaspari, Damir Jamsek. Safety-Critical Systems Built with COTS. IEEE Computer, November 1996.

[Sai04] Anne Saita. BIOLOGY: Back to Nature? Information Security Magazine, (NewSCAN). TechTarget. July 2004.

[She01] Sarah A. Sheard. Evolution of the Frameworks. Quagmire, IEEE Computer, July 2001.

[Wan&05] L. Wang, K. Ch. Tan. Software Testing for Safety-Critical Applications. IEEE Instrumentation & Measurement Magazine, pp. 38-45, June 2005.

Biography

Andreas Gerstinger, Research Assistant, Institute of Computer Technology, Vienna University of Technology, Vienna, Austria. Telephone +43-1-58801 38457, e-mail: gerstinger@ict.tuwien.ac.at.

Andreas Gerstinger is a research assistant at the Institute of Computer Technology, which is part of the University of Technology in Vienna, Austria. He works in a project which is sponsored by two companies which both build systems for safety-critical domains. In this function, he is on the interface between academia and industry. Before that, he has worked for six years as a technical lead for safety and software quality at the Frequentis corporation. He holds a master's degree in computer science and telecommunication.

Thomas Novak, Research Assistant, Institute of Computer Technology, Vienna University of Technology, Vienna, Austria. Telephone +43-1-58801 3827 , e-mail: novakt@ict.tuwien.ac.at.

Thomas Novak was born in Vienna, Austria and received a master’s degree in electrical engineering from the Vienna University of Technology in Austria in 2005. His major field of study is functional safety and security in building automation. He is working as project assistant at the Institute of Computer Technology, Vienna University of Technology.
He is chair member of the Austrian Smart Card Association (ASA) and expert in CEN/TC247/WG4. Additionally, he is member of the IEEE IES technical committee building automation, control and management (TC BACM).